



Quick answers to common problems

Oracle SOA Suite 11g Performance Tuning Cookbook

Over 100 recipes to get the best performance from your
Oracle SOA Suite 11g infrastructure

Matt Brasier **Nicholas Wright**

[PACKT] enterprise 
PUBLISHING professional expertise distilled

www.allitebooks.com

Oracle SOA Suite Performance Tuning Cookbook

Over 100 recipes to get the best performance from
your Oracle SOA Suite 11g infrastructure

Matt Brasier

Nicholas Wright



BIRMINGHAM - MUMBAI

Oracle SOA Suite Performance Tuning Cookbook

Copyright © 2013 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the authors, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: July 2013

Production Reference: 1040713

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-84968-884-0

www.packtpub.com

Cover Image by Abhishek Pandey (abhishek.pandey1210@gmail.com)

Credits

Authors

Matt Brasier
Nicholas Wright

Project Coordinators

Abhishek Kori
Wendell Palmer

Reviewers

Sandeep Phukan
Sylvain GROSJEAN
Tumelo Mametsa

Proofreaders

Elinor Perry Smith
Stephen Silk

Acquisition Editors

Grant Mizen
Antony Lowe

Indexer

Hemangini Bari

Lead Technical Editor

Susmita Panda

Graphics

Valentina D'silva
Abhinash Sahu

Technical Editors

Pushpak Poddar
Rikita Poojari
Lubna Shaikh

Production Coordinator

Conidon Miranda

Cover Work

Conidon Miranda

About the Authors

Matt Brasier is a professional services consultant with 10 years of experience tuning Enterprise Java applications. He has 9 years of experience working with Enterprise Java middleware, and has always focused on the performance aspects of application servers and SOA platforms. Matt has spoken at numerous technical conferences, including *The Serverside Java Symposium* and *JUDCon*.

Matt is the head of consulting at C2B2 Consulting Ltd., an independent middleware consultancy specializing in SOA and JEE, providing professional services on the leading Java middleware platforms. C2B2 is an Oracle Gold Partner with Application Grid Specialization.

I would like to thank my partner, Caroline, and our dog, Minstrel, for their understanding while I was writing this book, my father for encouraging me at the start, and C2B2 for giving me the time and experience necessary to do it.

Nicholas Wright is a Java middleware consultant and technical evangelist with 9 years of experience from Embedded C real-time systems to Enterprise Java. He has served on a Java Community Process Expert Group, and spoken at conferences across the world.

Nick is a senior consultant at C2B2 Consulting Ltd., where he works with a wide range of commercial and open source JEE and SOA middleware platforms.

I would like to thank my family, friends, and colleagues for whom I have disappeared off the face of the Earth while working on this book. For your patience and understanding, I thank you. Mostly I must thank my partner, Kay Louise, for putting up with my continual slaving over a computer whenever I have any spare time, love you.

About the Reviewers

Sandeep Phukan has more than 9 years of experience in Integration Technologies and Java. Sandeep currently works as a Senior Consultant with Anatas Technology Services, Sydney, a leader in next generation Enterprise Architecture Solutions. Earlier, Sandeep worked as a Technical Lead for Nokia Siemens Network R&D, Bangalore, assisting on the integration of Next Generation Unified Convergent Billing Systems based on Oracle Fusion Middleware 11g.

Prior to this, Sandeep worked with Oracle SSI, Bangalore as a Senior Consultant, where his key involvement was Oracle AIA for Communications (Telco PIP). During this period he travelled to several countries around the world, aiding customers in developing customizations for Oracle AIA artifacts.

Sandeep is a certified Java Developer and has a keen interest in Low Bandwidth Transports and Distributed Algorithms. During his time at Oracle SSI, Sandeep was actively involved with the SSI Innovation Centre where he created a number of utility extensions to the Oracle SOA Suite. Sandeep also maintains an open source repository of several SOA utilities in the Google code repository <http://code.google.com/p/soalabs/>.

Sandeep is also an avid researcher in the area of SOA Governance. He has published a number of papers in the Oracle Technology Network (OTN) on technology neutral Designtime Dependency analysis of SOA Artifacts. One of his inventions on this subject (Graph Based Designtime Dependency Analysis) has recently been accepted for a Patent in the US Patents and Trademarks Office (USPTO).

I would like to thank my loving wife Aparupa for her patience during those late night researches, and my friends and colleagues at Anatas who were always there whenever I needed them.

Sylvain GROSJEAN is a principal Oracle Fusion Middleware consultant currently working as a freelancer in France.

Over the past 12 years, he collaborated with BEA, and then with Oracle Consulting as a solution architect and trainer helping multiple major French clients (Utilities, Transportation, Insurance, Healthcare, Telecom).

His primary focus is on Oracle Service Bus, Oracle SOA Suite, Oracle BPM, and ADF.

You can follow Sylvain through his own blog at <http://sgrosjean.blogspot.com>

Tumelo Mametsa is an Oracle Fusion Middleware expert currently consulting on various projects in South Africa and other African countries. He has over 10 years of experience in the Electronics and IT fields, respectively, with increasing level of responsibility. He is currently providing senior leadership for solutions architecture, enterprise architecture, business process engineering efforts, including SOA and Open Source Solutions for Africom IT Solutions.

His focus is on engineered systems using various technologies, primarily the Oracle Fusion Middleware Stack. He also provides training for Oracle on Oracle Fusion Middleware Products.

To my extended, supportive, loving Mametsa family (Mpho, Stlaks, Tshepo, Nunu, Letsie, Gontse), thanks for always being there for me. Much love to my darling wife Tsakani and kids, Tumisa, and Seetja. You guys make everything worthwhile. Thanks to my partners at Africom IT Solutions, you are the best team to work with by far. Much appreciation to all my friends (the Zoo) for individually supporting me in your unique ways, without you guys I would not be able to expertly juggle so many balls.

www.PacktPub.com

Support files, eBooks, discount offers and more

You might want to visit www.PacktPub.com for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<http://PacktLib.PacktPub.com>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- ▶ Fully searchable across every book published by Packt
- ▶ Copy and paste, print and bookmark content
- ▶ On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Instant Updates on New Packt Books

Get notified! Find out when new books are published by following [@PacktEnterprise](#) on Twitter, or the *Packt Enterprise* Facebook page.

Table of Contents

Preface	1
Chapter 1: Identifying Problems	5
Introduction	6
Identifying new size problems with jstat	7
Identifying permanent generation problems with jstat	9
Monitoring garbage collection with jstat	10
Identifying locking issues with jstack	12
Identifying performance problems with jstack	14
Identifying performance problems using VisualVM on HotSpot	16
Identifying performance problems using JRMCI on JRockit	21
Using JRockit flight recorder to identify problems	24
Monitoring JDBC connections with the WebLogic console	28
Identifying slow-running database queries	32
Identifying slow-running components with the Enterprise Manager	33
Chapter 2: Monitoring Oracle SOA Suite	35
Introduction	36
Installing the Hyperic server	37
Installing Hyperic agents	42
Configuring Hyperic to monitor SOA Suite 11g	46
Monitoring the SOA Suite server availability	51
Monitoring the JVM memory usage	54
Monitoring the platform CPU usage	58
Monitoring the data source usage	60
Monitoring open sockets	63
Monitoring committed transactions	66
Configuring alerts in Hyperic	69
Monitoring the system using the DMS servlet	72

Chapter 3: Performance Testing	75
Introduction	75
Installing Apache JMeter	77
Creating a web service test using JMeter	78
Running JMeter on multiple servers	81
Checking responses in JMeter tests	84
Monitoring SOA Suite while testing	88
Recording user web sessions with JMeter	92
Designing advanced load tests	94
Running performance tests from the Cloud	98
Chapter 4: JVM Memory	105
Introduction	105
Increasing the JVM heap size	106
Setting Xmx and Xms to the same value	111
Setting the size of the Permanent Generation heap	113
Calculating the total memory used by your application	115
Viewing the memory used using JRMCI for JRockit	119
Viewing the memory used using VisualVM for HotSpot	124
Setting the size of the thread stack	130
Chapter 5: JVM Garbage Collection Tuning	133
Introduction	133
Setting the new size	134
Setting the survivor ratio	136
Choosing a garbage collection algorithm in HotSpot	138
Choosing a garbage collection algorithm in JRockit	140
Turning on verbose garbage collection	142
Tuning to reduce the number of full garbage collections	144
Disabling the RMI garbage collector	148
Disabling explicit GC	149
Chapter 6: Platform Tuning	151
Introduction	152
Tuning global transaction timeouts	153
Increasing the HTTP accept backlog	155
Reducing the server logging level	157
Finding out which JVM you are using	159
Using large pages in Linux	160
Increasing the number of file descriptors in Linux	162
Tuning the SOA Suite EJB timeouts	163
Upgrading to a newer JVM	167

Setting the Linux kernel swappiness to low	169
Using the Oracle JRockit JVM	171
Running your domain in the production mode	173
Creating a boot.properties file	175
Chapter 7: Data Sources and JMS	179
Introduction	179
Setting the data source pool sizes	180
Configuring data source testing	183
Configuring data source growing and shrinking	185
Setting data source connection timeouts	187
Tuning database XA timeouts	188
Tuning connections in the native EIS database adapter	191
Configuring the WebLogic thread pool	194
Using JMS file persistence	197
Tuning JMS connection factories	200
Chapter 8: BPEL and BPMN Engine Tuning	205
Introduction	206
Tuning the dispatcher invoke threads	207
Tuning the dispatcher engine threads	211
Tuning the dispatcher system threads	214
Disabling BPEL monitors and sensors	217
Reducing the audit level	219
Changing a BPEL process to be transient	221
Reducing the completion persist level	223
Disabling persistence on invocation	224
Setting the audit store policy	226
Reducing audit trail size threshold	228
Increasing large document threshold	229
Reducing SOA infra log levels	230
Purging data from the BPEL store	232
Ensuring automatic database statistics gathering is enabled	235
Tuning Oracle database parameters	236
Tuning the Oracle database	239
Chapter 9: Mediator and BAM	247
Introduction	247
Setting Mediator Parallel Metrics Level	248
Setting Mediator Parallel Worker Threads	250
Setting Mediator Parallel Maximum Rows Retrieved	250
Setting Parallel Locker Thread Sleep	251
Using batched delivery in the BAM Adapter	252

Chapter 10: Rules and Human Workflow	255
Introduction	255
Disabling automatic release timers globally	256
Choosing the correct workflow service client	258
Preventing looping and inefficient rule execution	259
Tuning rule execution	263
Chapter 11: SOA Application Design	267
Introduction	267
Using BPEL process parallelization	269
Using non-blocking service invocations in BPEL flows	271
Turning off payload validation and composite state monitoring	273
Designing BPEL processes to reduce persistence	275
Using parallel routing rules in Mediator components	276
Setting HTTP timeouts for external services	278
Tuning BPEL adapter properties	280
Chapter 12: High Performance Configuration	283
Introduction	283
Configuring a cluster of SOA Suite servers	284
Configuring an OHS load balancer	288
Tuning for deployment on a virtualized infrastructure	290
Using distributed JMS destinations in a cluster	292
Using JMS bridges to improve the enqueue speed	297
Choosing deployment hardware	304
Index	307

Preface

Oracle SOA Suite 11g forms the basis of the SOA infrastructure for many organizations. While there is a wealth of information on administering SOA Suite applications already available, there is very little independent information available regarding performance tuning. With the chief goal of software and hardware vendors being to sell more licenses and hardware, the topic of getting the most from your existing investment often falls by the wayside. This book aims to address that by giving clear, concise, and simple recipes that will help you identify performance bottlenecks in your application, and then resolve them.

We work from the ground up, starting with how to identify common SOA Suite 11g performance problems and introducing the tools that you need, and working up the stack through JVM tuning and application server tuning, before covering the BPEL, BPMN, and mediator stacks. Finally, we have chapters on application design and deployment considerations.

What this book covers

Chapter 1, Identifying Problems, starts out by looking at some of the common performance problems that can occur in the Oracle SOA Suite applications, and how to identify them. We look at using a variety of tools to understand what is going on inside the JVM, and to identify the particular resource bottleneck that is causing poor performance.

Chapter 2, Monitoring Oracle SOA Suite, looks at how we can use the VFabric Hyperic tool from VMWare to monitor the performance of our SOA Suite infrastructure, and what metrics we should be keeping an eye on.

Chapter 3, Performance Testing, looks at using the Apache JMeter tool to solve performance problems, by triggering them.

Chapter 4, JVM Memory, starts right at the bottom of the Oracle SOA Suite 11g stack. We look at how JVM memory is allocated, and how you can increase and decrease the memory available to your application, as well as other JVM tuning tips. The low-level areas of Oracle SOA Suite are often overlooked, so we cover the basics of Java memory management in enough detail to give those unfamiliar with it sufficient background to understand what they need to be concerned about.

Chapter 5, JVM Garbage Collection Tuning, teaches us how the Java virtual machine garbage collector does a very complicated job, and that there are a number of ways of optimizing it for the kinds of payloads that Oracle SOA Suite 11g usually has to handle.

Chapter 6, Platform Tuning, looks at some of the tuning options available in the operating system and Oracle WebLogic server that can improve the performance of SOA Suite 11g applications.

Chapter 7, Datasources and JMS, covers tips related to JMS and Datasource tuning that can improve the performance of resource-intensive SOA Suite applications.

Chapter 8, BPEL and BPMN Engine Tuning, primarily focuses on how we can improve the performance of the engine itself, including database tuning recipes, and helps those who find that the bottleneck of their SOA Suite system is the BPEL or BPMN application itself.

Chapter 9, Mediator and BAM, focuses on tuning the Oracle SOA Suite Mediator, and also includes some information on tuning the BAM Adapter for SOA Suite application that connects to a BAM service.

Chapter 10, Rules and Human Workflow, focuses on tuning the rules and human-workflow components, and our recipes focus on how to reduce resource contention.

Chapter 11, SOA Application Design, focuses on design-time recipes that will be of more interest to architects. We also discuss how design-time decisions can have an impact on the performance of a SOA Suite application, and covers best practices for designing the key components.

Chapter 12, High Performance Configuration, takes a higher-level look at the Oracle SOA Suite-deployment architectures, focusing on how we can use cluster and load balance to scale our application out to meet performance requirements.

What you need for this book

We obviously assume that you have an Oracle SOA Suite 11g application, and are looking to tune it. So we expect that you have the necessary software and licenses from Oracle. In addition to SOA Suite 11g from Oracle, which is used throughout the book, we also make use of the VFabric Hyperic HQ monitoring tool from VMWare, and the JMeter load-testing tool from Apache. Both Hyperic and JMeter are freely available from their respective vendors.

Who this book is for

The target audience for this book are Oracle SOA Suite administrators of all levels of experience. Whether you normally administer databases and have had SOA Suite handed to you because it has Oracle in the name, or whether you have years of experience in SOA administration, we are sure you will find something useful in this book. While the primary audience are the administrators, there is plenty of useful information—to think about when designing applications—for application architects as well.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "Batching_Limit_Lower determines the minimum number of messages in the batch."

A block of code is set as follows:

```
HugePages_Total:    <number of pages>
HugePages_Free:     <number of pages>
Hugepagesize:       <page size, in kB>
```

Any command-line input or output is written as follows:

```
cd %MIDDLEWARE_HOME%/user_projects/domains/soa_domain/
```

New terms and important words are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this:

"Set the property **Metrics Level** to **Disabled**."

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to feedback@packtpub.com, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.PacktPub.com>. If you purchased this book elsewhere, you can visit <http://www.PacktPub.com/support> and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from <http://www.packtpub.com/support>.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at questions@packtpub.com if you are having a problem with any aspect of the book, and we will do our best to address it.

1

Identifying Problems

This chapter looks at some of the tools and techniques that you can use to identify the areas where any performance bottlenecks are in your SOA Suite application. It covers the following recipes:

- ▶ Identifying new size problems with jstat
- ▶ Identifying permanent generation problems with jstat
- ▶ Monitoring garbage collection with jstat
- ▶ Identifying locking issues with jstack
- ▶ Identifying performance problems with jstack
- ▶ Identifying performance problems using VisualVM on HotSpot
- ▶ Identifying performance problems with JRMCI on JRockit
- ▶ Using JRockit flight recorder to identify problems
- ▶ Monitoring JDBC connections with the WebLogic console
- ▶ Identifying slow running database queries
- ▶ Identifying slow running components with the Enterprise Manager

Introduction

When asked to look into performance issues with a SOA Suite installation, we commonly start by collecting log files from the various system components, looking for clues on any errors or points of contention. The more advanced among us may even have collected Java garbage collection logs and runtime data, and then reviewed it for the same reasons. We encourage these actions as a starting point, but what do we do when the information isn't available in log files? The goal of this chapter is to bridge the gap between knowing there's an issue and identifying it, by introducing a range of tools and techniques for looking at scenarios, before thinking about how to take a remedial action.

The remainder of this book contains a number of recipes that will help you improve the performance of your SOA Suite application by tuning various aspects of the infrastructure, but the changes will generally only make a difference in the area in which the application is slow.

The first step in being able to performance tune your SOA Suite application is to be able to identify where the performance bottlenecks are. Performance is always governed by the availability of resources, and there is always some bottleneck in an application, which prevents it from performing faster. These resources can include the following:

- ▶ CPU time
- ▶ Network bandwidth
- ▶ Input/Output (IO) time
- ▶ Memory
- ▶ Availability of data to process

Memory is not generally a significant bottleneck on its own, but not having enough memory available to your SOA Suite application results in excessive Java garbage collection, which is a CPU-intensive activity, resulting in contention on the CPU. The most common bottlenecks in SOA Suite applications are IO time (waiting for data to be written to some resource, such as a disk or the network) and availability of data (waiting for data to be provided by some external system or user). However, all of the resources mentioned in the preceding list can cause bottlenecks in a SOA Suite application.

It is worth noting that different use cases can have different bottlenecks, so you need to tune for a realistic set of use cases; this is something that is discussed in more detail in a later performance tuning chapter.

We'll start by looking at some tools bundled in the various SOA Suite Java Virtual Machines for diagnosing Java issues, and look at some other common causes of resource contention in SOA Suite. The recipes in this chapter offer suggestions on next steps for common issues, but as mentioned earlier, later chapters will target specific areas of the SOA infrastructure in more detail, once you get over the hurdle of knowing where to start looking.

Identifying new size problems with jstat

jstat is a **Java Virtual Machine (JVM)** command-line tool, which shows you a number of statistics regarding how the JVM is performing. In this recipe, we will use it to understand how large the new size is, and how frequently it is filling up.

Getting ready

You will need the SOA Suite installed for this recipe, and will need permission to execute the domain control scripts, as well as the JVM tools. This recipe assumes that your SOA Suite application is running under a normal load.

The tools `jps` and `jstat` are included with both the HotSpot and JRockit JVMs. For brevity, this recipe assumes that you have the relevant `bin` directory on your path. If you do not, simply use the fully qualified paths to the relevant `bin` directory.

How to do it...

1. Use JPS to determine the process ID of your SOA Suite server.

```
jps -v
```

The process ID is the number in the first column. The SOA Suite server can be identified by the command-line options that the JVM has. If you have multiple SOA Suite servers running on the same machine, you can identify the relevant server by the `-Dweblogic.Name` parameter, as shown in the following screenshot:

```
E:\Oracle\Middleware\jdk160_29\bin>jps -v
440 Server -Xms1024m -Xmx1024m -XX:PermSize=128m -XX:MaxPermSize=512m -XX:+DisableExplicitGC -XX:NewSize=256m -XX:MaxNewSize=256m -XX:SurvivorRatio=8 -XX:+UseParallelGC -Dweblogic.Name=AdminServer -Djava.security.policy=E:\Oracle\MIDDLE~1\WLSEU~1.3\server\lib\weblogic.policy -Xverify:none -da -Dplatform.home=E:\Oracle\MIDDLE~1\WLSEU~1.3 -Dwls.home=E:\Oracle\MIDDLE~1\WLSEU~1.3\server -Dweblogic.home=E:\Oracle\MIDDLE~1\WLSEU~1.3\server -Dcommon.components.home=E:\Oracle\MIDDLE~1\ORACLE~1 -Djrf.version=11.1.1 -Dorg.apache.commons.logging.Log=org.apache.commons.logging.impl.Jdk14Logger -Ddomain.home=E:\Oracle\MIDDLE~1\USER_P~1\domains\SOA_DO~1 -Djrockit.optfile=E:\Oracle\MIDDLE~1\ORACLE~1\modules\oracle.jrf_11.1.1\jrockit.optfile.txt -Doracle.server.config.dir=E:\Oracle\MIDDLE~1\USER_P~1\domains\SOA_DO~1\config\FMWCON~1\servers\AdminServer -Doracle.domain.config.dir=E:\Oracle\MIDDLE~1\USER_P~1\domains\SOA_DO~1\config\FMWCON~1 -Digf.arisidbeans.carmilloc=E:\Oracle\MIDDLE~1\USER_P~1\domains\SOA_DO~1\config\FMWCON~1\carmil -
1936 Jps -Dapplication.home=E:\Oracle\Middleware\jdk160_29 -Xms8m
```

2. Use the `jstat` command to view the sizes of the survivor spaces and Eden:

```
jstat -gc -h10 <pid> 2000
```

The metrics we are interested in are `S0C`, `S1C`, `EC`, and `OC`. These are the sizes (capacity) of the two survivor spaces, the Eden space and the old generation, all in KB.

3. Check that `S0C` and `S1C` have values of 10000 or higher.
4. Check that `EC` has a size of between 20 percent and 50 percent of `OC`.

How it works...

`jstat` is a JVM tool that can be used to view a number of runtime statistics regarding the JVM. More detail on the JVM and its memory layouts is available in the recipes in *Chapter 5, JVM Garbage Collection Tuning*. The option `-gc` prints the statistics about garbage collection, including the capacity and utilization of each memory pool. In the preceding example, the parameter `-h10` prints the headers every 10 lines, to make the output easier to read, and 2000 is the time in milliseconds between each sample (2 seconds).

What we are looking for are problems caused by the new generation memory spaces (Eden and the two survivor spaces) having capacities that are too small. This will cause garbage collection to occur more frequently than necessary, resulting in poor performance. Due to their nature, SOA Suite applications usually involve a lot of object allocation, but many of these objects generally do not last very long (the duration of a request), so they can be collected while they are still in the Eden or one of the survivor spaces. For this reason, we generally want a SOA Suite application to have an Eden size that is between 10 percent to 33 percent of the total heap size. From our experience, this is a good starting point for an Eden space that needs to cope with lots of short-lived objects. The survivor spaces should be large enough to hold the objects generated by one or two large requests, and we have found that values smaller than 10 MB (around 10000 in `jstat`) can cause problems.

There's more...

If you have set the new size (or new ratio) and survivor ratio values on the command line, then the values you see in the output of `jstat` should match those you specified.

See also

- The *Setting the survivor ratio* and *Setting the JVM new size* recipe in *Chapter 5, JVM Garbage Collection Tuning*

Identifying permanent generation problems with jstat

The permanent generation is a special part of the HotSpot JVM's memory, which stores "class templates" for the JVM class loader to use when creating and removing objects.

jstat is a JVM command-line tool, which shows you a number of statistics regarding how the JVM is performing.

In this recipe, we use jstat to understand the size of the permanent generation.

Getting ready

You will need SOA Suite installed for this recipe, and will need permission to execute the domain control scripts, as well as the HotSpot JVM tools. This recipe assumes that your SOA Suite application is running such that we may run jstat against it.

The tools `jps` and `jstat` are included with both the HotSpot and JRockit JVMs. For brevity, this recipe assumes that you have the relevant `bin` directory on your path. If you do not, simply use the fully qualified paths to the relevant `bin` directory.

This recipe assumes you are using HotSpot, as JRockit does not have a permanent generation.

How to do it...

1. Use `JPS` to determine the process ID of your SOA Suite server; see step 1 of the *Identifying New Size problems with jstat* recipe.
2. Use the `jstat` command to view the sizes of the survivor spaces and Eden:

```
jstat -gc -h10 <pid> 2000
```

The metrics we are interested in are `PC` and `PU`. `PC` is the capacity (size) of the permanent generation (in KB), and `PU` is the utilization of the permanent generation (in KB).

3. We want to see that `PU` is around 80 percent or less of `PC`.

How it works...

`jstat` is a JVM tool that can be used to view a number of runtime statistics regarding the JVM. The option `-gc` prints the statistics about garbage collection, including the capacity and utilization of each memory pool. In the preceding example, the parameter `-h10` prints the headers every 10 lines, to make the output easier to read, and `2000` is the time in milliseconds between each sample (2 seconds).

Permanent generation is the section of JVM memory that is used to store classes and other objects that cannot be represented as Java types. The larger a SOA Suite application is, the more classes it will contain, and the more permanent generation it will need. If the permanent generation is too small, classes will need to be unloaded and reloaded in the permanent generation, causing full garbage collections to occur, and a performance hit. We want permanent generation to be large enough to hold all the classes in your SOA Suite application, with a little overhead for other resources, so if the utilization of the permanent generation is around 80 percent of its capacity, we should have sufficient overhead to prevent classes needing to be unloaded.

See also

- ▶ The *Setting the permanent generation size* recipe

Monitoring garbage collection with `jstat`

`jstat` is a JVM command-line tool, which shows you a number of statistics regarding how the JVM is performing. In this recipe, we use `jstat` to understand the size of the permanent generation.

Getting ready

You will need the SOA Suite installed for this recipe, and will need permission to execute the domain control scripts, as well as the JVM tools. This recipe assumes that your SOA Suite application is running under a normal load.

The tools `jps` and `jstat` are included with both the HotSpot and JRockit JVMs. For brevity, this recipe assumes that you have the relevant `bin` directory on your path. If you do not, simply use the fully qualified paths to the relevant `bin` directory.

This recipe assumes you are using HotSpot, as JRockit does not have a permanent generation.

How to do it...

1. Use JPS to determine the process ID of your SOA Suite server, see step 1 of the *Identifying New Size Problems with jstat* recipe.
2. Use the `jstat` command to view the sizes of the survivor spaces and eden:

```
jstat -gc -h10 <pid> 2000
```

The metrics we are interested in are `YGCC`, `YGCT`, `FGCC`, and `FGCT`. These are the young garbage collection count, young garbage collection time, full garbage collection count, and full garbage collection time.

How it works...

`jstat` is a JVM tool that can be used to view a number of runtime statistics regarding the JVM. The option `-gc` prints statistics about garbage collection, including the capacity and utilization of each memory pool. In the preceding example, the parameter `-h10` prints the headers every 10 lines, to make the output easier to read, and `2000` is the time in milliseconds between each sample (2 seconds).

Since garbage collection is a "stop the world" activity, any time spent garbage collecting is not the time spent in executing your business logic. We therefore want to minimize the amount of time spent in performing garbage collection. See the chapter on garbage collection for more details on what we are looking to achieve with garbage collection tuning.

There's more...

In *Chapter 5, JVM Garbage Collection Tuning*, we also pass the JVM a startup option to enable verbose GC logging output, so we can capture this data all the time.

We can then run free tools such as **GCviewer** (<http://www.tagtraum.com/gcviewer.html>) on this data output to visualize the data trends. Later in this chapter, we will introduce using real-time monitoring tools bundled with the JVM to view this same data.

See also

- ▶ The *Identifying performance problems using VisualVM on HotSpot* and *Identifying performance problems with jrmc on JRockit* recipes
- ▶ The *Tuning to reduce the number of major garbage collections* and *Turning on verbose garbage collection* recipes in *Chapter 5, JVM Garbage Collection Tuning*

Identifying locking issues with jstack

As SOA Suite is a heavily multi-threaded Java application, where locks are used to synchronize resources. If the threads encounter issues sharing these locks, then the impact on performance can be severe.

jstack is a HotSpot JVM tool that allows you to view thread dumps, which can be a useful way of identifying locking problems in your application.

Getting ready

You will need SOA Suite installed for this recipe, and will need permission to execute the domain control scripts, as well as the JVM tools. This recipe assumes that your SOA Suite application is running, and under a normal load.

The tools `jps` and `jstack` are included with the HotSpot JVM. If you're using JRockit, then see the *There's more..* section of this recipe for the alternative command to use. For brevity, this recipe assumes that you have the relevant JVM `bin` directory on your command line path. If you do not, simply use the fully qualified paths to the relevant `bin` directory.

How to do it...

1. Use `JPS` to determine the process ID of your SOA Suite server; see step 1 of the *Identifying New Size Problems with jstat* recipe.
2. Use the `jstat` command to create a thread dump for the process. We specify the `-l` parameter to display additional information about locks, and redirect the output to a file, so we can inspect it more easily:

```
jstack -l <pid> >jstack-output.txt
```

This will generate a file called `jstack-output.txt`. Alternatively `kill -3 <pid>` can be used on Linux or *Ctrl + Break* in the console in Windows.

3. Open `jstack-output.txt` in your favorite text editor. There are two main types of locking problem we are looking for. The first is a deadlock, where two threads each own one or more locks, but are both waiting on each other's locks. This will look something similar to the following screenshot:

```

"[STANDBY] ExecuteThread: '5' for queue: 'weblogic.kernel.Default (self-
java.lang.Thread.State: WAITING (on object monitor)
  at java.lang.Object.wait(Native Method)
  - waiting on <0x23f622d0> (a weblogic.work.ExecuteThread)
  at java.lang.Object.wait(Object.java:485)
  at weblogic.work.ExecuteThread.waitForRequest(ExecuteThread.java:2
  - locked <0x60fd1fe8> (a weblogic.work.ExecuteThread)
  at weblogic.work.ExecuteThread.run(ExecuteThread.java:226)

Locked ownable synchronizers:
  - None

"[STANDBY] ExecuteThread: '6' for queue: 'weblogic.kernel.Default (self-
java.lang.Thread.State: WAITING (on object monitor)
  at java.lang.Object.wait(Native Method)
  - waiting on <0x60fd1fe8> (a weblogic.work.ExecuteThread)
  at java.lang.Object.wait(Object.java:485)
  at weblogic.work.ExecuteThread.waitForRequest(ExecuteThread.java:2
  - locked <0x23f622d0> (a weblogic.work.ExecuteThread)
  at weblogic.work.ExecuteThread.run(ExecuteThread.java:226)

Locked ownable synchronizers:
  - None

```

Note that each thread is waiting on the lock held by the other thread. If you see thread deadlocks, this generally requires that you redesign your application to acquire locks in the correct order.

The second type of problem we are looking for is lock contention. This occurs when many threads are all waiting to get access to the same lock. When this occurs, you will see many threads all waiting on the same lock. This usually also requires application code changes to resolve.

How it works...

The jstack tool comes with the HotSpot JVM, and is used to produce stack dumps for a running JVM. The output of stack dumps contains information on any locks held by each thread, which we can use to diagnose common locking-related problems.

There's more...

There are two ways of performing locking in a Java application. The first is by using the `synchronized` keyword, and is the most common. Locks obtained using this method appear in the thread dumps as threads marked as - `locked <object reference>`, and threads waiting to enter a synchronized block that they do not own the lock to are marked as - `waiting on <object reference>`. The second mechanism for obtaining locks is using the `java.util.concurrent.locks` package, and locks obtained in this way do not appear in a standard thread dump. However, by specifying the `-l` parameter to `jstack`, we are able to view the information on which threads hold locks. This information is displayed in the locked ownable synchronizers section below each thread dump.

Free tools are available to examine lock dumps, but we will not discuss them in this book.

The techniques in this recipe can be replicated on the JRockit JVM with the command `jrcmd <pid> print_threads`, where `pid` can be identified using JRockit's `jps` in the same way as with the HotSpot JVM.

Identifying performance problems with jstack

Jstack is a JVM tool that will generate thread dumps for a running JVM. We can use this as a way to do some lightweight profiling of our running SOA Suite application, to identify where performance bottlenecks are.

Getting ready

You will need the SOA Suite installed for this recipe, and will need permission to execute the domain control scripts, as well as the JVM tools. This recipe assumes that your SOA Suite application is running under a normal load.

The tools `jps` and `jstack` are included with the HotSpot JVM. If you're using JRockit then see the *There's More...* section of this recipe for the alternative command to use. For brevity, this recipe assumes that you have the relevant `bin` directory on your path. If you do not, simply use the fully qualified paths to the relevant `bin` directory.

How to do it...

1. Use JPS to determine the process ID of your SOA Suite server; see step 1 of the *Identifying New Size Problems with jstat* recipe.
2. Execute a slow performing use case on your SOA Suite server.

3. While the use case is executing, use the `jstat` command to create a thread dump for the process. We redirect the output to a file to inspect it more easily:

```
jstack<pid> >jstack-output-1.txt
```

This will generate a file called `jstack-output-1.txt`. Alternatively, `kill -3 <pid>` can be used on Linux or *Ctrl + Break* in the console in Windows.

4. Repeat step 3 a number of times, changing the number of the generated file each time, to create a sequence of thread dumps. I suggest aiming to create thread dumps at intervals of one or two seconds, for a period of 30 seconds, or the amount of time that it takes to execute your slow use case.
5. Open the output files in your favorite text editor.

To identify the poor performing code, we are looking for thread stacks that appear frequently across the files. These will either be operations that are executed multiple times, or operations that are executed once but take a long time to complete.

How it works...

A thread dump is a snapshot of what the every thread in the application is doing at a particular moment in time. By taking multiple thread dumps over a period of time, we can build up a picture of where the application is spending its time. This approach is essentially how a sampling profiler works, so we are effectively doing manual profiling of the application.

It is common to find that the cause of poor performance is waiting on data from external systems, such as a database or web service. In this case, what we will see in our thread dumps is that there will be many instances where a thread is in JVM code that is reading data from a socket. This is the thread waiting on a response from the remote system, and we know that we should target the external system in order to improve the performance.

We can quickly look through large numbers of stacks by noting that, in general, threads of interest to us will be those that are running the application code. We can identify the threads of interest, as they will generally have two properties; they will be executing the code from the packages that are used in your application, and they will have stack traces that are longer than those of threads that are currently idle.

There's more...

What we are doing here is essentially what a sampling profiler tool would do, although we are doing it manually. While there may be cases in which a sampling profiler would be a preferable way of doing this, we have found that doing the profiling manually gives a number of benefits. Firstly, it uses only tools that already exist on the server—there is nothing that needs installing, and given that we may be doing this on a live system, this usually means less paperwork. Secondly, we have more control over how and when we generate the thread dumps, which helps reduce the overhead on the application.

If we find this technique is something that we do frequently, it may be worth writing a script that will generate the thread dumps for us. We have found this to be a good compromise between using a sampling profiler and manually sampling, using thread dumps.

It is worth noting that WebLogic can report Java threads as "stuck" if they run for a longer period of time than a configurable timeout. This itself is not indicative of a problem, as long-running activities (such as database or file polling) can be valid things for application components to do. Observing thread dumps will tell us definitively whether we need to take further action.

Generating heap dumps is also a good technique for investigating performance issues, and free tools such as the **Memory Analyzer Toolkit** can generate reports on object allocation and aid with detecting memory leaks. We do not discuss heap dumps explicitly in this book, because SOA Suite is an Oracle supported application, so issues with the core components themselves should be raised with Oracle support, but there are plenty of good resources online to get use this technique if you desire.

See also

- ▶ The *Identifying performance problems using VisualVM on HotSpot*, *Identifying performance problems using JRMCI on JRockit*, and *Using JRockit Flight Recorder to identify problems* recipes

Identifying performance problems using VisualVM on HotSpot

VisualVM is a powerful graphical tool that comes with the HotSpot JVM. It has a number of views, which can be useful for diagnosing performance problems with SOA Suite. This recipe looks at a number of these features.

Getting ready

You will need SOA Suite installed for this recipe, and will need permission to execute the domain control scripts, as well as the JVM tools. This recipe assumes that your SOA Suite application is running under a normal load, or a load sufficient to demonstrate any performance problems.

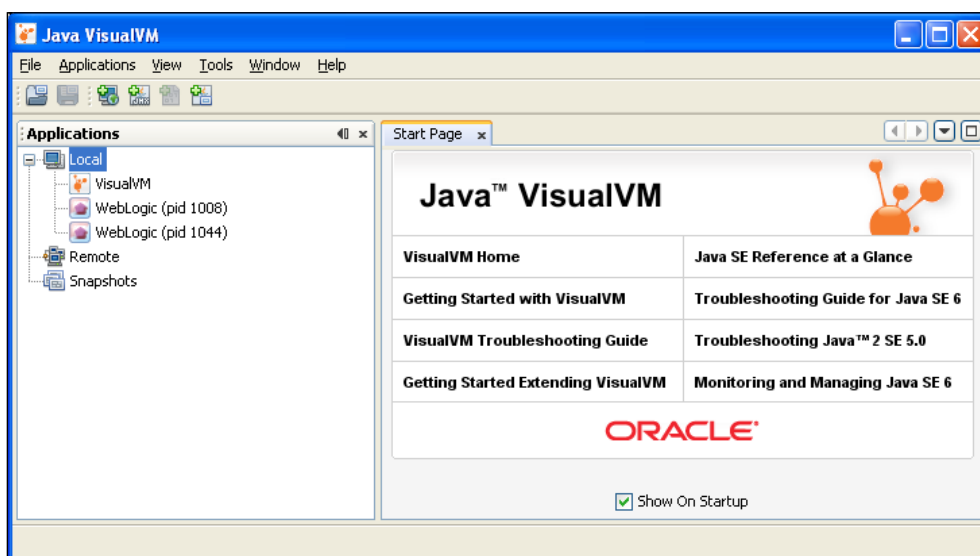
The tool `jvisualvm` is included with the HotSpot JVM, JRockit has a different tool described in a different recipe. For brevity, this recipe assumes that you have the relevant JVM `bin` directory on your path. If you do not, simply use the fully qualified paths to the relevant `bin` directory.

How to do it...

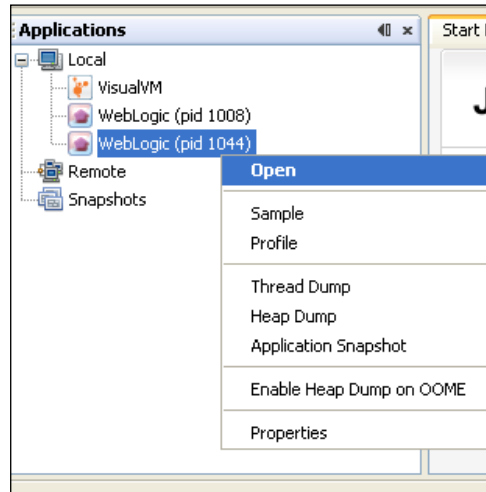
1. Use JPS to determine the process ID of your SOA Suite server, see step 1 of the *Identifying New Size Problems with jstat* recipe. This will let us identify a target Java process in a host running multiple JVMs. We're trying to identify the WebLogic admin server to use with Visual VM.
2. Start VisualVM by using the `jvisualvm` command:

```
jvisualvm
```

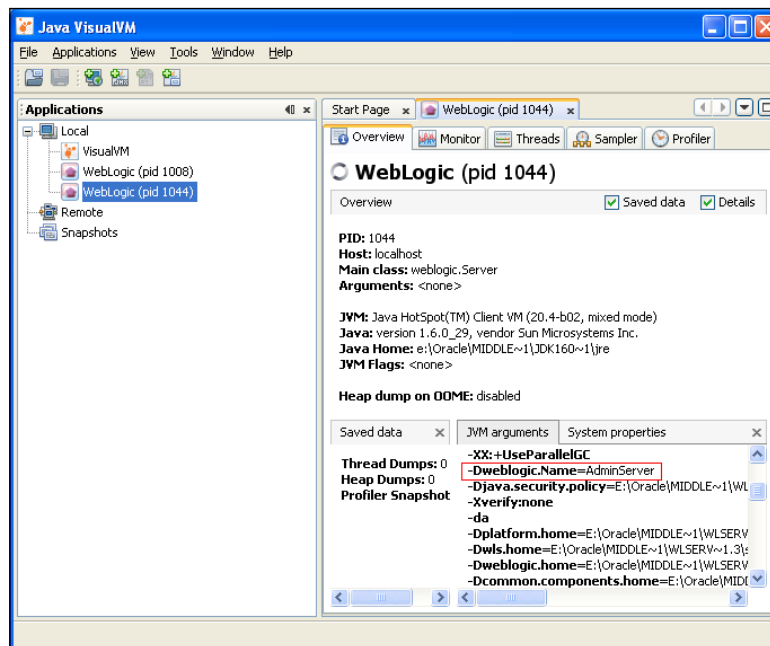
If this is the first time you have run VisualVM, it will first perform calibration on your server. Once the calibration is completed, the VisualVM homepage will open as shown:



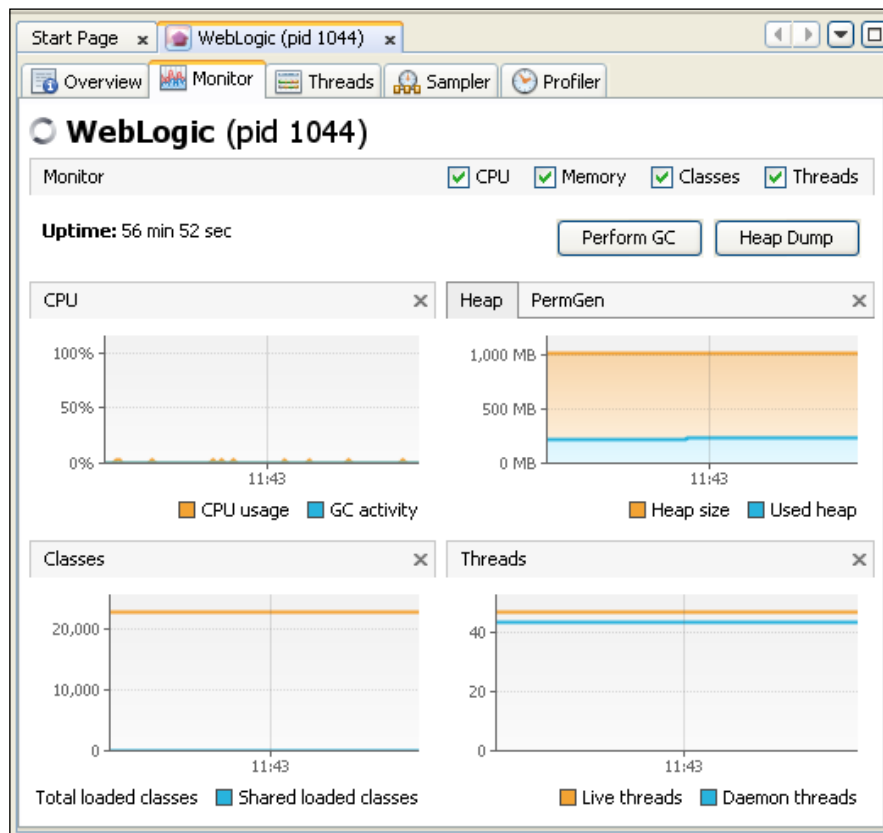
3. Select the WebLogic process that has the process ID you identified in step 1. Either double-click on it, or right-click and select **Open**.



4. VisualVM will open the **Overview** page for the **WebLogic** Server instance. You should check that the server name matches the one you were expecting.

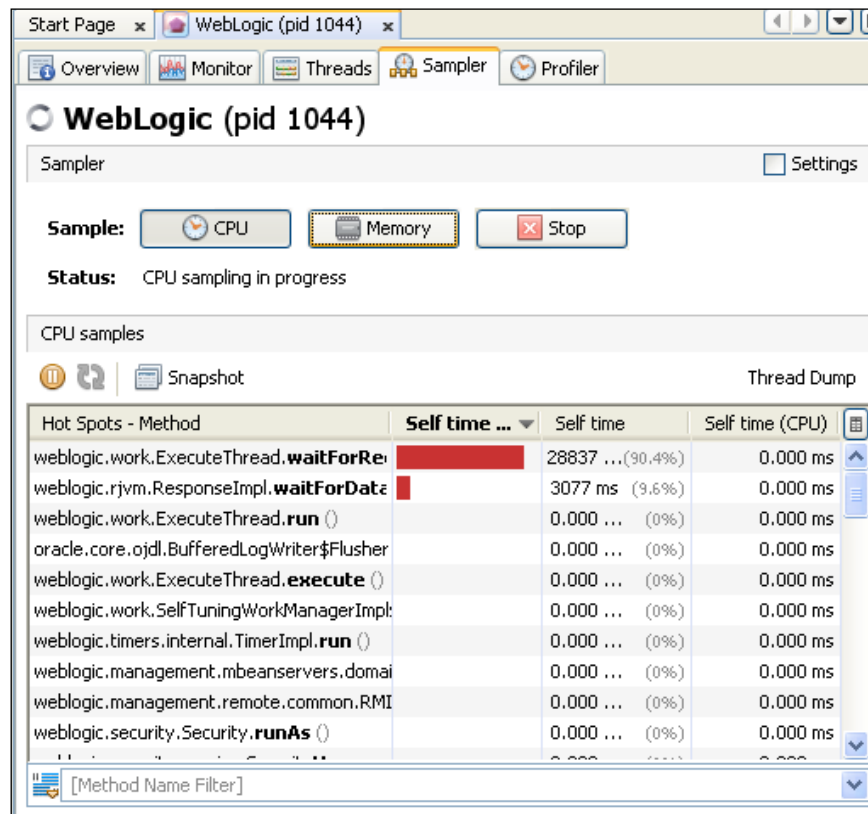


5. Switch to the **Monitor** tab, which provides an overview of the performance of the application.



6. There are a number of things that we want to check on this tab:
 - ❑ The CPU usage should be relatively low (below 30 percent, ideally) and the amount of CPU usage spent on the GC activity should also be low.
 - ❑ When garbage collection occurs, **Used Heap** should drop to a significant amount.

- VisualVM contains two profilers that can be used to identify performance problems. We prefer to use the sampling profiler, which is available on the **Sampler** tab.



- Click on the **CPU** button to begin profiling your application. The table will show which Java methods are taking the most time.

How it works...

VisualVM hooks into the JVM to read the statistics about memory usage, garbage collection, and other internals, and displays them graphically for the user. Many of these metrics are useful in identifying where a performance problem lies.

An application with high CPU usage may be CPU bound, but before just putting it on a host with more or faster CPUs, we need to identify what it is that is using the CPU. Garbage collection is a common culprit, so the graph showing the amount of CPU time spent running the garbage collector is particularly useful. If garbage collection is not the culprit, then the sampling profiler will show us where the CPU time is being spent. This is essentially a graphical version of using the `jstat` and `jstack` tools mentioned in other recipes.

There's more...

Additional plugins can be installed for VisualVM to display the information in more convenient ways, or to display additional information.

The **Profiler** can also be used to profile your application, but it imposes a much higher overhead, and so is more likely to affect the results.

See also

- ▶ The *Identifying performance problems using JRMCM on JRockit* recipe

Identifying performance problems using JRMCM on JRockit

JRockit Mission Control is a graphical management console for the JRockit JVM, which comes with SOA Suite 11g. We can use it to help identify a number of performance problems that we might encounter with SOA Suite when running on JRockit.

Getting ready

You will need SOA Suite installed for this recipe, and will need permission to execute the domain control scripts, as well as the JVM tools. This recipe assumes that your SOA Suite application is running, and under a normal load or a load sufficient to demonstrate any performance problems.

The tool `jrmc` is included with the JRockit JVM; HotSpot has a different tool described in a different recipe. For brevity, this recipe assumes that you have the relevant JVM `bin` directory on your path. If you do not, simply use the fully qualified paths to the relevant `bin` directory.

How to do it...

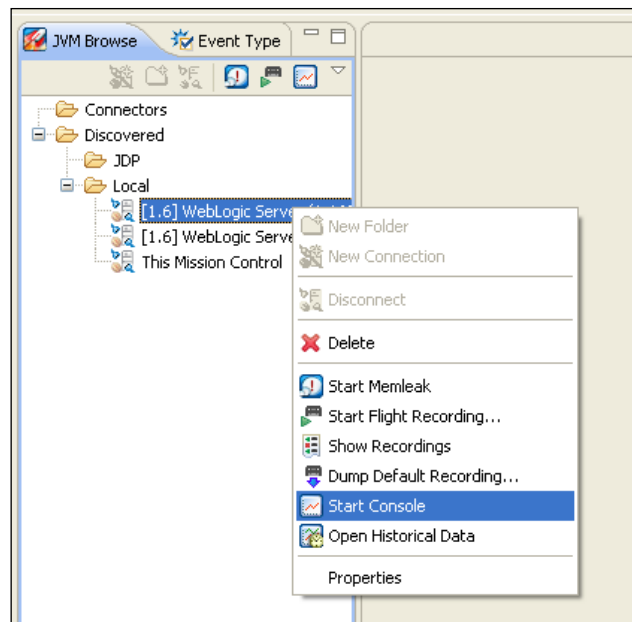
1. Use JPS to determine the process ID of your SOA Suite server; see step 1 of the *Identifying new size problems with jstat* recipe. This will let us identify a target Java process in a host running multiple JVMs. We're trying to identify the WebLogic Admin server to use with JRockit Mission Control.
2. Launch JRockit Mission Control by executing the command `jrmc` from the JVM `bin` directory:

```
jrmc
```

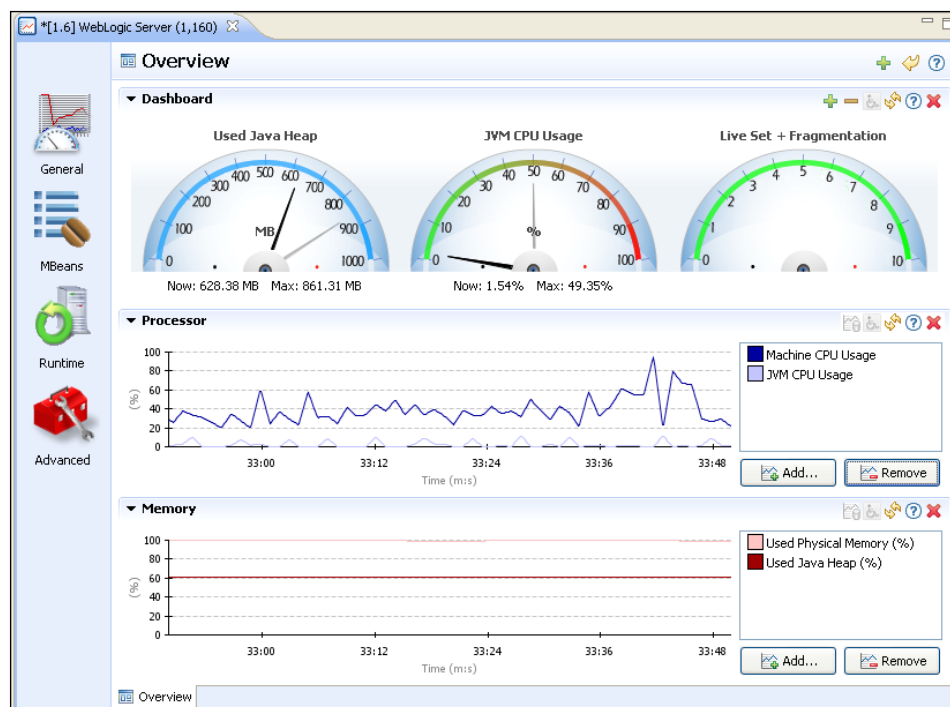
If this is the first time you have run `jrmc`, it will open the JRockit Mission Control Welcome page:



3. Click on **X** next to the **Welcome** tab in the top-left to close the welcome screen. This will display the birds-eye view, which can be selected at any time from the **Window** menu.
4. Select the WebLogic instance that you want to connect to, based on the process ID that you established in step one. Right-click on the instance and select **Start Console**.



5. This will display the JRockit Mission Control console for the selected WebLogic instance:



6. There are a number of things that we are looking for on this page:
 - ❑ **JVM CPU Usage** should be low, not higher than 30 percent ideally.
 - ❑ **Used Java Heap (%)** should increase as memory is used, but should come down again when garbage collection occurs.

How it works...

JRockit Mission control gathers management metrics made available by JRockit at runtime, and makes these visible graphically. The **Mission Control** console gives us an overview of the CPU and memory usage of the application with graphs of historical data. This gives us an overview that we can use to determine whether applications are having memory- or CPU-related performance problems.

Mission Control can also monitor JVMs remotely; refer to the JRockit documentation for guidelines on achieving this at <http://www.oracle.com/technetwork/middleware/jrockit/documentation/index.html>.

See also

- The *Using JRockit Flight Recorder to identify problems* recipe

Using JRockit flight recorder to identify problems

JRockit Flight Recorder is a component of JRockit Mission control that can be used to record detailed statistics about your application, to be viewed later.

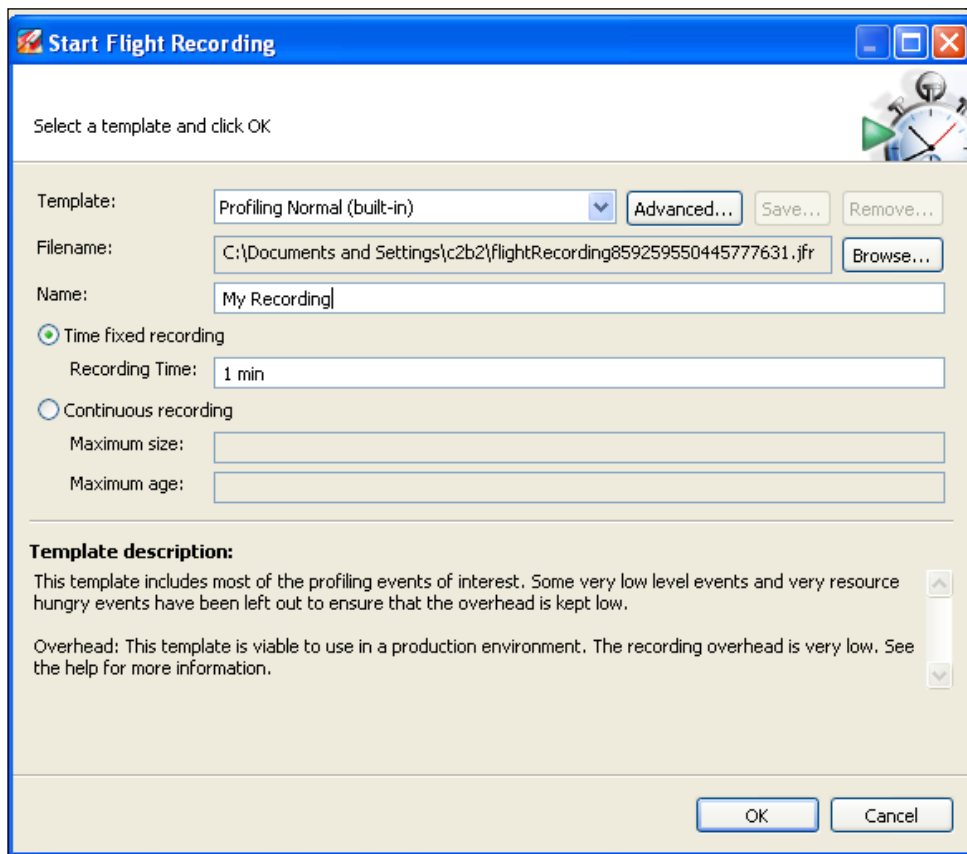
Getting ready

You will need SOA Suite installed for this recipe, and will need permission to execute the domain control scripts, as well as the JVM tools. This recipe assumes that your SOA Suite application is running under a normal load or a load sufficient to demonstrate any performance problems.

The tool `jrmc` (of which Flight Recorder is a component) is included with the JRockit JVM; HotSpot has a different tool called VisualVM described in a different recipe. For brevity, this recipe assumes that you have the relevant JVM `bin` directory on your path. If you do not, simply use the fully qualified paths to the relevant `bin` directory.

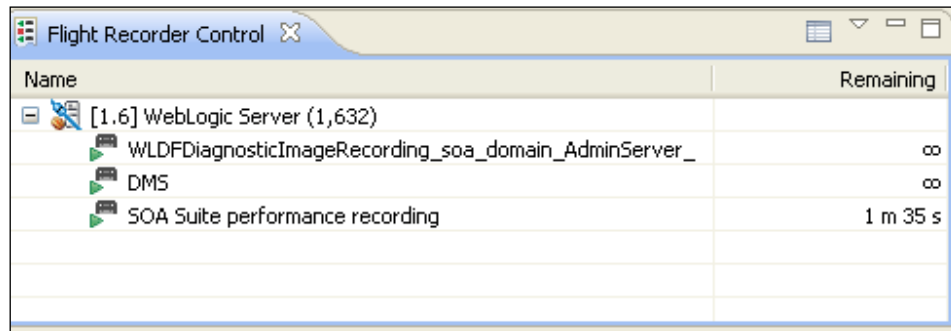
How to do it...

1. Start JRockit Mission Control by following steps from 1 through 3 of the *Identifying performance problems using JRMC on JRockit* recipe.
2. Select the WebLogic instance that you want to connect to, based on the process ID that you established in step 1. Right-click on the instance and select **Start Flight Recording**. This will display the flight recording settings dialog box.



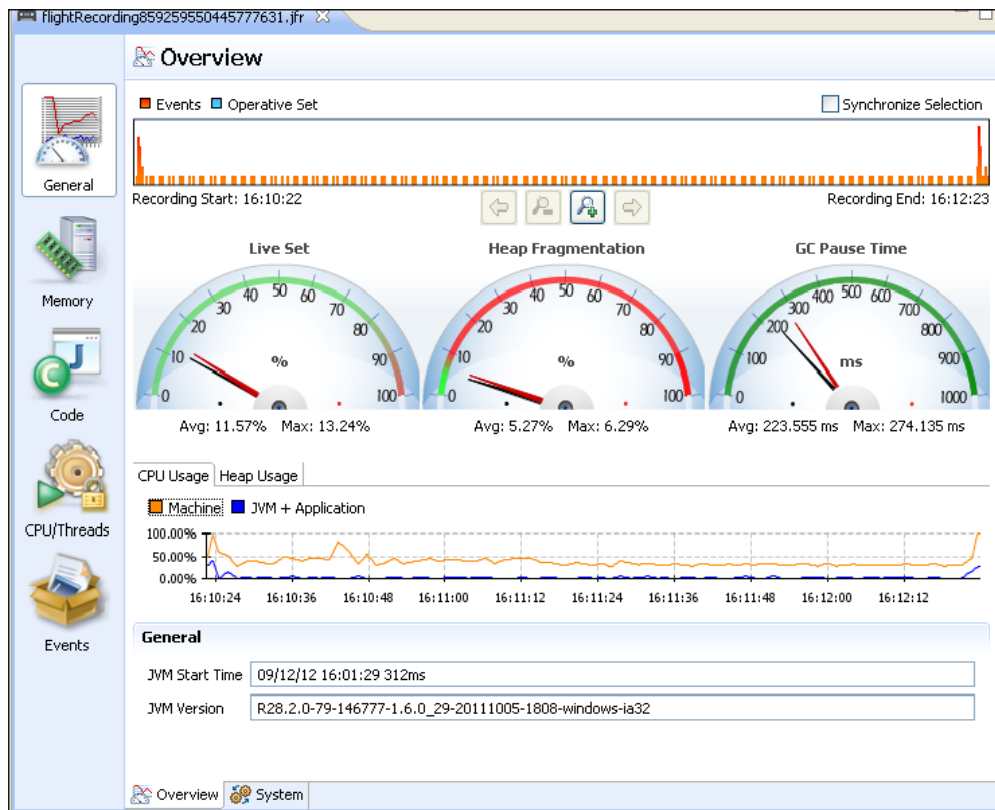
3. Accept the default **Template** and **Filename** settings, and set **Name** to be something descriptive. **Recording Time** should be long enough to encapsulate a use case that is considered to be slow. Once you are happy with the settings, click on **OK** and the recording will begin.

- While recording is in progress, you can see the remaining time from the **Flight Recorder Control** view at the bottom of the screen.



Name	Remaining
[1.6] WebLogic Server (1,632)	
WLDiagnosticImageRecording_soa_domain_AdminServer_	∞
DMS	∞
SOA Suite performance recording	1 m 35 s

- Once the flight recorder finishes recording, it will open the **Overview** screen as shown:



This screen displays the overview of the recorded data. The event strip at the top of the screen can be used to select a specific part of the timeline to view. Clicking on **Synchronize Selection** will keep the selected time range the same across all the viewed screens.

6. Select the **Memory** tab; this tab shows us a number of interesting statistics about the JVM memory during the runtime. Notice that for more detailed information, you can select one of the subtabs at the bottom of the screen.

On this page, we want to check a number of things:

- ❑ That the average and maximum **Main Pause** times are reasonable; for example, below 10 s
 - ❑ That the heap usage comes down by a reasonable amount when garbage collection occurs, and that it is not just "bouncing around near the top of the graph"
7. Select the **CPU/Threads** tab. This tab shows the statistics regarding CPU usage and threads. We want to check that the Application + JVM CPU usage does not make up the majority of the total CPU usage, and that the total CPU usage is low (ideally at 30 percent or below).

How it works...

JRockit Flight Recorder collects a large number of detailed statistics about an application execution, and stores them so that they can be analyzed later. This method allows a more detailed analysis to be performed then, if you are trying to observe the statistics in real time, you can go back and "replay" the gathered statistics to focus on certain time periods. Flight recorder can be executed with a number of profiles; the Normal profile has the lowest overhead, so is most suitable for use in production environments. The other profiles gather more information, but have a higher overhead, so are more suitable when you can reproduce the problem in a test environment. One of the advertised powerful features of JRockit is that the Normal collection profile adds virtually no overhead to the JVM execution time, as the collected metrics are used internally by JRockit during execution for runtime optimization.

It can be helpful to take Flight Recorder profiles of healthy systems, so that if a future release introduces a performance problem, you have a baseline to compare it to. This can make it much easier to identify the area in which performance is likely to be suffering. With SOA Suite in mind, it is worthwhile using Flight Recorder when running load tests (see *Chapter 3, Performance Testing*) against the WebLogic servers.

See also

- *The Identifying performance problems with JRockit Mission Control recipe*

Monitoring JDBC connections with the WebLogic console

One of the things that can cause poor application performance is not having enough connections to the database available. Since Oracle SOA Suite is a heavily database intensive application, it can cause significant performance problems, particularly if using lots of transient SOA BPEL processes (see *Chapter 8, BPEL and BPMN Engine Tuning*, for more on this topic). In this recipe, we will look at how to use the WebLogic console to view a number of available database connections.

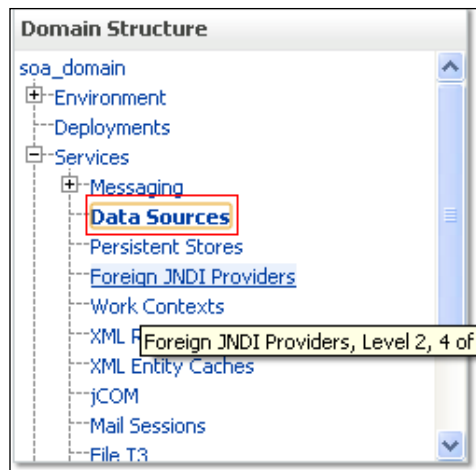
Getting ready

For this recipe, you will need to know the URL and login credentials for the WebLogic administration console that runs on the admin server in your SOA Suite domain. The default URL for the console is `http://<servername>:7001/console`. The username and password will have been specified when you created the domain.

You will need SOA Suite installed and running, and to have loaded it with representative load or live data, or be able to create a realistic load on the application.

How to do it...

1. Connect to the WebLogic administration console at `http://localhost:7001/console`; replace `localhost` and `7001` with the hostname and port of the server if it is not running locally on `7001`. If this is the first time you have accessed the WebLogic console since starting the server, WebLogic will first deploy the console application, which may take a few minutes.
2. Log in to the console by using your administration credentials.
3. Open the **Services** category on the left-hand navigation pane, and select **Data Sources**.



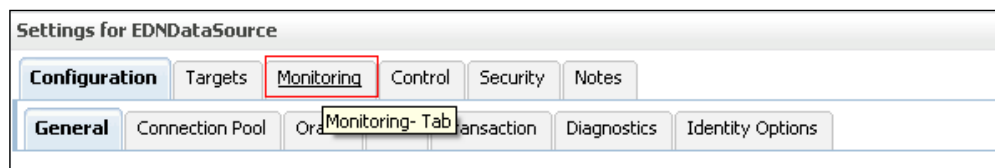
4. This will display a list of all the data sources in the main panel.

Data Sources (Filtered - More Columns Exist)

New ▾ Delete Showing 1 to 7 of 7 Previous | Next

<input type="checkbox"/>	Name ↕	Type	JNDI Name	Targets
<input type="checkbox"/>	EDNDataSource	Generic	jdbc/EDNDataSource	soa_server1
<input type="checkbox"/>	EDNLocalTxDataSource	Generic	jdbc/EDNLocalTxDataSource	soa_server1
<input type="checkbox"/>	mds-owsm	Generic	jdbc/mds/owsm	AdminServer, soa_server1
<input type="checkbox"/>	mds-soa	Generic	jdbc/mds/MDS_LocalTxDataSource	AdminServer, soa_server1
<input type="checkbox"/>	OraSDPMDDataSource	Generic	jdbc/OraSDPMDDataSource	soa_server1
<input type="checkbox"/>	SOADataSource	Generic	jdbc/SOADataSource	soa_server1
<input type="checkbox"/>	SOALocalTxDataSource	Generic	jdbc/SOALocalTxDataSource	soa_server1

5. Select the first of these data sources, which will load its properties page. Select the **Monitoring** tab and the **Statistics** subtab.



6. This will display the statistics for the data source. If you see nothing on this page, then it is likely that none of the servers on which this data source exists are running.
7. Click on the **Customise this table** button, and add the following columns to the view: **Active Connections Current Count**, **Active Connections High Count**, **Current Capacity**, **Number Available**, **Waiting for Connection Current Count**, and **Waiting for Connection High Count**. Click on **Apply**.

Customize this table

Filter

Filter by Column: Server Criteria:

View

Column Display:

Available:

- ☐ Connection Delay Time
- ☐ Connections Total Count
- ☐ Current Capacity High Count
- ☐ Failed Affinity Based Borrow Count
- ☐ Failed RCLB Based Borrow Count
- ☐ Failed Reserve Request Count
- ☐ Failures To Reconnect Count

Chosen:

- ☐ JDBC Driver
- ☐ Active Connections Current Count
- ☐ Active Connections High Count
- ☐ Current Capacity
- ☐ Number Available
- ☐ Waiting for Connection Current Count
- ☐ Waiting for Connection High Count

Number of rows displayed per page: 10 Maximum Results Returned: All

Apply **Reset**

Deployed Instances of this Data Source (Filtered - More Columns Exist)

Showing 0 to 0 of 0 Previous | Next

Server	Enabled	State	JDBC Driver
There are no items to display			

Showing 0 to 0 of 0 Previous | Next

8. Now, when you view the **Statistics** panel, it should show the following new statistics:

Deployed Instances of this Data Source (Filtered - More Columns Exist)

Showing 1 to 1 of 1 Previous | Next

Server	Enabled	State	JDBC Driver	Active Connections Current Count	Active Connections High Count	Current Capacity	Number Available	Waiting for Connection Current Count	Waiting for Connection High Count
soa_server1	true	Running	oracle.jdbc.xa.client.OracleXADataSource	0	0	0	0	0	0

Showing 1 to 1 of 1 Previous | Next

9. Check the statistics for any problems, see the *How it Works...* section of this recipe for more details about what to look for.
10. Repeat steps from 5 through 9 for each data source.

How it works...

The Oracle WebLogic application server exposes management and monitoring metrics using the JMX standard, and the Oracle WebLogic console makes use of these to display information on components such as the database connection pool sizes. It is worth noting that most connection pools have an initial size of zero. So if you view the pool size before the application has been used, then you will see zero for all of the statistics we are monitoring. If this is the case, then you should run some load through your application and check back again.

These connection pools are used by SOA Suite components to talk to the database for many purposes including storing process instance state, restoring state, storing logging information, and so on. If WebLogic runs out of these connections, then requests have to queue, waiting for a connection to become available, eventually timing out if no connection becomes available. By monitoring the number of connections in use, you can increase the maximum number of connections in the pool, if you notice that they are running out.

We are looking for a number of different things in the connection statistics:

- ▶ Active Connections Current Count equal to Current Capacity, which indicates that every connection from the pool is currently in use. This means that there are no free connections. If the Current Capacity is the maximum number of connections that the pool will hold, then new requests will have to wait for connections.
- ▶ Active Connections High Count equal to Current Capacity, which indicates that at some point, every connection from the pool was in use. Active Connections High Count gives the highest value that Active Connections Current Count has had since the server was restarted (or the statistics reset), so can be useful for diagnosing the cause of performance problems that happened in the past, and the system has now recovered.
- ▶ Number Available equal to zero, which indicates that the pool is empty (unless the pool has not been initialized yet). This will usually occur when all the other conditions are true.
- ▶ Waiting for Connection Current Count or Waiting for Connection High Count not equal to zero, indicating that at some point there have been threads waiting to get a connection from the pool. As threads will only wait for connections if there are none in the pool, then we would expect to only see this if all of the other conditions are also true.

These are all indicators of the same underlying problem, which is that there are insufficient available connections to the database. This is usually caused by one of three things—either a sudden burst of traffic has used up more than normal, the database is running slower than normal, or there is a connection leak in the application. To find out which of these is the case, you should look at the output of the thread dumps and database statistics.

There's more...

As well as being able to see this information via the WebLogic console, it is also made available via the JMX standard and via the WebLogic scripting tool, which is based on Jython, allowing you to use many tools to obtain and process connection pool sizes and more. We will discuss more about JMX in *Chapter 2, Monitoring Oracle SOA Suite*, and look at some of the tools that can be used for obtaining JMX metrics.

See also

- ▶ The *Identifying slow-running database queries* and *Identifying performance problems with JStack* recipes
- ▶ *Chapter 2, Monitoring Oracle SOA Suite*

Identifying slow-running database queries

SOA Suite applications make heavy use of the database. Identifying the database queries that are running slowly can allow you to identify areas where performance improvements can be made by tuning or reducing the amount of database access.

Getting ready

You will need SOA Suite installed and running, and have loaded it with representative load or live data.

This recipe requires that you are able to log in to the SOA Suite database as an administrator and are able to execute the necessary SQL statements.

How to do it...

1. Log in to the SQL database as an administrator.
2. Execute the following SQL query:

```
SELECT * FROM
  (SELECT
    sql_fulltext,
    sql_id,
    child_number,
    disk_reads,
    executions,
    first_load_time,
    last_load_time
  FROM v$sql
```

```
ORDER BY elapsed_time DESC)
WHERE ROWNUM < 10
/
```

This will return the top 10 recent slowest SQL statements, although statements eventually get removed from the `v$sql` table. So, this will contain statements that have been recently executed.

How it works...

Oracle SOA Suite makes very heavy use of the database, being able to identify the queries that take the longest time, which allows these to be optimized. The `v$sql` table in Oracle is a virtual table that contains the statistics about the recently executed SQL statements, so it can be used to identify the statements that take the longest to execute.

There are a number of ways of discovering slow-running database queries, but we have selected the preceding method because it is simple and easy to understand. More complex statistics can be used to identify exactly what it is that is causing queries to run slowly.

Once you have identified slow-running database queries, the steps you need to take to resolve them depends upon what the queries are. If they are application-related queries, then you can focus on changing the application code, but if they relate to Oracle SOA Suite itself, then you will have to look at things like reducing the amount of logging and persistence in your application, or rearchitecting it to behave differently.

See also

- ▶ *Chapter 7, Data Sources and JMS*

Identifying slow-running components with the Enterprise Manager

We can use Oracle Enterprise Manager Fusion Middleware Control to view the slow-performing components of our SOA Suite middleware infrastructure, allowing us to target our performance improvements at those components.

Getting ready

This recipe assumes that you have Oracle SOA Suite installed and running, and are able to generate sufficient load against it to demonstrate any performance problems.

You must have the Oracle Enterprise Manager component installed in your domain, and you will need to know the administration username and password for the domain.

How to do it...

1. Connect to Oracle Enterprise Manager running on the admin server on the URL `http://<hostname>:<port>/em`; by default, this will be `http://localhost:7001/em` if you are connecting from the same host.
2. Log in to Oracle Enterprise Manager by using your administration credentials.
3. Open **Application Deployments**, right-click on `soa_infra`, and select **Performance Summary**.
4. Check that request processing time is not high.

How it works...

Oracle Enterprise Manager is an Oracle product that is used to manage and monitor components of the SOA Suite infrastructure. Some of its features are licensed separately to Oracle SOA Suite. So, if you are unsure, you should check with your Oracle account manager to check whether you are licensed or not to use it.

Oracle Enterprise Manager can be used to manage many components within Oracle SOA Suite, and here we use it to view the performance summary for the SOA infrastructure components. In this instance, we view the request processing time, which tells us how long each request is taking to be processed. It is averaged across a large number of components, so you are really looking to see that it is not higher than usual. This of course requires you to know what usual is, so like all of these statistics, it is worth understanding how the system behaves when there are no performance problems.

There's more...

If you get a 404 error when attempting to connect to the Oracle Enterprise Manager Fusion Middleware Control console, it is likely that you did not add it to your game when you created your domain. You can add it to your domain subsequently by running the configuration manager, selecting your current domain, and extending it to add Oracle Enterprise Manager, although you should be aware that this may overwrite any configuration changes that you have made to the startup scripts.

2

Monitoring Oracle SOA Suite

We will cover the following recipes in this chapter:

- ▶ Installing the Hyperic server
- ▶ Installing Hyperic agents
- ▶ Configuring Hyperic to monitor SOA Suite 11g
- ▶ Monitoring the SOA Suite server availability
- ▶ Monitoring the JVM memory usage
- ▶ Monitoring the platform CPU usage
- ▶ Monitoring the data source usage
- ▶ Monitoring open sockets
- ▶ Monitoring committed transactions
- ▶ Configuring alerts in Hyperic
- ▶ Monitoring the system using the DMS Servlet

Introduction

In this chapter, we will look at the importance of monitoring your Oracle SOA Suite application. The topics covered include how to configure monitoring, what to monitor, and how to interpret the data that your monitoring tool returns.

Performance monitoring is the key to any successful tuning strategy. In order to improve the performance, you need to be able to measure before and after any changes you make. This can, of course, be done with simple performance testing tools, but these only give you the top-level results. What we want to do is dig deeper down and work out where time is being spent, and which monitoring tools allow us to do.

The term monitoring covers a wide range of tools and techniques, from simple systems that just tell us if a process is running or not, through complex user-experience monitoring tools that break down each user request into component parts, and can show what time and resources were involved in generating the response. The most value is obtained by tools that can look inside your SOA Suite application, giving a breakdown of how busy individual components and services are. These types of monitoring tools are often referred to as **Application Monitoring tools**.

The other huge advantage of having good monitoring tools in place is the ability to store metrics, so that you can go back and look at past figures, to see if performance has changed over time or whether what you are looking at is something that has always been there.

A lot of the recipes in this chapter use the VFabric Hyperic HQ tool from VMWare. This tool is part of VMWare's suite of middleware, and is available in the community and enterprise versions. We have chosen to use Hyperic, because it is the best general-purpose monitoring suite that we have found, allowing monitoring at all levels of the Oracle SOA Suite stack. It is also one of the easiest to use, and the enterprise version comes with many powerful features.

There are also many other monitoring tools, which can provide similar features, all of which would be suitable for monitoring our Oracle SOA Suite 11g applications. If you already have an existing application monitoring tool that provides the same information that we get here, then there is no need to replace it. We chose not to use the monitoring features of Oracle Enterprise Manager for a number of reasons. Firstly, we feel that the monitoring is not as comprehensive as provided by dedicated application monitoring tools, secondly it does not provide an extensible framework, and finally the licensing is expensive and restrictive. If you have a larger budget for monitoring, then the tools by Dynatrace and CA (Introscope) are excellent tools, but they come at a price that puts them out of the reach of most projects.

The monitoring in this chapter makes use of JMX, which has been a core component of the Java SE platform since Java 5. JMX provides an API and protocol for exposing management metrics and operations from Java applications. JMX centers on the concept of the Management Bean, or MBean, which is a Java Bean responsible for exposing management metrics and operations for a component or service. MBeans can be written by anyone, and contain any metrics, operations, or logic that you wish; they just need to implement a simple interface. MBeans are registered with an MBean Server using a name (their object name), which acts as a directory, allowing other systems to look up and access the MBeans. Protocol adapters then make the MBean server and its MBeans available over a variety of protocols. This simple, yet powerful API has made it very simple to expose management interfaces for Java applications, and nearly all modern Java infrastructure components will have a JMX interface.

By writing your own MBeans, you can expose management information about the internals of your own application, and make that information available to monitoring tools, such as Hyperic.

The key thing to take away from this chapter is the type of things that you should be monitoring, rather than having to use specific tools.

Installing the Hyperic server

Hyperic is a monitoring tool that can be used to collect, store, display, and alert on metrics from Oracle SOA Suite 11g. It has the server and agent components, and in this recipe, we will look at installing the server.

Getting ready

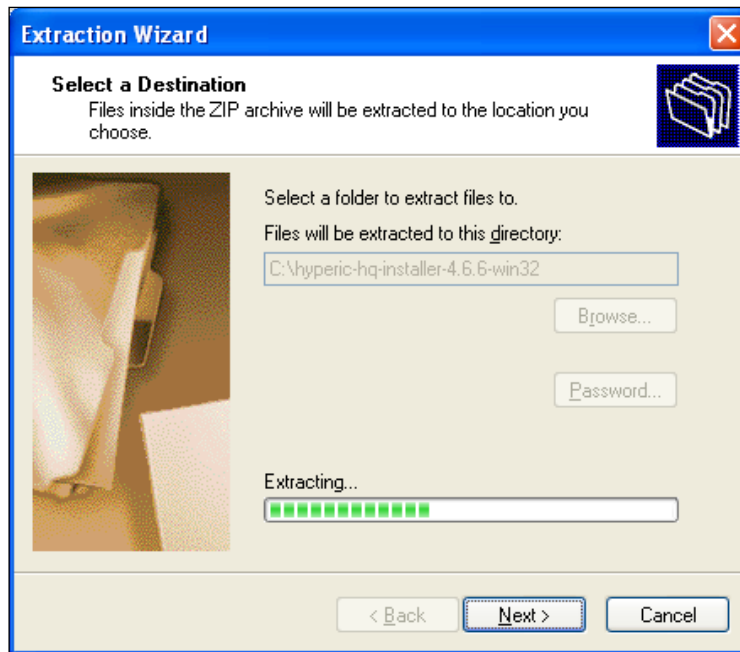
You will need a machine on which to install the server, and the permissions to install the software on it. Since the overhead of running a Hyperic server is noticeable but not huge, we recommend running it on a dedicated machine, or at least a machine that is not hosting the critical parts of your infrastructure. The server needs to meet a few requirements, such as a static IP address for communication with the agents, a Java 1.5 or higher JVM, Linux, Solaris, Mac OS X or Windows 2003 or a later operating system, a 1 GHz processor with at least 1 GB RAM (4 recommended), and at least 1 GB of free disk space. For more details on the requirements, see the documentation at <https://support.hyperic.com/display/DOC/Installation+Requirements>.

You will need to download the Hyperic HQ installer (either community or enterprise) from <http://www.hyperic.com/hyperic-open-source-download>. This recipe uses the community (open source) version; so if you are using Hyperic enterprise, you may find a few differences in the installation process.

How to do it...

Follow these steps to install the Hyperic server:

1. Extract the Hyperic installer to a temporary location.



2. Run the installer for your platform by using either `setup.bat` or `setup.sh`.

```
C:\WINDOWS\system32\cmd.exe
Please ignore references to missing tools.jar
Unable to locate tools.jar. Expected to find it in C:\DOCUME~1\c2b2\LOCALS~1\Tem
p\lib\tools.jar
Buildfile: C:\hyperic-hq-installer-4.6.6-win32\hyperic-hq-installer-4.6.6\instal
ler\bin\..\data\setup.xml
Loading taskdefs...
Taskdefs loaded
Initializing Hyperic HQ 4.6.6 Installation...
Choose which software to install:
1: Hyperic HQ Server
2: Hyperic HQ Agent
You may enter multiple choices, separated by commas.
-
```

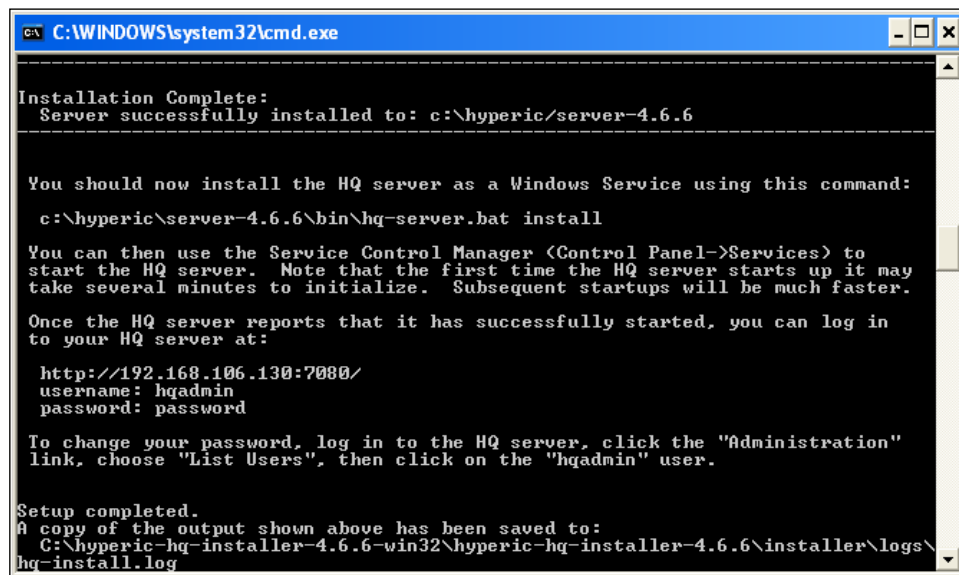
3. Select to install both the server and the agent, by entering 1, 2 at the prompt.
4. When prompted, specify the installation location for the server. This location must exist, as the installer will not create it.
5. Specify the SMTP server to be used for sending e-mails, also the e-mail address to be used for the "from" address, or accept the defaults.
6. Select 1 to use an encrypted database password.
7. Enter a name for the Hyperic admin user, or accept the default. Specify a password for the user, and finally an e-mail address.
8. Enter a location for the agent to be installed in.
9. The installer will now install the Hyperic server, which may take some time.

```

C:\WINDOWS\system32\cmd.exe
Taskdefs loaded
Initializing Hyperic HQ 4.6.6 Installation...
Choose which software to install:
1: Hyperic HQ Server
2: Hyperic HQ Agent
You may enter multiple choices, separated by commas.
1,2
HQ server installation path [default 'C:\Program Files\']:
c:\hyperic
Enter the fully qualified domain name of the SMTP server that HQ will use to send email messages [default '192.168.106.130']:
Enter the email address that HQ will use as the sender for email messages [default 'hqadmin@168.106.130']:
Choices:
1: Yes
2: No
Would you like to use an auto generated encryption key to encrypt the database password? [default '1']:
1
What should the username be for the initial admin user? [default 'hqadmin']:
What should the password be for the initial admin user?:
(again):
What should the email address be for the initial admin user? [default 'hqadmin@168.106.130']:
HQ agent installation path [default 'c:\hyperic\']:
Loading install configuration...
Install configuration loaded.
Preparing to install...
Validating agent install configuration...
Validating server install configuration...
Checking server webapp port...
Checking server secure webapp port...
Verifying admin user properties
Validating server DB configuration...
Installing the agent...
Looking for previous installation
Unpacking C:\hyperic-hq-installer-4.6.6-win32\hyperic-hq-installer-4.6.6\hyperic-hq-agent-4.6.6-win32.zip to: c:\hyperic\agent-4.6.6...

```

10. Once the installer completes, it will display a message telling you to install Hyperic as a service.



```
C:\WINDOWS\system32\cmd.exe

Installation Complete:
Server successfully installed to: c:\hyperic\server-4.6.6

You should now install the HQ server as a Windows Service using this command:
c:\hyperic\server-4.6.6\bin\hq-server.bat install

You can then use the Service Control Manager <Control Panel>Services> to
start the HQ server. Note that the first time the HQ server starts up it may
take several minutes to initialize. Subsequent startups will be much faster.

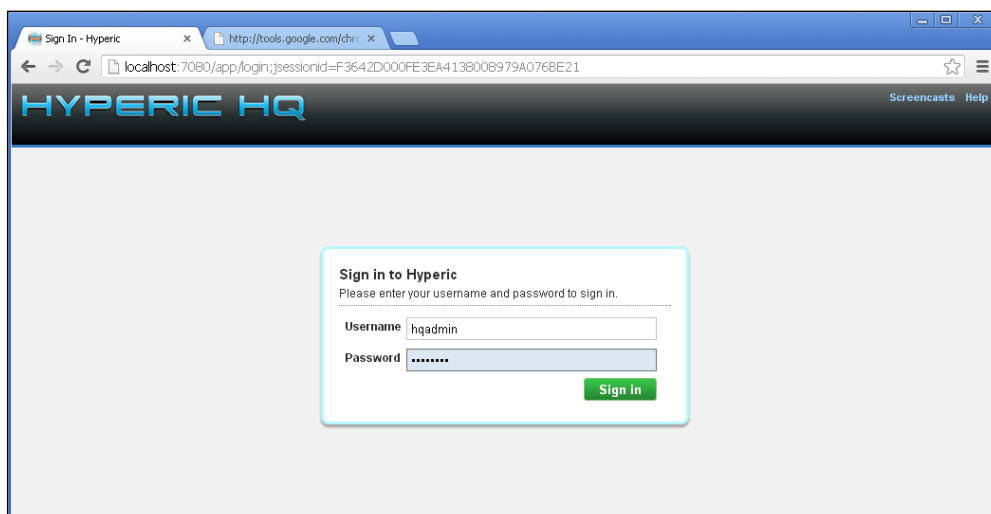
Once the HQ server reports that it has successfully started, you can log in
to your HQ server at:

http://192.168.106.130:7080/
username: hqadmin
password: password

To change your password, log in to the HQ server, click the "Administration"
link, choose "List Users", then click on the "hqadmin" user.

Setup completed.
A copy of the output shown above has been saved to:
C:\hyperic-hq-installer-4.6.6-win32\hyperic-hq-installer-4.6.6\installer\logs\
hq-install.log
```

11. After the service is installed, you can start Hyperic by using the `hq-server.bat` or `hq-server.sh` script.
`hq-server.bat start`
12. Once the server starts, you can connect to it by using the URL provided at the end of the installation script. This is usually `http://localhost:7080/`.



13. You can log in with the username and password that you specified during installation, to see the Hyperic dashboard.



How it works...

Hyperic HQ has an architecture based on a server and agents. The server component consists of a database, used to store configuration data together with the historical metrics that have been gathered. The server also provides a web application, which can be used to view the data and configure the collection of metrics and the creation of alerts.

The server is implemented by using a Tomcat-based web server, and is bundled with a **Java Runtime Environment (JRE)** to run it. It also ships with its own embedded PostgreSQL database for storing configuration data and metrics.

There's more...

In this recipe, we installed the community version of Hyperic, and accepted most of the default settings. In a production setup, you may want to consider using a more resilient deployment by using a more reliable database and using real SSL certificates, rather than the demo self-signed certificates that are distributed with the Hyperic server.

See also

- *The Installing Hyperic agents and Configuring Hyperic to monitor SOA Suite 11g recipes*

Installing Hyperic agents

Hyperic agents need to be installed on each machine that you wish to monitor. The agent will report the metrics back to the Hyperic server. In this recipe, we will learn how to install a Hyperic agent.

Getting ready

You will need to install the Hyperic agent install ZIP file plus the necessary permissions to log in, and install the software on the machine that runs your Oracle SOA Suite application. The Hyperic agent will need to be installed as the same user as Oracle SOA Suite, in order to detect it properly.

How to do it...

We can follow these steps to install our Hyperic agents:

1. Extract the Hyperic agent ZIP file to the location where you want it to be installed.
2. Navigate to the `bin` directory:

```
cd bin
```
3. Run the install task of the relevant `hq-agent` script—either `hq-agent.sh` or `hq-agent.bat`:

```
hq-agent.bat install
```
4. Run the start task of the relevant `hq-agent` script—either `hq-agent.sh` or `hq-agent.bat`:

```
hq-agent.bat start
```
5. The agent will start up, and start prompting for its configuration.

```

C:\WINDOWS\system32\cmd.exe - hq-agent.bat start

E:\hyperic\hyperic-hq-agent-4.6.6-win32\hyperic-hq-agent-4.6.6\bin>hq-agent.bat
install
wrapper ! Hyperic HQ Agent service installed.

E:\hyperic\hyperic-hq-agent-4.6.6-win32\hyperic-hq-agent-4.6.6\bin>hq-agent.bat
start
wrapper ! Starting the Hyperic HQ Agent service...
wrapper ! Waiting to start...
wrapper ! Hyperic HQ Agent started.
[ Running agent setup ]
What is the HQ server IP address: _

```

6. Enter the IP address of the Hyperic server.
7. Answer yes to using a secure communication between the agent and server.
8. If you have not changed the SSL port for your Hyperic Server, accept the default port; otherwise enter your custom port.
9. Enter the administration username for your Hyperic server.
10. Enter the password for your Hyperic administration user.
11. Accept the default IP for the agent bind address.
12. Accept the default port for the agent port.
13. If you are using the default self-signed SSL certificates, the agent will prompt for whether you wish to accept these. Enter yes.

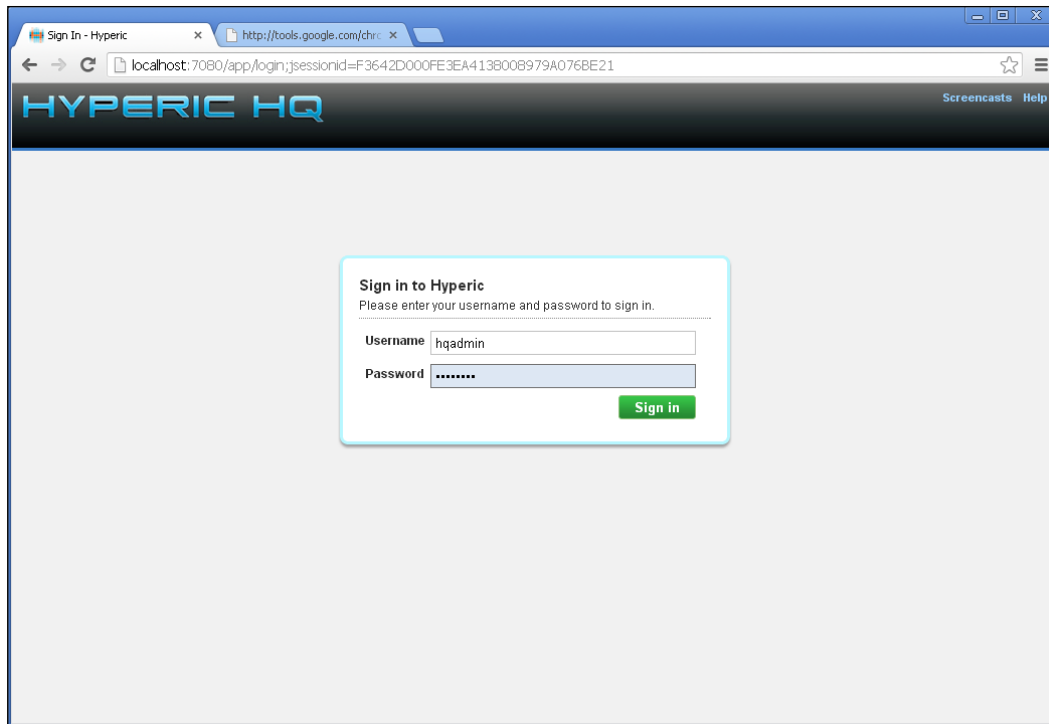
```

C:\WINDOWS\system32\cmd.exe

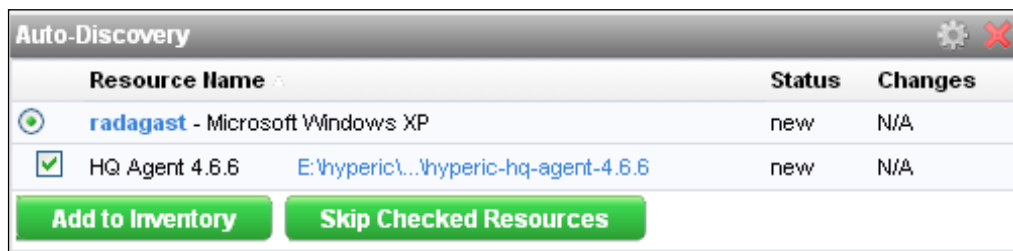
wrapper ! Hyperic HQ Agent started.
[ Running agent setup ]
What is the HQ server IP address: 192.168.106.130
Should Agent communications to HQ always be secure [default=yes]: yes
What is the HQ server SSL port [default=7443]:
- Testing secure connection ... Success
What is your HQ login [default=hqadmin]:
What is your HQ password:
What IP should HQ use to contact the agent [default=192.168.106.129]:
What port should HQ use to contact the agent [default=2144]:
- Received temporary auth token from agent
- Registering agent with HQ
The server to agent communication channel is using a self-signed certificate and
could not be verified
Are you sure you want to continue connecting? [default=no]: yes
- HQ gave us the following agent token
1356372451516-1003282569436907839-2518535934802335588
- Informing agent of new HQ server
- Validating
- Successfully setup agent
E:\hyperic\hyperic-hq-agent-4.6.6-win32\hyperic-hq-agent-4.6.6\bin>

```


14. The agent install will complete.
15. Use your web browser to connect to the Hyperic server console (usually `http://localhost:7080/console`):



16. Enter your administration username and password to log in.
17. You will see that the agent has been discovered by the server, and is listed at the top-right of the dashboard:



18. Click on the **Add to Inventory** button to accept the agent's request to send metrics to the Hyperic server.

How it works...

Many monitoring tools, including Hyperic, use a server and agent architecture. In this architecture, a lightweight agent process runs on each box to be monitored, collecting metrics and sending them back to the server, where they are processed and stored. An agent therefore needs to be installed on each machine that is running one of our WebLogic servers hosting SOA Suite 11g applications.

Hyperic agents are responsible for discovering resources, gathering metrics, and reporting them back to the Hyperic server. To do this, agents need to be installed on each machine that has resources to be monitored, which in our case, means WebLogic servers that are running Oracle SOA Suite.

The agent is implemented as a Java application, and is distributed as a ZIP file that can be extracted into a location. Scripts are provided for both the Windows and Linux versions of the agent; the Windows agent is run by installing it as a Windows service, and then starting that service. When it starts for the first time, it will prompt for its configuration data. This will be stored, and so the agent will no longer prompt for the configuration data once it has been set up. The agent is essentially a container for running plugins, with each plugin knowing how to discover and monitor a specific type of resource (such as a SOA Suite server). The monitoring of SOA Suite is accomplished by the WebLogic plugin, which uses the **Java Management eXtensions (JMX)** interface to connect to WebLogic and retrieve metrics. These metrics are the ones that the JMX subsystem in WebLogic and SOA Suite is already collecting. So the overhead of monitoring the platform is limited to that imposed by connecting to WebLogic, requesting the metrics, and then forwarding them to the Hyperic server. This allows the agents to have a very small overhead on the boxes they monitor (typically no more than 5 percent), although the overhead varies according to the number of metrics that are being collected.

When the agent first starts up and communicates with the server, the server places it in a list of resources that have been discovered but not yet added to the inventory. We need to log in to the server and accept the agent and its discovered resources into the inventory before any monitoring will occur.

There's more...

In these sections we have installed the Hyperic server and agent to use the default self-signed SSL certificates for secure communication. For a production deployment, it is recommended that these certificates are not used, and that you use real SSL certificates, or generate your own set of self-signed certificates.

See also

- ▶ *The Installing the Hyperic server and Configuring Hyperic to monitor SOA Suite 11g recipes*

Configuring Hyperic to monitor SOA Suite 11g

Once the Hyperic server and agents have been installed, there are a few more steps that are needed to get the Hyperic server to monitor the WebLogic server instances that run your Oracle SOA Suite 11g application.

Getting ready

You will need to have the Hyperic server installed and configured, and the Hyperic agent installed and imported into the inventory. You will also need your WebLogic server instances running for Hyperic HQ to be able to discover them. You will also need read and write access for both the Oracle SOA Suite and VMWare Hyperic installations.

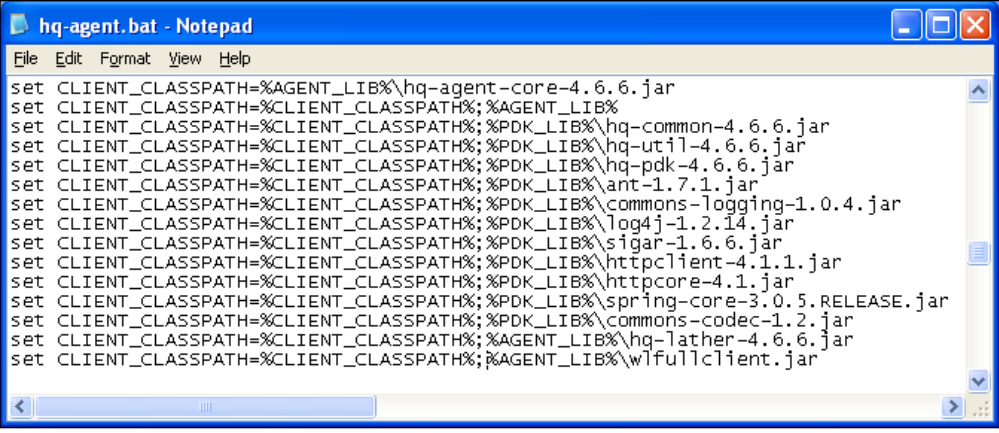
How to do it...

The following steps will configure your Hyperic server and agents to monitor WebLogic and SOA Suite 11g:

1. Navigate to the WebLogic server `lib` directory:

```
cd MIDDLEWARE_HOME/wlserver_10.3/server/lib
```
2. Execute the `jar` builder tool to create a `wlfullclient.jar` file:

```
java -jar ../../../../modules/com.bea.core.jarbuilder.1.7.0.0.jar
```
3. Copy the generated `wlfullclient.jar` file into the `bundles/agent-4.6.6\lib` directory of your Hyperic agent.
4. Edit the file `HQ_AGENT_DIR/bundles/agent-4.6.6/bin/hq-agent.bat` or `hq-agent.sh`. Locate the section that sets the agent classpath, and add `wlfullclient.jar` to it.

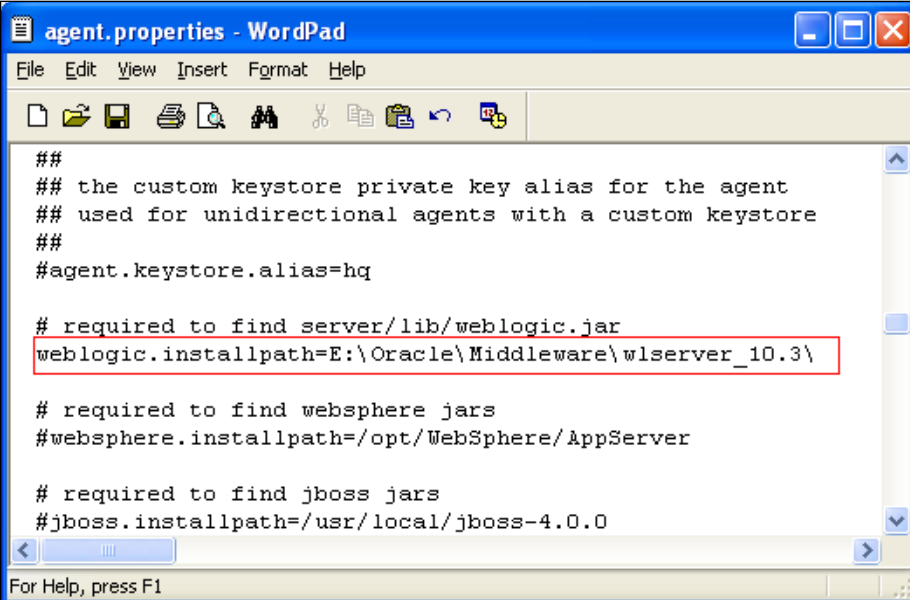


```

set CLIENT_CLASSPATH=%AGENT_LIB%\hq-agent-core-4.6.6.jar
set CLIENT_CLASSPATH=%CLIENT_CLASSPATH%;%AGENT_LIB%
set CLIENT_CLASSPATH=%CLIENT_CLASSPATH%;%PDK_LIB%\hq-common-4.6.6.jar
set CLIENT_CLASSPATH=%CLIENT_CLASSPATH%;%PDK_LIB%\hq-util-4.6.6.jar
set CLIENT_CLASSPATH=%CLIENT_CLASSPATH%;%PDK_LIB%\hq-pdk-4.6.6.jar
set CLIENT_CLASSPATH=%CLIENT_CLASSPATH%;%PDK_LIB%\ant-1.7.1.jar
set CLIENT_CLASSPATH=%CLIENT_CLASSPATH%;%PDK_LIB%\commons-logging-1.0.4.jar
set CLIENT_CLASSPATH=%CLIENT_CLASSPATH%;%PDK_LIB%\log4j-1.2.14.jar
set CLIENT_CLASSPATH=%CLIENT_CLASSPATH%;%PDK_LIB%\sigar-1.6.6.jar
set CLIENT_CLASSPATH=%CLIENT_CLASSPATH%;%PDK_LIB%\httpclient-4.1.1.jar
set CLIENT_CLASSPATH=%CLIENT_CLASSPATH%;%PDK_LIB%\httpcore-4.1.jar
set CLIENT_CLASSPATH=%CLIENT_CLASSPATH%;%PDK_LIB%\spring-core-3.0.5.RELEASE.jar
set CLIENT_CLASSPATH=%CLIENT_CLASSPATH%;%PDK_LIB%\commons-codec-1.2.jar
set CLIENT_CLASSPATH=%CLIENT_CLASSPATH%;%AGENT_LIB%\hq-lather-4.6.6.jar
set CLIENT_CLASSPATH=%CLIENT_CLASSPATH%;%AGENT_LIB%\wlfuillclient.jar

```

5. Edit the file HQ_AGENT_DIR/conf/agent.properties, and set the weblogic.installpath property to the location of the wls_server_10.3 directory in your middleware home location.



```

##
## the custom keystore private key alias for the agent
## used for unidirectional agents with a custom keystore
##
#agent.keystore.alias=hq

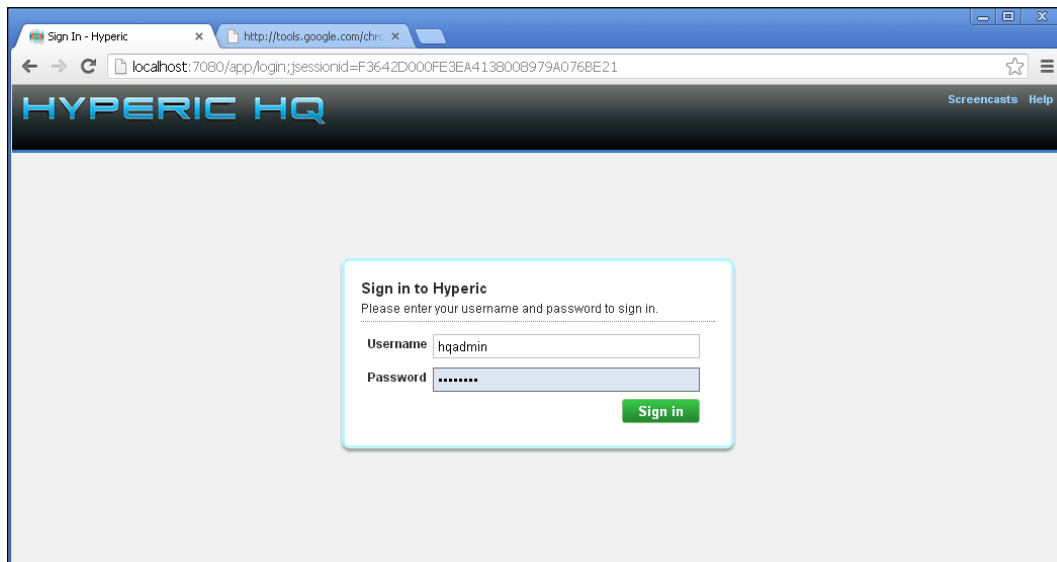
# required to find server/lib/weblogic.jar
weblogic.installpath=E:\Oracle\Middleware\wls_server_10.3\

# required to find websphere jars
#websphere.installpath=/opt/WebSphere/AppServer

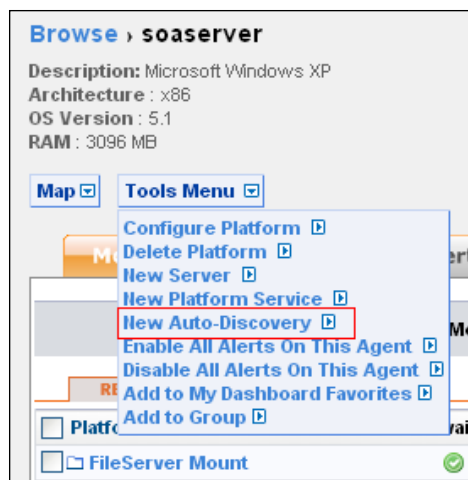
# required to find jboss jars
#jboss.installpath=/usr/local/jboss-4.0.0

```

6. Restart the Hyperic agent by using the script in HQ_AGENT_DIR/bin:
hq_agent.bat start
hq_agent.bat stop
7. Log in to the Hyperic console by using your administration username and password.



8. Navigate to the relevant platform that hosts your SOA domain Administration server from the **Resources** tab and select **Browse**. Click on the **Tools** menu, and select **New Auto-Discovery**.



9. Select the checkbox for **Select All Server Types** and scroll to the bottom of the page. Enter the relevant details for the drives to scan, and click on **Ok**. This will start the agent running a new auto-discovery on the platform.

Quick Auto-Discovery Scan

Click 'OK' to start an immediate system scan on all resource types. This will include a scan of the Windows registry and process table. If you also want to perform a filesystem scan or if you'd like to limit the resource types to be discovered, fill out the remainder of the form below.

Extended Filesystem Scan Configuration

* Scan For Server Types: ☒ **Select All Server Types**

<input type="checkbox"/> .NET 1.1	<input type="checkbox"/> .NET 2.0	<input type="checkbox"/> .NET 3.0
<input type="checkbox"/> Active Directory	<input type="checkbox"/> ActiveMQ 4.0	<input type="checkbox"/> ActiveMQ 5.0
<input type="checkbox"/> ActiveMQ 5.1	<input type="checkbox"/> ActiveMQ 5.2	<input type="checkbox"/> ActiveMQ 5.3
<input type="checkbox"/> ActiveMQ 5.4	<input type="checkbox"/> ActiveMQ Embedded 5.0	<input type="checkbox"/> ActiveMQ Embedded
<input type="checkbox"/> ActiveMQ Embedded 5.2	<input type="checkbox"/> ActiveMQ Embedded 5.3	<input type="checkbox"/> ActiveMQ Embedded
<input type="checkbox"/> Alfresco 2.0.x	<input type="checkbox"/> Apache 1.3	<input type="checkbox"/> Apache 2.0
<input type="checkbox"/> Apache 2.2	<input type="checkbox"/> Apache httpd	<input type="checkbox"/> Apache Tomcat 5.5
<input type="checkbox"/> Apache Tomcat 6.0	<input type="checkbox"/> Apache Tomcat 7.0	<input type="checkbox"/> Apache-ERS 2.3
<input type="checkbox"/> Apache-ERS 2.4	<input type="checkbox"/> Apache-ERS 3.x	<input type="checkbox"/> Apache-ERS 4.x
<input type="checkbox"/> ColdFusion 6.x	<input type="checkbox"/> ColdFusion 7.x	<input type="checkbox"/> DB2 JDBC 9.x
<input type="checkbox"/> DB2 JDBC Database Manager 9.x	<input type="checkbox"/> Exchange 2000	<input type="checkbox"/> Exchange 2003
<input type="checkbox"/> Exchange 2007	<input type="checkbox"/> Exchange 2010	<input type="checkbox"/> Exchange 5.5
<input type="checkbox"/> Exchange Transport 2007	<input type="checkbox"/> Exchange Transport 2010	<input type="checkbox"/> Geronimo 1.0
<input type="checkbox"/> GlassFish 9.x	<input type="checkbox"/> HQ Agent	<input type="checkbox"/> JIS 4.x
<input type="checkbox"/> JIS 5.x	<input type="checkbox"/> JIS 6.x	<input type="checkbox"/> JIS 7.x
<input type="checkbox"/> Informix 10.0	<input type="checkbox"/> Planet Admin 4.1	<input type="checkbox"/> Planet Admin 6.0
<input type="checkbox"/> JBoss 3.2	<input type="checkbox"/> JBoss 4.0	<input type="checkbox"/> JBoss 4.2
<input type="checkbox"/> JBoss 4.3	<input type="checkbox"/> JBoss 5.0	<input type="checkbox"/> JBoss 5.1
<input type="checkbox"/> JPress 6.0	<input type="checkbox"/> JPress 7	<input type="checkbox"/> JPress Host Controller

10. The auto-discovery will discover the Admin Server resources; you can add them to the inventory by going to **Dashboard**, selecting them in the **Auto-Discovery** portlet, and selecting **Add to Inventory**.

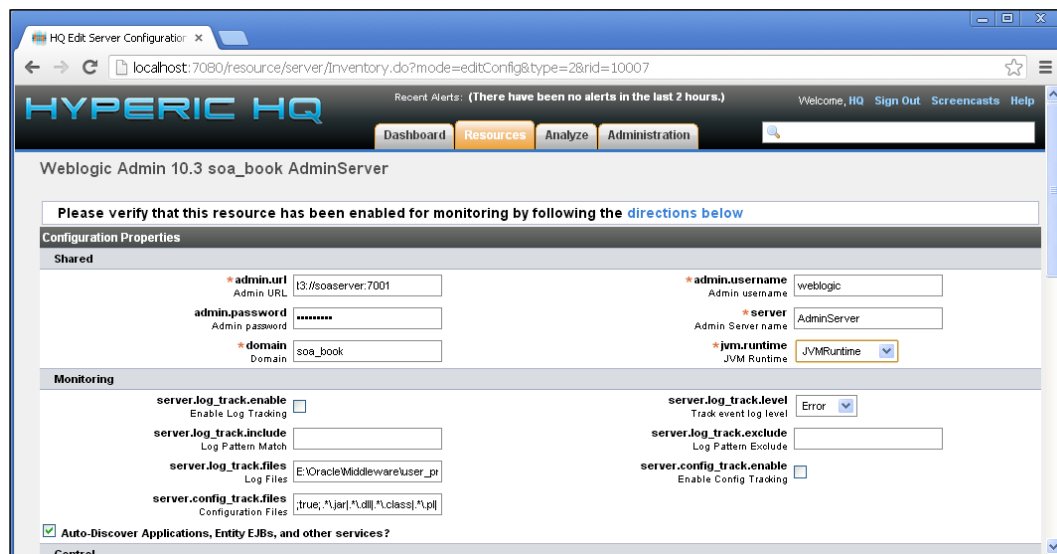
Resource Name	Status	Changes
<input checked="" type="checkbox"/> soaserver - Microsoft Windows XP	modified	server set changed
<input checked="" type="checkbox"/> Weblogic Admin 1...book AdminServer E:\Oracle\...\domains\soa_book	new	N/A

Add to Inventory **Skip Checked Resources**

11. Select **Resources** and then **Browse**, and select the platform that hosts the SOA domain administration server. You will notice that in the **Resources** panel, the **WebLogic AdminServer** resource has a gray question mark next to it. We need to configure it with the correct username and password. Click on **AdminServer** in the **Resources** panel, and you will see a message appear telling you that **Configuration Properties** need to be set. Click on the words **Configuration Properties** to go to the configuration page.



12. Enter the correct configuration properties. The ones you probably need to change are the password and the JVM Runtime setting.



13. Scroll to the bottom of the page and click on **Ok**; the configuration properties will be updated, and Hyperic will start collecting the metrics.
14. The plugin will automatically discover the managed servers that are defined in the domain and start monitoring these too.

How it works...

The Hyperic HQ agent contains a plugin to allow it to discover the WebLogic server instances; it discovers the admin server by finding the running process, and then interrogates the admin server to find each managed server in the domain. Before the agent can discover WebLogic, we need to make a few configuration changes to allow the agent to talk to the WebLogic server instance. These include setting the `weblogic.installpath` property and placing `wlfullclient.jar` on the classpath. These changes are needed to allow the agent to communicate with the WebLogic administration server.

Once the agent has discovered the WebLogic administration server, we need to go into the Hyperic console to provide the password necessary for the agent to communicate with the server. The agent will then automatically discover the managed servers and add them to the relevant platforms, and then any resources underneath the servers will be added.

There's more...

In order to associate the managed servers with platforms that are managed by Hyperic, the agent uses the listen address of the managed server to match against a platform name. This means that the listen address of the managed server needs to be set to the hostname of the platform, as returned by the `hostname` command. If the listen address is set to an IP address or is empty (to bind to all network interfaces), the managed server will not be added to the inventory, as the plugin will be unable to work out which platform to associate the server with.

See also

- ▶ The *Installing the Hyperic server* and *Installing Hyperic agents* recipes

Monitoring the SOA Suite server availability

One of the most obvious things to have Hyperic HQ monitor for you is whether your servers are available; that is, whether they are running and responding to requests. In this recipe, we will learn how to view the availability information for our SOA Suite 11g instances.

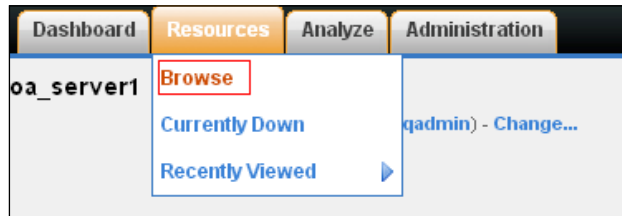
Getting ready

You will need to install Oracle SOA Suite and the Hyperic HQ server and agents for this recipe. Both Hyperic HQ and Oracle SOA Suite will need to be running. You will also need the login credentials for the Hyperic HQ console.

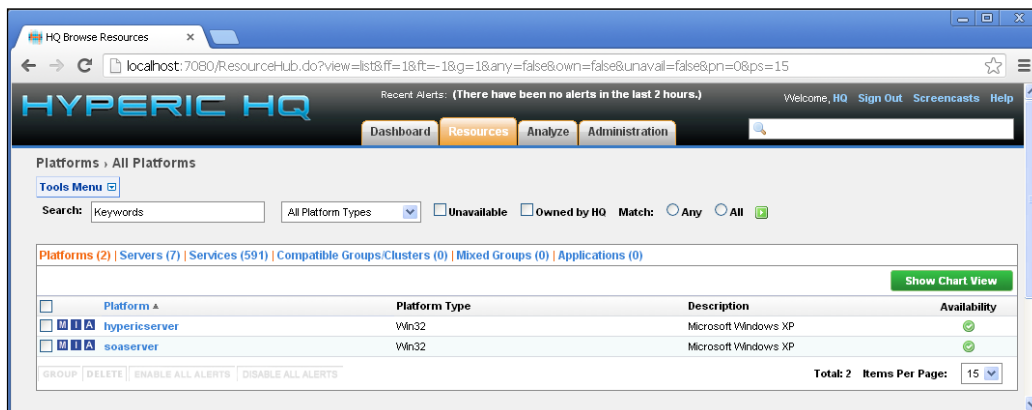
How to do it...

These steps show us how to monitor system availability with Hyperic:

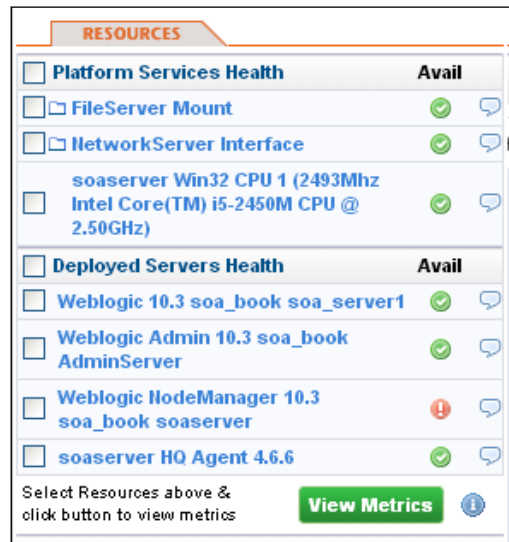
1. Log in to the Hyperic HQ console.
2. Open the **Resources** menu, and select **Browse**.



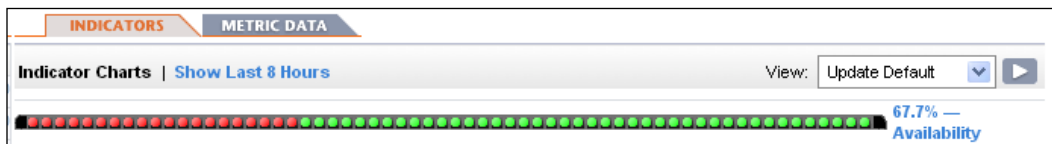
3. Select the platform that has the server you want to monitor, which should be one of your SOA Suite managed servers.



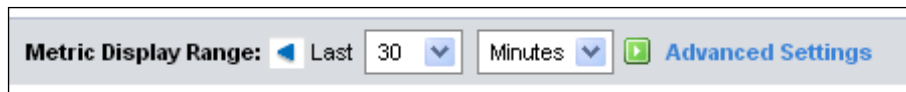
- On the **Resources** panel on the left-hand side, select the WebLogic managed server that you wish to monitor.



- The **Availability** track is shown at the top of the center panel, with red dots indicating periods when the server was not available.



- Use the time settings at the top of the graph panel to select the time period you are interested in.



How it works...

The Hyperic agent determines the availability, which lets us know whether it can connect to the WebLogic server; if the agent can get a response from the server, then it is considered to be available. The availability metric data is sent by the agent to the Hyperic HQ server, which stores it in its database and makes it available via the Web console.

There's more...

All resources that are monitored by Hyperic have availability data associated with them, including the platform itself. We have found that the most useful of these to monitor are the WebLogic server instances that run your SOA Suite applications, but monitoring the platform availability can also be useful.

The availability track uses a sequence of dots to show the availability of the resource during the selected time period. A green dot indicates that the resource was 100% available during the time period that the dot represents. A red dot indicates that the resource was less than 100% available for the time period.

See also

- ▶ The *Installing the Hyperic server, Installing Hyperic agents, and Configuring Hyperic to monitor SOA Suite 11g* recipes
- ▶ The *Monitor garbage collection using jstat* recipe in *Chapter 1, Identifying Problems*

Monitoring the JVM memory usage

Many of the common causes of performance problems have their roots in memory management, so monitoring the memory usage of the servers in your SOA Suite domain can alert you to many performance problems before they become problematic. In this recipe, we will see how we can view graphs of the current and historical memory usage of our SOA Suite 11g application.

Getting ready

You will need to install Oracle SOA Suite and the Hyperic HQ server and agents for this recipe. Both Hyperic HQ and Oracle SOA Suite will need to be running. You will also need the login credentials for the Hyperic HQ console.

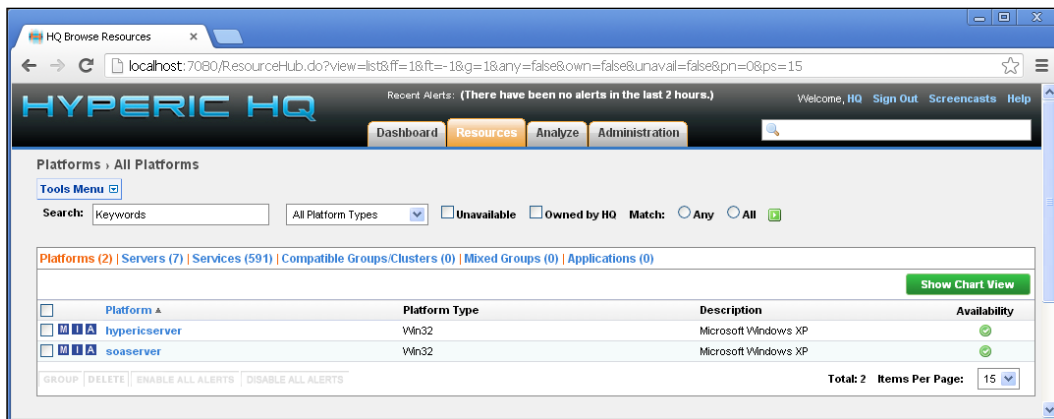
How to do it...

Follow these steps to monitor the memory usage of our SOA Suite 11g application.

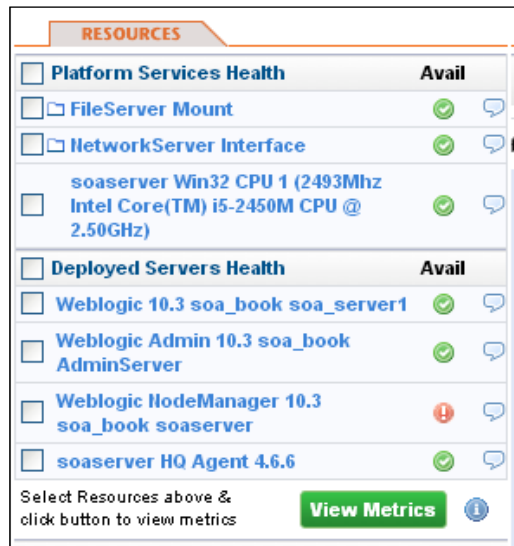
1. Log in to the Hyperic HQ console.
2. Open the **Resources** menu, and select **Browse**.



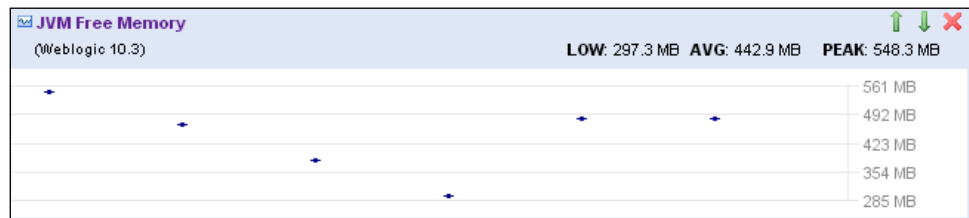
3. Select the platform that has the server you want to monitor, which should be one of your SOA Suite managed servers.



- On the left-hand side of the **Resources** panel, select the WebLogic managed server that you wish to monitor.



- The graph pane in the center of the page contains the graph for the **JVM Free Memory** metric.



- Use the time settings at the top of the graph panel to select the time period you are interested in.



7. Click on the graph title to view a more detailed graph.



How it works...

The Hyperic HQ agent periodically connects to the WebLogic server instances that are in its inventory, and uses the JMX service to obtain metrics on a number of measurements, including the JVM free memory. It reports this data back to the server, which stores it in its database. We can then view this collected data by using the Hyperic console.

JVM free memory is an interesting metric to monitor, although one that is hard to set up alerting on, because of the nature of JVM memory management; memory will be used up until there is none free, and then the garbage collector will kick in and free up a large chunk. This gradual use of memory, and then a sudden freeing up results in the familiar saw tooth pattern of JVM memory graphs, which are described in other chapters. This does not mean that monitoring the free JVM memory is useless, as we can identify memory leaks by the fact that the saw tooth pattern frees up less and less memory each time the garbage collector is run.

There's more...

Since memory usage patterns can change a lot depending upon the usage of the application, it is very useful to have an understanding of what the graph of JVM free memory looks like during normal use. If you know what the usage typically looks like on a Monday morning, you are in a much better position to identify an abnormally high or low memory usage on a particular Monday morning.

See also

- ▶ The *Installing the Hyperic server, Installing Hyperic agents, and Configuring Hyperic to monitor SOA Suite 11g* recipes
- ▶ The *Monitor garbage collection using jstat* recipe in *Chapter 1, Identifying Problems*

Monitoring the platform CPU usage

In this recipe, we are going to learn how to view the CPU usage of the platforms running SOA Suite 11g. CPU usage is one of the key areas where bottlenecks can occur, and knowing that our CPU is running at 100 percent allows us to focus our efforts into working out what is using the CPU.

Getting ready

You will need to install Oracle SOA Suite and the Hyperic HQ server and agents for this recipe. Both Hyperic HQ and Oracle SOA Suite will need to be running. You will also need the login credentials for the Hyperic HQ console.

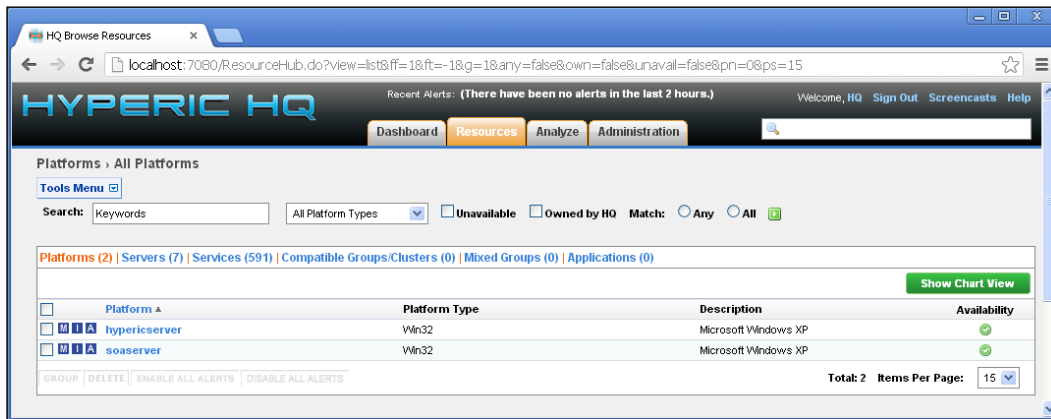
How to do it...

These steps will show us how to monitor the CPU usage of our SOA Suite platforms:

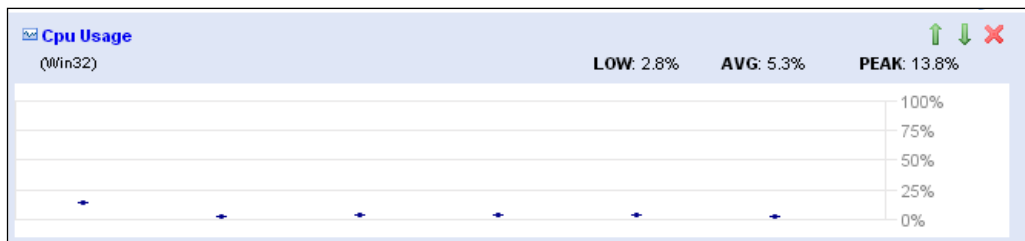
1. Log in to the Hyperic HQ console.
2. Open the **Resources** menu, and select **Browse**.



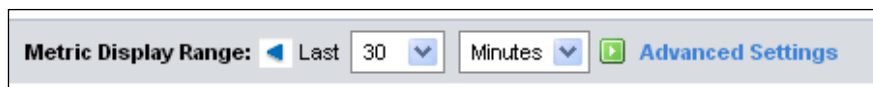
3. Select the platform that has the server you want to monitor, which should be one of your SOA Suite managed servers.



4. The CPU usage will be displayed in the graph area in the center of the screen.



5. Use the display range settings at the top of the graph pane to select the time period you are interested in.



How it works...

The Hyperic agent uses the **SIGAR** library to collect metric data related to operating systems, including the CPU usage. This data is sent by the agent to the Hyperic server, which stores it in the database. We can then use the Hyperic console to browse this information.

When monitoring the CPU usage, we obviously look to ensure that the system is not running at 100 percent CPU usage, which implies that there is a performance problem somewhere in the system. CPU usage is a good metric to set alerts on, because it should not be high in normal circumstances, and it is always worth investigating the incidents of high CPU usage.

There's more...

As with monitoring JVM memory usage, having a good understanding of what your CPU usage pattern usually looks like can be very useful for diagnosing an abnormal behavior. For example, many organizations have an anti-virus or batch process that runs overnight or during quiet periods, which can use significant amount of the CPU when running. If you are not aware that this is a normal behavior, it can be easy to mistake it for the cause of a performance problem, and waste significant time looking for a CPU bottleneck, where the true performance bottleneck is elsewhere.

See also

- ▶ The *Installing the Hyperic server*, *Installing Hyperic agents*, and *Configuring Configuring Alerts in Hyperic* recipes

Monitoring the data source usage

Oracle SOA Suite applications make a very heavy use of database connections, so if the connection pools get exhausted, then performance can suffer dramatically. By monitoring the data source usage, you can ensure that the number of available connections is sufficient.

Getting ready

You will need to install Oracle SOA Suite and the Hyperic HQ server and agents for this recipe. Both Hyperic HQ and Oracle SOA Suite will need to be running. You will also need the login credentials for the Hyperic HQ console.

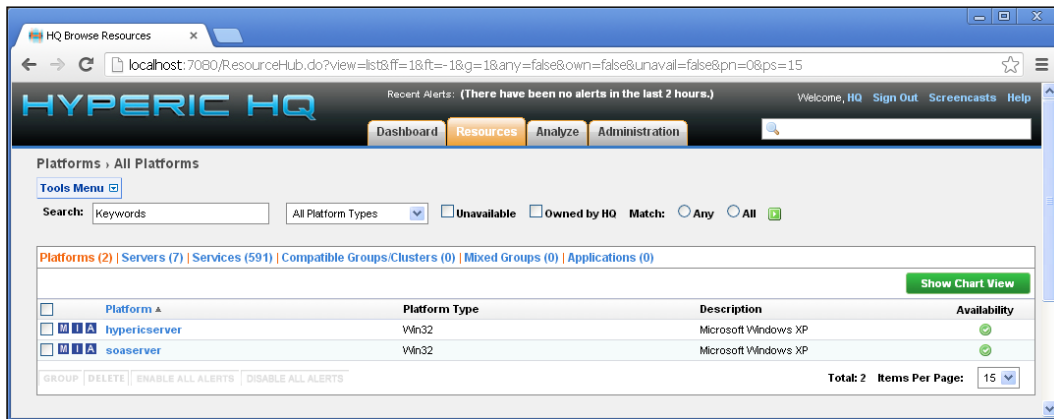
How to do it...

By following these steps, we can monitor the data source usage:

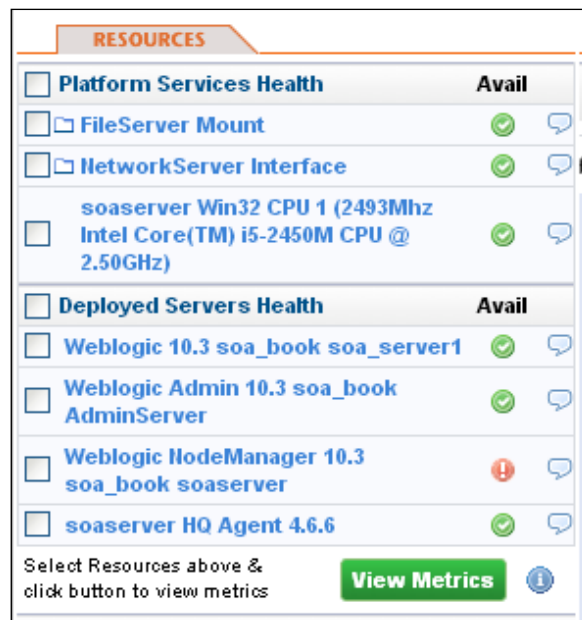
1. Log in to the Hyperic HQ console.
2. Open the **Resources** menu, and select **Browse**.



3. Select the platform that has the server you want to monitor, which should be one of your SOA Suite managed servers.

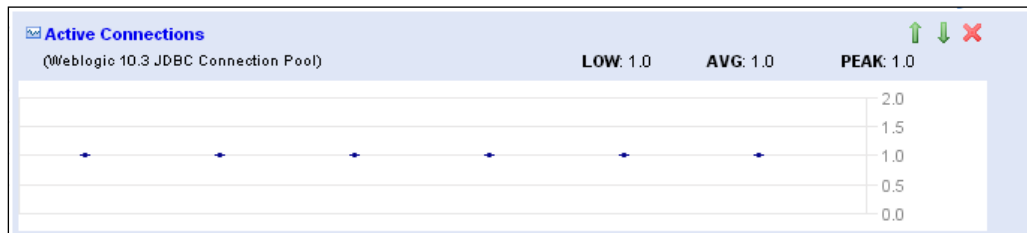


4. On the left-hand side of the **Resources** panel, select the WebLogic managed server that you wish to monitor.



5. Select the data sources entry in the **Resources** pane on the left-hand side.
6. Select the data source you are interested in from the **Resources** pane on the left-hand side.

7. The graph pane will now display the metrics for the data source, including the number of active connections.



8. Use the display range settings at the top of the graph pane to select the time period you are interested in.

Metric Display Range: Last Minutes [Advanced Settings](#)

How it works...

The Hyperic HQ Agent periodically connects to the WebLogic server instances that are in its inventory, and uses the JMX service to obtain metrics on a number of measurements, including the data source usage. It reports this data back to the server, which stores it in its database. We can then view this collected data by using the Hyperic console.

The metrics we are interested in are the individual metrics for each data source, rather than the aggregate metrics that Hyperic also provides. These individual metrics allow us to see which data sources are getting low on available connections, and do something about it before they run out. Because connection pools should not usually be running near their maximum limits, this is a good candidate for configuring alerting.

There's more...

A common problem with data sources is the application code that leaks connections, which occurs when there is a use-case within the application that results in a connection being taken from the pool, but never returned. This is usually found in edge-cases because it is often quickly spotted in testing in the main use-cases. The symptoms of a connection leak are that the baseline number of active connections keeps rising, giving the appearance of something similar to a set of steps.

See also

- ▶ The *Installing the Hyperic server, Installing Hyperic agents, and Configuring Configuring Alerts in Hyperic recipes*

Monitoring open sockets

Sockets and file handles are resources that are limited by the operating system, and can get exhausted in busy SOA Suite applications. This recipe shows us how to monitor the open sockets.

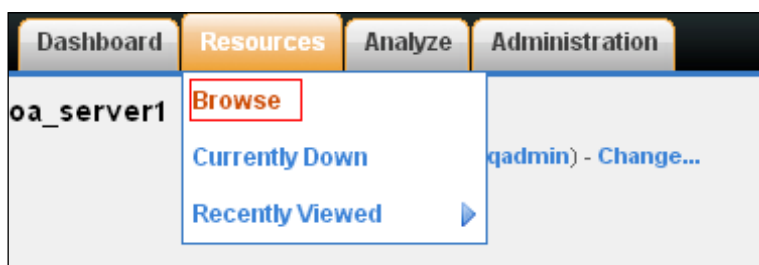
Getting ready

You will need to install Oracle SOA Suite and the Hyperic HQ server and agents for this recipe. Both Hyperic HQ and Oracle SOA Suite will need to be running. You will also need the login credentials for the Hyperic HQ console.

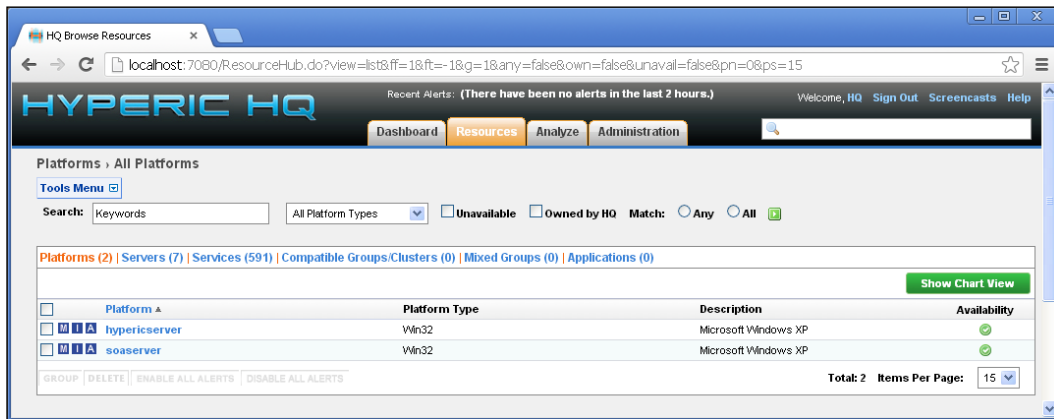
How to do it...

By following these steps, we can monitor the open sockets:

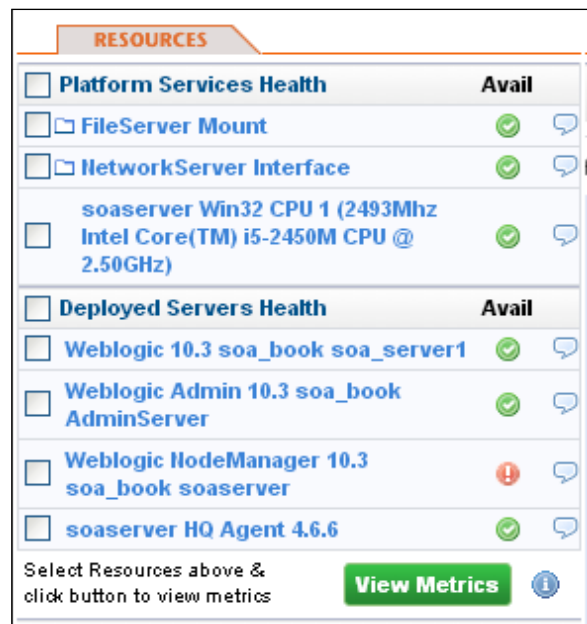
1. Log in to the Hyperic HQ console.
2. Open the **Resources** menu, and select **Browse**.



3. Select the platform that has the server you want to monitor, which should be one of your SOA Suite managed servers.



4. On the left-hand side of the **Resources** panel, select the WebLogic managed server that you wish to monitor.



5. Scroll down the available graphs to find the display of the number of open sockets.
6. Use the display range settings to view the time period you are interested in.



How it works...

The Hyperic HQ agent periodically connects to the WebLogic server instances that are in its inventory, and uses the JMX service to obtain metrics on a number of measurements, including the number of open sockets. It reports this data back to the server, which stores it in its database. We can then view this collected data by using the Hyperic console.

The number of open sockets can give an indication of how busy the system is, and as sockets are a resource that is often limited, this can cause a number of performance problems, as sockets are needed for much of the internal and external communication that goes on with an SOA Suite 11g application. Running out of available sockets can cause some very strange behavior that can be hard to diagnose if we don't realize the underlying cause.

There's more...

In Linux and Unix operating systems, each open socket uses a file descriptor, and the number of file descriptors available to a particular user is controlled by the operating system. As threads and open files also use file descriptors, and Oracle SOA Suite uses a lot of all the three resources, then running out of file descriptors can be a common cause of performance problems on Linux- or Unix-based operating systems. For this reason, it is especially important to monitor the number of open sockets on these environments.

See also

- ▶ The *Installing the Hyperic server, Installing Hyperic agents, and Configuring Configuring Alerts in Hyperic* recipes
- ▶ The *Increasing the number of file descriptors in Linux* recipe in Chapter 6, *Platform Tuning*

Monitoring committed transactions

The number of committed transactions on a server gives a good indication of how busy it is, so this can be a good indicator of the user load on a system. This recipe will show us how to view the Hyperic charts and metrics related to the Java Transaction API subsystem.

Getting ready

You will need to install Oracle SOA Suite and the Hyperic HQ server and agents for this recipe. Both Hyperic HQ and Oracle SOA Suite will need to be running. You will also need the login credentials for the Hyperic HQ console.

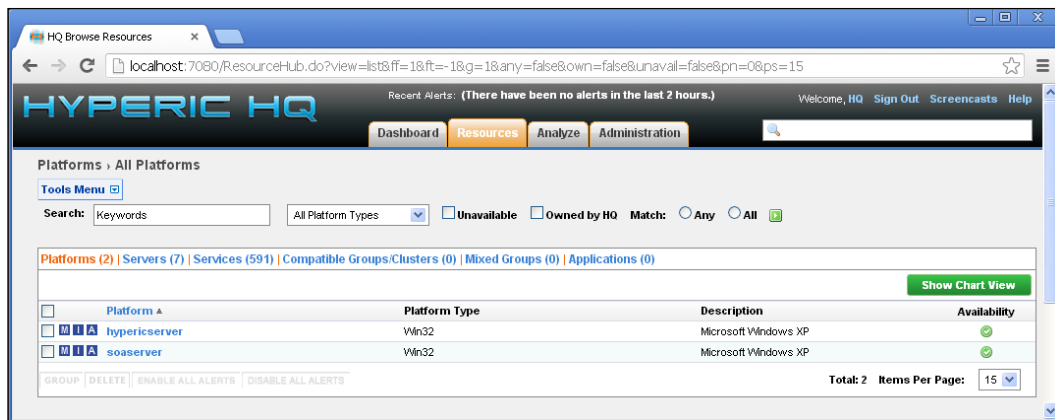
How to do it...

Follow these steps to monitor the number of committed transactions:

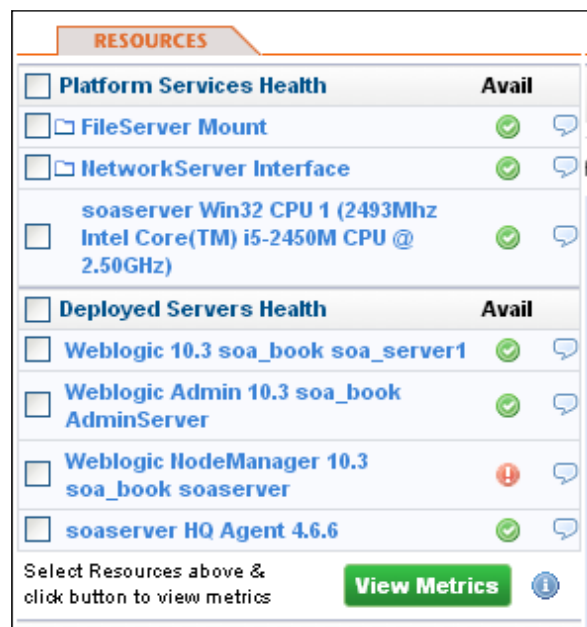
1. Log in to the Hyperic HQ console.
2. Open the **Resources** menu, and select **Browse**:



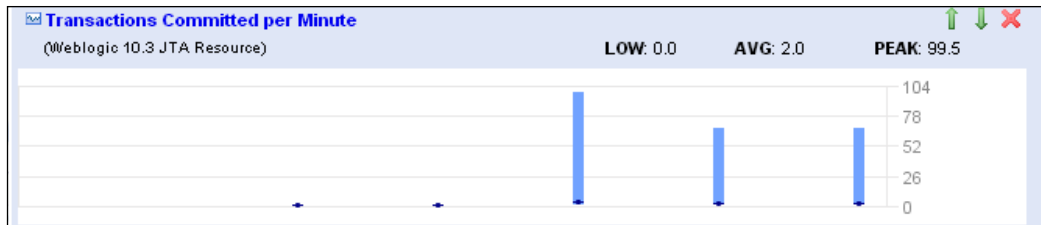
3. Select the platform that has the server you want to monitor, which should be one of your SOA Suite managed servers.



- On the left-hand side of the **Resources** panel, select the WebLogic managed server that you wish to monitor.



5. Select the **JTA** resource from the **Resources** pane on the left-hand side.
6. The graph pane will show a number of resources related to JTA transactions, including the number of transactions committed per minute.



7. Use the display range settings to view the time period you are interested in.

Metric Display Range: ◀ Last 30 Minutes ▶ [Advanced Settings](#)

How it works...

The Hyperic HQ agent periodically connects to the WebLogic server instances that are in its inventory, and uses the JMX service to obtain metrics on a number of measurements, including the JTA transaction statistics. It reports this data back to the server, which stores it in its database. We can then view this collected data by using the Hyperic console.

The number of committed JTA transactions are a very good indicator of how busy a system is. We can therefore use it to detect when an abnormally high load affects our system, to be aware of the potential performance problems that this may cause.

There's more...

WebLogic server does have a limit on the number of concurrent JTA transactions that can be going on at any one time, so this metric can be a useful indicator of whether our system is reaching that limit.

See also

- ▶ The *Installing the Hyperic server, Installing Hyperic agents, and Configuring Configuring Alerts in Hyperic* recipes

Configuring alerts in Hyperic

Monitoring an application provides information on how that application is behaving, and it can be compared to the old data to understand when an application is behaving abnormally, but if all we have is graphical monitoring, we need someone to sit and watch the metrics looking for abnormal behavior. Alerting allows us to have the monitoring framework notify us when a metric breaches some boundary condition, meaning we can receive an e-mail or other notification when something needs looking at, and don't have to sit looking at the metrics.

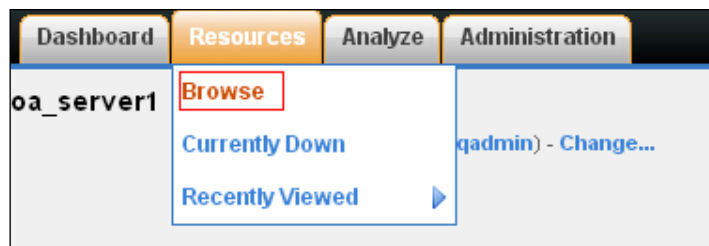
Getting ready

You will need to install Oracle SOA Suite and the Hyperic HQ server and agents for this recipe. Both Hyperic HQ and Oracle SOA Suite will need to be running. You will also need the login credentials for the Hyperic HQ console. Hyperic will need to be configured with working e-mail settings to be able to send out alert e-mails.

How to do it...

We can set up alerts in Hyperic by following these steps:

1. Log in to the Hyperic HQ console.
2. Open the **Resources** menu, and select **Browse**.



3. Select the platform that has the server you want to set an alert on, which should be one of your SOA Suite managed servers.
4. Navigate to the relevant resource, click on the name of the metric on the graph pane in the center of the screen; this will show the detailed graph:



5. Click on **Define New Alert** on the menu at the top-right of the graph.



6. This will take you to the alert definition window. Enter the details such as the name, priority, and description, and then set the condition that you wish the alert to fire on. Finally, set whether you wish the alert to fire once, or every time the condition is true, or be damped for a time after it fires.

HYPERIC HQ Recent Alerts: (There have been no alerts in the last 2 hours.) Welcome, HQ Sign Out Screenshots Help

Dashboard Resources Analyze Administration

Edit hypericserver: New Alert Definition

Alert Properties

Name: Priority:

Description: Active: ☒ Yes ☐ No

Condition Set

If Conditions: ☒ Metric: is (absolute value)

☐ value changes

☐ Inventory Property: value changes

☐ Events/Logs Level: and match substring (optional, 150 chars max):

☐ Config changed and match file name (optional, 150 chars max):

Enable Action(s): ☒ Each time conditions are met

☐ Once every times conditions are met within a time period of minutes

☒ Generate one alert and then disable alert definition until fixed

Configure Actions for this Alert Definition after clicking "OK".

01/04/2013 07:38 PM SYSTEM About Hyperic Version 4.6.6 Copyright © 2004-2012 VMware, Inc. www.hyperic.com

7. Click on **Ok**; this will take you to the alert summary screen:

HYPERIC HQ Recent Alerts: (There have been no alerts in the last 2 hours.) Welcome, HQ Sign Out Screenshots Help

Dashboard Resources Analyze Administration

hypericserver: High CPU Usage: Alert Definition

[<< Return to Alert Definitions](#)

Alert Properties

Name: High CPU Usage Priority: 1 - Medium

Description: CPU Usage is high Active: Yes

Date Created: 01/04/2013 07:38 PM

Date Modified: 01/04/2013 07:38 PM

Condition Set

If Conditions: Cpu Usage > 8,000.0%

Enable Action(s): Each time conditions are met

Generate one alert and then disable alert definition until fixed

Escalation

☐ Email

[<< Return to Alert Definitions](#)

Total: 0 Items Per Page: 15

01/04/2013 07:38 PM SYSTEM About Hyperic Version 4.6.6 Copyright © 2004-2012 VMware, Inc. www.hyperic.com

8. At the bottom of the summary screen is the notifications section. Click on **Notify Other Recipients** and then **Add To List...**. This will allow you to enter the e-mail addresses of everyone you wish to be notified when the alert condition is true.

How it works...

Alerts are generated by the server when it receives metrics from agents. It filters the stream of incoming metrics and checks whether any will cause an alert to fire. So, depending upon how frequently agents are gathering metrics and reporting them to the server, it may take a few minutes before an alert is fired. When an alert is triggered, the associated actions are processed, which will normally involve sending e-mails to users or mailing lists, and possibly notifying Hyperic HQ users via their dashboards.

In this recipe, we configured a simple e-mail notification, although it is possible to have more complex actions, especially if you wish to write a custom action plugin. One of the most common actions to perform is to raise an SNMP event to a corporate-level incident management system, which can be accomplished by using the OpenNMS action. Hyperic has a plugin-based architecture, so it is possible to produce almost any other notification methods if you wish.

There's more...

The fact that alerts are generated on the server side means you need to give yourself sufficient time to be able to react to a condition, although it is still necessary to avoid false positives. This means that setting the alert thresholds can be quite a complex task, which may require several iterations to get it correct.

In addition setting alerts on metric conditions, it is also possible to set them up for configuration changes or errors being logged to a log file.

For more information about configuring alerts, see the Hyperic documentation at <https://support.hyperic.com/display/DOC/Defining+Alerts>.

See also

- ▶ *The Installing the Hyperic server, Installing Hyperic agents, and Configuring Hyperic to monitor SOA Suite 11g recipes*

Monitoring the system using the DMS servlet

The **DMS Spy servlet** provides access to information from the Oracle Dynamic Monitoring System. It is not strictly a monitoring tool, as it does not display historical metrics or trends, but has a wide range of metrics, so is nevertheless useful.

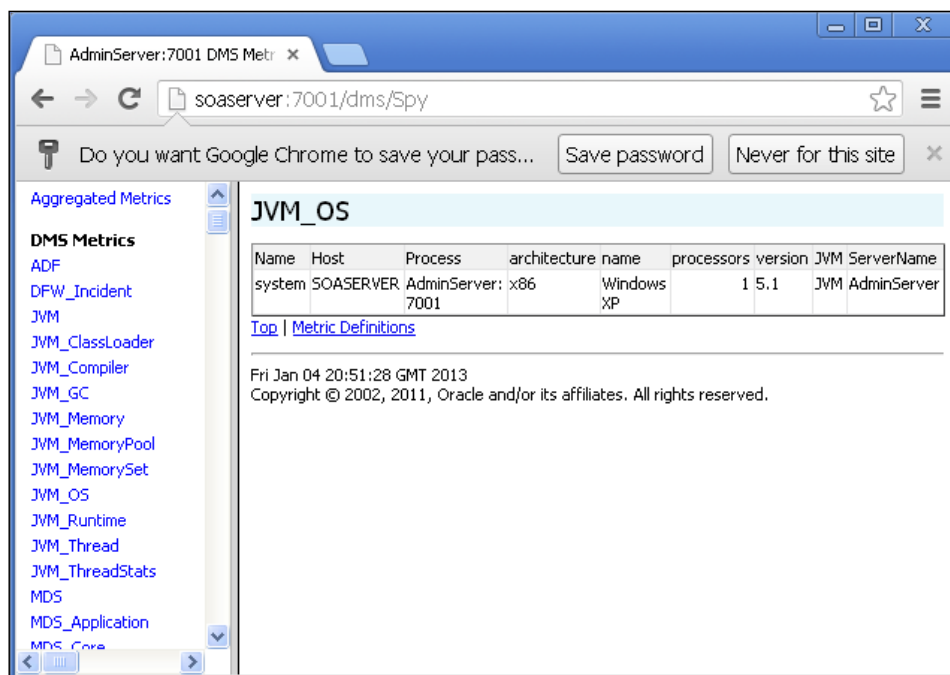
Getting ready

You will need to have your WebLogic SOA Domain administration server running, and will need to know the administration credentials for this recipe.

How to do it...

These steps show us how to use the DMS servlet:

1. Open a web browser, and point it at `http://servername:port/dms/Spy`.
For example, `http://localhost:7001/dms/Spy`.
2. A login dialog box will pop up; enter your domain administration credentials.
3. The DMS Spy servlet will open, displaying a list of available metrics:



4. Select the metric you want from the available categories on the left, and it will be displayed in the main pane.

How it works...

The Oracle **Dynamic Monitoring System (DMS)** is a library that pulls together metrics from a number of different sources, and exposes them in a common way. This does not only include JMX metrics from the servers, but also other metrics. In fact, you can add your own metrics to the DMS by adding sensors into your Oracle SOA Suite application. We recommend doing this if you wish to track performance metrics from within your application processes.

The DMS Spy servlet provides a simple way to view the metrics made available by the DMS, and is deployed by default on the administration server in a SOA Suite domain. On its own, it is not very powerful, but the simple format of the pages means that you can build simple "screen scraping" tools that will pull the metrics from the DMS servlet and make them available.

There's more...

A majority of the information available via the DMS servlet is also exposed via JMX, which is a much easier way of accessing it, and because you can also expose your own metrics over JMX, we have not seen many organizations that make use of the DMS servlet.

As JMX is a standards-based protocol, with many tools available for retrieving metrics (such as Hyperic, discussed elsewhere in this chapter), we would recommend using this over DMS if you wish to expose your own monitoring information about your application. Having said that, the DMS servlet can be a useful tool in identifying performance problems, and the fact that it is deployed by default in SOA domain environments makes it worth mentioning here.

3

Performance Testing

In this chapter, we will look at some ways in which we can test the performance of our SOA Suite applications. We will look at the following recipes:

- ▶ Installing Apache JMeter
- ▶ Creating a web service test using JMeter
- ▶ Running JMeter on multiple servers
- ▶ Checking responses in JMeter tests
- ▶ Monitoring SOA Suite while testing
- ▶ Recording user web sessions with JMeter
- ▶ Designing advanced load tests
- ▶ Running performance tests from the Cloud

Introduction

The goal of performance testing is to load the system sufficiently so that we can be confident that we will not face any issues in production. This also goes hand-in-hand with monitoring; testing in a representative environment will give us an indication as to how the system will behave. We can then monitor the production environment for the same behaviors, and will be better placed to describe the patterns we see. Load testing should not be performed as a "success or fail" activity, but should be used to develop a deeper understanding of how user requests use up the resources available to the SOA Suite application. It is important that any monitoring performed on a testing system does not interfere with its performance, but we strongly advocate having comprehensive monitoring in your production environment. Replicating this monitoring in the test environment should be sufficient to give you all the metrics that you need.

When designing a set of tests, it is important to ensure that the tests match the real life pattern of use as much as possible; this includes parameters such as the following:

- ▶ Number of requests per second
- ▶ Size of requests
- ▶ Pattern of requests

While testing, we want to monitor the behavior of the test system, looking at things such as the following:

- ▶ CPU usage
- ▶ Free memory
- ▶ Resource usage (database connections, JMS connections, and file handles)
- ▶ Disk usage

For more information on the metrics refer to *Chapter 2, Monitoring Oracle SOA Suite*.

SOA Suite comes bundled with a set of testing tools; the `soa-infra` web application allows you to fire `http` requests on composites (go to `http://our-soa-server-1-ip:7001/soa-infra`, and give it a whirl), and Enterprise Manager Fusion Middleware Control will allow us to test the flow of a SOA composite, and ensure that all of the components are available and linked together. However, this leaves us with a gap—how can we apply sufficient levels of load to be confident that our composite will stand up to the production load? While Enterprise Manager provides a basic end-to-end testing framework, it is not primarily a testing framework; it can affect how your application behaves and its results are not always reliable. So in this chapter, we'll focus on using the open source tool **Apache JMeter**. We will start with how to create simple performance tests for your SOA application, and then move on to designing a suite of tests that will enable us to simulate a realistic user load.

When performance testing a system, our goal is to measure the performance of the system under test, not the performance testing framework itself, and we want to minimize any overhead that the performance testing and monitoring frameworks have on the system. It is therefore important that we ensure we have sufficient load generating capacity, so that when we say that the limit of the system is 5000 requests per second, we are sure that it is the limit of our SOA Suite application, and not the load testing framework; that is 5000 requests per second. For this reason, we have recipes in this chapter on using multiple boxes to generate load for testing, and on using Cloud servers, such as EC2, to generate the load.

Installing Apache JMeter

To enable our performance testing, we'll install the open source tool Apache JMeter.

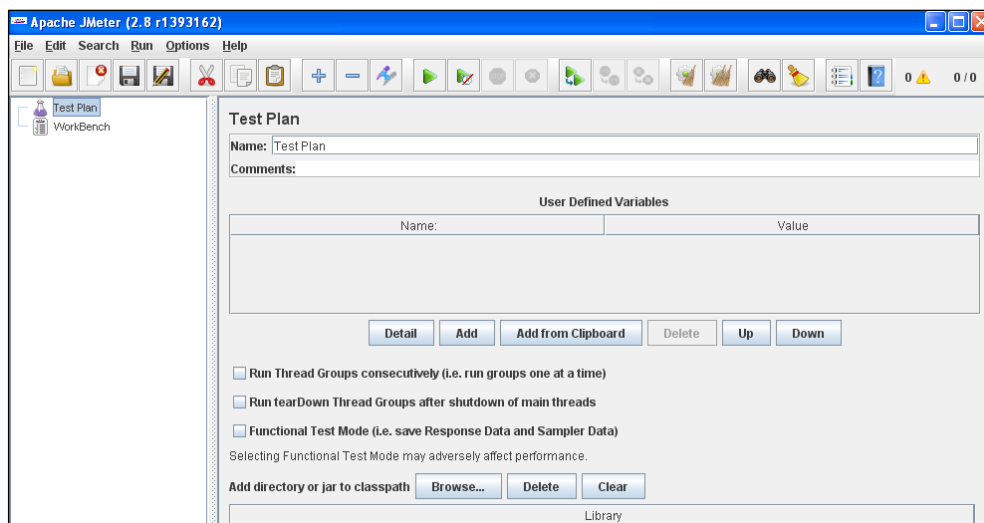
Getting ready

You'll need to install Java JDK Version 6 or higher on a machine to follow this recipe.

How to do it...

Follow these steps to complete the installation of Apache JMeter:

1. Navigate to http://jmeter.apache.org/download_jmeter.cgi, and select the latest binary version for download. If you're on Windows, it will be easier to proceed with the ZIP file.
2. The next step will assume that you've selected the ZIP version, or that you know what you're doing with the `tar.gz` file. Unzip the downloaded file. On Windows, you can right-click and select **Extract All**; on Linux, you can use the `unzip` or `gunzip` commands.
3. Place the extracted contents in a suitable location. The rest of the recipe will use the location `C:\performance-testing\apache-jmeter-2.8`.
4. Within the `bin` directory, run `jmeterw.cmd` on Windows, or `jmeter.sh` on Linux (if using the ZIP bundle, remember to set the file as executable first via the `chmod` command).
5. We now have JMeter installed and are ready to use.



How it works...

JMeter is simply a series of JAR files that are bootstrapped by wrapper scripts. There are no complicated installers or dependencies to maintain; all we need is an installation of Java to get going! This methodology extends into the rest of JMeter, so don't be put off if some of the recipes in this chapter don't seem glamorous; this is a powerful tool.

See also

- ▶ The *Creating a web service test by using JMeter* recipe

Creating a web service test using JMeter

In this recipe, we'll create a JMeter test for a SOA composite.

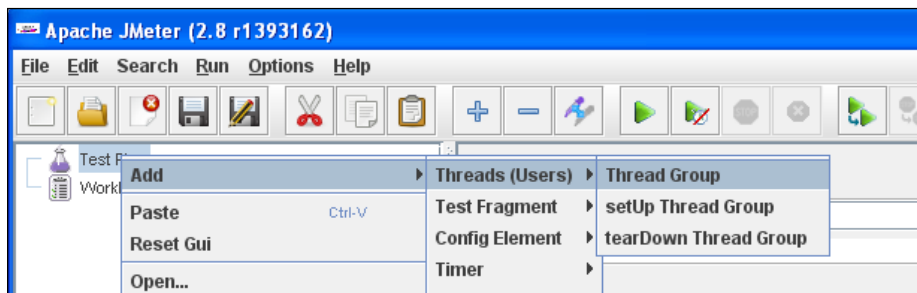
Getting ready

You'll need to complete the *Installing Apache JMeter* recipe. We've provided a composite to use in this recipe that is available on the book's website. You can use your own, but the steps will follow the example recipe.

How to do it...

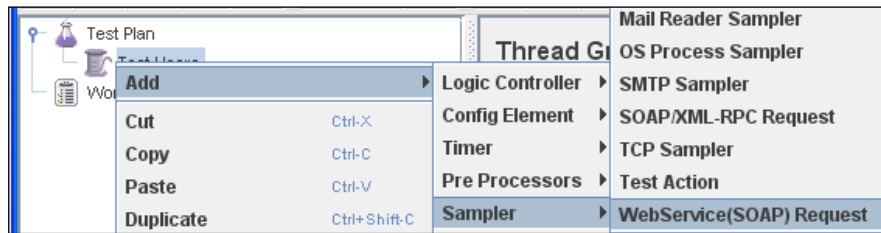
Follow these steps to create a simple web service test:

1. Start JMeter as per step 4 of the *Installing Apache JMeter* recipe.
2. You'll be presented with the JMeter GUI. In the window pane on the left, right-click on the **Test Plan** flask icon and select **Add | Threads | Thread Group**. Name the **Thread Group** as **Test Users** by changing the text in the name box as shown:



3. Set **Thread Properties** so that **Number of Threads** is 10 and **Ramp-Up Period** is 15.

- Right-click on the **Test Users** thread group and select **Add | Sampler | WebService (SOAP) Request**.



- Depending upon how the composite to test is loaded on the system, the WSDL location will be different. If you've deployed the test composite for this chapter to a SOA server on the local machine, it will be located at something like `http://127.0.0.1:8001/soa-infra/services/default/Project1/bpelprocess1_client_ep?WSDL`. Place this in the WSDL URL textbox and select **Load WSDL**. In the **Web Methods** drop-down box, select **Process**, and click on **Configure**. Finally, set **Timeout** to a suitable value such as 2000 milliseconds. The JMeter GUI should appear as follows:

WebService(SOAP) Request

Name: WebService(SOAP) Request

Comments:

WSDL helper

WSDL URL:

Web Methods:

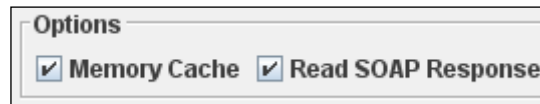
Protocol [http]: Server Name or IP: Port Number: Timeout:

Path: SOAPAction: ☒ Maintain HTTP Session

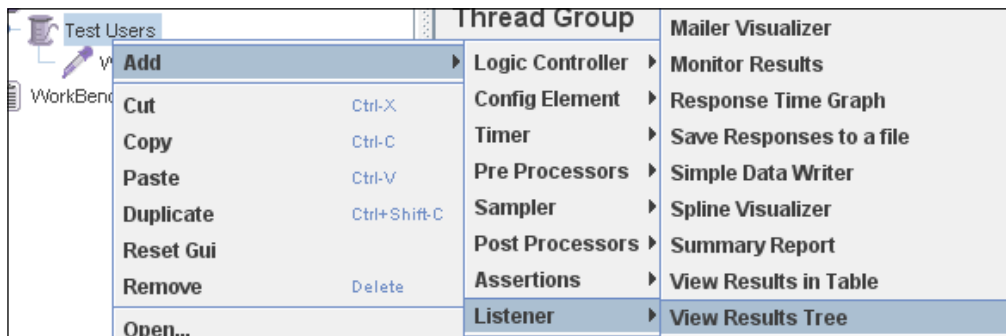
- Place the following text into the **Soap/XML-RPC Data** box:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/
envelope/">
<soap:Body>
  <ns1:process xmlns:ns1="http://xmlns.oracle.com/
SOAApplication1/Project1/BPELProcess1">
    <ns1:input>1</ns1:input>
  </ns1:process>
</soap:Body>
</soap:Envelope>
```

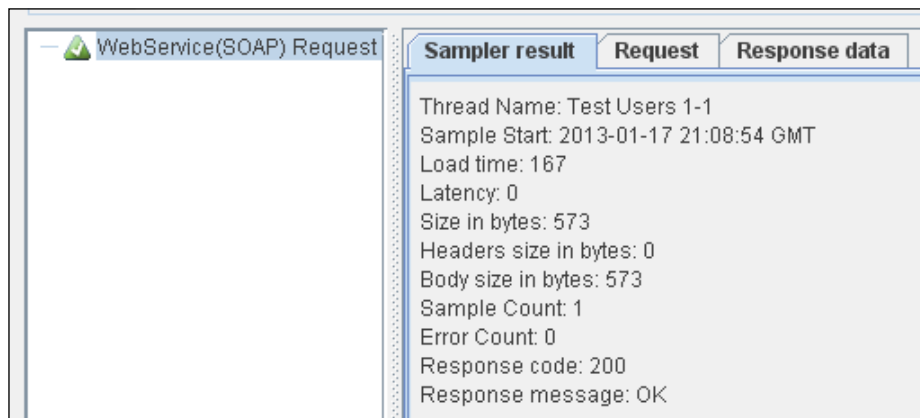
7. Ensure that the **Options** section has the **Read SOAP Response** checkbox selected, as follows:



8. Right-click on the **Test Users** thread group and select **Add | Listener | View Results Tree**.



9. Select **File | Save**, and give the plan a suitable name.
10. Click on the green run arrow or go to **Start | Run** to start the test. Click on the **View Results Tree** item to browse the test result, and data sent and received is in the **Request** and **Response data** tabs. If all goes well, you should see a green icon next to **Web Service Request**, as follows:



How it works...

JMeter uses the concept of samplers, controllers, and listeners to control the sending and capturing of data to and from endpoints. Thread groups then encapsulate the load element of the test, allowing us to control volumes and the rate at which they escalate.

In this recipe, we created a single thread test, which executed only once. We used a SOAP sampler to make it send a SOAP XML test to our SOA Suite composite, by parsing the WSDL exposed by our composite and firing a static SOAP envelope at its exposed address.

We then added a listener to capture the response from the composite, and validated that we received a HTTP 200 response.

There's more...

If you had issues publishing data to your composite, then a good alternative to create a single SOAP message test is to use the Fusion Middleware Control Enterprise Manager application to generate the test SOAP message. To do this, click on the **Test** button within the composite view. This can give you the exposed WSDL address, endpoint invoked, and request and response information that you can place within JMeter, ensuring that all of the values are correct. Enterprise Manager has a useful, single execution test option, however, this is not suitable for load testing. SOA Infra app exposes a test application, however, this is a beta JavaScript stub and is not suitable for large scale, distributed load testing.

There are a lot of other samplers and elements that we can add to a test plan; of particular use is the ability to save test results for later analysis. The JMeter documentation is the best place to learn about what they can do and how they work.

Running JMeter on multiple servers

In this recipe, we'll extend the *Creating a web service test using JMeter* recipe to run our JMeter test from multiple locations.

Getting ready

You will need to install Oracle SOA Suite 11g for this recipe.

You'll need to have completed the *Installing Apache JMeter* recipe for a single machine, and have the SOAP composite test configured and working correctly from the *Creating a web service test using JMeter* recipe.

You'll need network access to the SOA Suite server hosting your composite, and the machine hosting the first JMeter installation.

We'll refer to the installation folder for JMeter as `$JMETER_HOME` for this recipe.

How to do it...

By following these steps, we can configure JMeter to use multiple machines to generate load on your application:

1. Follow the steps in the *Installing Apache JMeter* recipe to install JMeter on the second machine. Ensure that you install the same version of JMeter on the second machine. We'll refer to the original instance of JMeter, which hosts the saved test plan from the *Creating a web service test using JMeter* recipe, as JMeterServer1, and the new installation as JMeterServer2.
2. Determine the IP of JMeterServer2. On Windows, you can do this from the command line by using the `ipconfig` command, as follows:

```
C:\Documents and Settings\c2b2>ipconfig

Windows IP Configuration

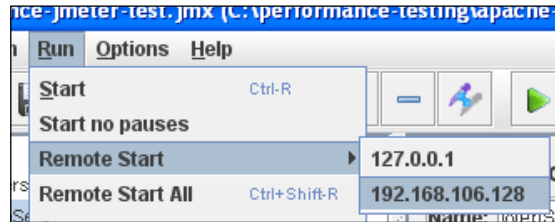
Ethernet adapter Local Area Connection:

    Connection-specific DNS Suffix  . : 
    IP Address. . . . . : 192.168.106.128
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . :
```

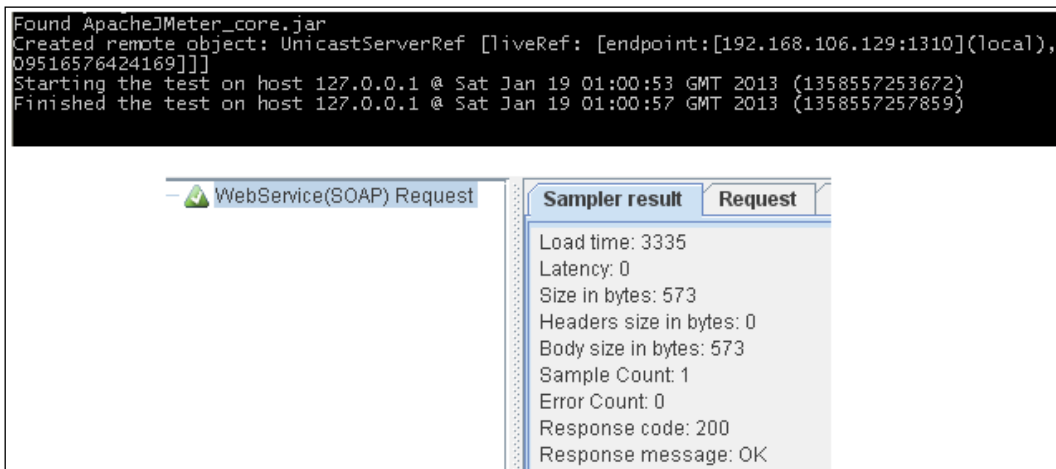
3. On JMeterServer2, open a command-line terminal and change the directory to `$JMeter_HOME`.
4. Run `bin/jmeter-server.bat` on Windows. On Linux, run `bin/jmeter-server.sh`.
5. On JMeterServer1, within the folder `$JMeter_HOME/bin`, open the file `jmeter.properties`, edit the line starting with `remote_hosts=127.0.0.1` and add a comma, then the IP for JMeterServer2.
`remote_hosts=127.0.0.1,192.168.106.128`
6. On JMeterServer1, start JMeter with `$JMeter_HOME/bin/jmeterw.cmd` on Windows, or `jmeter.sh` on Linux.
7. Open the saved plan from the *Creating a web service test using JMeter* recipe. In the `WebService(SOAP)` request step, we need to ensure that the IP for the request points at the SOA suite server and not localhost, otherwise JMeterServer2 will not be able to communicate with it. Alter the **Server Name or IP** box to add your server's IP as shown:

Server Name or IP: 192.168.106.129

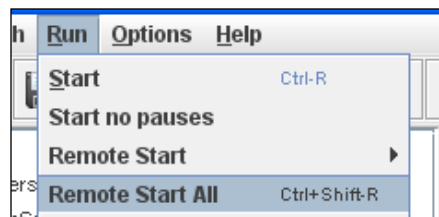
8. In the JMeter GUI, select JMeterServer2's IP from **Run | Remote Start**.



9. Ensure that the test results are valid by examining the command-line output from the JMeter server program on JMeterServer2, and the GUI result listener on JMeterServer1:



10. On JMeterServer1, run `$JMeter_HOME/bin/jmeter-server.bat` on Windows; `$JMeter_HOME/bin/jmeter-server.sh` on Linux.
11. The test can now be run across all available nodes from the GUI by selecting **Run | Remote Start All**.



12. If the test is successful, there will be two valid SOAP test requests—one for each JMeterServer that ran the test.

How it works...

JMeter is capable of running in a "headless" mode, where Java processes act as load test agents, which are then orchestrated by a command process. In this recipe, we ran the JMeter GUI as the command process and two agents; one agent ran on the same host as the JMeter GUI and we ran another agent on a remote host.

The JMeter GUI process sends the test plan to the agents over the RMI. When the tests are complete, they are sent back to the GUI process and processed for visualization. Note that each JMeter agent runs the whole test plan; the total load is not distributed across them.

We have now initialized a multi-node load test environment that we can scale (by adding more hosts running JMeter), to place increasing loads on our SOA composites.

There's more...

There are some limitations on the network relationships between agents and the JMeter command node. The JMeter website has details on the things to consider for increasing the scale of distributed load test. See <http://jmeter.apache.org>.

See also

- ▶ The *Checking responses in JMeter tests* and *Running performance tests from the Cloud* recipes

Checking responses in JMeter tests

In this recipe, we'll use some techniques to validate JMeter test results rather than just checking manually for an HTTP 200 response code.

Getting ready

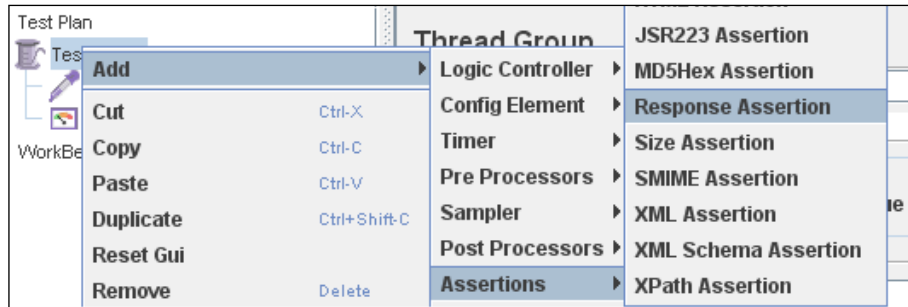
You will need to install Oracle SOA Suite 11g for this recipe.

This recipe assumes that you have a loaded test plan. You can complete the *Creating a web service test using JMeter* and *Running JMeter on multiple servers* recipes, in order to have access to a working test plan against a composite. Alternatively you can use these steps with your own working test plan.

How to do it...

By following these steps, you can use assertions to check that the responses contain the correct data.

1. In the test plan, right-click on the **Test Users** thread group, and select **Add | Assertions | Response Assertion**.



2. In **Assertion**, ensure that the checkboxes **Main sample only**, **Test Response**, and **Contains** are selected.
3. Click on **Add** in **Patterns to Test** in the textbox, and enter `<result>2</result>`. Your test plan should look similar to the following screenshot:

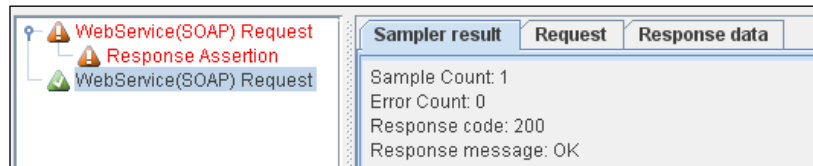
The screenshot shows the 'Response Assertion' configuration dialog box. The 'Name' field is 'Response Assertion'. The 'Comments' field is empty. The 'Apply to' section has 'Main sample only' selected. The 'Response Field to Test' section has 'Text Response' selected. The 'Pattern Matching Rules' section has 'Contains' selected. The 'Patterns to Test' section contains the text '<result>2</result>'.

4. In step 3, we've purposely added an assertion that will fail for the input to our test composite. Run the test by clicking on the green run icon, and navigate to **Result Listener Tree View**.

- Expand the error result tree and view the assertion error. We can see why our response check failed—there was no result of 2 in the response.

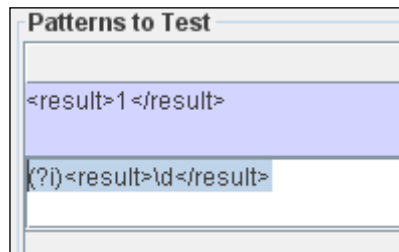


- Change the assertion to check that it matches the expected result. Do this by double-clicking on the pattern to test for in **Response Assertion**, and change it to `<result>1</result>`.
- Run the test again and view the response listener; it will now contain our valid web service test result as well as the one for which the assertion failed.



- Response assertions are Perl 5 style regular expressions; we'll add another pattern to test for a `<result>` element that contains any number. Click on **Add**, and in the **Patterns To Test** box, add the following:

```
(?i)<result>\d</result>
```
- Run the test again, and the result should contain no errors if the pattern is correct.



How it works...

Assertions can be used in JMeter to check the responses in our tests. In the event that an assertion fails, we can determine the cause of the failure in a results listener. We purposefully set an incorrect assertion to begin with, and saw this in action. Assertions are relatively intensive for JMeter to process, so they should be used sparingly, checking for key text that you expect to be in a successful response, rather than trying to perform complex regular expressions or analysis on the returned data. If you use assertions, be sure to monitor the CPU usage of the host running JMeter; if it maxes out, your tests are limited by the performance of the testing host, rather than the SOA Suite host. If this happens, you should add extra JMeter hosts to increase the load, or save the response data and run assertions after the test by using scripting or some other framework.

The assertion type that you can add to JMeter tests can interrogate responses in one of the two ways. Selecting the **Contains** or **Matches** checkboxes enables the use of Perl5-style regular expressions. Choosing **Equals** or **Substring** will match plaintext only, and will do so in a case-sensitive way.

For the assertions added in this recipe, we not only exclusively used the **Contains** checkbox, but also used both plaintext and regular expressions—this is perfectly valid.

If you want to use **Matches** and **Contains** response checks, you can do this by using a second assertion test element.

There's more...

There are a large number of assertion elements that can be added to our JMeter tests; they are documented at http://jmeter.apache.org/usermanual/component_reference.html#assertions. Of particular note are XML schema assertions, which check that a response conforms to a DTD or XML schema. XPath assertions can also be used to extract values from our SOAP responses, and BSF assertions that can use a scripting language, such as groovy, to check the returned data. These complex assertion types should be used sparingly and with care, as they are very CPU-intensive.

See also

- *The Running JMeter on multiple servers recipe*

Monitoring SOA Suite while testing

In this recipe, we'll look at tying together running tests that generate larger loads with starting points for monitoring SOA composite performance.

Getting ready

You will need to install Oracle SOA Suite 11g for this recipe with the Enterprise Manager Fusion Middleware Control console.

You'll need to complete the *Installing Apache JMeter* recipe, deploy the test composite available from the book website to a SOA Suite server, have a working test plan as in the *Creating a web service test using JMeter* recipe, and have access to the Enterprise Manager and WebLogic web consoles.

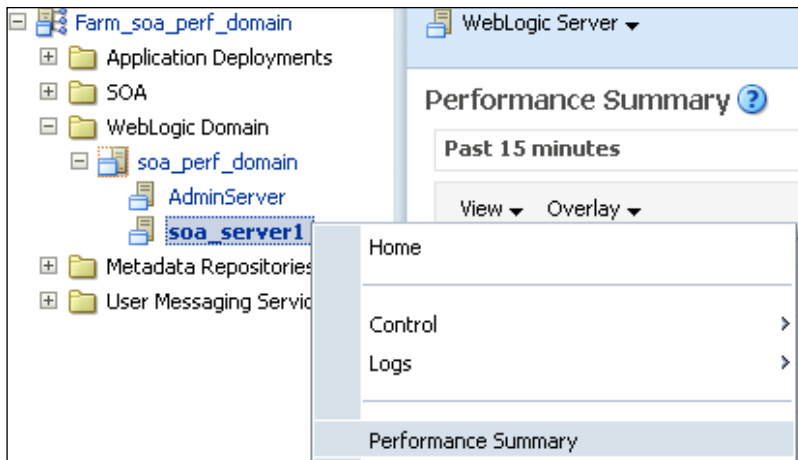
How to do it...

These steps will show you some basic monitoring that can be performed on SOA Suite while tests are running:

1. Load the sample JMeter for this chapter (or create your own). In the JMeter test, select **Test Users Thread Group**. Set **Number of users (Threads)** to a higher value than 1. As a guide, if the host running this test is powerful, then try 100, otherwise start with a value around 50. We only want a small value to start with.
2. Set **Ramp-Up Period (in seconds)** to a value similar to the number of users. Set **Loop Count** to a value of around 50, so that threads will loop and generate a sustained load.

Thread Properties	
Number of Threads (users):	140
Ramp-Up Period (in seconds):	120
Loop Count:	<input type="checkbox"/> Forever 50

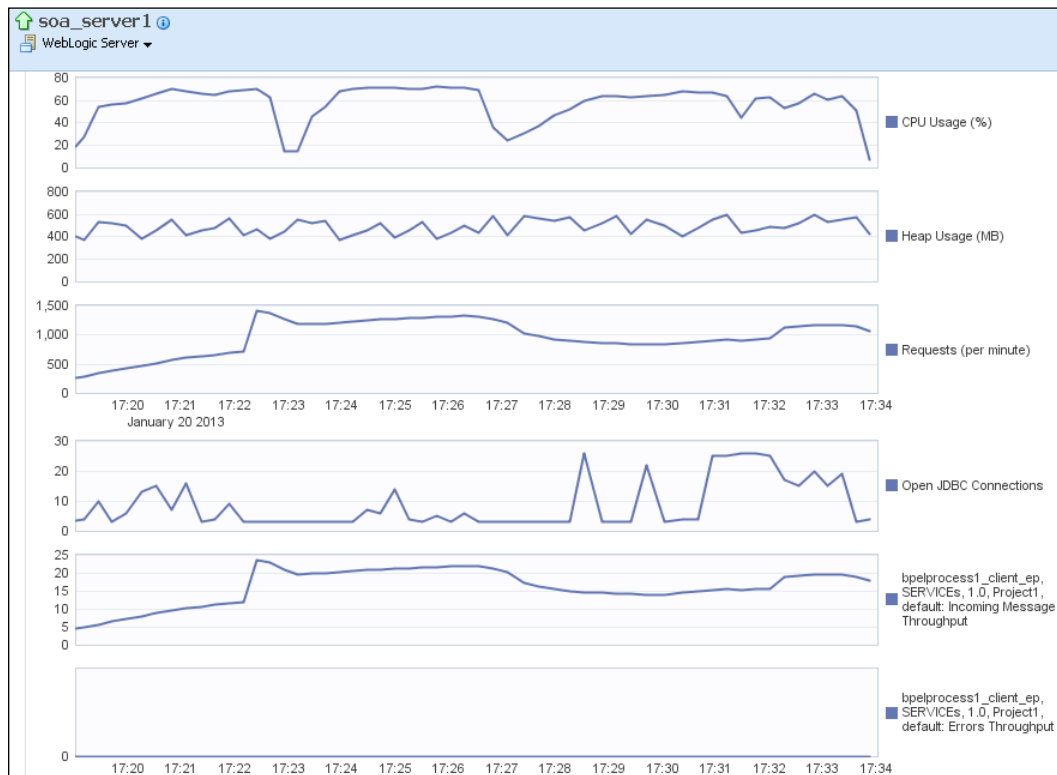
3. Log in to Enterprise Manager on `http://admin-server-host:<admin-port>/em` (the default port is 7001).
4. In the toolbar on the left of the screen, expand the **Weblogic Domain** folder, expand the domain running your composite, and right-click on the server under test. Select **Performance Summary**.



5. Click on the **Show Metric Palette** button, and select metrics to monitor. Good starting candidates are **JVM Heap**, **Memory Usage**, **SOA Composite Errors**, and **Incoming Message Throughput**. Finally, in the **Server Overview** folder, there is a summation of the JDBC connections **Open JDBC Connections**.
6. In another tab, log in to the WebLogic Administration Console `http://admin-server-host:<admin-port>/console` and under **Environment | Servers | SOA Server to monitor | Monitoring | JDBC**, you'll see a breakdown view of the JDBC connections. Click on **Customize this table**, and ensure that **Prep Stmt Cache Current Size** and **Active Connections Current Count** are added to the chosen list.
7. Now, run the test JMeter load test a few times, gradually increasing the number of users (thread) in each test, and monitor the output in Enterprise Manager and the WebLogic admin console.

Performance Testing

- Try to observe any limiting factors within the SOA suite server being monitored. In the following screenshot, we can see that we're approaching a large amount of CPU consumption on this host, but the throughput is steady and the JDBC connection pools are not yet at capacity:



- Use the WebLogic console to monitor the breakdown of connections:

JDBC DataSource Runtime Statistics (Filtered - More Columns Exist)										
Showing 1 to 7 of 7 Previous Next										
Name	Type	Server	State	Active Connections Average Count	Active Connections Current Count	Active Connections High Count	Current Capacity	Leaked Connection Count	Prep Stmt Cache Current Size	Waiting For Connection Current Count
EDNDDataSource	Generic	soa_server1	Running	1	1	2	1	0	1	0
EDNLocalTxDataSource	Generic	soa_server1	Running	2	2	3	2	0	5	0
mds-owsm	Generic	soa_server1	Running	0	0	2	1	0	2	0
mds-soa	Generic	soa_server1	Running	0	0	2	1	0	1	0
OracleSDPMDDataSource	Generic	soa_server1	Running	0	0	1	1	0	1	0
SOADataSource	Generic	soa_server1	Running	4	0	25	26	0	211	0
SOALocalTxDataSource	Generic	soa_server1	Running	3	0	27	28	0	126	0
Showing 1 to 7 of 7 Previous Next										

How it works...

In this recipe, we looked at the main way to control and vary user load in JMeter; by controlling the thread group that represents concurrent user load. There are other ways to form this load by using looping controllers that we did not explore here.

We looked at the results in a new JMeter listener. Whilst JMeter can give us information on the interaction profile with our SOA composites, the view of results it presents ends at the entry point of our SOA infrastructure.

We need to combine the information in *Chapter 2, Monitoring Oracle SOA Suite*, to guide us on where to begin analyzing the SOA Suite components, to identify bottlenecks and candidates for tuning.

In this section, we looked at creating an overview page in the Enterprise Manager Fusion Middleware Control tool available with SOA Suite. This tool gives us a short term component metric view; not every system will have the same performance choke points, so the dashboard components we've used in this recipe should be altered to the needs of the system under test.

We also leveraged the WebLogic tool to give us a further breakdown of the internal metrics for the WebLogic SOA Server's JDBC connection pool. We can use this to further decompose points of contention in our system.

There's more...

Unfortunately, Enterprise Manager Fusion Middleware Control only allows for a short window of monitoring until the browser session times out. It is good practice to record metrics in a permanent store of information, for posterity and tallying of test results with system changes. *Chapter 2, Monitoring Oracle SOA Suite*, contains many recipes to set up monitoring of your Oracle SOA Suite application. If you configure this monitoring in your testing environment, you will have a large amount of monitoring data available to work with.

See also

- ▶ The *Monitoring JVM memory usage* and *Monitoring platform CPU usage* recipes in *Chapter 2, Monitoring Oracle SOA Suite*

Recording user web sessions with JMeter

In this recipe, we'll look at simply configuring JMeter to capture user web activity for replaying in a load test.

Getting ready

You'll need to complete the *Installing Apache JMeter* recipe.

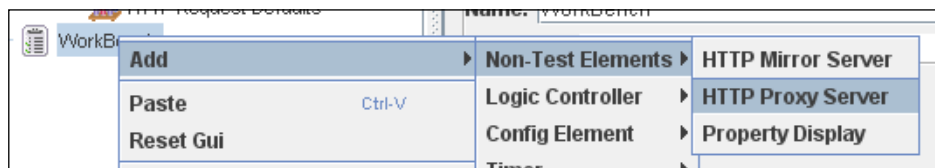
We'll be using a website in this recipe, so you'll need access to a webpage to test.

You'll also need to know how to change your browser's proxy settings.

How to do it...

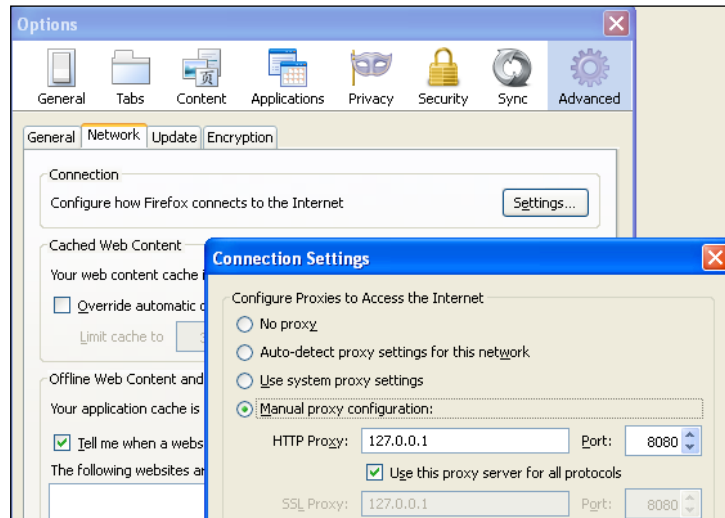
By following these steps, you can record a browser web session to replay as a JMeter test:

1. Start JMeter with `$JMeter_HOME/bin/jmeterw.cmd` on Windows, or `jmeter.sh` on Linux.
2. Right-click on the test plan and select **Add | Threads | Thread Group**.
3. Right-click on the thread group and select **Add | Config Element | HTTP Request Defaults**. In **Server Name** or **IP** section, enter `www.c2b2.co.uk` or the hostname of the server you will be making requests to.
4. Right-click on the **Workbench** item under the test plan, and select **Add | Non-Test-Elements | HTTP Proxy Server**.

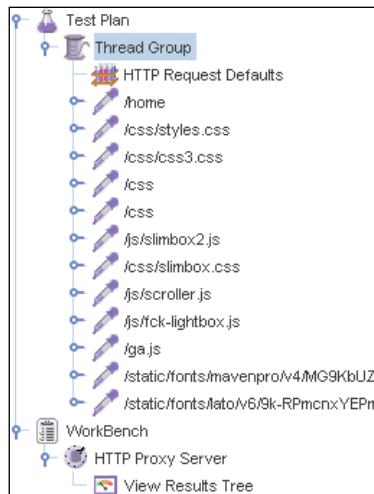


5. Right-click on **HTTP Proxy Server**, select **Add | Listener | View Results Tree**. This is to allow us to observe the recording in flight.
6. Left-click on **HTTP Proxy Server**. In **Global Settings**, select a port that is not in use on that host; 8080 is the default. In URL patterns to exclude add `.*\.ico`, `.*\.gif`, `.*\.jpg`, `.*\.png` to prevent our test capturing the image requests.
7. In **HTTP Proxy Server**, click on **Start** at the bottom of the control pane to begin recording.

8. Next, on any machine with network access to the host running the JMeter proxy, configure a browser or host to utilize the JMeter proxy server. In Firefox, this is achieved by going to **Tools | Options | Advanced**. Select the **Network** tab and click on **Settings**. Select the **Manual Proxy Configuration** checkbox, and enter the host and port for JMeter:



9. Generate some traffic by going to `www.c2b2.co.uk`. All of the requests going through the proxy can be monitored in **View Results Tree** under **HTTP Proxy Server**.
10. The recorder will add the requests that match your include and exclude patterns under your thread group in the test plan. These can then be saved to a test a plan for replay at a later date.



How it works...

JMeter contains a built-in proxy that can be used to capture user HTTP requests, and store them straight into a test plan, which can then be replayed at a later date.

In this test, we configured a proxy and recorded a test plan that can be replayed directly, or modified to add parameterized URLs to substitute values and user per-thread cookies. This gives us a powerful way to capture traffic from tools such as the Enterprise Manager test console, to use in load tests.

Designing advanced load tests

In this recipe, we'll look at the ways in which you can create more varied tests that we can use to more closely replicate realistic user loads.

Getting ready

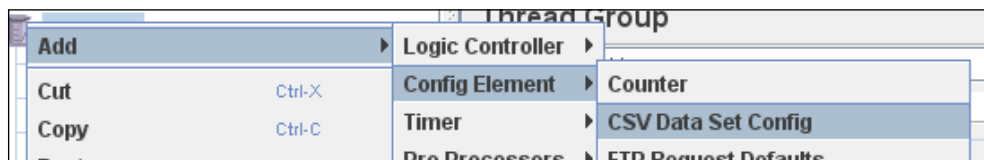
You will need to install Oracle SOA Suite 11g for this recipe.

You'll need to complete the *Installing Apache JMeter* recipe. We'll also be reusing the JMeter test from the *Creating a web service test using JMeter* recipe, which you can reload for this recipe. We'll use a CSV dataset, which you can get from the book's website, or you can create your own.

How to do it...

These steps will demonstrate some of the more advanced features of JMeter testing:

1. Start JMeter with `$JMETER_HOME/bin/jmeterw.cmd` on Windows, or `jmeter.sh` on Linux.
2. Open the saved test plan from the *Creating a web service test by using JMeter* recipe, or load the sample provided with the book source code. Modify the test, if required, to use the correct hostname and port information in the `WebService(SOAP) Request`.
3. Right-click on **Response Assertion** and click on **Remove**.
4. Right-click on the test plan, and select **Add | Config Element | CSV Data Set Config**.



- In the CSV dataset config, set **Filename** to point to the location of the CSV file **load-test-data.csv**. Set **Variable Names** to **RequestValue, ResponseValue**.

Configure the CSV Data Source

Filename: C:\performance-testing\apache-jmeter-2.8\load-test-data.csv

File encoding:

Variable Names (comma-delimited): RequestValue,ResponseValue

Delimiter (use '\t' for tab):

Allow quoted data?: False

Recycle on EOF ?: True

Stop thread on EOF ?: False

Sharing mode: All threads

- In the **Webservice(SOAP) Request** element, change the request value to be loaded by the CSV, by using the parameter `${RequestValue}`. Also, in the **Options** section, disable the **Memory Cache** checkbox as shown:

WebService message

Soap/XML-RPC Data

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
  <ns1:process xmlns:ns1="http://xmlns.oracle.com/SOAApplication1/Project1/BPELProcess1">
    <ns1:input>${RequestValue}</ns1:input>
  </ns1:process>
</soap:Body>
</soap:Envelope>
```

File with SOAP XML Data (overrides above text)

Filename

Use random messages SOAP

Message(s) Folder

Options

☐ Memory Cache ☒ Read SOAP Response ☐ Use HTTP Proxy Server Name or IP:

- In the **Test Users** element, set the number of threads to 10 and **Ramp-Up Period (in seconds)** to 40.

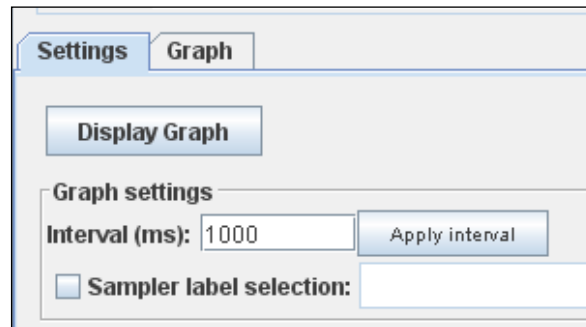
Thread Properties

Number of Threads (users): 10

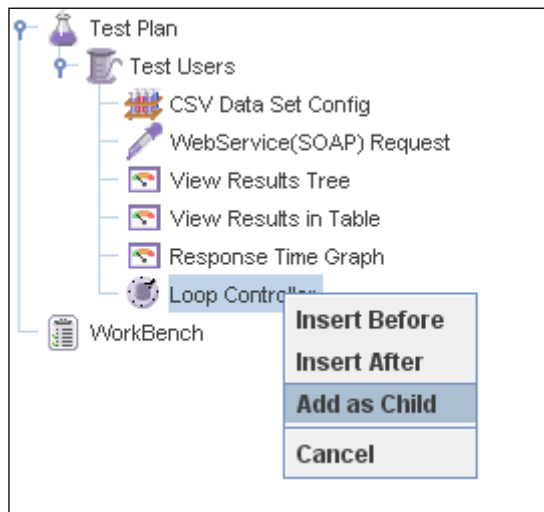
Ramp-Up Period (in seconds): 40

Loop Count: ☐ Forever 1

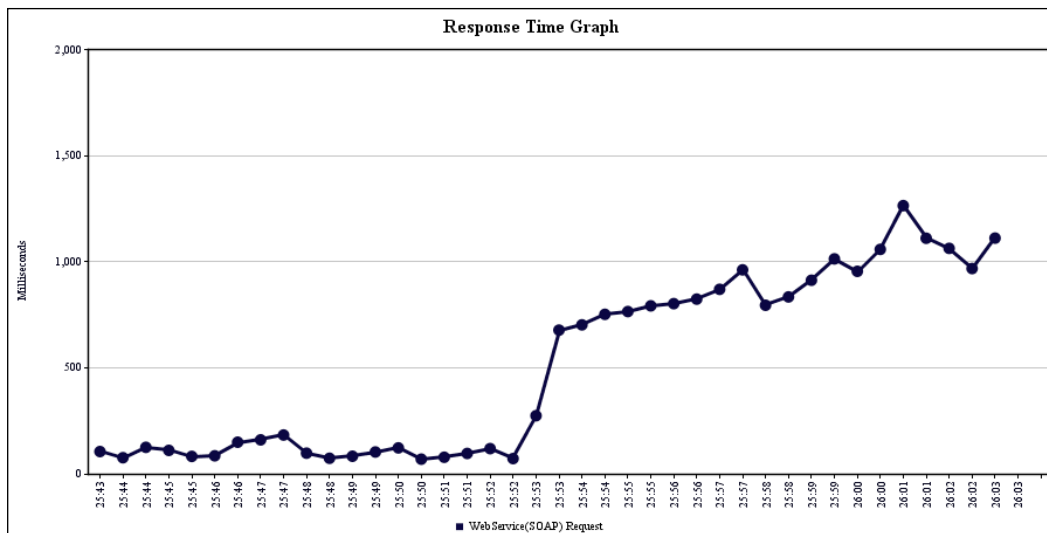
8. Right-click on **Test Users** and select **Add | Listener | Graph Results**. In the graph settings, set the display interval to 1000, and click on **Apply Interval**.



9. Right-click on **Test Users** and select **Add | Logic Controller | Loop Controller**. Set **Loop Count** to 3.
10. Left-click on **WebService(SOAP) Request**, and drag it onto **Loop Controller**. Select **Add as Child**.



11. Right-click on **Loop Controller**, and select **Add | Timer | Constant Timer**. Set the value to 300.
12. Run the test by clicking on the green start icon; when completed, view the test results by clicking on **Response Time Graph**, and selecting the **Graph** tab.



How it works...

In this test, we mutated the base test developed in an earlier recipe to add data substitution on the fly, a looping element, and a simple graphing element. The test composite provided with this book will have responded by providing an increase in the response time after the fifth input change from the dataset, which is viewable in the listeners; in particular, the graph we plotted. This is the sort of information we hope to distil from systems, by providing mutating data.

We created the test data in advance of the test. This prevented JMeter from having to perform expensive randomizations mid-test, which allows it to focus on simply performing the test actions.

We added a looping element to each iteration to ensure that each execution's response time was not a one off. We could increase the distribution of these randomizations and vary the looping mechanisms, to test only certain parts of the application based on conditions specified in the "if" controllers.

This will help us to design a variety of different load generating tests that we can use in conjunction with monitoring tools, to gather information on the performance of our SOA composites.

There's more...

There are a large number of variations available in these sorts of tests. A number of types of randomizations are available in the pause controllers. SOAP messages can be pre-created and loaded from a folder, and operating system samplers can execute commands on systems to collect data to graph at a later date. It's best to experiment with the wide range of options available by using the knowledge gained in this chapter, to determine the best way to replicate more advanced load profiles.

Running performance tests from the Cloud

In this recipe, we'll use **Amazon Elastic Cloud Compute (EC2)** instances to allow us to put large loads on the system under test, and simulate realistic user traffic from multiple locations.

Getting ready

You will need to install Oracle SOA Suite 11g for this recipe.

You'll need to have access to an Amazon web services account to create a Cloud test server; at the time of writing this is free for the resources we need in this recipe.

You'll need to complete the *Installing Apache JMeter*, *Creating a web service test by using JMeter*, and *Running JMeter on multiple servers* recipes to be comfortable with setting up the test servers.

For ease of use, we'll use JMeter in command-line mode. We'll make use of the test plan created in the *Creating a web service test* recipe.

How to do it...

Follow these steps to configure a JMeter server running on Amazon Elastic Compute Cloud:

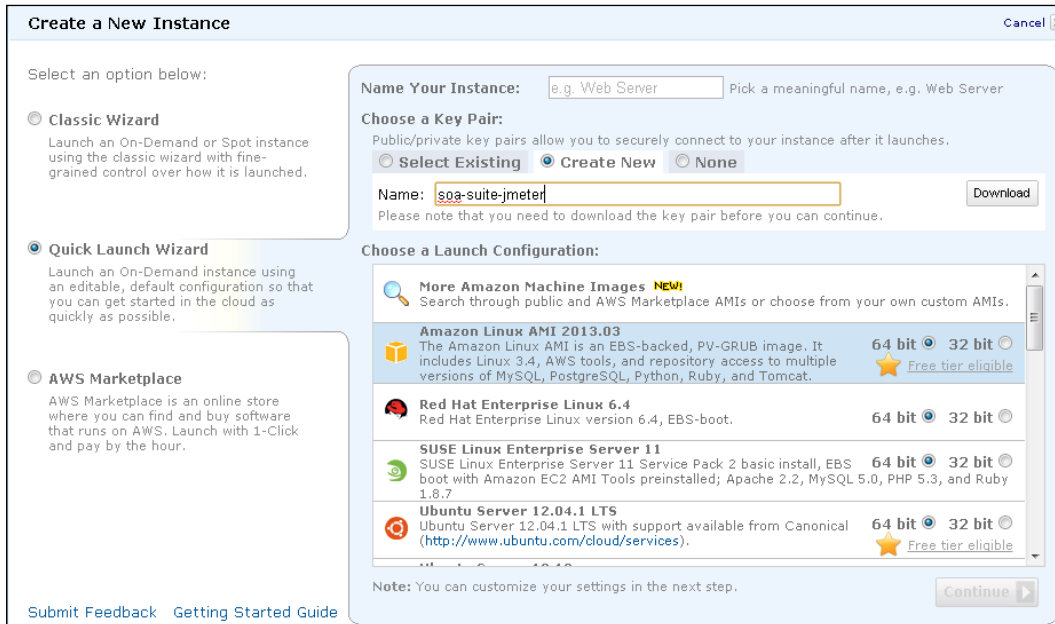
1. Log in to AWS at <https://console.aws.amazon.com/ec2>, and select **EC2** from the main menu if prompted.
2. Click on **Launch Instance**.

To start using Amazon EC2 you will want to launch a virtual server, known as an Amazon EC2 instance.

Launch Instance 

Note: Your instances will launch in the region.

3. In the new instance wizard, select **Quick Launch**. Name the instance JMeterCloudServer1 in the **Name Your Instance** textbox. In the **Choose a Key Pair** section, choose **Create New**, and click on the **Download** button. If you're on Windows, you'll need to use a remote shell, such as Putty to perform some extra steps with the key file you download; these are detailed in the AWS user guide at <http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/putty.html>.



Create a New Instance Cancel

Select an option below:

- ☐ **Classic Wizard**
Launch an On-Demand or Spot instance using the classic wizard with fine-grained control over how it is launched.
- ☒ **Quick Launch Wizard**
Launch an On-Demand instance using an editable, default configuration so that you can get started in the cloud as quickly as possible.
- ☐ **AWS Marketplace**
AWS Marketplace is an online store where you can find and buy software that runs on AWS. Launch with 1-Click and pay by the hour.

[Submit Feedback](#) [Getting Started Guide](#)

Name Your Instance: Pick a meaningful name, e.g. Web Server

Choose a Key Pair:
Public/private key pairs allow you to securely connect to your instance after it launches.







☐ Select Existing ☒ **Create New** ☐ None

Name: Download

Please note that you need to download the key pair before you can continue.

Choose a Launch Configuration:

More Amazon Machine Images NEW
Search through public and AWS Marketplace AMIs or choose from your own custom AMIs.

 Amazon Linux AMI 2013.03 The Amazon Linux AMI is an EBS-backed, PV-GRUB image. It includes Linux 3.4, AWS tools, and repository access to multiple versions of MySQL, PostgreSQL, Python, Ruby, and Tomcat.	64 bit <input checked="" type="radio"/> 32 bit <input type="radio"/>	 Free tier eligible
 Red Hat Enterprise Linux 6.4 Red Hat Enterprise Linux version 6.4, EBS-boot.	64 bit <input checked="" type="radio"/> 32 bit <input type="radio"/>	
 SUSE Linux Enterprise Server 11 SUSE Linux Enterprise Server 11 Service Pack 2 basic install, EBS boot with Amazon EC2 AMI Tools preinstalled; Apache 2.2, MySQL 5.0, PHP 5.3, and Ruby 1.8.7	64 bit <input checked="" type="radio"/> 32 bit <input type="radio"/>	
 Ubuntu Server 12.04.1 LTS Ubuntu Server 12.04.1 LTS with support available from Canonical (http://www.ubuntu.com/cloud/services).	64 bit <input checked="" type="radio"/> 32 bit <input type="radio"/>	 Free tier eligible

Note: You can customize your settings in the next step. Continue

4. Select **Amazon Linux AMI**. Ensure that the instance selected has **Free tier eligible** written next to it. Ensure that the instance is a micro type.

- Click on **Edit Details** and add a security group to allow access on a port between the hosts. You need to add ports for 2159, 1099, 4445, and 22. Details on creating security groups are available in the Amazon AWS documentation.

Create a New Instance Cancel

Amazon Linux AMI 2012.09 (ami-c37474b7)
 Platform: Amazon Linux
 Architecture: x86_64
 The Amazon Linux AMI 2012.09 is an EBS-backed, PV-GRUB image. It includes Linux 3.2, AWS tools, and repository access to multiple versions of MySQL, PostgreSQL, Python, Ruby, and

Click **Save details** in order to save your changes and return to the review screen.

☐ Instance Details
☐ Modify Tags
☒ **Security Settings**

Group name: Description: Create

Create a new rule:

Port range:
 (e.g., 80 or 49152-65535)

Source:
 (e.g., 192.168.2.0/24, sg-47ad482e, or 1234567890/default)

+ Add Rule

TCP Port (Service)	Source	Action
22 (SSH)	0.0.0.0/0	Delete
2159	0.0.0.0/0	Delete
4445	0.0.0.0/0	Delete
1099	0.0.0.0/0	Delete

☐ Advanced Details
☐ Storage Device Configuration

Save details Launch

- Launch the instance by clicking on the final **Launch** button. Right-click on the instance in the EC2 console, and click on **Connect** to get connection instructions for Putty or an SSH terminal.
- Repeat steps 1 to 6 to create another instance, and call it `JMeterCloudServer2`. Ensure that the already created key and security groups are reused.
- The Amazon Linux AMI has OpenJDK Java pre-installed. If you selected another EC2 instance type, install Java.
- Install JMeter as shown in the *Installing Apache JMeter* recipe. Use `wget` from the command line with the link to the JMeter download binary distribution, from the download page at http://jmeter.apache.org/download_jmeter.cgi, and then extract JMeter with the command `tar zxvf apache-jmeter-2.8.tgz`. In the rest of the recipe, we'll assume you've installed JMeter in `/home/ec2-user/apache-jmeter-2.8/`, and refer to this as `$JMeter_HOME`.

10. On JMeterCloudServer1 in \$JMETER_HOME/bin, edit `jmeter.properties`, and set the following parameters:
- ❑ `remote_hosts` to be the hostname of JMeterCloudServer2
 - ❑ Uncomment `client.rmi.localport=2159`, and set this to be 2159 (`client.rmi.localport=2159`)
 - ❑ Uncomment `server.rmi.localport`, and set this to be 2159 (`server.rmi.port=2159`)

11. Create a file called `test-plan.jmx`, and upload the test plan created in the *Creating a web service test recipe*. Edit the test plan to match your public IP and port that can be accessed from the Cloud; in our test plan, this is set in the properties `HTTPSampler.domain`, `HTTPSampler.port`, and `WebServiceSampler.wsdl_url`. We have placed a template test plan with the properties set to `$CHANGE_ME` in this chapter's source download.

12. On JMeterCloudServer1, run the test plan by using the following command:

```
java -jar ApacheJMeter.jar -n -t test-plan.jmx -l my-test-log.txt
```

13. Check the result log for a result, as follows:

```
<sample t="207" lt="0" ts="1358983533714" s="true"
lb="WebService(SOAP) Request" rc="200" rm="OK" tn="Test Users 1-1"
dt="text" by="573">
  <assertionResult>
    <name>Response Assertion</name>
    <failure>false</failure>
    <error>false</error>
  </assertionResult>
</sample>
```

`rc=200` indicates that the HTTP response was successful, and `assertionResult <error>false</error>` tells us that the assertion was successful.

14. On JMeterCloudServer2 in \$JMETER_HOME/bin, edit `jmeter.properties` by setting `server.rmi.localport=2159`.
15. Start JMeter on JMeterCloudServer2, by running `$JMETER_HOME/bin/jmeter-server.sh`. You should see output similar to the following code snippet:

```
[ec2-user@ip-10-226-115-140 bin]$ ./jmeter-server
Using local port: 2159
Created remote object: UnicastServerRef [liveRef:
[endpoint: [10.226.115.140:2159] (local), objID: [-
2c446135:13c69f551bd:-7fff, 9050801633329340775]]]
```

16. On JMeterCloudServer1, run the following JMeter command to send the test plan to the agent on server 2 to execute:

```
java -jar ApacheJMeter.jar -n -t test-plan.jmx -l my-test-log.txt  
-R 10.226.115.140
```

You should see output similar to the following:

```
Created the tree successfully using test-plan.jmx  
Configuring remote engine for 10.226.115.140  
Using remote object: UnicastRef [liveRef:  
[endpoint:[10.226.115.140:2159] (remote),objID:[-  
2c446135:13c69f551bd:-7fff, 9050801633329340775]]]  
Starting remote engines  
Starting the test @ Thu Jan 24 20:30:17 UTC 2013  
Remote engines have been started  
Waiting for possible shutdown message on port 4445  
Tidying up remote @ Thu Jan 24 20:30:19 UTC 2013  
... end of run
```

17. Check the agent command line on JMeterCloudServer2 for an equivalent execution message:

```
Starting the test on host 10.226.115.140 @ Thu Jan 24 21:03:38 UTC  
2013 (1358975018304)  
  
Finished the test on host 10.226.115.140 @ Thu Jan 24 21:03:39 UTC  
2013 (1358975019169)
```

18. Check the log file on JMeterCloudServer1 for the test response for 200 success code and no errors:

```
<sample t="382" lt="0" ts="1358975019724" s="true"  
lb="WebService(SOAP) Request" rc="200" rm="OK" tn="Test Users 1-1"  
dt="text" by="573">  
  <assertionResult>  
    <name>Response Assertion</name>  
    <failure>false</failure>  
    <error>false</error>  
  </assertionResult>  
</sample>
```

How it works...

To enable new users to test drive EC2 for free, Amazon lets you create and start a free server instance on their Cloud infrastructure. In this recipe, we leveraged this capability to allow you to create and upload a test plan to the Cloud (you can learn more about setting this up at <http://aws.amazon.com/free/>).

We downloaded and installed JMeter onto two Cloud servers, tested our composite remotely, and then configured JMeter from the command line to run in a distributed mode in the Cloud. We can now add multiple agents as remote targets to `JMeterCloudServer1` via the `-R` command-line flag; these would run the test loads and send the results back.

There are some limitations to consider, however. There must be sufficient processing power on the master node to cope with the number of agents streaming data to it. Memory and network bandwidth will also need to be monitored if heavy load tests are to be used.

There's more...

We can now stretch our imagination to think of scenarios, such as running multiple, different, concurrent types of load tests across multiple hosts in the Cloud.

To add business value, these distributed tests should be enhanced to output their data to file for aggregation and posterity. Try adding this functionality via listeners!

See also

- ▶ *The Recording user web sessions with JMeter and Monitoring SOA Suite while testing recipes*

4

JVM Memory

This chapter looks at some of the things we can do to tune the memory settings of the **Java Virtual Machine (JVM)**. We will look at how to perform the following actions:

- ▶ Increasing the JVM heap size
- ▶ Setting Xmx and Xms to the same value
- ▶ Setting the size of the Permanent Generation heap
- ▶ Calculating the total memory used by your application
- ▶ Viewing the memory used using JRMCM for JRockit
- ▶ Viewing the memory used using VisualVM for HotSpot
- ▶ Setting the size of the thread stack

Introduction

JVM manages memory for us so that we can focus on writing our applications. It is pretty good at it too, but it cannot fully optimize its behavior while running programs. To help it, we can give it explicit rules to follow as well as hints on how to operate. Unfortunately, the number of options available for these hints and rules is wide ranging and not always fully documented.

In this chapter, we will explore some of the options available to us and look at how we can monitor the effect they have on your SOA Suite installation. This will allow us to analyze its behavior at runtime and monitor the effect of our applied tuning options. We will look at options for the HotSpot and JRockit JVMs (for more details on making a selection between the two, see the *Using the Oracle JRockit JVM* recipe in *Chapter 6, Platform Tuning* end parenthesis).

Memory management is a combination of art and science—more is not always better, and so, selecting your memory settings should always be performed as part of a tuning exercise (which is why we covered identifying problems, monitoring, and testing in our initial chapters). Our goal is to ensure that the application has enough memory to handle as much data as it is provided with, but not so much that garbage collection takes a long time (see *Chapter 5, JVM Garbage Collection Tuning*).

Increasing the JVM heap size

The Java heap contains all of the objects (instances of classes) that the JVM is currently, or was recently working with. It is therefore the memory space that the JVM uses for storing and working with data. Increasing the heap size for the JVM allows more objects to exist in the heap space. Getting the heap sized optimally will reduce the chance that full "stop the world" garbage collections will occur, thus increasing the processing time available (what we call JVM throughput).

Getting ready

For this recipe, you will need to have Oracle SOA Suite 11g installed on a server. This requirement is the same whether you're using the Hotspot or the JRockit JVM. Note that you will need to perform the following steps on each machine that runs SOA Suite instances.

How to do it...

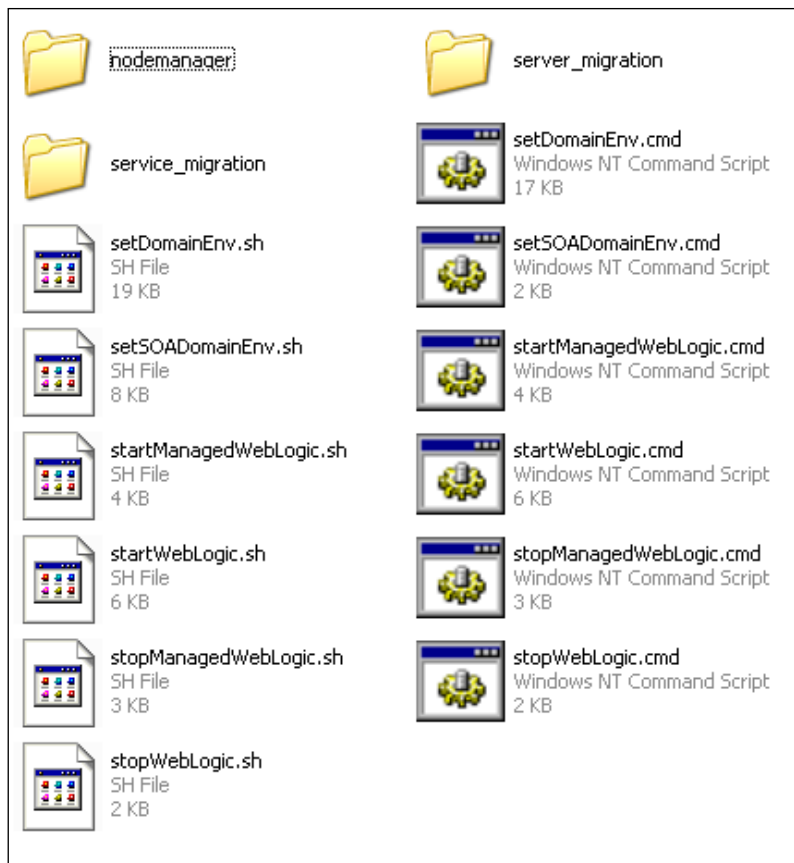
Follow these steps:

1. Navigate to the domain's home directory:

```
cd %MIDDLEWARE_HOME%/user_projects/domains/soa_domain/
```

Replace `soa_domain` with the relevant domain.

2. Within the `bin` folder you will see a number of scripts. Depending on your platform, you will need to edit a different file—`setSOADomainEnv.cmd` on Windows, `setSOADomainEnv.sh` on Linux. Note that it is a good idea to make a backup of these files before editing them.



3. Open the file and locate the following section:

```
set JAVA_OPTIONS=%JAVA_OPTIONS%
set DEFAULT_MEM_ARGS=-Xms 512m -Xmx1024m
set PORT_MEM_ARGS=-Xms 768m -Xmx1536m
```



Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

4. In `DEFAULT_MEM_ARGS` and `PORT_MEM_ARGS`, set the value of `Xmx` to your new value. A good rule of thumb is to stick to powers of 2, with 1.5 MB as a good starting value. This gives us 1,536 as a new JVM maximum heap size.

5. Start the server and view the output to verify your change. Here, I set `Xmx` to 1280 in order to show a change from the default value on a 32-bit machine.

```
starting weblogic with Java version:
java version "1.6.0_29"
Java(TM) SE Runtime Environment (build 1.6.0_29-b11)
Java HotSpot(TM) Client VM (build 20.4-b02, mixed mode)
Starting WLS with line:
e:\Oracle\MIDDLE~1\JDK160~1\bin\java -client -Xms512m -Xmx1280m
```

6. Be careful—if you set it too large, you will see a message, similar to the one given here:

```
Error occurred during initialization of VM
Could not reserve enough space for object heap
Could not create the Java virtual machine.
```

How it works...

Why do we set both `USER_MEM_ARGS` and `PORT_MEM_ARGS`? The scripts configure the environment to start the Java process. When they do this, they check a flag `JAVA_USE_64BIT` to see if your environment is 64-bit enabled. If the flag is set then the value of `DEFAULT_MEM_ARGS` is used for your memory settings, if not then it is presumed that you are in a 32-bit environment and `PORT_MEM_ARGS` is used instead. This allows you to write portable scripts that will work across heterogeneous environments that have different memory settings. As we didn't follow the variable naming convention, we apologize for their confusing names.

Setting the correct flag in the startup scripts increases the maximum size of the Java heap. This best maximum size should be determined by testing, but for a busy application running on a 64-bit JVM, 4 GB is a good starting point. Because the operating system and other processes also need to make use of the system memory available on the box, we would recommend setting `Xmx` to no more than 75 percent of the total system memory.

Note that by default the flag `JAVA_USE_64BIT` is set to `false` in `%WL_HOME%\common\bin\commEnv.cmd`.

If you are not using a 64-bit JVM, but want to use heap sizes larger than 32 bits (or take advantage of some of the other performance enhancements in the 64-bit JVM), you can follow the instructions given here:

1. Navigate to the Oracle website to download a 64-bit JVM. The quickest way here is to search for `JRockit 64 bit download`. Choose the appropriate JVM for your platform from the given list. As I'm on Windows, this will be **Windows x86-64 binary**; for Linux, this would be **Linux x86-64 binary**.

2. Follow the installation instructions for the downloaded file. As we'll be using this JVM only for our WebLogic domain, for convenience we recommend installing it in the Oracle installation directory %MIDDLEWARE_HOME%.
3. Now we can use this JVM in SOA Suite. Navigate to the domain's home directory:

```
cd %MIDDLEWARE_HOME%/user_projects/domains/soa_domain/
```

Replace soa_domain with the relevant domain, as required.

4. Within the bin folder, you will see a number of scripts. Depending on your platform, you will need to edit a different file—setDomainEnv.cmd on Windows; setDomainEnv.sh on Linux. Note that it is a good idea to make a backup of these files before editing them.
5. Open the file and locate the following section:

```
set BEA_JAVA_HOME=C:\Oracle\webLogic10_36\jrockit_160_29_D1.2.0-10
set SUN_JAVA_HOME=C:\Oracle\webLogic10_36\jdk160_29
```

6. Edit BEA_JAVA_HOME to be the location of your 64-bit JVM. As per the installation screen in step 2, I set mine to be:

```
C:\Oracle\WebLogic10_36\jrockit-jdk1.6.0_33-R28.2.4-4.1.0
```

7. Finally, we will edit a 64-bit toggle for this domain. Open the commEnv.cmd file on Windows or commEnv.sh on Linux, located in the MIDDLEWARE_HOME%\wlserver_10.3\common\bin directory.

Locate the following section and change the value to true:

```
@rem JAVA_USE_64BIT, true if JVM uses 64 bit operations
set JAVA_USE_64BIT=false
```

8. Now, start the domain with the startWeblogic.cmd command on Windows or the startWeblogic.sh command on Linux. If the change to 64 bit has been successful, you will see a message similar to the one in the following screenshot for Jrockit (note the x86_64 in the JVM name, and that the maximum heap size is now happily beyond the 4 GB 32-bit limit):

```
starting weblogic with Java version:
java version "1.6.0_33"
Java(TM) SE Runtime Environment (build 1.6.0_33-b03)
Oracle JRockit(R) (build R28.2.4-14-151097-1.6.0_33-20120618-1634-windows-x86_64, compiled mode)
Starting WLS with line:
C:\Oracle\WEBLOG~2\JROCKI~1.0\bin\java -jrockit -Xms512m -Xmx5120m -Dweblogic.Name=AdminServer
```

If you have been performing these steps with the Hotspot JVM, the only change is to set `SUN_JAVA_HOME`, and you will see the following message (note the 64-Bit in the JVM name, and that the maximum heap size is now happily beyond the 4 GB 32-bit limit):

```
starting weblogic with Java version:
java version "1.6.0_35"
Java(TM) SE Runtime Environment (build 1.6.0_35-b10)
Java HotSpot(TM) 64-Bit Server VM (build 20.10-b01, mixed mode)
Starting WLS with line:
C:\Programs\Java\JDK16~2.0_3\bin\java -client -Xms256m -Xmx5120m
```

There's more...

There are more flags to set the fine-grain heap-sizing options in the `setDomainEnv.cmd` or `setDomainEnv.sh` file, but you would need to set these yourself in the final memory-argument string as they are overridden in the SOA Suite startup script by a check against the `USER_MEM_ARGS` flag. It is also possible to set the memory arguments and JVM start parameters by specifying remote start parameters in the WebLogic console. These parameters will only take effect when the WebLogic node manager is used to start and stop servers, so we don't discuss it in detail here (although it is the recommended way of configuring memory parameters for your server in a production environment). You can find instructions for configuring the node manager in the WebLogic Server documentation, at http://docs.oracle.com/cd/E21764_01/apirefs.1111/e13952/taskhelp/machines/BindToProtectedPortsOnServersStartedByNodeManager.html.

The properties that you configure to set memory parameters are the same as the ones you configure when using the node manager; you just enter them via the WebLogic console rather than in the scripts.

Because large heaps are not always better and because modern hardware often comes with tens of GBs of memory, many SOA Suite systems will run multiple WebLogic Server instances on each host, each with a smaller heap (for example, on a box with 16 GB RAM, running 3 WebLogic Server instances, each with 4 GB RAM). For more details on configuring domains with multiple servers, see *Chapter 12, High Performance Configuration*.

See also

- ▶ The *Setting Xmx and Xms to the same value* recipe
- ▶ The *Setting the size of the Permanent Generation heap* recipe
- ▶ The *Calculating the total memory used by your application* recipe
- ▶ The *Configuring a cluster of SOA Suite servers* and *Choosing deployment hardware* recipes in *Chapter 12, High Performance Configuration*

Setting Xmx and Xms to the same value

Setting the `Xmx` and `Xms` parameters configures the maximum (`mx`) and minimum (`ms`) sizes of the heap; setting them to the same value fixes the heap to a pre-determined size.

Getting ready

You will need to have Oracle SOA Suite 11g installed for this recipe. This command is the same whether you're using the Hotspot or the JRockit JVM.

How to do it...

To configure your `Xmx` and `Xms` parameters, perform the following steps:

1. Navigate to the domain's home directory:

```
cd %MIDDLEWARE_HOME%/user_projects/domains/soa_domain/
```

Replace `soa_domain` with the relevant domain, as required.
2. Within the `bin` folder, you will see a number of scripts. Depending on your platform, you will need to edit a different file—`setSOADomainEnv.cmd` on Windows; `setSOADomainEnv.sh` on Linux. Note that it is a good idea to make a backup of these files before editing them.
3. Open the file and locate the following section:

```
set JAVA_OPTIONS=%JAVA_OPTIONS%
set DEFAULT_MEM_ARGS=-Xms512m -Xmx1024m
set PORT_MEM_ARGS=-Xms768m -Xmx1536m
```

4. In `DEFAULT_MEM_ARGS` and `PORT_MEM_ARGS`, set the value of `Xms` and `Xmx` parameters to your new value. A good rule of thumb is to stick to powers of 2, with 1.5 MB as a good starting value. This gives us 1,536 as a new JVM Maximum heap size.
5. Start the server and view the output to verify your change. Here, I set `Xms` and `Xmx` to 1280 in order to show a change from the default value on a 32-bit machine):

```
starting weblogic with Java version:
java version "1.6.0_29"
Java(TM) SE Runtime Environment (build 1.6.0_29-b11)
Java HotSpot(TM) Client VM (build 20.4-b02, mixed mode)
Starting WLS with line:
e:\Oracle\MIDDLE~1\JDK160~1\bin\java -client -Xms1088m -Xmx1088m
```

How it works...

By setting both `Xmx` and `Xms` to the same value, you are preventing the JVM from expanding its heap dynamically. This can be a good thing as it will prevent pauses when the JVM hits its memory ceiling, which, if these were different values, would cause a potentially lengthy memory allocation by the operating system. The swapping of memory pages to disk, or multi-tenant virtualized environments are good examples of instances where the operating system might take some time to achieve this allocation. Furthermore, the process of expanding memory up to the value of `Xmx` will happen incrementally, based on the application's memory needs. Each time an expansion happens, a potentially lengthy full garbage collection will be triggered.

On the downside, hitting the memory ceiling will not always cause a full "stop the world" garbage collection to occur, which introduces pauses during the execution of our code.

We usually find that it's better to give the garbage collector the overhead room to operate rather than wait for memory allocations by the operating system to complete.

There's more...

The maximum memory that you can allocate by using the `Xmx` and `Xms` parameters depends on the system and JVM that you are using. On 32-bit JVMs, the limiting factor is the requirement that all of the JVM's memory be contiguous (one big block). This contiguous block is used for heap, stack, native, permanent, and all other classes of memory used by the JVM. This means that, in practice, on a 32-bit JVM on Windows (which can only allocate around 2 GB of contiguous memory), the maximum is around 1.5 GB for `Xmx`. On other platforms, such as Linux, the limit is a little higher, but still not much more than 2.5 GB.

With a 64-bit JVM, you can address much more memory. So you can specify much larger heap sizes, although doing so is not necessarily a good idea (due to the duration of time garbage collection takes). In practice, heaps much above 8 GB become difficult for the garbage collector to work with. We have seen SOA Suite applications using heaps of up to 64 GB—garbage collection pauses were in tens of seconds, which caused significant problems for the application.

We discuss memory sizing in detail in *Chapter 5, JVM Garbage Collection Tuning*.

See also

- ▶ The *Setting the maximum heap size* recipe
- ▶ The *Setting the size of the Permanent Generation heap* recipe
- ▶ The *Calculating the total memory used by your application* recipe

Setting the size of the Permanent Generation heap

Permanent Generation (PermGen) is a special part of the Java heap that contains the class templates for all of the Java classes on your classpath. The size of the Permanent Generation heap can be controlled via the `-XX:PermSize` and `-XX:MaxPermSize` flags. Note that this is not a feature of the JRockit JVM (which stores its classes in the heap)—as such this recipe does not apply if you have chosen that JVM.

Getting ready

You will need to have Oracle SOA Suite 11g installed for this recipe. This command is the same whether you're using the Hotspot or JRockit JVM.

How to do it...

Perform the following steps to configure the size of the Permanent Generation heap:

1. Navigate to the domain's home directory:

```
cd %MIDDLEWARE_HOME%/user_projects/domains/soa_domain/
```

Replace `soa_domain` with the relevant domain, as required.
2. Within the `/bin` folder, you will see a number of scripts. Depending on your platform you will need to edit a different file—`setSOADomainEnv.cmd` on Windows; `setSOADomainEnv.sh` on Linux. Note that it is a good idea to make a backup of these files before editing them.
3. Open the file and locate the following section. Note that I have manually placed the `@REM` comment; your file will not contain this line:

```
@REM This section will only run on a non JRockit VM
if "%JAVA_VENDOR%" == "Oracle" goto OracleJVM
set DEFAULT_MEM_ARGS=%DEFAULT_MEM_ARGS% -XX:PermSize=128m -XX:MaxPermSize=512m
set PORT_MEM_ARGS=%PORT_MEM_ARGS% -XX:PermSize=256m -XX:MaxPermSize=512m
```

4. In `DEFAULT_MEM_ARGS` and `PORT_MEM_ARGS`, set the value of `-XX:PermSize` and `-XX:MaxPermSize` to your new values. A good rule of thumb is to stick to powers of 2, with 256 MB as the minimum and 1,024 MB as the maximum starting values.

5. Start the server and view the output to verify your change. Here, I set the minimum to 256 MB and maximum to 640 MB in order to show a change from the default values on a 32-bit machine.

```
starting weblogic with Java version:
java version "1.6.0_29"
Java(TM) SE Runtime Environment (build 1.6.0_29-b11)
Java HotSpot(TM) Client VM (build 20.4-b02, mixed mode)
Starting WLS with line:
e:\Oracle\MIDDLE~1\JDK160~1\bin\java -client -Xms512m -Xmx1024m -XX:PermSize=256m -XX:MaxPermSize=640m
```

How it works...

The Permanent Generation heap space (PermGen) is a feature of the Hotspot JVM, designed to prevent the bloat of the Tenured (Old) Java heap space, and thus allows the garbage collection routines to ignore these classes during their main execution runs.

However, there is a problem with this design—the PermGen space forms a part of the total memory required by the Java process. In other words, you must add this to the `XX:MaxPermSize` maximum heap space (and other values; see the *Calculate the total memory used by your application* recipe) to determine the full memory usage of your application. Setting the PermGen space to large values, especially on 32-bit machines, subtracts from the total memory available for your applications objects. This explains why PermGen is, by default, set to a small value.

So why increase it? And what's a `java.lang.OutOfMemoryError: PermGen space` exception? The answers to these questions are to do with class loading in Java, which is a deep subject, beyond the scope of this chapter. It should suffice to say that there are two main points to consider when setting these values:

- ▶ Large applications with a large number of classes will require more PermGen space to store their class templates. SOA Suite is a large application, hence our recommendation to increase the value.
- ▶ Every time you redeploy an application, a new class loader is used. This will load all the classes in that application in isolation. Thus, if you are redeploying a large application, you effectively have two instances of each of your classes in the PermGen memory space until the old application is un-deployed (and sometimes not even then, but that's another topic!). SOA Suite itself is not redeployed regularly, so this is less of an issue for us here.

The recommendation is to set this value high enough for SOA Suite to run smoothly but not so high that you are wasting memory that could be used for your heap objects.

See also

- ▶ The *Calculating the total memory used by your application* recipe

Calculating the total memory used by your application

The Java process consumes more memory than simply the maximum heap size. This recipe will show you how to determine the expected native memory size based on the arguments you supply in the startup arguments, and your knowledge of the application.

Getting ready

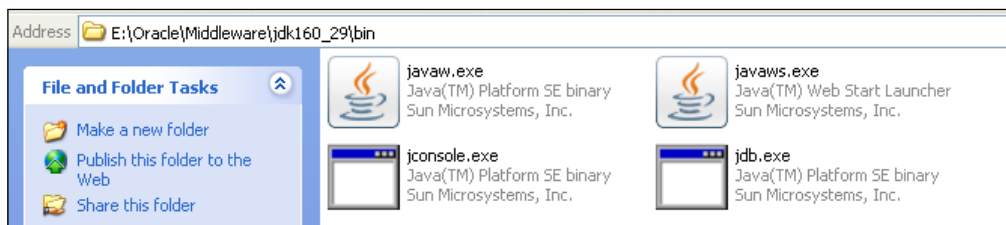
For this recipe, we'll be using the administration server of the SOA Suite server. Note that the arguments we use in this recipe are for demonstration purposes only; your production requirements may be higher! We'll need to have the administration server running so that we can make some measurements using standard Java tooling before we can make any changes. We'll also be using the Hotspot JVM.

How to do it...

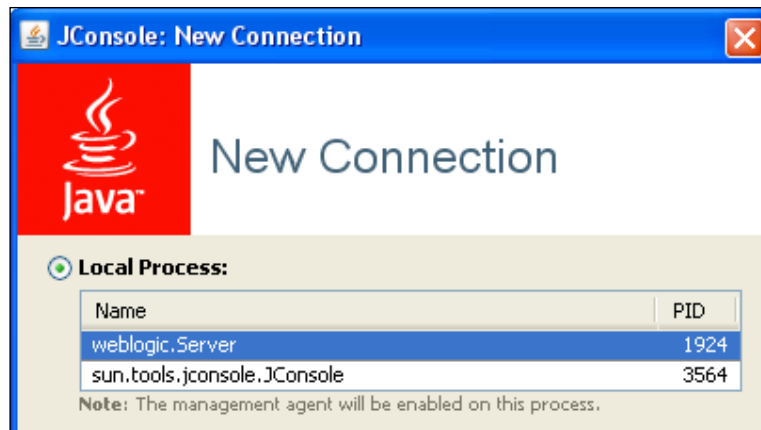
Follow these steps to calculate the memory used by your application:

1. We'll set appropriate values for the SOA Suite administration server by modifying the stock startup scripts. Navigate to the domain's home directory:
`%MIDDLEWARE_HOME%/user_projects/domains/soa_domain/`
 Replace `soa_domain` with the relevant domain, as required.
2. Start the administration server by running the `startWeblogic.cmd` script.
3. Within the `bin` folder, open the `setSoaDomainEnv.cmd` file. We'll edit the file shortly, but first we'll study the values we wish to set.
4. We'll set the maximum and minimum Java heap sizes to the same value. This sets the heap memory size to a predefined static value so it cannot grow or shrink.
`-Xms1024m -Xmx1024m`
5. Add to the 1,024 MB heap a predefined Permanent Generation memory size, which we'll set at 128 MB. Our total is now 1,152 MB.
`-XX:PermSize=128m -XX:MaxPermSize=128m`
6. Set the JIT code cache size to 16 MB using. This gives us a total of 1,168 MB.
`-XX:ReservedCodeCacheSize=16m`

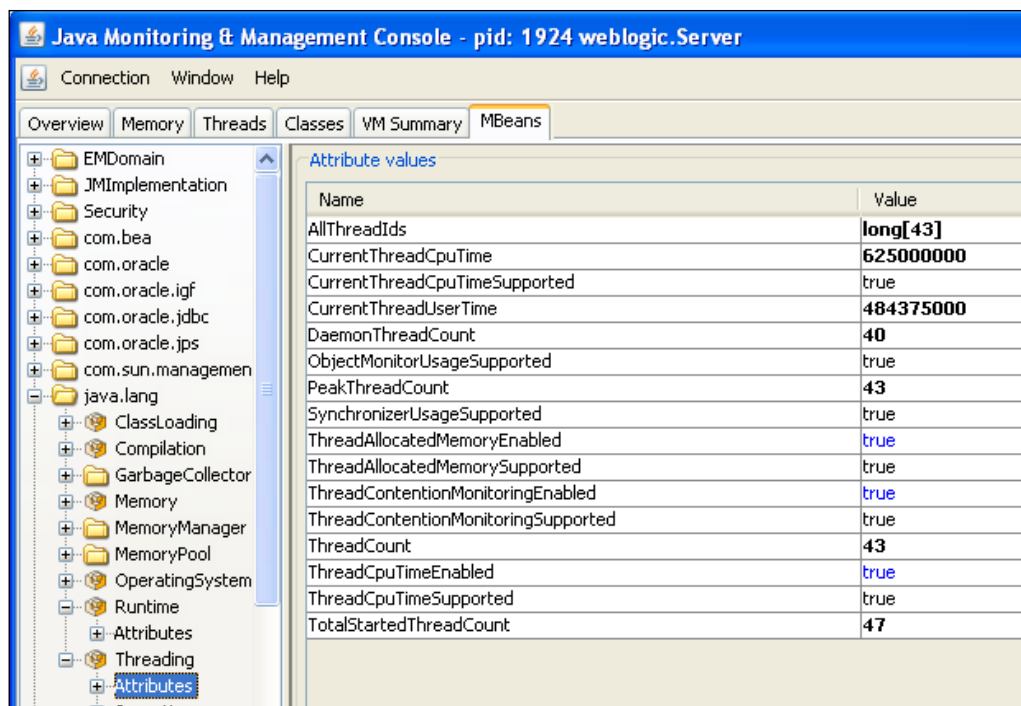
7. Set the thread stack size to a known value. We'll use 128 KB as a starting value. SOA Suite makes heavy use of threads to service your business logic requests; as such we can't just add this value to our running memory count
-Xss128k
8. In order to calculate the total amount of memory used by the threads in the JVM, we need to determine the number of threads in the WebLogic administration server process. We can do this using the jconsole application, which we'll start on the same machine as the administration server (so that we do not have to set up any remote access policies). Within the %MIDDLEWARE_HOME%/your_hotspot_jdk_version/bin directory, open `jconsole.exe`



When jconsole is running, attach it to the administration server by selecting the process from the list and clicking on **Connect**



Within jconsole, select the **mbean** tab, and then open the **java.lang** folder and the **Threading** package. Select **Attributes**; you'll see a **ThreadCount** attribute



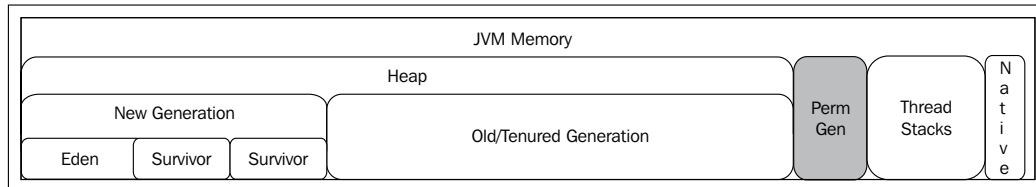
We multiply this by our new Thread Stack Size value (128 KB) to give us a value of 5.504 MB. This gives us a running total of 1,173 MB.

- Finally, make the changes to the file and run the program. Here are the complete changes we make in the `setSoaSuiteDomainEnv.cmd` file:

```
set JAVA_OPTIONS=%JAVA_OPTIONS%
set DEFAULT_MEM_ARGS=-Xms1024m -Xmx1024m -Xss128k
set PORT_MEM_ARGS=-Xms1024m -Xmx1024m -Xss128k
@REM This section will only run on a non JRockit VM
if "%JAVA_VENDOR%" == "Oracle" goto OracleJVM
set DEFAULT_MEM_ARGS=%DEFAULT_MEM_ARGS% -XX:PermSize=128m
-XX:MaxPermSize=128m -XX:ReservedCodeCacheSize=16m
set PORT_MEM_ARGS=%PORT_MEM_ARGS% -XX:PermSize=128m
-XX:MaxPermSize=128m -XX:ReservedCodeCacheSize=16m
```

How it works...

Our Java processes use more memory than the object memory heap space alone. Internally, the memory layout is as shown in the following diagram:



Unfortunately, it can be hard to determine exactly how an operating system natively handles the memory of our Java process. However, there are options that we can use to control the JVM's memory allocation.

The total native memory size of the Java process is calculated by adding together the following:

- ▶ Maximum Object Heap Size (`Xmx`): The area of memory used for holding our Java objects.
- ▶ The PermGen Memory Size (`XX:MaxPermSize`): The area of memory to hold Java class templates.
- ▶ The Thread Stack Size (multiplied by the number of runtime threads) (`XX:ThreadStackSize` or `Xss`): This configures the maximum thread call stack depth.
- ▶ The JIT Code Cache Size (`XX:ReservedCodeCacheSize`): This is the area of memory reserved for optimizations performed by the JVM in response to your code paths.
- ▶ Native memory: The native memory from the JVM itself, plus any native libraries that are being used (look up JNI on the Internet).

We can now make a much closer approximation of the total memory size of our Java process. This will allow us to size our JVMs according to the available native memory on a physical or virtual host.

You can see from the recipe that our final native memory calculation is nearly 150 MB higher than the maximum heap space, which is not a small change if you're trying to co-locate JVMs onto a host.

You can monitor native memory using tools such as Process Explorer on Windows, the `top` command on Linux, or the `ps` and `prstat` commands on Solaris.

Viewing the memory used using JRMCM for JRockit

JRockit Mission Control (JRMCM) is a visualization and management tool for the JRockit JVM. This recipe will show how to connect to local and remote JRockit JVMs to view this information.

Getting ready

For this recipe, you will need to have the JRockit JVM installed on a local machine. We'll use the JRMCM application bundled with the JRockit release.




We'll attach JRockit to the WebLogic administration server to visualize its memory usage at runtime. To do this, you can use the recipe in the previous chapter to change the start-up script to use JRockit.

We will extend the recipe by connecting to a remote JRockit VM to show the steps required to view remote JVM data.

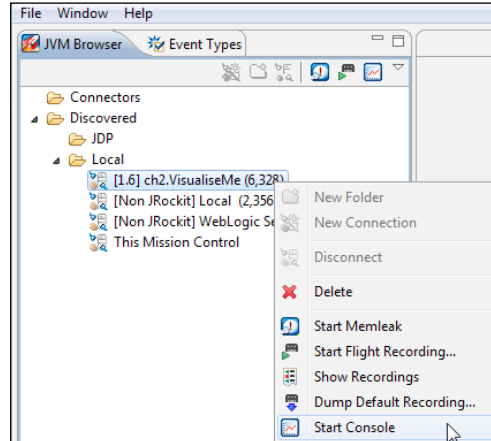
How to do it...

Perform the following steps to use JRockit to monitor your JVM memory:

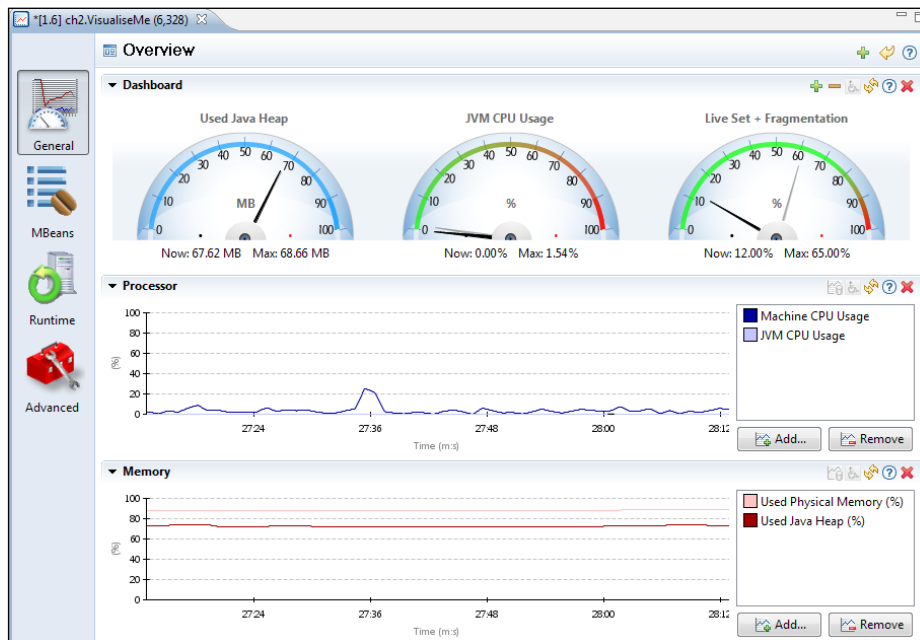
1. Navigate to the domain's home directory:
`%MIDDLEWARE_HOME%/user_projects/domains/soa_domain/`
 Replace `soa_domain` with the relevant domain, as required.
2. Start the administration server by running the `startWeblogic.cmd` script.
3. Navigate to the JRockit JVM directory within the middleware home directory:
`%MIDDLEWARE_HOME%/your_jrockit_jdk_version/`
 Replace `your_jrockit_jdk_version` with the relevant version number, as required.
4. Within the `bin` folder, you will see a number of programs and some native libraries. Open `jrmc.exe` on Windows; on Linux, this would simply be the `jrmc` executable file.

 <code>jrcmd.exe</code>	05/10/2011 18:12	Application	197 KB
 <code>jrmc.exe</code>	05/10/2011 18:12	Application	221 KB
 <code>irunscript.exe</code>	05/10/2011 18:15	Application	197 KB

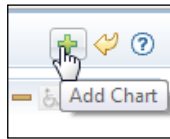
- When booted for the first time, you may be presented with the welcome **Start** page. Click on **x** in the tab to close this and we'll see the local Java processes to which we have permission to attach JRMCI; it will even note the different JVM types. We can see that here we have a WebLogic instance and the JRMCI instance that we can load into JRMCI, as well as our custom piece of code (which you can download from the book site) that simply loops around, allocating and releasing memory.



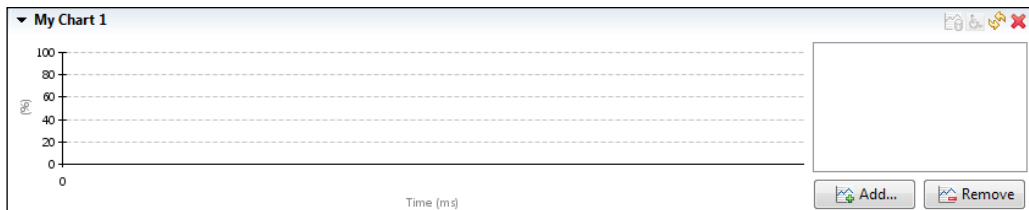
- Load the WebLogic process (by right clicking and selecting **Start Console**). You'll see the default dashboard with the default CPU dials and memory graphs:



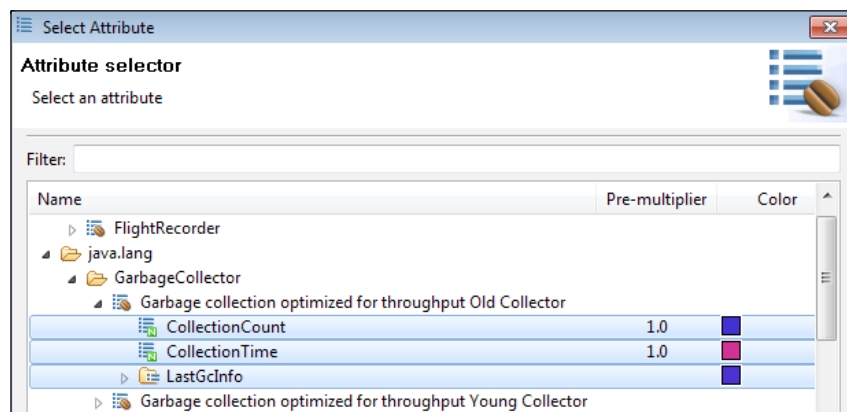
7. More information is available for our dashboards. We can add a chart to the display by clicking on the **Add Chart** button:



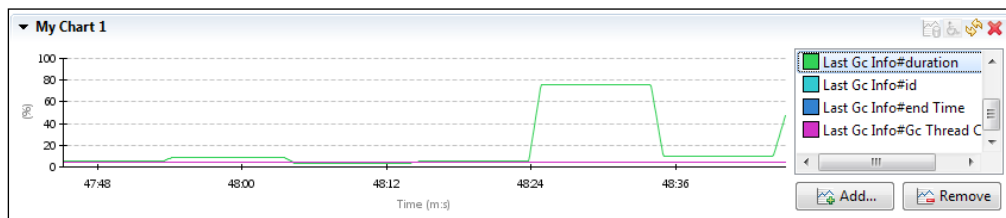
This will add a blank dashboard as shown in the following screenshot:



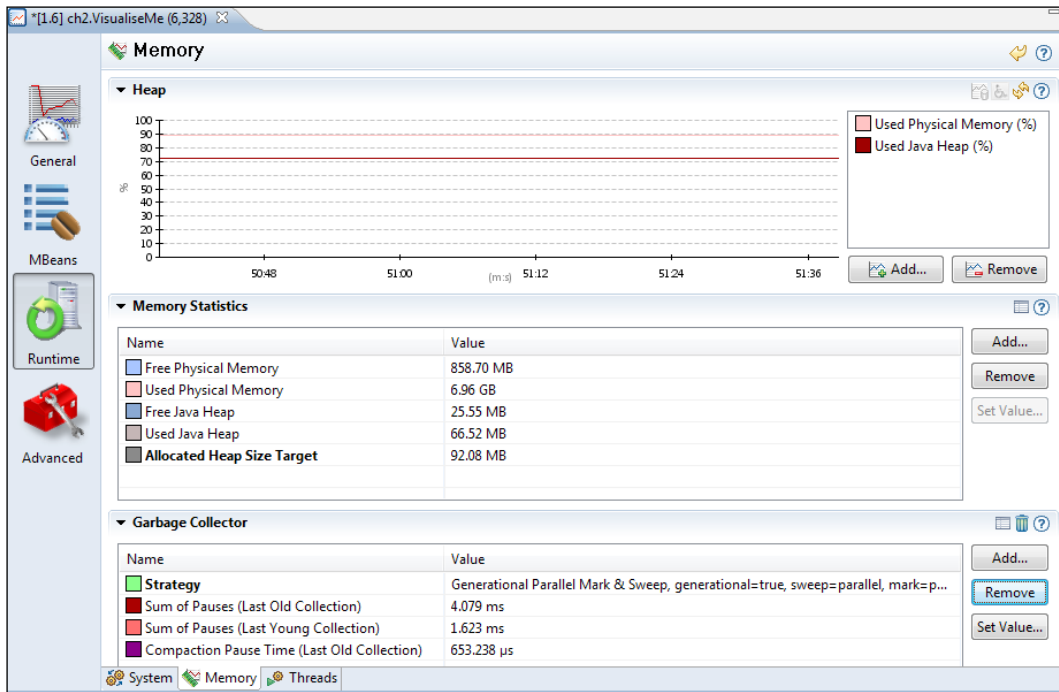
8. We can now add metrics of interest that are available in the JVM. Click on the **Add** button and use **Attribute selector** to determine appropriate metrics. Here, we select metrics on garbage collections.



9. JRMC will then collect and graph the data for us.



10. The runtime view and **Memory** tab show us details on the memory consumption of the program as follows:



How it works...

JRockit contains a bundled application called **Mission Control**, an application based on the Eclipse platform. This allows collation of instrumentation collected by JRockit at runtime by default. The upshot of this is that the only overhead on the JRockit JVM when using Mission Control is in the transmission of these metrics to the monitoring platform.

JRockit leverages the Eclipse platform to allow the user to dynamically alter the view palettes to display the information of interest to your operation teams.

There's more...

JRMC has lots of powerful tooling, such as flight recording to allow capture and replay of JRockit behavior, and a built-in memory leak detector. See the documentation for more details, at <http://www.oracle.com/technetwork/middleware/jrockit/documentation/index.html>.

While JRMC is very useful for diagnosing and monitoring problems in operational systems, its use as a day-to-day monitoring tool should be discouraged because its overhead is higher than other available tools, and it does not automatically store historical data.

If you are going to expose remote access to JRockit in production, we recommend securing remote access as described in the JRockit documentation, by at least securing the data with passwords and enforcing encryption over external links.

Connecting to a remote JRockit JVM

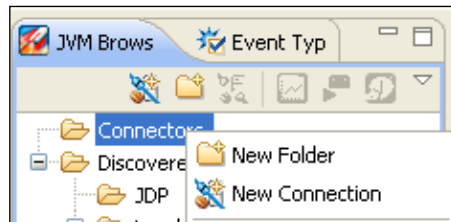
By default, JRockit does not allow JRMC connections from remote hosts. This can be changed to allow a remote instance of JRMC to monitor metrics. We will show the required change to enable this here using the code used in *Chapter 2, Monitoring Oracle SOA Suite* as an example.

Follow these steps:

1. The following parameter needs to be added as a start-up parameter to your WebLogic Server, or a similar one, with ports for your environment:
`-Xmanagement:ssl=false,authenticate=false,port=7099`
2. Run the start script as normal. If successful, you will see a line like this in the console output:

```
[[INFO ][mgmnt ] Remote JMX connector started at address radagast:7099
[[INFO ][mgmnt ] Local JMX connector started
```

3. We can now connect via JRMC by right-clicking on **Connectors** and selecting **New Connection**, as shown in the following screenshot. Enter the details of the host and the port we have exposed of 7099. Right-click and select **show console**, as demonstrated in the local connections section:



Viewing the memory used using VisualVM for HotSpot

The VisualVM application lets you view real-time JVM data such as memory and thread usage and garbage collection cycles. This recipe will show how to connect to local and remote JVMs in order to view this information.

Getting ready

For this recipe, you will need to have the Hotspot JVM installed on your local machine. We'll use the VisualVM application built in to the JDK 6 release so there is no need to download the available, standalone VisualVM distribution.

We'll attach VisualVM to the WebLogic Administration server to visualize its memory usage at runtime. To do this, you need to ensure that the Administration server is configured to start with the Hotspot JVM.

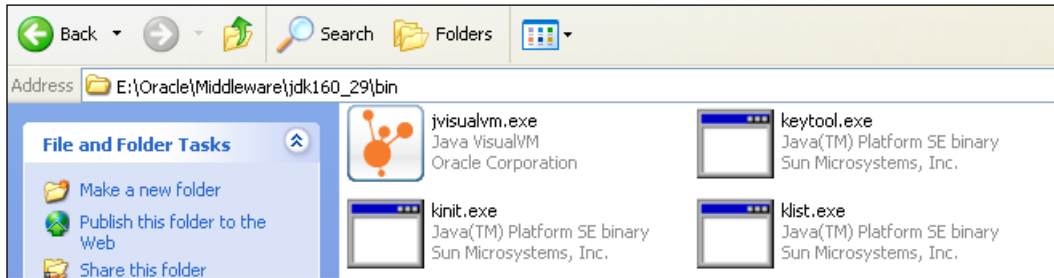
We will extend the recipe by connecting to a remote Hotspot VM to show the steps required to view remote JVM data.

How to do it...

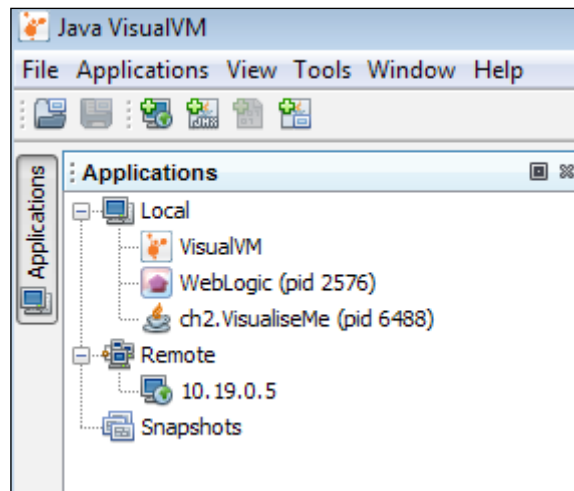
Perform the following steps to view the memory used by Hotspot, using VisualVM:

1. Navigate to the domain's home directory:
`%MIDDLEWARE_HOME%/user_projects/domains/soa_domain/`
Replace `soa_domain` with the relevant domain, as required.
2. Start the administration server by running the `startWeblogic.cmd` script.
3. Navigate to the Hotspot JVM directory within the middleware home directory:
`%MIDDLEWARE_HOME%/your_hotspot_jdk_version/`
Replace `your_hotspot_jdk_version` with the relevant version number, as required.

4. Navigate to the `bin` directory and open `jvisualvm.exe`. The following screenshot shows this for a Windows machine:

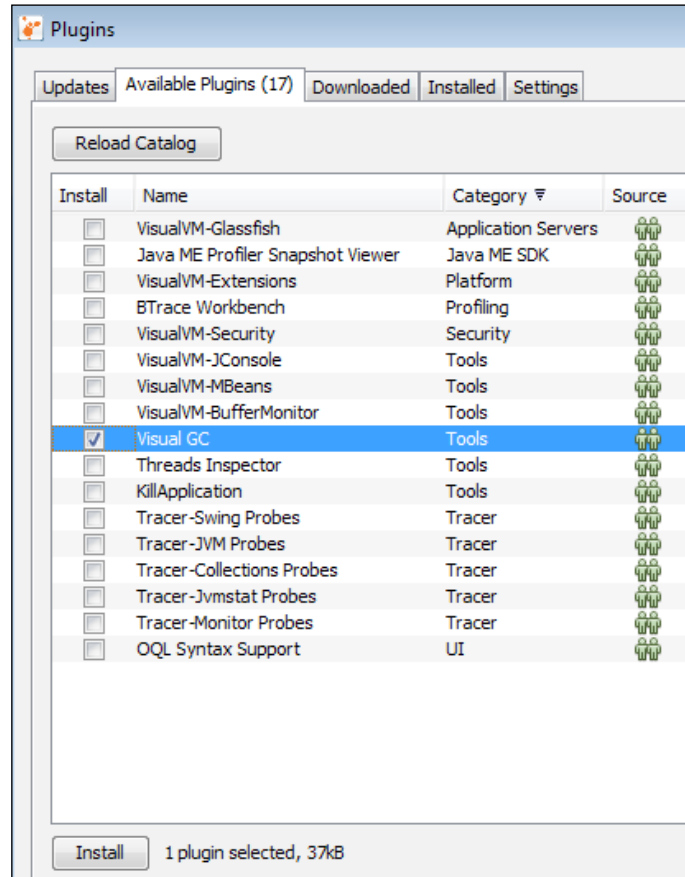


5. When booted for the first time, we'll see the local Java processes to which we have permission to attach VisualVM. If you don't see your process then ensure that you are running VisualVM as a user with privileges to access the WebLogic process. We can see in the following screenshot a WebLogic instance and the VisualVM instance to which we can connect VisualVM:

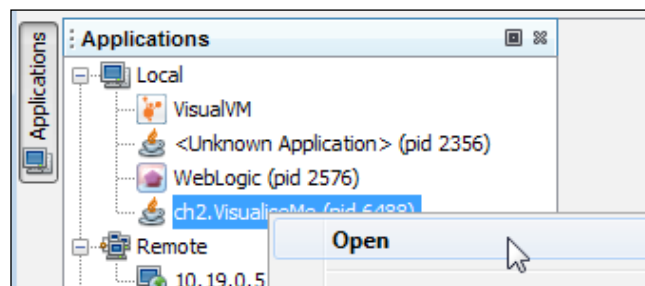


6. If you don't see this view, click on the **Applications** button on the left gutter to display it.

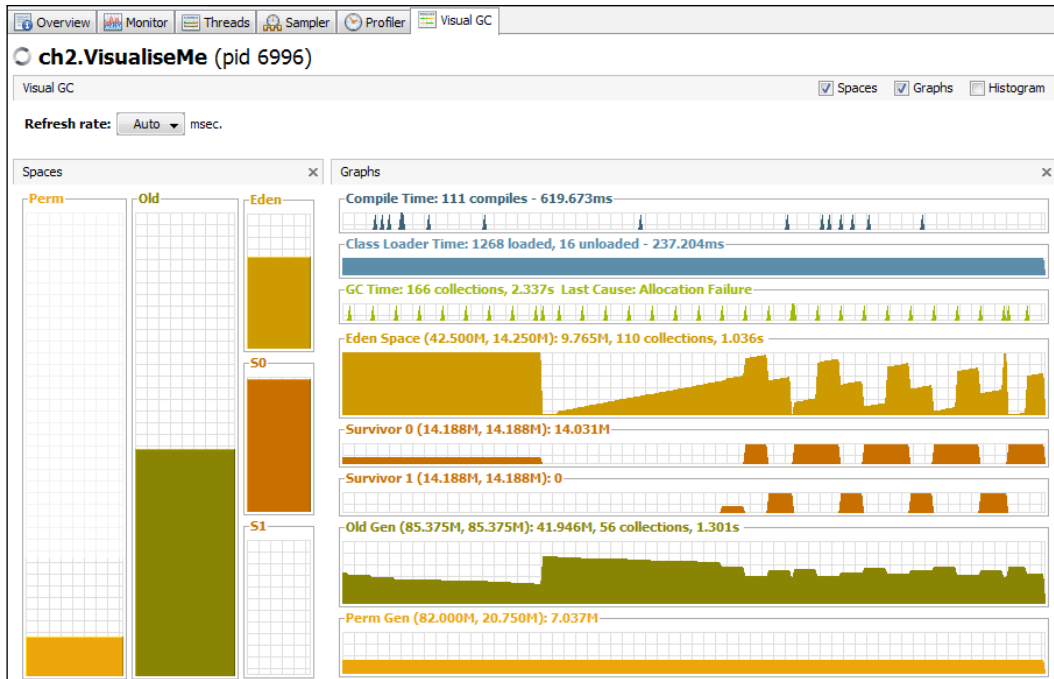
7. We'll install a plugin to visualize the GC cycles and memory usage. Go to **Tools | Plugins**. In **Available Plugins**, select **Visual GC** as shown in the following screenshot, and then click on **Install**:



8. After accepting the license agreement, restart VisualVM to enable the plugin.
9. Double-click on the test program or right-click on it and select **Open**.



10. Now you are able to move between tabs to view visual data on the JVM. Let's look at the tab for our new plugin, the **Visual GC** tab:



11. On the left are presented the current memory size snapshots; on the right, GC's timings are graphed on a timeline showing JVM area memory sizes.

How it works...

VisualVM is a tool bundled with the JDK download package that lets you view detailed information on the internal mechanics of JVM. You can run heap dumps, thread stack listings or, as we did in this recipe, load in one of the many plugins and extrapolate the runtime data to interpret how your program is running on the existing JVM. This is a powerful tool for tuning your application as you can see how the memory is being managed in response to your application code and JVM tuning flags. For instance, the VisualGC plugin that we loaded graphed JVM young and tenured memory sizes going back in time. These were plotted alongside JVM **Just In Time (JIT)** compilation and GC timings; this is a good indicator of how frequently your garbage collector has to run.

JVM Statistics Monitoring Tool Daemon (jstatd) is a powerful mechanism for exposing Java processes. But bear in mind that there is no native authentication in this application. Firewall-access policies are recommended to supplement securing your machine.

There's more...

Instead of using jstatd, we could have used **Java management extensions (JMX)** to remotely access individual processes, which is arguably more secure. It depends on your security needs and environment. JMX usages with VisualVM details are available at http://visualvm.java.net/jmx_connections.html.

Connecting to a remote Hotspot JVM

On the host running your Java process, you need to enable jstatd. This provides a remote interface to access the JVM over an RMI server. The server requires a security policy file to run, which can aid in securing this process. This is described in this section.

Note that jstatd must be running with the same user credentials as the Java process that you wish to monitor.

1. Within `%MIDDLEWARE_HOME%`, browse to the Hotspot installation location:

`%MIDDLEWARE_HOME%/your_hotspot_jdk_version/`

Replace `your_hotspot_jdk_version` with the relevant version number, as required.

2. Within the `bin` folder, create a file named `jstatd.policy`. Add the following content to the file:

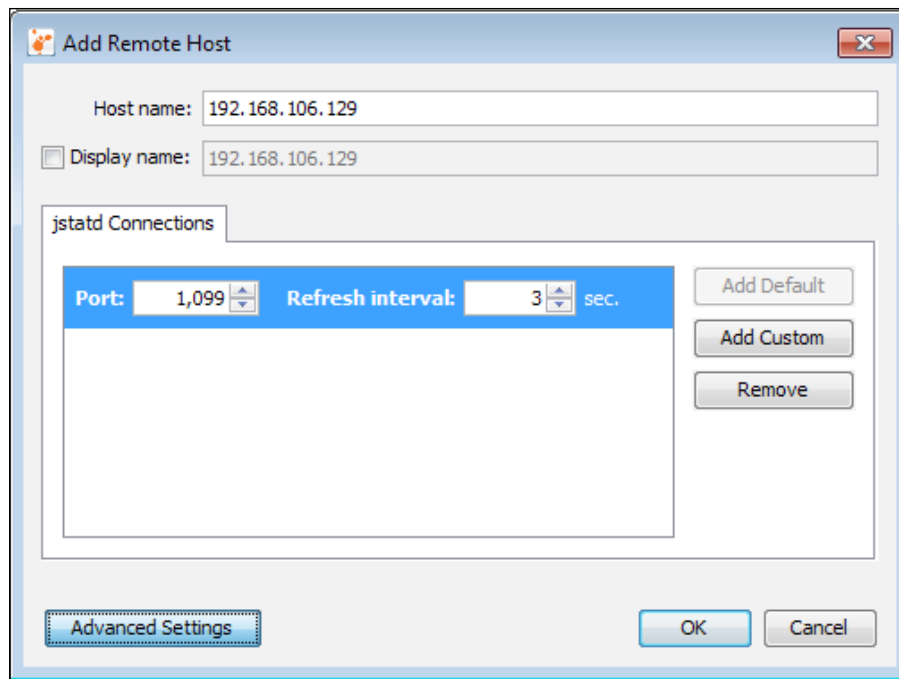
```
grant codebase "file:${java.home}/../lib/tools.jar" {  
    permission java.security.AllPermission;  
};
```

3. Now start a command-line terminal, navigate to the `bin` folder and run the following command:

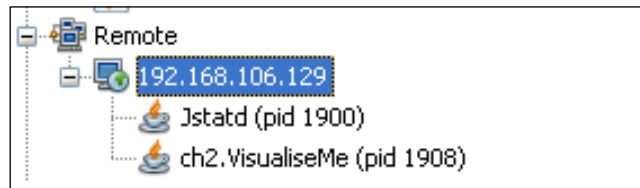
```
jstatd -J-Djava.security.policy=jstatd.policy
```

If started correctly, there will be no response from the program, it will just sit there.

4. We can now connect via VisualVM from a remote host by navigating to **File | Add Remote Host:**



5. You will be able to open all of the processes that jstatd has permission to access on the remote machine.



Setting the size of the thread stack

Every thread in your Java process has its own stack. The size of this stack can be controlled via the `-Xss` setting.

Getting ready

You will need to have Oracle SOA Suite 11g installed for this recipe. This command is the same whether you're using the Hotspot or the JRockit JVM.

How to do it...

Perform the following steps to configure the size of the thread stack:

1. Navigate to the domain's home directory:

```
cd %MIDDLEWARE_HOME%/user_projects/domains/soa_domain/
```


Replace `soa_domain` with the relevant domain, as required.
2. Within the `bin` folder, you will see a number of scripts. Depending on your platform you will need to edit a different file—`setSOADomainEnv.cmd` on Windows; `setSOADomainEnv.sh` on Linux. Note that it is a good idea to make a backup of these files before editing them.
3. Open the file and locate the following section. Note that I have manually placed the `@REM` comment; your file will not contain this line:

```
set JAVA_OPTIONS=%JAVA_OPTIONS%
set DEFAULT_MEM_ARGS=-Xms512m -Xmx1024m
set PORT_MEM_ARGS=-Xms768m -Xmx1536m
```

4. At the end of the `DEFAULT_MEM_ARGS` and `PORT_MEM_ARGS` lines, add the parameter `-Xss [n] k` where `[n]` is your stack size in KB. You can also specify a stack size in MB but this is not recommended on 32-bit machines. After the edit, your line will look like this, with 128 KB as my example value:

```
set JAVA_OPTIONS=%JAVA_OPTIONS%
set DEFAULT_MEM_ARGS=-Xms512m -Xmx1024m -Xss128k
set PORT_MEM_ARGS=-Xms768m -Xmx1536m -Xss128k
```

5. Start the server and view the output to verify your change:

```
starting weblogic with Java version:
java version "1.6.0_29"
Java(TM) SE Runtime Environment (build 1.6.0_29-b11)
Java HotSpot(TM) Client VM (build 20.4-b02, mixed mode)
Starting WLS with line:
e:\Oracle\MIDDLE~1\JDK160~1\bin\java -client -Xms512m -Xmx1024m -Xss128k
```

How it works...

The JVM is a multi-threaded application, and WebLogic and SOA Suite are built to take advantage of this, using threads (essentially, parallel sequences of instructions) to allow multiple tasks or requests to be processed at the same time. In most operating systems, threads are a service provided by the operating system, and so when the application requests new threads, the JVM looks to the operating system to provide the resources. This can take some time, so modern application servers create many threads at startup and keep them idle, giving them tasks to do as and when requests come in. This means that even an idle SOA Suite application will have many threads. Threads can communicate with each other by the use of variables and locks, which prevents multiple threads from accessing the same data at the same time.

Java thread stack size is used for storing information on the chain of methods (signatures, parameters, and return addresses) that are called in the thread, along with local variables. This has a default value depending on the JVM type, and whether you're running a 32 or a 64-bit version. Defaults for each JVM can change across releases, so we will not discuss them here.

Having a stack size that is too small for normal operations will prevent the thread from calling the chain of functions that it is requesting; you will encounter a `java.lang.StackOverflowError` if the stack cannot handle the method-call chain. Setting this value to a higher number will allow the method chains to complete successfully.

There's more...

There are VM-specific options of this command, such as `-XX:ThreadStackSize=[n]` for Hotspot, where `[n]` is the size in KB.

You need to be careful when setting this value. Each thread will use an amount of the native operating system's memory equal to the size of the thread stack; this is in addition to the process's maximum Java heap size. These will add to the Java heap size on your machine, which may explain why the Java process memory value returned by your operating system is higher than you expect. Also, operating systems can round this value off to quite coarse numbers. Typically, 64 KB is the least amount of stack space allowed per thread.

The JVM operating system defaults are usually sensible values, which we recommend starting with, and setting them only if you see issues or want explicit control of native memory size.

See also

- ▶ The *Calculating the total memory used by your application* recipe

5

JVM Garbage Collection Tuning

In this chapter, we will cover the following recipes:

- ▶ Setting the new size
- ▶ Setting the survivor ratio
- ▶ Choosing a garbage collection algorithm in Hotspot
- ▶ Choosing a garbage collection algorithm in JRockit
- ▶ Turning on verbose garbage collection
- ▶ Tuning to reduce the number of major garbage collections
- ▶ Disabling the RMI garbage collector
- ▶ Disabling explicit GC

Introduction

Poorly tuned **garbage collection (GC)** is one of the most common causes for interruptions in Java applications, and this includes Oracle SOA Suite 11g. This chapter is devoted to looking at some techniques that will help you tune the garbage collector in your JVM.

Garbage collection is the process and set of techniques by which the JVM frees up memory that holds the data which is no longer relevant, making it available to applications again. By removing the burden of tracking (when allocated memory is no longer needed) from the developer, Java increased the productivity of application developers, and because many application bugs are caused by memory leaks and other memory problems, the stability of applications also improved. However, the process of tracking which memory allocations are no longer needed is non-trivial; a method that works well on one set of circumstances may not work well in another.

There are a large number of settings related to the JVM garbage collector that can be tweaked. This chapter looks at some of the ones that you are most likely to tune.

Setting the new size

Getting the new size correct is probably the single biggest thing that can be done to prevent poor garbage collection performance.

Getting ready

This recipe assumes that you are starting WebLogic using the `startWebLogic` or `startManagedWebLogic` scripts. If you are using the node manager or some other mechanism, simply apply the startup parameter to the relevant place. You will need the file system permissions to edit the WebLogic start scripts for this recipe.

This recipe assumes that you are using the HotSpot JVM, and if you are using JRockit, see the *There's more...* section of this recipe, for the parameter you need.

How to do it...

To set the HotSpot heap memory's new size, perform the following steps:

1. Navigate to the domain's `bin` directory:
2. Open the `setSOADomainEnv.cmd` or `setSOADomainEnv.sh` script in a text editor.
3. Locate the following lines for a Windows system:

```
SET DEFAULT_MEM_ARGS=%DEFAULT_MEM_ARGS% -XX:PermSize=128m -  
XX:MaxPermSize=512m  
  
SET PORT_MEM_ARGS=%PORT_MEM_ARGS% -XX:PermSize=128m -  
XX:MaxPermSize=512m
```

Or the equivalent on a Linux or a Unix system.

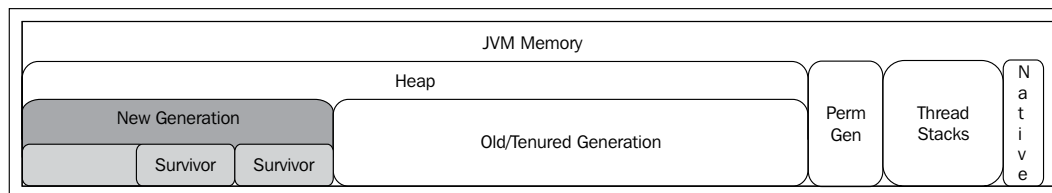
4. Edit the previous lines to add the parameter `-XX:NewSize=256m`
`-XX:MaxNewSize=256m`:

```
SET DEFAULT_MEM_ARGS=%DEFAULT_MEM_ARGS% -XX:PermSize=128m -  
XX:MaxPermSize=512m -Xmx1024m -XX:NewSize=256m -XX:MaxNewSize=256m  
  
SET PORT_MEM_ARGS=%PORT_MEM_ARGS% -XX:PermSize=128m -  
XX:MaxPermSize=512m -XX:NewSize=256m -XX:MaxNewSize=256m
```
5. Save the file.

6. Start WebLogic with the `startWebLogic` and `startManagedWebLogic` scripts, depending on whether this is a managed server.

How it works...

The new size is the amount of memory allocated to the new spaces in the JVM, where new objects are allocated. The new space is a part of the total heap (so the value must be less than the total heap), so in our experience we have found that a value between one quarter and one-third of the total heap works best for Oracle SOA Suite. Modern Java garbage collectors are all generational collectors; memory is split into a number of different generations, with objects being initially created in the new generation, and once they have survived a number of garbage collections, they are then promoted to the old generation. The new generation itself is split into an Eden space, where new objects are created, and two survivor spaces, where objects live until they are promoted to the old generation. Each time the Eden generation fills up, it is garbage collected along with the active survivor space, and any object that survives is moved into the other survivor space. This copy style of garbage collection is fast, but leaves memory segments unused. Thus, it is not space efficient:



We have set two parameters here: `-XX:NewSize` is the initial new generation (Eden plus two survivor spaces) and `-XX:MaxNewSize` is the maximum size that it can grow to. By setting these two values to the same amount, we are forcing the new size to be an exact size, thus preventing the JVM from pausing in order to resize it.

The objective of tuning the new size is to ensure that minor garbage collections do not occur too frequently, and that there is still plenty of space for the old generation to hold objects that are long lived.

There's more...

JRockit uses different garbage collection algorithms and methods as compared to HotSpot, but still has concepts similar to HotSpot. Just like HotSpot, it uses a new part of the heap, which it calls the "nursery," and an old heap to which long-lived objects are promoted after a garbage collection. The JRockit nursery size can be set by using the `-Xns256m` command-line parameter.

See also

- ▶ The *Setting Xmx and Xms to the same value* recipe in *Chapter 4, JVM Memory*
- ▶ The *Setting the survivor ratio* recipe
- ▶ The *Tuning to reduce the number of full garbage collections* recipe

Setting the survivor ratio

Survivor ratio determines the size of the survivor spaces, as a ratio of the total new space, and can be tuned to ensure that the survivor spaces are large enough to hold all the objects that survive "young" garbage collections.

Getting ready

This recipe assumes you are using the HotSpot JVM. JRockit manages memory differently and does not have an equivalent parameter.

For more details, refer to the *Getting ready* section of the *Setting the new size* recipe.

How to do it...

To set the heap survivor ratio for the HotSpot JVM, perform the following steps:

1. Navigate to the domain's bin directory:

```
cd %MIDDLEWARE_HOME%/user_projects/domains/soa_domain/bin
```
2. Open the `setSOADomainEnv.cmd` or `setSOADomainEnv.sh` script in a text editor.
3. Locate the following lines:

```
SET DEFAULT_MEM_ARGS=%DEFAULT_MEM_ARGS% -XX:PermSize=128m -  
XX:MaxPermSize=512m  
  
SET PORT_MEM_ARGS=%PORT_MEM_ARGS% -XX:PermSize=128m -  
XX:MaxPermSize=512m
```

Or the equivalent on a Linux or a Unix system.

4. Edit the previous lines to add the parameter `-XX:SurvivorRatio=8`:

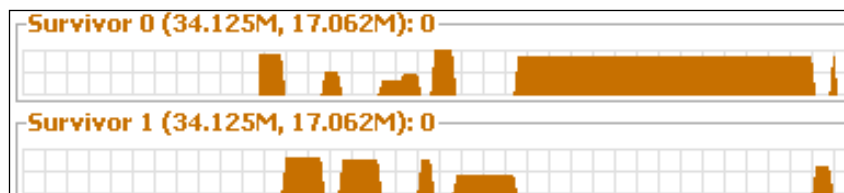
```
SET DEFAULT_MEM_ARGS=%DEFAULT_MEM_ARGS% -XX:PermSize=128m -  
XX:MaxPermSize=512m -Xmx1024m -XX:SurvivorRatio=8  
  
SET PORT_MEM_ARGS=%PORT_MEM_ARGS% -XX:PermSize=128m -  
XX:MaxPermSize=512m -XX:SurvivorRatio=8
```

5. Save the file.
6. Start WebLogic using the `startWebLogic` and `startManagedWebLogic` scripts.

How it works...

The survivor spaces are a part of the new generation area of the JVM heap. Objects are allocated in Eden, and when a minor garbage collection occurs, live objects (if any) are copied from Eden and the active survivor space, and pasted into the passive survivor space. The passive survivor space is then marked as active, and the now-empty survivor space, which was previously active, becomes passive. The entirety of the Eden space can now be considered empty (because all the live objects were moved) and available to have new objects allocated into it (refer to the figure in the *How it works...* section of the *Setting the new size* recipe).

The survivor ratio determines the size of each of the survivor spaces, as a ratio of the total new size. A survivor ratio of 1:8 means that each survivor space is one-tenth of the new total size, because there are two survivor spaces. We have found that 8 is a good starting point for tuning, but you should benchmark your application to determine the best value for you. The following screenshot shows how only one survivor space is in use at a time:



See also

- ▶ The *Setting the new size* recipe
- ▶ The *Setting Xmx and Xms to the same value* and *Viewing the memory used using VisualVM for HotSpot* recipes in *Chapter 4, JVM Memory*

Choosing a garbage collection algorithm in HotSpot

Choice of garbage collection algorithm can have a huge impact on the performance of an application, which is why HotSpot provides a number of choices. This recipe explains how to set the garbage collection algorithm for HotSpot, and then discusses the trade-offs between the algorithm choices at a high level.

Getting ready

This recipe assumes you are using the HotSpot JVM. There is a separate recipe covering JRockit.

For more details, refer to the *Getting ready* section of the *Setting the new size* recipe.

How to do it...

To set the HotSpot garbage collector for a WebLogic Server, perform the following steps:

1. Navigate to the domain's bin directory:

```
cd %MIDDLEWARE_HOME%/user_projects/domains/soa_domain/bin
```
2. Open the `setSOADomainEnv.cmd` or `setSOADomainEnv.sh` script in a text editor.
3. Locate the following lines:

```
SET DEFAULT_MEM_ARGS=%DEFAULT_MEM_ARGS% -XX:PermSize=128m -  
XX:MaxPermSize=512m  
  
SET PORT_MEM_ARGS=%PORT_MEM_ARGS% -XX:PermSize=128m -  
XX:MaxPermSize=512m
```

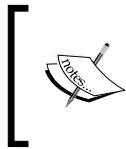
Or the equivalent on a Linux or a Unix system.

4. Edit the previous lines to add the parameter `-XX:+UseParallelGC`:

```
SET DEFAULT_MEM_ARGS=%DEFAULT_MEM_ARGS% -XX:PermSize=128m -  
XX:MaxPermSize=512m -XX:+UseParallelGC  
  
SET PORT_MEM_ARGS=%PORT_MEM_ARGS% -XX:PermSize=128m -  
XX:MaxPermSize=512m -XX:UseParallelGC
```
5. Save the file.
6. Start WebLogic using the `startWebLogic.cmd` or `startWebLogic.sh` scripts.

How it works...

The `setSoaDomainEnv` script sets up a number of parameters, including memory-related parameters that are passed to the JVM when starting WebLogic. It is also possible to set the memory arguments and the JVM startup parameters—by specifying remote startup parameters in the WebLogic console. These parameters will only take effect when the WebLogic node manager is used to start and stop servers, and so we don't discuss it in detail here, although it is the recommended way of configuring memory parameters for your server in a production environment.



You can find the instructions for configuring the node manager in the WebLogic Server documentation at http://docs.oracle.com/cd/E21764_01/apirefs.1111/e13952/taskhelp/machines/BindToProtectedPortsOnServersStartedByNodeManager.html.

HotSpot provides a number of garbage collection algorithms, and you can choose one among them by setting the relevant flags on the command line at start-up. We recommend starting out with the parallel garbage collector, using the `-XX:UseParallelGC` flag.

There's more...

As we have mentioned, there are multiple choices for garbage collection algorithm, depending on your application, you may find that one of the algorithms works better for you. The following algorithms are available:

- ▶ **Concurrent mark sweep:** It is set by using the `-XX:+UseConcMarkSweepGC` flag. This is a low pause garbage collector, best used for real-time or fat-client applications, where subsecond response times are important. While low pause times are important, this algorithm can perform badly under heavy load, so we do not normally recommend it for the SOA Suite.
- ▶ **Parallel garbage collection:** It is set by using the `-XX:UseParallelGC` flag. This algorithm uses multiple cores to perform garbage collection, and is the default algorithm for server-class machines. We have found it to be the best performing algorithm for the majority of SOA Suite installations, so we recommend it as a starting point.
- ▶ **Garbage first:** It is set by using the `-XX:+UseG1GC` flag. This is a new garbage collector intended as a replacement for concurrent mark sweep, for use in applications that need large heaps and low pause times. As it is still new, we have not had much chance to investigate how well it works in real life SOA Suite deployments.

See also

- ▶ The *Choosing a garbage collection algorithm in JRockit* recipe

Choosing a garbage collection algorithm in JRockit

Choice of garbage collection algorithm can have a large impact on the performance of an application, JRockit provides you with two options to specify whether performance or pause times are more important to your application.

Getting ready

This recipe assumes you are using the JRockit JVM. There is a separate recipe for using the HotSpot JVM.

For more details, refer to the *Getting ready* section of the *Setting the new size* recipe.

How to do it...

To set the JRockit garbage collector for a WebLogic Server, perform the following steps:

1. Navigate to the domain's bin directory:

```
cd %MIDDLEWARE_HOME%/user_projects/domains/soa_domain/bin
```
2. Open the `setSOADomainEnv.cmd` or `setSOADomainEnv.sh` script in a text editor.
3. Locate the following lines:

```
SET DEFAULT_MEM_ARGS=%DEFAULT_MEM_ARGS% -XX:PermSize=128m -  
XX:MaxPermSize=512m  
  
SET PORT_MEM_ARGS=%PORT_MEM_ARGS% -XX:PermSize=128m -  
XX:MaxPermSize=512m
```

Or the equivalent on a Linux or a Unix system.

4. Edit the previous lines to add the parameter `-Xgcprio:throughput`:

```
SET DEFAULT_MEM_ARGS=%DEFAULT_MEM_ARGS% -XX:PermSize=128m -  
XX:MaxPermSize=512m -Xgcprio:throughput  
  
SET PORT_MEM_ARGS=%PORT_MEM_ARGS% -XX:PermSize=128m -  
XX:MaxPermSize=512m -Xgcprio:throughput
```
5. Save the file.
6. Start WebLogic using the `startWebLogic.cmd` or `startWebLogic.sh` scripts.

How it works...

The `setSoaDomainEnv` script sets up a number of parameters, including the memory-related parameters that are passed to the JVM when starting WebLogic.

JRockit allows you to choose either throughput or pause time as the priority for runtime execution. Throughput priority will cause JRockit to self-optimize its settings to try and get the most out of your application—that is to spend as little time as possible collecting garbage. Pause time priority focuses on reducing the impact of each individual garbage collection by keeping pause times down. This will usually result in more frequent but faster garbage collections, with a higher total amount of time spent in garbage collection. **Pause time priority** is normally most appropriate for "fat-client" style applications and real-time systems, because its performance degrades rapidly if it is unable to keep up with the rate that garbage is created.

To set pause time priority, you would use the parameter `-Xgcprio:pausetime` rather than `-Xgcprio:throughput`.

There's more...

JRockit and HotSpot have very different ways of selecting the garbage collection algorithm, because JRockit makes a dynamic decision to select the garbage collection algorithm it uses based on your priorities, whereas HotSpot allows you to directly specify the garbage collection algorithms.

We have found that the JRockit approach works well for organizations that want to specify the general behaviors of the garbage collector without having to understand the details of garbage collection algorithms, and their uses. If you wish to specify an algorithm directly, there are a number of legacy garbage collectors available, and you can specify them with the following flags:

- ▶ `-Xgc:parallel` for parallel garbage collection
- ▶ `-Xgc:gencon` for generational concurrent garbage collection
- ▶ `-Xgc:singlecon` for a concurrent and non-generational garbage collector

See also

- ▶ *The Choosing a garbage collection algorithm in HotSpot recipe*

Turning on verbose garbage collection

Verbose garbage collection is a feature that writes log files detailing when garbage collection occurred, how long it took, and what was collected. While we would not recommend having it permanently turned on in a live system, it can be useful when trying to debug garbage collection related problems.

Getting ready

This recipe assumes you are using the HotSpot JVM, if you are using JRockit, see the *There's more...* section of this recipe for the parameter you need.

For more details, refer to the *Getting ready* section of the *Setting the new size* recipe.

How to do it...

To enable verbose garbage collection logging, perform the following steps:

1. Navigate to the domain's bin directory:
`cd %MIDDLEWARE_HOME%/user_projects/domains/soa_domain/bin`
2. Open the `setSOADomainEnv.cmd` or `setSOADomainEnv.sh` script in a text editor.
3. Locate the following lines:

```
SET DEFAULT_MEM_ARGS=%DEFAULT_MEM_ARGS% -XX:PermSize=128m -  
XX:MaxPermSize=512m  
  
SET PORT_MEM_ARGS=%PORT_MEM_ARGS% -XX:PermSize=128m -  
XX:MaxPermSize=512m
```

Or the equivalent on a Linux or a Unix system.

4. Edit the previous lines to add the parameter `-verbose:gc -Xloggc:gc.log`:

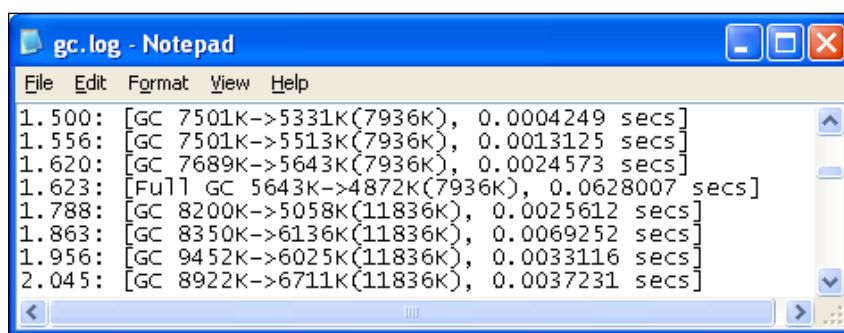
```
SET DEFAULT_MEM_ARGS=%DEFAULT_MEM_ARGS% -XX:PermSize=128m -  
XX:MaxPermSize=512m -Xmx1024m -verbose:gc -Xloggc:gc.log  
  
SET PORT_MEM_ARGS=%PORT_MEM_ARGS% -XX:PermSize=128m -  
XX:MaxPermSize=512m -verbose:gc -Xloggc:gc.log
```

If you already have additional parameters on these lines, then leave them in place.

5. Save the file.
6. Start WebLogic using the `startWebLogic.sh` and `startManagedWebLogic.sh` scripts.

How it works...

We set two parameters in this recipe: `-verbose:gc` enables verbose garbage collector output, and `-Xloggc:gc.log` sends the output to a file called `gc.log` in the root of the domain directory. You can specify a different file name, or even a path to a location, by specifying a different location after `-Xloggc:`. In a system that is performing well, the overhead of outputting garbage collection timings is not high (because there should not be many garbage collections occurring), and the information that is contained can be useful. The garbage collection log file indicates the type of garbage collection that occurred (minor or full), the starting and ending sizes, and the time that the garbage collection took.



The format of the output is as follows:

```
{time}: [{GC Type} {Starting size}->{Final size}({Amount freed}), {time taken} secs]
```

- ▶ **Time:** It is the time in seconds since the JVM started
- ▶ **GC Type:** It is either GC (minor) or Full GC (full)
- ▶ **Starting size:** It is the size of the heap before garbage collection
- ▶ **Final size:** It is the size of the heap after garbage collection
- ▶ **Amount freed:** It is the difference between the final and starting sizes
- ▶ **Time taken:** It tells how long, in seconds, the garbage collection took

This information is useful when tuning garbage collection, because we can see how much memory gets freed up by minor and full collections, and how long they take.

On a Linux or a Unix environment, the time taken is further split into system, user, and real time, so you can understand where any potential bottleneck is.

There's more...

JRockit also supports verbose garbage collection output, using the parameter `-Xverbose:gc`.

See also

- ▶ The *Viewing the memory used using VisualVM for HotSpot* recipe in *Chapter 4, JVM Memory*
- ▶ The *Tuning to reduce the number of full garbage collections* recipe

Tuning to reduce the number of full garbage collections

The objective of garbage collection tuning in an SOA Suite application is to reduce the overhead of garbage collection. This chapter looks at how we do that at a high level.

Getting ready

This recipe discusses the process of tuning garbage collection for an application in the SOA Suite; as such it uses a number of techniques described elsewhere in the book. This recipe assumes that you are using the HotSpot JVM, and if you are using the JRockit JVM, the same principals apply. The starting parameters suggested are based on using a 64 bit JVM. If you are using a 32-bit JVM, you will need to reduce these. This recipe also assumes that you are starting servers using the `startWebLogic` and `startManagedWebLogic` scripts. If you are using a different mechanism to start servers, such as the node manager, you should make the changes elsewhere.

You will need the file system permissions to edit the WebLogic startup scripts for this recipe.

How to do it...

We now perform the following steps to set JVM parameters, and use command-line tools to evaluate the number of JVM garbage collections:

1. Navigate to the domain's `bin` directory, as shown in the following command line:

```
cd %MIDDLEWARE_HOME%/user_projects/domains/soa_domain/bin
```
2. Open the script `setSOADomainEnv.cmd` or `setSOADomainEnv.sh` in a text editor.
3. Locate the following lines:

```
SET DEFAULT_MEM_ARGS=-Xms=512m -Xmx=1024m
```

```
SET PORT_MEM_ARGS=-Xms=512m -Xmx=1024m
```

Or the equivalent on a Linux or a Unix system.

4. Set `Xms` and `Xmx` to the same value:

```
SET DEFAULT_MEM_ARGS=-Xms=1536m -Xmx=1536m
SET PORT_MEM_ARGS=-Xms=1536m -Xmx=1536m
```

5. Locate the following lines:

```
SET DEFAULT_MEM_ARGS=%DEFAULT_MEM_ARGS% -XX:PermSize=128m -
XX:MaxPermSize=512m
SET PORT_MEM_ARGS=%PORT_MEM_ARGS% -XX:PermSize=128m -
XX:MaxPermSize=512m
```

6. Set the following parameters to reduce the occurrence of events that cause a major garbage collection:

- ❑ `-XX:+UseParallelGC`
- ❑ `-XX:+DisableExplicitGC`
- ❑ `-XX:NewSize=384m -XX:MaxNewSize=384m`
- ❑ `-XX:PermSize=384m`

```
SET DEFAULT_MEM_ARGS=%DEFAULT_MEM_ARGS% -XX:PermSize=384m -
XX:MaxPermSize=512m -XX:+UseParallelGC -XX:+DisableExplicitGC -
XX:NewSize=284m -XX:MaxNewSize=384m
SET PORT_MEM_ARGS=%PORT_MEM_ARGS% -XX:PermSize=384m -
XX:MaxPermSize=512m -XX:+UseParallelGC -XX:+DisableExplicitGC -
XX:NewSize=284m -XX:MaxNewSize=384m
```

7. Save the file.
8. Start WebLogic using the start scripts.
9. Start `jstat` and monitor garbage collections (see the *Identifying new size problems with jstat* recipe in *Chapter 1, Identifying Problems*, for instructions on using this tool).
10. Use your performance-testing framework to load the application with a normal load (see the *Designing advanced load test* recipe in *Chapter 3, Performance Testing*, for an example on using JMeter).
11. Use `jstat` to monitor the garbage collection performance; in particular, you want to check the frequency of full garbage collections (FGCC) and their duration (FGCT divided by FGCC), and the cause of those garbage collections (GCC and LGCC).

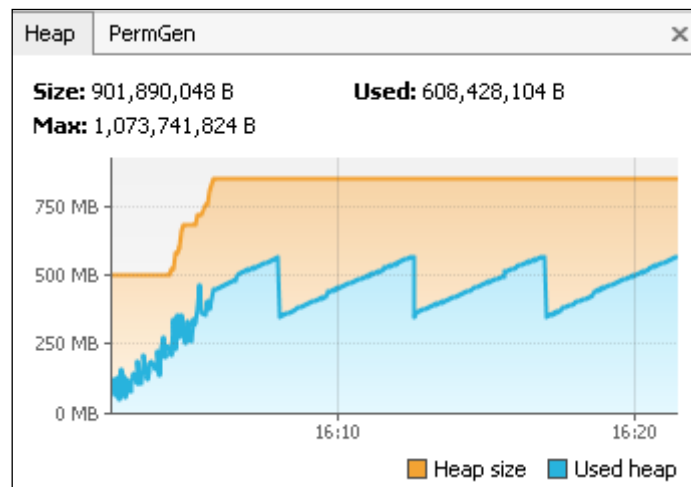
You are aiming for full garbage collections to occur every few hours under normal load, and take a few seconds. If they occur more frequently, try increasing your heap size or your new generation size.

12. Use your performance testing framework to place an extreme load on the application.
13. Use `jstat` again to monitor the garbage collection performance, looking for the same things as in step 11.

How it works...

When the JVM runs the garbage collector, it is what is called a "stop the world event", that is, everything stops while the garbage collector runs, and this includes all of your application code. No requests will be received or responded to; no processing occurs. It is therefore obvious that we want to reduce the number of times that this occurs, but more importantly for a SOA Suite application to reduce the amount of time spent collecting garbage as a percentage of total application time, which is called throughput.

This trade-off between tuning for reduced pauses and tuning for throughput is often a source of confusion for people who are not familiar with garbage collection tuning; it means there are no "best" garbage collection settings that work for everything, because all applications have different requirements. In general, SOA Suite applications are best tuned to improve throughput. A 2-second pause every few hours while a full garbage collection is performed is usually preferable to a pause of a few hundred milliseconds every few seconds, because more "work" gets done overall.



Regardless of whether we decide to tune for throughput or for low pause times, reducing the number of full garbage collections is always a key goal, because the full garbage collections take significantly longer than minor garbage collections. In order to reduce the number of collections, we need to understand what can cause a full garbage collection to occur. The following list provides the main causes:

- ▶ **Allocation failure:** The JVM wanted to allocate some space for an object, but not enough space was available. This can occur in the Eden generation when creating a new object, or in another generation when moving an object as part of generational garbage collection.
- ▶ **System.gc:** Application code invoked the garbage collector directly, which is usually not a good thing.
- ▶ **A change to the JVM heap size:** If you have not set `Xmx` and `Xms` to the same value, then when the JVM grows its heap towards `Xmx`, a full garbage collection will be triggered.

So, to reduce the number of full garbage collections, we need to work to reduce how frequently these triggers occur. The first two causes can be prevented altogether by disabling explicit GC, and setting `Xmx` and `Xms` to the same value. Once we have set both of these parameters, we focus our efforts on reducing the occurrence of allocation failures.

Allocation failures happen when an attempt is made to allocate space for an object, but there is insufficient space in the relevant memory space. This can happen in a number of different spaces, listed as follows:

- ▶ Failure to allocate a new object in Eden
- ▶ Failure to move an object from Eden to a survivor space or from one survivor space to another, during a minor GC
- ▶ Failure to move an object from a survivor space to the old generation
- ▶ Failure to find space in the permanent generation to store the definition of a class file (HotSpot only)

The fourth point can be dealt with by correctly sizing the permanent generation based on your application size, whereas the other three points come down to correctly tuning the size of your total heap, and the Eden and survivor spaces within that.

See also

- ▶ The *Setting the new size* recipe
- ▶ The *Setting the survivor ratio* recipe
- ▶ The *Choosing a garbage collection algorithm in HotSpot* recipe
- ▶ The *Turning on verbose garbage collection* recipe
- ▶ The *Choosing a garbage collection algorithm in JRockit* recipe
- ▶ The *Viewing the memory used using VisualVM for HotSpot* recipe in Chapter 4, *JVM Memory*
- ▶ The *Viewing the memory used using JRMCI for JRockit* recipe in Chapter 4, *JVM Memory*

Disabling the RMI garbage collector

The RMI garbage collector is a task that periodically runs an explicit garbage collection in order to free up objects involved in remote method calls. By default, this runs once per minute, which is very frequent for a system where most RMI objects are going to be quite long-lived.

Getting ready

For more details, refer to the *Getting ready* section of the *Setting the new size* recipe.

How to do it...

To disable the RMI garbage collector, consider the following steps:

1. Navigate to the domain's bin directory:

```
cd %MIDDLEWARE_HOME%/user_projects/domains/soa_domain/bin
```
2. Open the script `setSOADomainEnv.cmd` or `setSOADomainEnv.sh` in a text editor.
3. Locate the following line:

```
SET EXTRA_JAVA_PROPERTIES=%EXTRA_JAVA_PROPERTIES% -Dums.oracle.  
home=%UMS_ORACLE_HOME%
```

Or the equivalent on a Linux or a Unix system.

4. Edit the previous line to add the following properties:

```
□ -Dsun.rmi.dgc.client.gcInterval=3600000  
□ -Dsun.rmi.dgc.server.gcInterval=3600000  
  
SET EXTRA_JAVA_PROPERTIES=%EXTRA_JAVA_PROPERTIES% -Dums.oracle.  
home=%UMS_ORACLE_HOME% -Dsun.rmi.dgc.client.gcInterval=3600000  
-Dsun.rmi.dgc.server.gcInterval=3600000
```

If you already have additional parameters on these lines, then leave them in place.

5. Save the file.
6. Start WebLogic using the `startWebLogic.sh` and `startManagedWebLogic.sh` scripts.

How it works...

Remote Method Invocation (RMI) is the low-level framework that Java uses to allow code running in one JVM to call code running in another JVM. Because developers are not responsible for tracking unused memory in Java, it then falls to the JVM to track which objects might be still being used by a remote JVM, and this is the task of the RMI garbage collector.

In a SOA Suite application, most of the code that is accessed by a remote JVM will be the code that provides services, and therefore will be relatively long-lived, and will not become available for garbage collection very often. By reducing the frequency of the RMI garbage collector, we can reduce the number of RMI garbage collections, and keep the CPU overhead of this relatively intensive task down. The interval is specified in milliseconds, and the default interval is 60 seconds, but we would suggest trying a value like 1 hour (the value of 3,600,000 seconds and above).

See also

- The *Disabling explicit GC* recipe

Disabling explicit GC

Garbage collection is a very CPU-intensive activity; while it is occurring, no application logic can be executed. While the Java API provides a method of invoking the garbage collector from application code, it is best practice to let only the JVM decide when to perform garbage collection. If an application contains explicit calls to `System.gc()`, then these can significantly impact performance. Luckily, Java includes a flag that allows us to disable calling the garbage collector programmatically.

Getting ready

This recipe assumes you are using the HotSpot JVM, if you are using JRockit, see the *There's more...* section of this recipe for the parameter you need.

For more details, refer to the *Getting ready* section of *Setting the new size* recipe.

How to do it...

To disable explicit programmatic garbage collection, perform the following steps:

1. Navigate to the domain's `bin` directory:

```
cd %MIDDLEWARE_HOME%/user_projects/domains/soa_domain/bin
```
2. Open the `setSOADomainEnv.cmd` or `setSOADomainEnv.sh` script in a text editor.
3. Locate the following lines:

```
SET DEFAULT_MEM_ARGS=%DEFAULT_MEM_ARGS% -XX:PermSize=128m -  
XX:MaxPermSize=512m  
  
SET PORT_MEM_ARGS=%PORT_MEM_ARGS% -XX:PermSize=128m -  
XX:MaxPermSize=512m
```

Or the equivalent on a Linux or a Unix system.

4. Edit the previous lines to add the parameter `-XX:+DisableExplicitGC`:

```
SET DEFAULT_MEM_ARGS=%DEFAULT_MEM_ARGS% -XX:PermSize=128m -  
XX:MaxPermSize=512m -Xmx1024m -XX:+DisableExplicitGC  
  
SET PORT_MEM_ARGS=%PORT_MEM_ARGS% -XX:PermSize=128m -  
XX:MaxPermSize=512m -XX:+DisableExplicitGC
```

If you already have additional parameters on these lines, then leave them in place.
5. Save the file.
6. Start WebLogic using the `startWebLogic.sh` and `startManagedWebLogic.sh` script.

How it works...

In a Java application, it is possible to programmatically invoke the garbage collector using a call to `System.gc()`. This call will cause the JVM to make its best effort to run the garbage collector before control returns to the application. In practice, this means that a full garbage collection will be run, which is a time-consuming and CPU-intensive operation. While it might make sense to programmers to attempt to clean up objects when they're no longer needed in application code, in reality, the JVM has a much clearer view on how much memory is used, and how much it should be able to reclaim, and can do a much better job at deciding when to perform garbage collections. Because of the "stop the world" nature of a full garbage collection, and the fact that calls to `System.gc()` can cause more full garbage collections to run, this often causes worse performance rather than an improvement.

By disabling calls to `System.gc()`, the calls are ignored and the JVM performs its garbage collection when memory thresholds are reached. If the developers of the SOA Suite application you are using have tried to be helpful by calling the garbage collector for you, you can often improve performance by disabling explicit garbage collection calls.

There's more...

In JRockit, the flag to be used is `-XXnoSystemGC`, which has the same effect.

See also

- ▶ *The Tune to reduce the number of full garbage collections recipe*
- ▶ *The Disable the RMI garbage collector recipe*

6

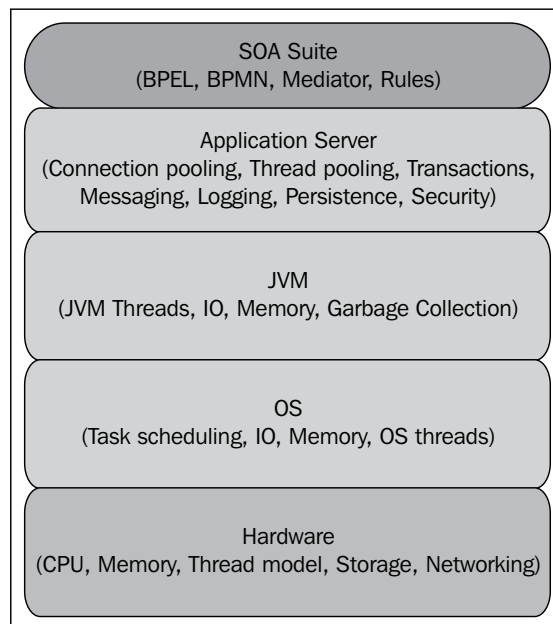
Platform Tuning

In this chapter, we will look at the following recipes:

- ▶ Tuning global transaction timeouts
- ▶ Increasing the HTTP accept backlog
- ▶ Reducing the server logging level
- ▶ Finding out which JVM you are using
- ▶ Using large pages in Linux
- ▶ Increasing the number of file descriptors in Linux
- ▶ Tuning the SOA Suite EJB timeouts
- ▶ Upgrading to a newer JVM
- ▶ Setting the Linux kernel swappiness to low
- ▶ Using the Oracle JRockit JVM
- ▶ Running your domain in the production mode
- ▶ Creating a boot.properties file

Introduction

This chapter looks at some of the things you can do to tune the underlying platform on which the SOA Suite runs. SOA Suite is an enterprise Java application, and therefore runs on top of a stack consisting of hardware, operating system, JVM, and application server. Each of these layers builds upon resources provided by the layers below them, and can be tuned to improve the performance of Oracle SOA Suite.



The previous diagram shows the layers in the SOA Suite stack, and examples of the services that they provide to the layer above it. This chapter gives some of the key techniques for tuning the operating system, the application server, and selecting a JVM. JVM memory and garbage collection tuning are such key concepts that they get their own chapters.

Performance gains obtained by tuning the underlying platform components can be significant because they impact all of the layers above it; so platform tuning is one of the most important aspects when tuning an Oracle SOA Suite 11g installation.

Tuning global transaction timeouts

The global transaction timeout, set at the application server level, defines how long a transaction has to complete before it is terminated by the application server. If this value is set too low, then performance can suffer.

Getting ready

For this recipe, you will need to know the URL and login credentials for the WebLogic administration console that runs on the admin server in your SOA Suite domain. The default URL for the console is `http://<servername>:7001/console`. The username and password will have been specified when you created the domain.

If your domain is in the production mode, or does not have **Automatically Acquire Lock and Activate Changes**, then you will need to acquire the domain configuration lock before making these changes, and apply the changes after saving.

How to do it...

To tune global transaction timeouts, perform the following steps:

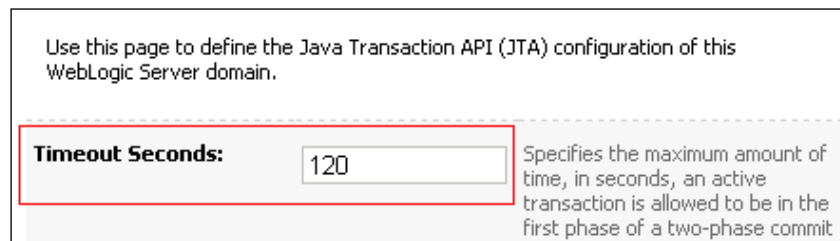
1. Connect to the WebLogic administration console at `http://localhost:7001/console`. Replace `localhost` and `7001` with the hostname and port of the server, if it is not running locally on `7001`. If this is the first time you have accessed the WebLogic console since starting the server, WebLogic will first deploy the console application, which may take a few minutes.
2. Log in to the console using your administration credentials.
3. Click on the name of your domain at the root of the navigation tree in the left-hand pane; for example, **soa_domain** is at the root of the tree, as shown in the following screenshot:



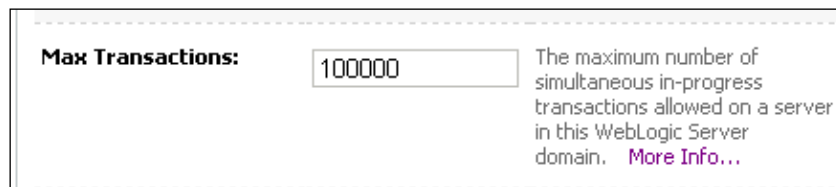
4. Select the **JTA** tab under the **Configuration** tab in the main pane.



5. Increase the value of `TimeoutSeconds` to a value that ensures that the slowest external systems you talk to will complete in time. We would recommend a setting of at least 120.



6. Scroll down to the `Max Transactions` field, and increase this to 100000:



7. Scroll to the top of the page and click on **Save**.

How it works...

The global **Java Transaction API (JTA)** timeout sets the time that WebLogic will wait for transactions to complete. In the SOA Suite, this affects how long we wait for external systems to return data, which in a large SOA environment can sometimes take longer than 30 seconds. By increasing this timeout slightly, we reduce the chances that these calls will fail due to a slow external system. Failed transactions are usually retried by the originating system, meaning that we create additional load on an already heavily loaded system.

The **Max Transactions** setting determines the maximum number of concurrent transactions that can be in progress at a time. In a busy SOA Suite system, it is not difficult to reach the default, out-of-the-box limit of 10000 transactions, which can cause calls to fail, often requiring that they are replayed, and generating additional load on an already busy system.

See also

- ▶ The *Setting datasource connection timeouts* recipe in *Chapter 7, Data Sources and JMS*
- ▶ The *Tuning connections in native eis db adapter* recipe in *Chapter 7, Data Sources and JMS*
- ▶ The *Tuning SOA Suite EJB timeouts* recipe

Increasing the HTTP accept backlog

The HTTP accept backlog determines how many HTTP requests can be queued by the WebLogic server if there are not enough threads available to service them.

Getting ready

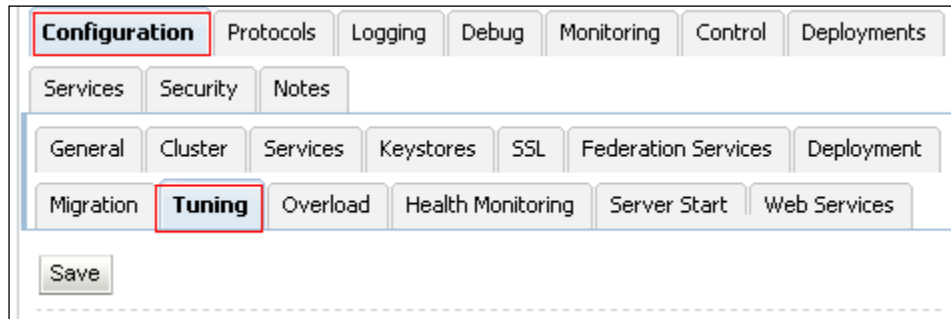
For details on this, refer to the *Getting ready* section of the *Tuning global transaction timeouts* recipe.

How to do it...

To increase the HTTP accept backlog, consider the following steps:

1. Connect to the WebLogic administration console at `http://localhost:7001/console`. Replace `localhost` and `7001` with the hostname and port of the server if it is not running locally on `7001`. If this is the first time you have accessed the WebLogic console since starting the server, WebLogic will first deploy the console application, which may take a few minutes.
2. Log in to the console using your administration credentials.
3. Navigate to **Environment**, and then **Servers** on the left-hand navigation pane.
4. Select the server with the SOA infrastructure deployed (if there are multiple servers with the SOA infrastructure deployed, you need to follow steps 4-x for each of them). Out-of-the-box, this will be a server called `soa_server1`.

5. Navigate to the **Tuning** tab under the **Configuration** tab:



6. Scroll down, and set the parameter `Accept Backlog` to 1000, or a value determined by your performance testing:

A screenshot of a configuration field. It shows the label 'Accept Backlog:' followed by a text input box containing the value '1000'. The input box has a light gray border and a small vertical line on the right side.

7. Scroll to the top of the page and click on **Save**.
8. Repeat the steps from 3 through 8 for each server that has the SOA Suite infrastructure deployed.

How it works...

The accept backlog setting determines how many TCP requests will be queued up when there are no idle threads available to service the requests, and applies to requests on both the servers' normal and SSL ports. The default value is 300, so once 300 requests are queued up, any additional requests are refused, and the calling system or user receives a `Connection Refuse` response. This can cause performance problems because it often results in the calling system retrying the request from the start, placing more load on an already heavily loaded server.

There's more...

While you are on this page, you may also want to take a look at some of the other settings here. In particular, you may want to increase the **Stuck Thread Time** if your system has blocking calls to external systems that take a long time to respond.

Reducing the server logging level

While logging information to the files is useful, it generates a lot of disk I/O, which has a huge performance impact on the application. Reducing the amount of logging done by the servers can improve performance noticeably.

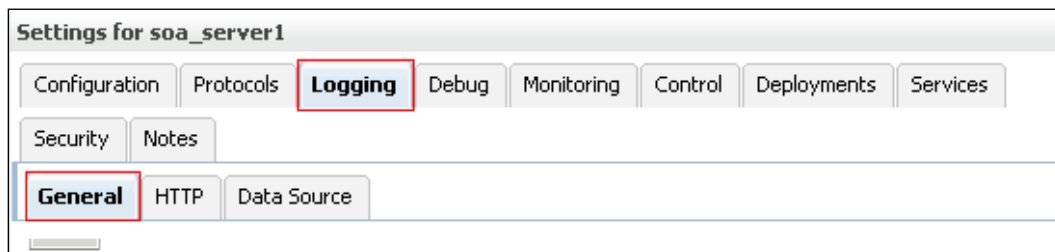
Getting ready

For details on this, refer to the *Getting ready* section of the *Tuning global transaction timeouts* recipe.

How to do it...

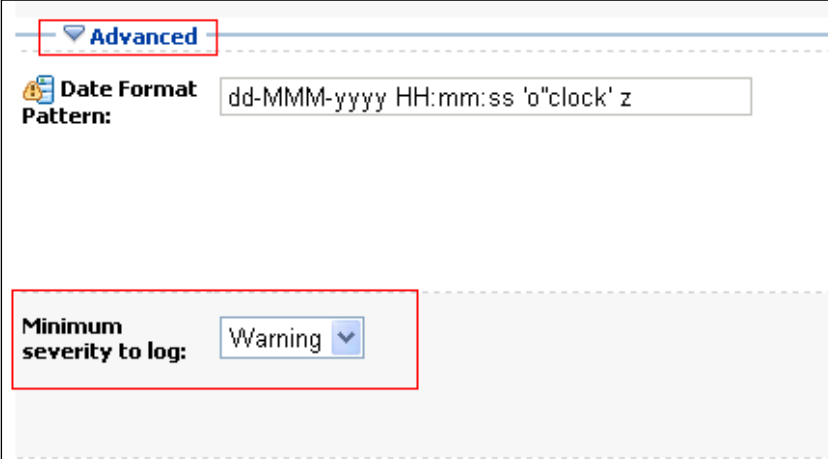
To reduce the server logging level, perform the following steps:

1. Connect to the WebLogic administration console at `http://localhost:7001/console`. Replace `localhost` and `7001` with the hostname and port of the server if it is not running locally on `7001`. If this is the first time you have accessed the WebLogic console since starting the server, WebLogic will first deploy the console application, which may take a few minutes.
2. Log in to the console with your administration credentials.
3. Navigate to **Environment**, and then **Servers** on the left-hand navigation pane.
4. Select the server with the SOA infrastructure. Out-of-the-box, this will be a server called `soa_server1`.
5. Navigate to the **General** tab under the **Logging** tab.



6. Scroll down to the bottom of the page, and expand the **Advanced** options.

7. Use the **Minimum severity to log** dropdown to set the severity as **Warning**.



The screenshot shows a configuration interface with a tab labeled "Advanced". Below the tab, there is a "Date Format Pattern" field with the value "dd-MMM-yyyy HH:mm:ss 'o'clock' z". Further down, there is a "Minimum severity to log:" label followed by a dropdown menu currently set to "Warning". Red boxes highlight the "Advanced" tab and the "Minimum severity to log:" dropdown.

8. Scroll to the top of the page and click on **Save**.
9. Repeat steps from 3 through 9 for each server with the SOA Suite infrastructure deployed.

How it works...

Writing log messages to disk requires a considerable amount of file I/O, which is often a synchronous operation, and as most disks are slow compared to the speed of the CPUs and the memory, this slows down the entire application. By increasing the logging severity to **Warning**, only messages at **Warning** and **Error** will be written to the server logs, which can provide a significant performance improvement, especially under heavy load.

There's more...

After setting the minimum severity to log to **Warning**, I would recommend setting the **Severity level** of the **Domain log broadcaster** to **Error**. This can be done further down on the same page, and will ensure that only error logs get sent to the domain log file.

See also

- ▶ The *Reducing SOA Infra log levels* recipe in *Chapter 8, BPEL and BPMN Engine Tuning*

Finding out which JVM you are using

Knowing which JVM you are using is a prerequisite to be able to pick one that might perform better.

Getting ready

You need to have installed WebLogic and the SOA Suite for this, and have access to the `startWebLogic` script.

How to do it...

To find out which JVM you are using, perform the following steps:

1. Navigate to the domain `home` directory.
2. Execute the `startWebLogic` script for the operating system you are using. This will be `startWebLogic.sh` on Linux/Unix environments, and `startWebLogic.cmd` on Microsoft Windows environments.
3. Look at the first few lines of the output from the script to see the JVM vendor and version:

```
*****
* To start WebLogic Server, use a username and *
* password assigned to an admin-level user. For *
* server administration, use the WebLogic Server *
* console at http://hostname:port/console *
*****
starting weblogic with Java version:
java version "1.6.0_29"
Java(TM) SE Runtime Environment (build 1.6.0_29-b11)
Java HotSpot(TM) Client VM (build 20.4-b02, mixed mode)
```

How it works...

The `startWebLogic` script executes the command `java -version` before starting the server, which prints out the JVM version and vendor. There are two JVMs that ship with Oracle SOA Suite 11g. HotSpot is the JVM originally developed by Sun Microsystems, and the descendent of the original Sun JVM. Oracle acquired this JVM with the purchase of Sun Microsystems. The second JVM is JRockit which was acquired by Oracle with its purchase of BEA Systems. JRockit was originally designed as a server-side JVM for application servers, and historically the two JVMs have sought to outperform each other in terms of benchmarks with each release.

There's more...

There are plans to merge the two JVM implementations into a single JVM called **HotRocket**, but work on this is ongoing. For now, you need to decide which JVM best suits your application, and our testing has shown that JRockit is usually faster, but we suggest running your own performance tests against the application to decide for yourself.

See also

- ▶ The *Upgrading to a newer JVM* recipe
- ▶ The *Using the Oracle JRockit JVM* recipe

Using large pages in Linux

Using large pages allows the JVM and the operating system to manage memory more efficiently, having to page memory in and out less frequently.

Getting ready

You need to have Oracle SOA Suite 11g installed on a Linux server for this recipe, and to have enabled **HugePages** in the operating system. This recipe assumes that you are starting WebLogic using the `startWebLogic` or `startManagedWebLogic` scripts, and if you are using the node manager or some other mechanism, simply apply the startup parameter to the relevant place. You will need the file system permissions to edit the WebLogic start script for this recipe.

This recipe assumes you are using the Sun (HotSpot) JVM, and if you are using JRockit, the parameter is different (for more information on this, see the *There's more...* section at the end of this recipe).

How to do it...

To use large pages in Linux, perform the following steps:

1. Check if HugePages support is enabled in the kernel:

```
cat /proc/meminfo | grep Huge
```

You are looking for the following lines:

```
HugePages_Total:      <number of pages>
HugePages_Free:       <number of pages>
Hugepagesize:         <page size, in kB>
```

2. Navigate to the domain's home directory:

```
cd %MIDDLEWARE_HOME%/user_projects/domains/soa_domain/
```

Replace `soa_domain` with the relevant domain as required.

3. Navigate to the bin directory.

```
cd bin
```

4. Edit the `startWebLogic.sh` if you are on a Linux/Unix environment or the `startWebLogic.cmd` if you are on Microsoft Windows.

5. Go to the bottom of the file, and locate the section that starts the JVM:

```
if "%WLS_REDIRECT_LOG%"==" " (
    echo Starting WLS with line:
    echo %JAVA_HOME%\bin\java %JAVA_VM% %MEM_ARGS% -Dweblogic.Name=%SERVER_
%JAVA_HOME%\bin\java %JAVA_VM% %MEM_ARGS% -Dweblogic.Name=%SERVER_NAME%
) else (
    echo Redirecting output from WLS window to %WLS_REDIRECT_LOG%
    %JAVA_HOME%\bin\java %JAVA_VM% %MEM_ARGS% -Dweblogic.Name=%SERVER_NAME%
```

6. Insert the startup arguments `-XX:+UseLargePages` after the `%MEM_ARGS%` or `$MEM_ARGS` parameter, as shown in the following screenshot:

```
if "%WLS_REDIRECT_LOG%"==" " (
    echo Starting WLS with line:
    echo %JAVA_HOME%\bin\java %JAVA_VM% %MEM_ARGS% -XX:+UseLargePages -Dweblogi
%JAVA_HOME%\bin\java %JAVA_VM% %MEM_ARGS% -XX:+UseLargePages -Dweblogic.Nam
) else (
    echo Redirecting output from WLS window to %WLS_REDIRECT_LOG%
    %JAVA_HOME%\bin\java %JAVA_VM% %MEM_ARGS% -XX:+UseLargePages -Dweblogic.Nam
```

7. Save the file.
8. Start WebLogic using the startup scripts already mentioned in the *Getting ready* section.

How it works...

Large or huge pages are a feature of the operating system that allow for more efficient management of memory by the operating system, but to get the best out of the feature, the application allocating the memory, needs to allocate memory to large pages. By setting this flag, you are telling the JVM that you are running on an operating system that supports large pages, and that it should use large pages for its memory management.

If you set this setting on an operating system that does not support large pages, an error is printed and the JVM starts without large page support.

There's more...

You can also set the parameter `-XX:LargePageSizeInBytes=<size>` to set the large page size to the same as the `HugePage` size in Linux. Remember to convert between the value reported by Linux (in KB), and the value used by the JVM in bytes.

If you are using the JRockit JVM, the parameter is `-XlargePages` rather than `-XX:UseLargePages`.

Increasing the number of file descriptors in Linux

As a multiuser operating system, Linux limits the number of file descriptors available to each user, with a default of 1024. As all access to files, sockets, threads, and others requires file descriptors, this limit is too low for a production SOA Suite application. Increasing the limit improves performance and stability.

Getting ready

You will need `root` access to the Linux server on which SOA Suite is installed for this recipe.

How to do it...

To increase the number of file descriptors in Linux, perform the following steps:

1. Connect to the Linux server as the user that runs the SOA Suite.
2. Verify that the current limit is 1024 by using the following command:

```
$ ulimit -n
1024
```

3. Now, log out.
4. Connect to the Linux server as `root`.
5. Edit the file `/etc/security/limits.conf`.
6. Scroll down the file and find the section that looks like this:

```
#<domain>      <type>  <item>          <value>
#
```

7. Add limits for the user that runs your SOA Suite applications like this:

```
#<domain>      <type>  <item>          <value>
#
weblogic        soft    nofile           65535
weblogic        hard    nofile           65535
```

Substitute `weblogic` with the username of the user that runs your SOA Suite server.

8. Save the file.
9. Again, log out.
10. Connect to the Linux server as the user that runs the SOA Suite.
11. Verify that the current limit is now 65535:

```
$ ulimit -n
65535
```

How it works...

Linux uses the `limits.conf` file to set soft (default value) and hard (maximum value) limits for a number of resources that need to be managed in a multiuser environment. The default value of 1024 is very low, and is based on a system where resources need to be carefully managed. Modern servers have many more file descriptors available, and most servers running SOA Suite are not likely to have large number of users logged in and trying to do other things.

The value of 65535 we chose is an example of a large number that should be big enough for our purposes, but any other large value could be used.

Tuning the SOA Suite EJB timeouts

Increasing the timeouts above the default values for a number of key SOA Suite EJBs will reduce the number of timeouts that may occur under heavy load. These timeouts can result in poor performance, as they cause requests to be retried on an already heavily loaded system.

Getting ready

For details on this, refer to the *Getting ready* section of the *Tuning global transaction timeouts* recipe.

How to do it...

To tune the SOA Suite EJB timeouts, perform the following steps:

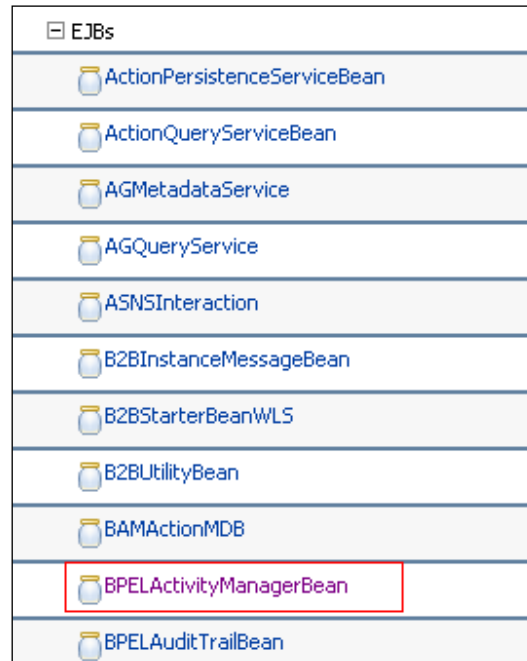
1. Connect to the WebLogic administration console at `http://localhost:7001/console`. Replace `localhost` and `7001` with the hostname and port of the server if it is not running locally on `7001`. If this is the first time you have accessed the WebLogic console since starting the server, WebLogic will first deploy the console application, which may take a few minutes.
2. Log in to the console with your administration credentials.
3. Select **Deployments** in the left-hand navigation pane.



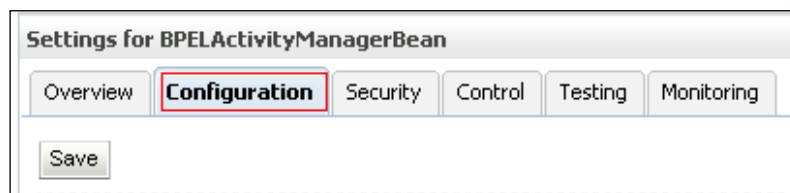
4. In the main pane, locate the **soa-infra** deployment, and click on the **+** icon before it to expand it.

<input type="checkbox"/>	OracleBamAdapter	New	Resource Adapter	329
<input type="checkbox"/>	soa-infra	New	Enterprise Application	350
	Modules			
	/b2b		Web Application	
	/bpm/services		Web Application	
	/integration/services		Web Application	

5. Scroll down the EJBs, and select the EJB called `BPELActivityManagerBean`.



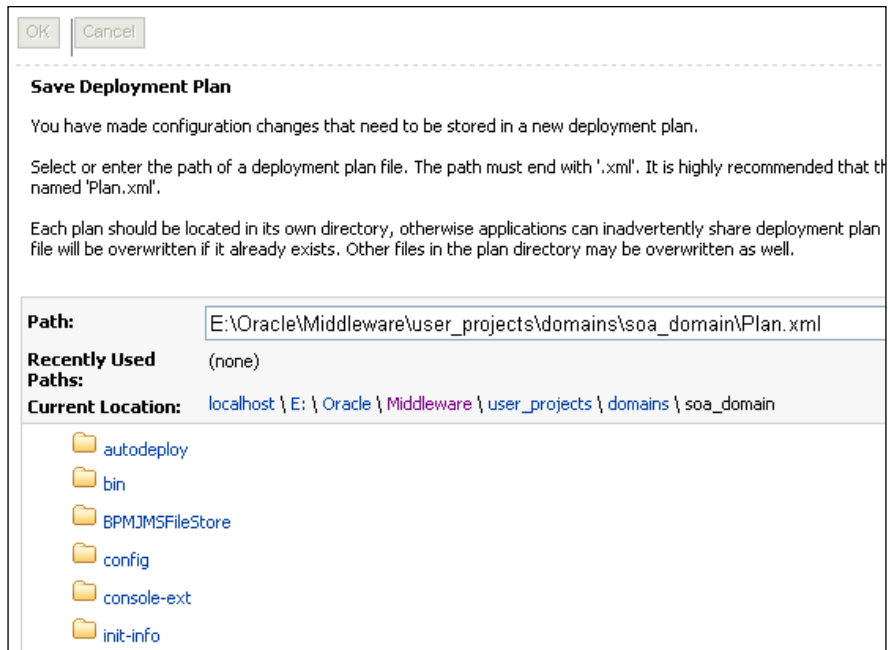
6. Click on the bean, and then select the **Configuration** tab.



7. Scroll down to the bottom of the properties, and increase the Transaction Timeout from 300 to 600 seconds.



8. Click on **Save**.
9. If this is the first bean you have changed the timeout for, you will be prompted to create a deployment plan. We suggest creating it in the domain's `home` directory.



The dialog box titled "Save Deployment Plan" contains the following elements:

- Buttons: OK, Cancel
- Text: "You have made configuration changes that need to be stored in a new deployment plan."
- Text: "Select or enter the path of a deployment plan file. The path must end with '.xml'. It is highly recommended that the file be named 'Plan.xml'."
- Text: "Each plan should be located in its own directory, otherwise applications can inadvertently share deployment plan files. Files in the plan directory may be overwritten as well."
- Path field: E:\Oracle\Middleware\user_projects\domains\soa_domain\Plan.xml
- Recently Used Paths: (none)
- Current Location: localhost \ E: \ Oracle \ Middleware \ user_projects \ domains \ soa_domain
- Directory list:
 - autodeploy
 - bin
 - BPMJMSFileStore
 - config
 - console-ext
 - init-info

10. Once you have selected the location for the deployment plan, click on **OK**.
11. Repeat the steps 3 through 8 for the following EJBs:
 - ☐ BPELServerManagerBean
 - ☐ BPELDeliveryBean
 - ☐ BPELInstanceManagerBean
 - ☐ BPELEngineBean
 - ☐ BPELProcessManagerBean
 - ☐ BPELDispatcherBean
 - ☐ BPELSensorValuesBean
 - ☐ BPELFinderBean

How it works...

Under heavy load, some of these EJBs can take a while to respond, and may eventually time out. If the request times out, it is likely that the originator of the request will retry, increasing load on an already heavily loaded system. By increasing the transaction timeout for these beans, you increase the amount of time that they will wait before a timeout occurs, giving them longer time to complete their task.

There's more...

Our value of 600 is just a starting point, if you see timeouts occurring on these beans, you might want to increase the value further above 600.

Upgrading to a newer JVM

Each new JVM release usually includes a large number of performance improvements, as the vendors constantly vie for the top spot on the performance benchmarks. By upgrading to the latest available JVM, you can take advantage of the latest performance improvements.

Getting ready

For this recipe you need to have installed SOA Suite, and have edit access to the start scripts for your domain. You also need to have downloaded and installed a newer JVM (the JDK version, not the JRE).

This recipe assumes that you are starting WebLogic using the scripts in the domain `bin` directory. If you are using another method such as the node manager, you should make the changes to `JAVA_HOME` via the relevant method (for the node manager, this is the WebLogic administration console).

How to do it...

To upgrade to a newer JVM, perform the following steps:

1. Navigate to the domain's `bin` directory, as shown in the following command line:

```
cd %MIDDLEWARE_HOME%/user_projects/soa_domain/bin
```
2. Edit the file `setDomainEnv.cmd` or `setDomainEnv.sh`. If you are using a newer version of HotSpot, add the line `set JAVA_VENDOR=Sun` or `set JAVA_VENDOR=Sun`, depending on whether you are on Microsoft Windows or Linux, respectively. If you are using a newer release of JRockit, you should set the Java vendor to Oracle.

3. Comment out the line `set SUN_JAVA_HOME=...` (or the line `set ORACLE_JAVA_HOME=...` if you are using JRockit). If you are using Linux, the line is just `SUN_JAVA_HOME=...`.
4. Add a line `set SUN_JAVA_HOME=<JDK path>` where `<JDK path>` is the fully qualified pathname of your JDK. If the path contains spaces, then this must be in quotes as shown.

```

setDomainEnv.cmd - Notepad
File Edit Format View Help
@REM (http://download.oracle.com/docs/cd/E23943_01/we
@REM
set JAVA_VENDOR=Sun
set COMMON_COMPONENTS_HOME=E:\oracle\middleware\oracle_11.1.1\server\bin
for %%i in ("%COMMON_COMPONENTS_HOME%") do set COMMON_COMPONENTS_HOME=%%i

set WL_HOME=E:\oracle\middleware\wlserver_10.3
for %%i in ("%WL_HOME%") do set WL_HOME=%%i

set BEA_JAVA_HOME=E:\oracle\middleware\jrockit_160_29

set SUN_JAVA_HOME=E:\oracle\middleware\jdk160_29
set SUN_JAVA_HOME=E:\java

set UMS_ORACLE_HOME=E:\oracle\middleware\oracle_soa1

set SOA_ORACLE_HOME=E:\oracle\middleware\oracle_soa1

```

5. Save the file.
6. Start WebLogic by running `startWebLogic.sh` or `startWebLogic.cmd`.
7. Check that the output lists the new Java version.

```

C:\WINDOWS\system32\cmd.exe - startWebLogic.cmd
e:\win\32;E:\oracle\MIDDLE~1\WLSEU~1.3\server\bin;e:\oracle\MIDDLE~1\modules\ORG
APA~1.1\bin;E:\java\jre\bin;E:\java\bin;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOW
S\System32\Wbem;E:\oracle\MIDDLE~1\WLSEU~1.3\server\native\win\32\oci920_8;E:\O
racle\middleware\Oracle_SOA1\soa\thirdparty\edifecs\XEngine\bin
*****
* To start WebLogic Server, use a username and *
* password assigned to an admin-level user. For *
* server administration, use the WebLogic Server *
* console at http://hostname:port/console *
*****
starting weblogic with Java version:
java version "1.7.0_07"
Java(TM) SE Runtime Environment (build 1.7.0_07-b11)
Java HotSpot(TM) Client VM (build 23.3-b01, mixed mode, sharing)
Starting WLS with line:
E:\java\bin\java -client -Xms512m -Xmx1024m -XX:PermSize=128m -XX:MaxPermSize=
512m -Dweblogic.Name=AdminServer -Djava.security.policy=E:\oracle\MIDDLE~1\WLSEU
~1.3\server\lib\weblogic.policy -Xverify:none -da -Dplatform.home=E:\oracle\M
IDDLE~1\WLSEU~1.3 -Dwls.home=E:\oracle\MIDDLE~1\WLSEU~1.3\server -Dweblogic.ho
me=E:\oracle\MIDDLE~1\WLSEU~1.3\server -Dcommon.components.home=E:\oracle\MIDD
LE~1\ORACLE~1 -Djrf.version=11.1.1 -Dorg.apache.commons.logging.Log=org.apache.c
ommons.logging.impl.Jdk14Logger -Ddomain.home=E:\oracle\MIDDLE~1\USER_P~1\domain
s\SOA_001 -Djrockit.optfile=E:\oracle\MIDDLE~1\ORACLE~1\modules\oracle.jrf_11.1
\jrockit\optfile.txt -Doracle.server.config.dir=E:\oracle\MIDDLE~1\USER_P~1\do

```

How it works...

The `setDomainEnv` script sets up a number of parameters that are used when starting the WebLogic servers, including the JDK that is used. By default this script selects between one of the two JDKs installed with the SOA Suite. By changing the value of `SUN_JAVA_HOME` you can point to a different HotSpot JDK, and the server will use this to start up. It is important to set the value of `JAVA_VENDOR` correctly because HotSpot and JRockit have a number of different parameters that are passed in by the script.

You could also use a newer version of JRockit, if you set the `JAVA_VENDOR` and `ORACLE_JAVA_HOME` variables appropriately.

There's more...

This file is created by the domain creation wizard, so if you make any changes to the domain with the wizard after you have edited the file, changes to the file will be lost.

While Oracle will only provide support if you are running on one of their supported JVMs, there is nothing that stops you from using a newer JVM in production, you will just have to replicate issues using an older JVM before raising support cases with Oracle.

See also

- ▶ The *Using the Oracle JRockit JVM* recipe
- ▶ The *Finding out which JVM you are using* recipe

Setting the Linux kernel swappiness to low

The swappiness level of the Linux kernel determines how likely it is to swap memory pages out. Setting a low value can improve the performance of Oracle SOA Suite.

Getting ready

You will need to be running Oracle SOA Suite 11g on a Linux box, and have `root` or `sudo` access in order to change the kernel settings.

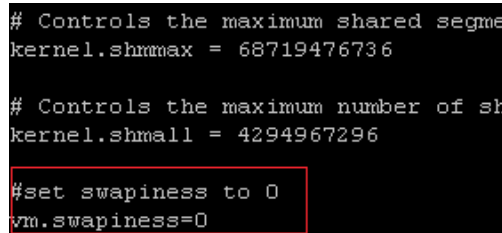
How to do it...

To set the Linux kernel swappiness to low, perform the following steps:

1. Log in to the box as root.
2. Check the current swappiness setting as follows:

```
$ cat /proc/sys/vm/swappiness
```

```
60
```
3. Edit the file `/etc/sysctl.conf`, and set the property `vm.swappiness` to a lower value. We suggest starting with 0 if you have plenty of physical memory on the machine.



```
# Controls the maximum shared segment size in bytes
kernel.shmmax = 68719476736

# Controls the maximum number of shared memory segments
kernel.shmall = 4294967296

#set swappiness to 0
vm.swappiness=0
```

4. Save the file.
5. Restart your Linux server for the change to take effect.
6. Check if the swappiness setting has been applied, like this:

```
$ cat /proc/sys/vm/swappiness
```

```
0
```

How it works...

The `vm.swappiness` property of the Linux kernel determines how likely the kernel is to swap a page out of memory. Swappiness is set as a value between 0 and 100, with 0 being a strong preference to avoid swapping pages out, and 100 being a strong preference to swap a page out whenever possible. The memory that is freed up by swapping pages out can be used by the kernel for a number of things. If there are multiple applications running on the system, and the memory is oversubscribed, it can be allocated to another process; otherwise it can be used to cache files in order to improve the file system performance. When a memory page is swapped out by the kernel, it must be swapped back in before it can be accessed by an application, and so the application must wait while the operating system performs this task. Java Application Servers generally run on dedicated servers, which have sufficient physical (or virtual) memory to allow the operating system and the JVM to exist in the memory at the same time without any pages being swapped out. The best performance is usually achieved by configuring a low swappiness value, preferring that existing pages are kept in the memory wherever possible.

See also

- ▶ The *Increasing the number of file descriptors in Linux* recipe
- ▶ The *Using large pages in Linux* recipe

Using the Oracle JRockit JVM

JRockit is a JVM specifically designed for high performance application servers, and can often deliver better performance than HotSpot. This recipe explains how to switch to the JRockit JVM if you are not already using it.

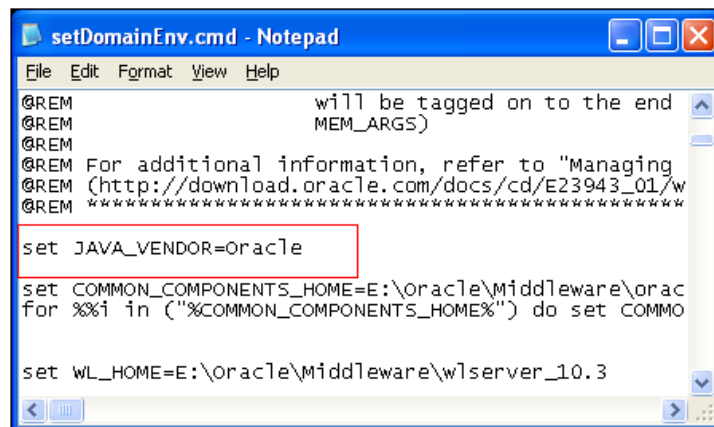
Getting ready

This recipe assumes that you are starting WebLogic using the `startWebLogic` or `startManagedWebLogic` scripts, and if you are using the node manager or some other mechanism, you will need to specify the new JVM home location through the console or another mechanism.

How to do it...

To use the Oracle JRockit JVM, perform the following steps:

1. Navigate to the domain `bin` directory, as shown in the following command line:
`cd %MIDDLEWARE_HOME%/user_projects/soa_domain/bin`
2. Edit the file `setDomainEnv.cmd` or `setDomainEnv.sh`, add the line `set JAVA_VENDOR=Oracle` or `set JAVA_VENDOR=Oracle`, depending on whether you are on Microsoft Windows or Linux.



3. Save the file.
4. Start WebLogic by running `startWebLogic.sh` or `startWebLogic.cmd`.
5. Check that the output lists JRockit as the JVM implementation (highlighted in the following screenshot):

```

Mark C:\WINDOWS\system32\cmd.exe - startWebLogic.cmd
\npatch_ocp371\profiles\default\native;E:\Oracle\MIDDLE~1\WLSEU~1.3\server\nativ
e\win32;E:\Oracle\MIDDLE~1\WLSEU~1.3\server\bin;e:\Oracle\MIDDLE~1\modules\ORG
APA~1.1\bin;e:\Oracle\MIDDLE~1\JROCKI~1.0-1\jre\bin;e:\Oracle\MIDDLE~1\JROCKI~1.
0-1\bin;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;E:\Oracle\MIDDLE
~1\WLSEU~1.3\server\native\win32\oci920_8;E:\Oracle\Middleware\Oracle_SOA1\soa
\thirdparty\edifecs\XEngine\bin
*****
* To start WebLogic Server, use a username and *
* password assigned to an admin-level user. For *
* server administration, use the WebLogic Server *
* console at http://hostname:port/console *
*****
starting weblogic with java version:
java version "1.6.0_29"
Java(TM) SE Runtime Environment (build 1.6.0_29-b11)
Oracle JRockit(R) (build R28.2.0-79-146777-1.6.0_29-20111005-1808-windows-ia32,
compiled mode)
Starting WLS with line:
e:\Oracle\MIDDLE~1\JROCKI~1.0-1\bin\java -jrockit -Xms512m -Xmx1024m -Dweblogi
c.Name=AdminServer -Djava.security.policy=E:\Oracle\MIDDLE~1\WLSEU~1.3\server\l
ib\weblogic.policy -Xverify:none -da -Dplatform.home=E:\Oracle\MIDDLE~1\WLSEU
~1.3 -Dwls.home=E:\Oracle\MIDDLE~1\WLSEU~1.3\server -Dweblogic.home=E:\Oracle\M
IDDLE~1\WLSEU~1.3\server -Dcommon.components.home=E:\Oracle\MIDDLE~1\ORACLE~1

```

How it works...

The `setDomainEnv` script sets up a number of parameters that are used to start WebLogic, including selecting between the two JVMs that ship with SOA Suite 11g. The script checks for the presence of a variable called `JAVA_VENDOR`. If it is set to the value `Oracle`, then JRockit is used. If the value is not `Oracle`, or the variable is not set, then the variable is set to `Sun`, and HotSpot is used.

There's more...

This file is created by the domain creation wizard, so if you make any changes to the domain with the wizard after you have edited the file, changes to the file will be lost.

There are a number of other ways that you could achieve the same thing. You could set the `JAVA_VENDOR` variable in your shell before calling the `startWebLogic` script, or you could directly set the `JAVA_HOME` variable either in the `setDomainEnv` script or in your shell.

See also

- ▶ The *Upgrading to a newer JVM* recipe
- ▶ The *Finding out which JVM you are using* recipe

Running your domain in the production mode

WebLogic domains can run in either the production or the development mode. The development mode is optimized for environments where application development is taking place and resources are scarcer. The production mode is optimized for running in an operational environment with more resources and higher performance requirements.

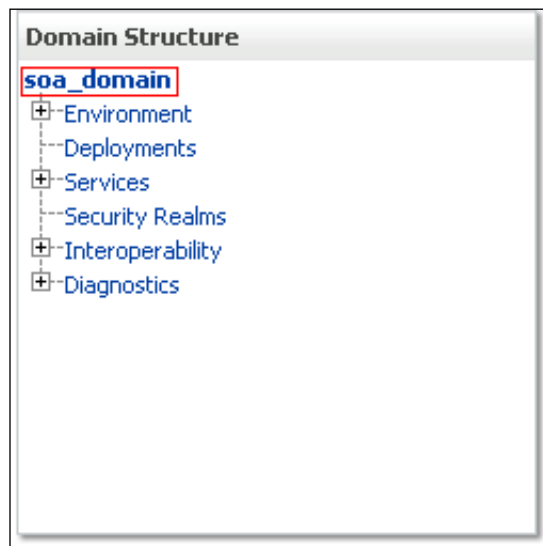
Getting ready

For details on this, refer to the *Getting ready* section of the *Tuning global transaction timeouts* recipe.

How to do it...

To run your domain in the production mode, perform the following steps:

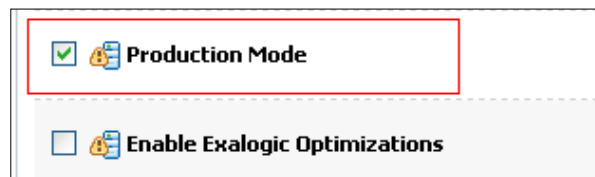
1. Connect to the WebLogic administration console at `http://localhost:7001/console`. Replace `localhost` and `7001` with the hostname and port of the server, if it is not running locally on `7001`. If this is the first time you have accessed the WebLogic console since starting the server, WebLogic will first deploy the console application, which may take a few minutes.
2. Log in to the console with your administration credentials.
3. Click on the name of your domain in the left-hand navigation pane.



4. Ensure that the **General** tab under the **Configuration** tab is selected.



5. Scroll down to the property **Production Mode**, and ensure that it is checked.



6. Scroll to the top of the page and click on **Save**.
7. Restart your WebLogic servers.

How it works...

Changing the production mode settings affects all the servers in the domain, and needs the admin server to be restarted to pick up the change. You may find that once you apply the production mode, you need to enter the WebLogic administration username and password to start the server. You can prevent this by creating a `boot.properties` file (described in the *Creating a boot.properties file* recipe).

In the production mode there are a number of things that change their default values, including the following:

- ▶ The default JDBC connection pool size is increased from 15 to 25
- ▶ The default JVM is JRockit
- ▶ Hot deployment is disabled
- ▶ Server log file rotation is enabled
- ▶ Using demo SSL certificates logs a warning



For more details on the production mode, see the Oracle WebLogic documentation at http://docs.oracle.com/cd/E23943_01/web.1111/e13814/wls_tuning.htm#PERFM177.

There's more...

Production mode changes the default values for a number of settings, but we talk about overriding many of them individually in other recipes in this book. There are some settings, such as hot deployment, that cannot be individually turned off, and therefore can only be changed by switching to the production mode.

While production mode selects JRockit as the default JVM, you can override this if you wish to use Hotspot or a different JVM. As usual, we recommend benchmarking with a number of different JVM versions and implementations to find out which one works best for you.

Once the production mode is enabled on a domain, it cannot be disabled from the console, and you will need to directly edit the `config.xml` file if you want to change a domain in the production mode back to the development mode.

See also

- ▶ The *Creating a boot.properties file* recipe
- ▶ The *Setting the data source pool sizes* recipe in *Chapter 6, Data Sources and JMS*
- ▶ The *Upgrading to a newer JVM* recipe
- ▶ The *Using the Oracle JRockit JVM* recipe

Creating a boot.properties file

In the production mode, the server will prompt for your administration username and password when you start the server. While this does not directly impact performance, it means that server restarts cannot be performed automatically, possibly reducing the number of running servers and impacting performance that way. This recipe looks at how you can create a `boot.properties` file which will allow servers to be restarted without prompts.

Getting ready

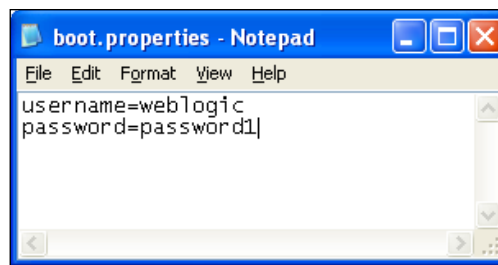
You will need to be able to log on to the host running your WebLogic servers as the user that owns the domain files.

How to do it...

To create a `boot.properties` file, perform the following steps:

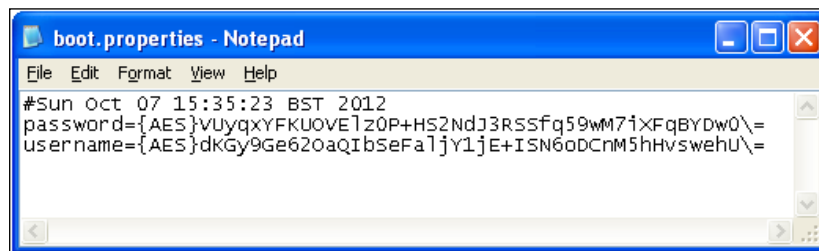
1. Navigate to the directory `%DOMAIN_HOME%/servers/AdminServer/security` as shown in the the following command line:

```
cd %DOMAIN_HOME%/servers/AdminServer/security
```
2. Create a new file called `boot.properties`, with two properties defined, username and password.



3. Save the file.
4. Start WebLogic using the following command:

```
cd %DOMAIN_HOME%/  
./startWebLogic.sh
```
5. Check that the `boot.properties` file has been updated with the encrypted values.



How it works...

In the production mode, WebLogic will check for a file called `boot.properties` in the `%DOMAIN_HOME%/servers/AdminServer/security` directory. If this file exists, it will read two properties from the file, username and password. This username and password is used to start the admin server. If the file content is plain text, then once the server has been started, the values are encrypted using the domain encryption credentials, and saved back to the file.

There's more...

While the file is updated with the encrypted values, this should not be considered as secure, as it is likely that if a user can read the `boot.properties` file, they can also read the files that contain the decryption keys. It therefore provides little in the way of actual security. This, unfortunately, is the "chicken and egg" scenario of the automated server restarts—if the server is to be started automatically, then all the material that it needs to start, including encryption key material, has to be available to the process starting the server.

See also

- ▶ The *Running your domain in the production mode* recipe

7

Data Sources and JMS

This chapter looks at some of the things you can do to tune the SOA Suite data sources, backing a database, and JMS resources. We will look at the following topics:

- ▶ Setting the data source pool sizes
- ▶ Configuring data source testing
- ▶ Configuring data source growing and shrinking
- ▶ Setting data source connection timeouts
- ▶ Tuning database XA timeouts
- ▶ Tuning connections in the native EIS database adapter
- ▶ Configuring the WebLogic thread pool
- ▶ Using JMS file persistence
- ▶ Tuning JMS connection factories

Introduction

In this chapter, we'll look at the aspects of SOA Suite that can be tuned when interacting with the backing database. Recipes for tuning the database itself can be found in *Chapter 8, BPEL and BPMN Engine Tuning*.

As we scale SOA Suite to cope with application load, it is likely that the underlying database will become a performance bottleneck. SOA Suite makes heavy use of the database, storing both the metadata for process instances as well as their payloads. Much of the tuning for the BPEL, BPMN, Mediator, and other components, which we discuss in later recipes, is based around reducing the amount of database I/O that happens. The primary use of the database by SOA Suite is for its "Dehydration Store," which is a phrase used to describe how process instances can be persisted to the database in order to be loaded at a later time. This can be done for a number of reasons: it allows process instances to persist across restarts of the server, and can allow long-running processes that are waiting for responses to be removed from memory, and re-loaded when the response comes back. This concept applies across all the components in SOA Suite (BPM, BPEL, Rules, Workflow, and Mediator) and so can be a source of contention for busy applications. In this chapter, we are focusing on making sure that when SOA Suite does need to connect to the database, it can do so quickly and efficiently. Large amounts of BPEL process execution for an instance can generate a large I/O demand on the database tables that back SOA Suite.

We'll also look in this chapter at the use of WebLogic JMS by SOA Suite, but we'll only be scratching the surface here. WebLogic JMS is a wide-ranging subject with lots of options; we'll focus on some important ones for using SOA Suite.

Setting the data source pool sizes

Datasource pooling is a critical feature in Java containers, allowing resource sharing among deployed components and applications. This recipe will show you how to control the pool sizes in WebLogic.

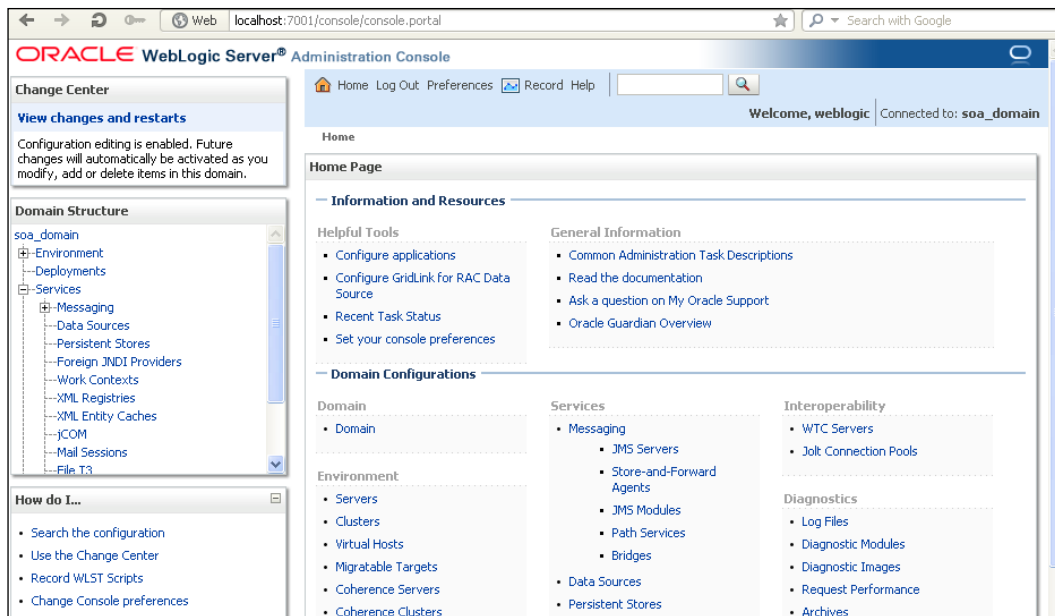
Getting ready

You will need to have Oracle SOA Suite 11g installed on a server for this recipe, as we'll be accessing the default data source used by SOA Suite.

How to do it...

Follow these steps to set your datasource pool sizes:

1. Open the SOA Suite WebLogic administration console and log in as administrator. If you have issues locating the console, see steps 1 and 2 in the *Tuning global transactions* recipe in *Chapter 6, Platform Tuning*. A default administration screen is shown here:



2. Navigate to the **Data Sources** section by performing either of the following actions:
 - ❑ Click on the **Data Sources** link under the **Services** drop-down list in the **Domain Structure** box located on the left of the screen
 - ❑ In the **Home Page** box in the center of the screen, under **Services** in **Domain Configurations**, select **Data Sources**
3. In the **Data Sources** screen, select **SOADataSource**. Within the **Connection Pool** tab, locate the pool capacity section.

Initial Capacity:	<input type="text" value="0"/>
Maximum Capacity:	<input type="text" value="50"/>
Minimum Capacity:	<input type="text" value="0"/>
Statement Cache Type:	<input type="text" value="LRU"/>
Statement Cache Size:	<input type="text" value="10"/>

4. Reduce the waiting time for new connection requests, by setting the **Initial Capacity:** and **Minimum Capacity:** fields to a value that is suitable for our database. A good example starting value is 20.
5. The **Maximum Capacity:** field controls the maximum number of connections this pool can grow to. In other words, the maximum number of concurrent connections that can be sustained by SOA Suite to the database. As this will be used by multiple threads, a good starting value is 50, if our database can support it.
6. Click on the **Save** button located at the bottom of the page.

How it works...

Pooling, sharing, and reuse of database connections is a powerful feature of Java Application Servers. However, the default settings for WebLogic are to values that are quite low. Under heavy load, we can expect the datasource pools to reach their maximum capacity fairly quickly. We set the **Maximum Capacity:** to a higher value to accommodate this.

To prevent cold startup delays on new initial-connection requests, we set the **Minimum Capacity:** value. Be careful with the value in the **Initial Capacity:** field as WebLogic will allocate these many connections to the database as soon as it boots; this can overload busy production databases with connection requests if many servers start up in a short space of time.

Note that the datasource pool sizes will apply to each running instance of WebLogic, so ensure that your database can stand up to a number of connections equal to:

$$(\text{Pool Maximum Capacity}) \times (\text{Number of Targets that the data source is applied to})$$

There's more...

There are many data sources in use by SOA Suite; the use of these vary, depending on the components you are utilizing in your application.

The tuning options available to you by default will yield maximum effect on the `SOADatasource`, `SOATxDatasource`, and `mds-soa` data sources. It is best to monitor the data sources to determine the options that are right for your usage footprint. For detailed information on data-source monitoring, see the *Monitoring the data source usage* recipe in *Chapter 2, Monitoring Oracle SOA Suite*. It is also possible to get a basic monitoring of the datasource pool size by clicking on the **Monitoring** tab in the WebLogic console, when on the page for a data source. By customizing the table, you can add a number of columns that give detailed monitoring information about the data source.

Also, controlling the size of the prepared statement cache can increase throughput on the database connections—a big win if you're performing lots of hydration and dehydration via the use of BPEL, for instance.

See also

- ▶ The *Tuning global transactions* recipe in *Chapter 6, Platform Tuning*
- ▶ The *Configuring data source testing* recipe
- ▶ The *Configuring data source growing and shrinking* recipe
- ▶ The *Setting data source connection timeouts* recipe
- ▶ The *Monitoring the data source usage* recipe in *Chapter 2, Monitoring Oracle SOA Suite*

Configuring data source testing

Database connections that are pooled can be configured to be tested by WebLogic as part of the managed-connection lifecycle. This recipe will show you how to configure testing of connections.

Getting ready

For this recipe, you will need to have Oracle SOA Suite 11g installed on a server as we'll be accessing the default data source used by SOA Suite.

How to do it...

By following the steps given here, we can configure data-source testing:

1. Open the SOA Suite WebLogic administration console and log in as administrator. If you have issues locating the console, see steps 1 and 2 in the *Tuning global transactions* recipe in *Chapter 6, Platform Tuning*.
2. Navigate to the **Data Sources** section by performing either of the following actions:
 - Click on the **Data Sources** link under the **Services** drop-down list in the **Domain Structure** box located on the left of the screen
 - In the **Home Page** box in the center of the screen, under **Services** in **Domain Configurations**, select **Data Sources**
3. In the **Data Sources** screen, select **SOADDataSource**. Within the **Connection Pool** tab, click on the **Advanced** link at the bottom of the page.

- Click on the **Advanced** option to display the advanced options:

Advanced

☒ Test Connections On Reserve

Test Frequency: 300

Test Table Name:
SQL SELECT 1 FROM DUAL

Seconds to Trust an Idle Pool Connection: 0

- We need to set the **Test Table Name:** field to an appropriate test table in our database. `SQL SELECT 1 FROM DUAL` is valid for Oracle.
- With **Test Table Name** set, we can now enable connection testing by ticking the **Test Connections on Reserve** checkbox.
- Set **Test Frequency:** to a suitable value, such as 300 seconds.
- Click on the **Save** button at located the bottom of the page.

How it works...

Connection testing allows the application server to validate that connections (that are created to the database) are valid and that they remain valid after a period of time.

Before adding database connections to the data-source connection pool, they are tested by executing the SQL we entered into the **Test Table Name** box. It should go without saying that this will add extra overhead on the creation of connections, both when the pool is first created and when growing its number of connections.

Setting the **Test Frequency** period adds an interval after which idle pool connections are tested. This helps to ensure that we do not have stale connections in the pool, which we could potentially allocate to a requested SOA Suite application thread. You can disable this with the value of 0 seconds.

Note that connections that fail the test will either not be added to the pool or will be removed from the pool, depending on whether this is a creation or frequency test. If connections continually fail then the pool will enter a suspended state, requiring administrative intervention.

Testing our connections adds to the overhead on the database; it must satisfy test queries in conjunction with concurrent production load.

There's more...

The **Test Connections On Reserve** value can also be set to ensure that connections are tested prior to handing them over to application threads, but adds even more query-based overhead on the database!

Another setting is available to further optimize testing of connections: setting **Seconds to Trust an Idle Pool Connection** to a nonzero value creates a window for the duration for which the connection will be trusted after a connection test passes. This enables periodic checks, and checks on reserve to be skipped prior to handing a connection to a requesting thread.

Configuring data source growing and shrinking

Data sources can dynamically alter their size to cope with changes in application load. This recipe will show you how to alter these settings.

Getting ready

You will need to have Oracle SOA Suite 11g installed on a server as we'll be accessing the default data source used by SOA Suite.

How to do it...

We can configure how data sources grow and shrink, by performing the following steps:

1. Open the SOA Suite WebLogic administration console and log in as administrator. If you have issues locating the console, see steps 1 and 2 in the *Tuning Global Transactions* recipe in *Chapter 6, Platform Tuning*.

2. Navigate to the **Data Sources** section by performing either of the following actions:
 - ❑ Click on the **Data Sources** link under the **Services** drop-down list in the **Domain Structure** box located on the left of the screen
 - ❑ In the **Home Page** box in the center of the screen, under **Services** in **Domain Configurations**, select **Data Sources**
3. In the **Data Sources** screen, select **SOADDataSource**. Within the **Connection Pool** tab, locate the pool capacity section.
4. Click on the **Advanced** option to display the advanced options.
5. This section contains two settings that are of interest for this recipe, **Maximum Capacity** and **Shrink Frequency**:

Maximum Capacity:	<input type="text" value="50"/>
Shrink Frequency:	<input type="text" value="900"/>

6. Setting **Maximum Capacity** controls the maximum number of connections that WebLogic will allow to exist for this connection pool.
7. Set **Shrink Frequency** to a non-zero value such as 300 to shrink a pool that has expanded.
8. Click on the **Save** button located at the bottom of the page.

How it works...

The data source's connection pool will grow in size incrementally from the values in **Initial Capacity** / **Minimum Capacity**, in response to incoming demand. We set the ceiling for this growth, using the **Maximum Capacity** field. Note that there is no way to control the rate at which connections are created, or to specify a step value to create a fixed number of connections at a time.

When the load being applied to your application falls, the number of connections will stay at the maximum number created until **Shrink Frequency** expires. This time interval begins from the time the pool first incrementally increases beyond the number specified in **Initial Capacity** / **Minimum Capacity**. It will repeat periodically until the pool reaches the value in **Minimum Capacity**.

Setting data source connection timeouts

WebLogic data sources' pools can be configured to prevent "leakage"—the process whereby connections are perceived as being in use when in reality the thread has stopped using it. Over time, this will cause the pool to be exhausted. In this case, it is said to have leaked connections!

This recipe will show how to configure WebLogic to mitigate against this scenario and reclaim these leaked connections.

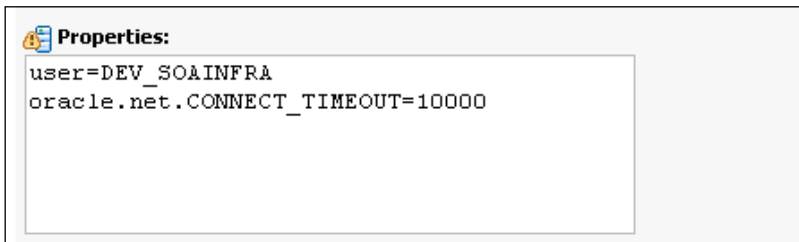
Getting ready

You will need to have Oracle SOA Suite 11g installed on a server as we'll be accessing the default data source used by SOA Suite.

How to do it...

The following steps will allow us to set the data-source connection timeouts:

1. Open the SOA Suite WebLogic administration console and log in as administrator. If you have issues locating the console, see steps 1 and 2 in the *Tuning Global Transactions* recipe in *Chapter 6, Platform Tuning*.
2. Navigate to the Data Sources section by performing either of the following actions:
 - ❑ Click on the **Data Sources** link under the **Services** drop-down list in the **Domain Structure** box located on the left of the screen
 - ❑ In the **Home Page** box in the center of the screen, under **Services** in **Domain Configurations**, select **Data Sources**
3. In the **Data Sources** screen, select **SOADDataSource**. Within the **Connection Pool** tab, locate the **Properties:** box:



4. The value `oracle.net.CONNECT_TIMEOUT` is configurable to set the timeout on the JDBC driver in milliseconds. A value of 10 seconds has been set here, which is quite low. A value of 30 seconds allows a little more grace on busy production systems.

5. Still within the **Connection Pool** tab, click on the **Advanced** link located at the bottom of the page. Locate the **Inactive Connection Timeout:** box:

Inactive Connection Timeout:	<input type="text" value="0"/>
-------------------------------------	--------------------------------

6. Set the **Inactive Connection Timeout:** value to 600. WebLogic will return inactive connections to the pool if idle for 600 seconds.

How it works...

Setting the **Inactive Connection Timeout:** value acts as a protection against applications that might leak connections. Should an application not call the `close` method on their database connection, perhaps because they have thrown an exception, this timeout will cause the connection to be closed by WebLogic and returned to the connection pool.

Note that setting **Inactive Connection Timeout:** to a low value will potentially cause WebLogic to reclaim connections that are in use by an application thread, which will have a destructive impact on the thread (which expects an open connection). A good starting point is a value of 10 minutes; this provides some insulation for long-running in-memory processes, while still enabling resource clean-up.

Tuning database XA timeouts

Database connections that are pooled can be configured to enable distributed transactions, with an associated timeout if other resources in the transaction do not complete in time. This recipe will show you how to set and configure this.

Getting ready

You will need to have Oracle SOA Suite 11g installed on a server as we'll be accessing the default data source used by SOA Suite.

We'll set XA timeouts on the default SOA datasource.

How to do it...

Follow the steps given here to tune XA timeouts:

1. Open the SOA Suite WebLogic administration console and log in as an administrator. If you have issues locating the console, see steps 1 and 2 in the *Tuning global transactions* recipe in *Chapter 6, Platform Tuning*.

2. Navigate to the **Data Sources** section by performing either of the following actions:
 - ❑ Click on the **Data Sources** link under the **Services** drop-down list in the **Domain Structure** box located on the left of the screen
 - ❑ In the **Home Page** box in the center of the screen, under **Services** in **Domain Configurations**, select **Data Sources**
3. In the **Data Sources** screen, select **SOADDataSource**. Within the **Transaction** tab, check the **Set XA Transaction Timeout** checkbox and enter a value of **3600** (or a value determined by your testing) in the timeout value box:

The screenshot shows the 'Settings for SOADDataSource' window with the 'Transaction' tab selected. The 'Save' button is at the top left. Below it, there is explanatory text about transaction protocols. Two checkboxes are checked: 'Use XA Data Source Interface' and 'Set XA Transaction Timeout'. At the bottom, the 'XA Transaction Timeout' is set to 3600.

Settings for SOADDataSource

Configuration Targets Monitoring Control Security

General Connection Pool Oracle ONS **Transaction**

Save

The transaction protocol for a JDBC data source determines how c
handled during transaction processing. Transactions within a JDBC
global (local).

This page enables you to define transaction options for this JDBC c

☒ **Use XA Data Source Interface**

☒ **Set XA Transaction Timeout**

XA Transaction Timeout:

4. Click on **Save** and restart the server for the changes to take effect.

How it works...

An XA transaction is a transaction that involves a number of different transactional resources, where those resources need to work together to complete the work. For example, it might involve multiple databases, or a database and a JMS queue. The XA transaction timeout determines how long WebLogic will wait for all of those resources to complete their work before timing out and rolling back the transaction. In this recipe, we have increased the timeout to a higher value, allowing resources more time to complete their work before the timeout occurs and the transaction rolls back. It might also be the case that you would want to reduce the timeout to cause the transaction to rollback faster. This scenario is not particularly common though, because rolling back the transaction usually returns an error to the calling system, which will often immediately retry in case it's a transient error. We suggest that you experiment with various values to find the one that suits your system best. Remember that you need to set the Oracle database settings to matching values, for this to work correctly.

There's more...

In order for this mechanism to not fail in your environment, you will need to ensure that the database's distributed timeout value (known as `distributed_lock_timeout`) is greater than the timeout set in WebLogic's XA timeout value, otherwise the database will timeout and rollback our transactions while WebLogic is in the middle of utilizing them. To view the value on our Oracle database, we can run (or ask the database administrator to run) the following command:

```
SQL>SHOW PARAMETER DISTRIBUTE_LOCK_TIMEOUT;
```

NAME	TYPE	VALUE
distributed_lock_timeout	integer	60

We can update this value using the following command:

```
SQL>ALTER SYSTEM SET DISTRIBUTE_LOCK_TIMEOUT=3600 SCOPE=SPFILE;
```

And then restart the database for the changes to take effect:

```
SQL>SHUTDOWN TRANSACTIONAL
```

```
SQL>STARTUP
```

```
...
```

```
SQL>SHOW PARAMETER DISTRIBUTE_LOCK_TIMEOUT;
```

NAME	TYPE	VALUE
distributed_lock_timeout	integer	3600

Ensure that this is performed on each node that houses your SOA Suite database instances (such as RAC cluster nodes).

We can set this for the other SOA Suite data sources as we need them to participate in XA transactions, such as `EDNDataSource` and `mds-soa`, but this really should be done based on the system usage. We recommend monitoring the database usage of these items for XA resource issues.

Tuning connections in the native EIS database adapter

SOA Suite enables interaction with database tables in BPEL processes, using database adapters. In this recipe, we will show you how to tune the stock `DbAdapter`.

Getting ready

You will need to have Oracle SOA Suite 11g installed on a server as we'll be tuning the default `DbAdapter` deployed to WebLogic when installing SOA Suite.

How to do it...

These steps show us how to tune the connections in the EIS adapter:

1. Open the SOA Suite WebLogic administration console and log in as administrator. If you have issues locating the console, see steps 1 and 2 in the *Tuning global transactions* recipe in *Chapter 6, Platform Tuning*.
2. Navigate to the **Deployments** section by performing either of the following actions:
 - Click on the **Deployments** link in the **Domain Structure** box located on the left of the screen
 - In the **Home Page** box in the center of the screen, under **Your Deployed Resources** in **Domain Configurations**, select **Deployments**
3. Select the **DbAdapter** deployment.

4. Within the **Configuration** tab, select the **Outbound Connection Pools** sub-tab. Select the **eis/DB/SOADemo** instance.

Settings for DbAdapter

Overview | Deployment Plan | **Configuration** | Security | Target

General | Properties | **Outbound Connection Pools** | Admin O...

This page displays a table of Outbound Connection Pool groups and its connection factory interface and the instances are listed by their JNDI group. Click the name of a group or instance to configure it. Automate

Outbound Connection Pool Configuration Table

New Delete

<input type="checkbox"/>	Groups and Instances
<input type="checkbox"/>	javax.resource.cci.ConnectionFactory
<input type="checkbox"/>	eis/DB/SOADemo
<input type="checkbox"/>	eis/DB/SOADemoLocalTx

New Delete

5. Select the **Connection Pool** tab and set the **Initial Capacity:** field to 10 and **Max Capacity:** to 500.

Settings for javax.resource.cci.ConnectionFactory

General | Properties | Transaction | Authentication | **Connection Pool** | Logging

Save

This page allows you to view and modify the pool parameters of this outbound connection.

Initial Capacity: 1

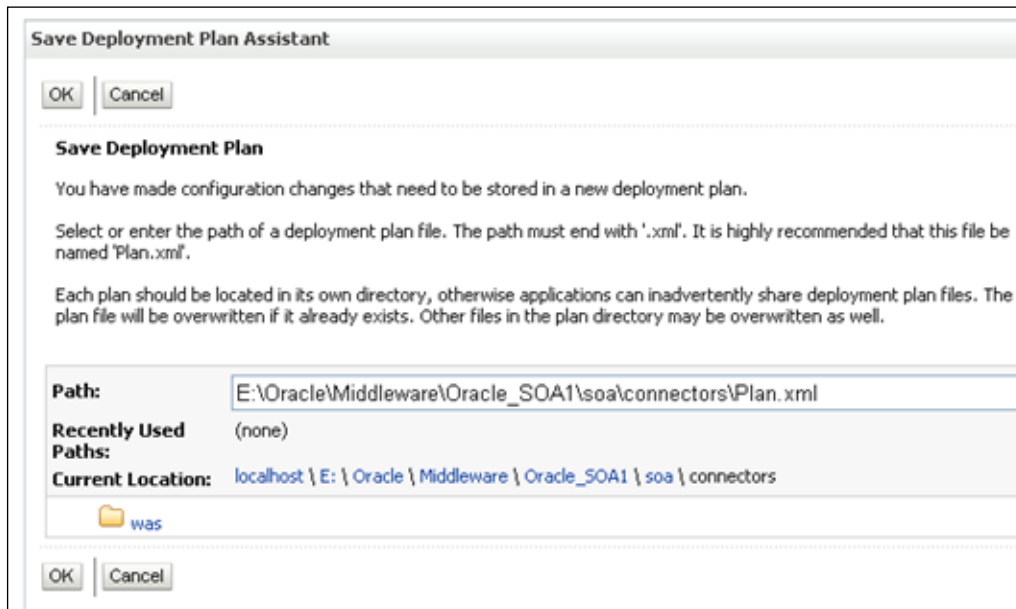
Max Capacity: 1000

Capacity Increment: 1

Shrinking Enabled: true

Shrink Frequency Seconds: 900

- Click on **Save** for your configuration changes and we'll be prompted to create a deployment plan to apply our changes permanently to the adapter via a deployment configuration for this domain.



How it works...

SOA Suite enables native access to resources such as file systems and databases, using JCA-compliant resource adapters.

SOA Suite packages a number of predefined resource adapters for interaction with database tables and the consumption of Oracle AQ message. For ease of configuration, we can tune these deployed artifacts using the WebLogic Administration console.

We can control the minimum and maximum sizes, which will help prevent delays when processing load and prevent the adapter from saturating the container's finite resources. We can also configure them to automatically "shrink" down in size when lesser load is being placed on the system.

When we make changes to these deployments, WebLogic allows us to apply these alterations on server restarts, using a deployment plan that is saved automatically for us, on the file system local to the WebLogic instance.

There's more...

As an alternative to using the WebLogic Administration console to configure resource adapters, we can manipulate the adapter archive files (.rar) manually on disk by modifying the `weblogic-ra.xml` configuration file. Note that this requires us to extract the archive, make the file change, and re-archive the .rar file; WebLogic must be restarted for changes to take effect.

We can also manually create new resource adapters using the JDeveloper tool.

Configuring the WebLogic thread pool

Application threads in SOA Suite are pooled in the underlying WebLogic container. In this recipe, we'll configure the main thread pool for a SOA Suite server instance.

Getting ready

You will need to have Oracle SOA Suite 11g installed on a server as we'll be accessing the default thread pools used by WebLogic for SOA Suite.

How to do it...

Follow these steps to configure our WebLogic thread pool:

1. Open the SOA Suite WebLogic administration console and log in as administrator. If you have issues locating the console, see steps 1 and 2 in the *Tuning global transactions* recipe in *Chapter 6, Platform Tuning*.
2. Navigate to the **Servers** section by performing either of the following actions:
 - ❑ Click on the **Servers** link under the **Environment** drop-down list in the **Domain Structure** box located on the left of the screen
 - ❑ In the **Home Page** box in the center of the screen, under **Environment** in **Domain Configurations**, select **Servers**
3. Select the **Administration** server. Click on the **Monitoring** tab and then the **Threads** sub-tab.

Settings for AdminServer

Configuration Protocols Logging Debug **Monitoring** Control Deployments

General Health Channels Performance **Threads** Timers Workload Security

Dump Thread Stacks

This page provides information on the thread activity for the current server.

The first table provides general information about the status of the thread pool. The second table provides information on the threads currently in the thread pool.

Note: The user ID displayed for individual threads is the user ID executing the thread during the last time it was active.

[Customize this table](#)

Self-Tuning Thread Pool (Filtered - More Columns Exist)

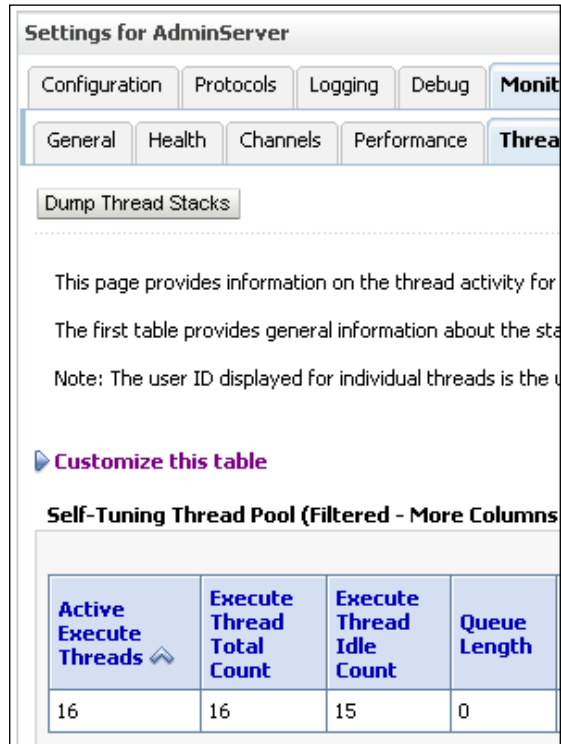
Active Execute Threads	Execute Thread Total Count	Execute Thread Idle Count	Queue Length	Pending User Request Count	Completed Request Count
4	5	3	0	0	761

- Here we can see that the total threads currently set to the **Self-Tuning Thread Pool** for this server is represented by the **Execute Thread Total Count** column. Our pool size is **5**, with **4** active threads.
- To reconfigure the default pool sizes, we need to add some Java `-D` properties to the command line. Navigate to the domain's home directory:
`%MIDDLEWARE_HOME%/user_projects/domains/soa_domain/`
 Replace `soa_domain` with the relevant domain, as required.
- Within the `bin` directory, open the `setSOADomainEnv.cmd` file on Windows (`setSOADomainEnv.sh` on Linux) and locate the following section:

```
set JAVA_OPTIONS=%JAVA_OPTIONS%
set DEFAULT_MEM_ARGS=-Xms1024m -Xmx1224m
set PORT_MEM_ARGS=-Xms1024m -Xmx1224m
```

- At the end of both the `DEFAULT_MEM_ARGS` and `PORT_MEM_ARGS` parameters, add both of the following `-D` parameters:
`-Dweblogic.threadpool.MinPoolSize=15`
`-Dweblogic.threadpool.MaxPoolSize=25`

- Restart the instance of WebLogic and browse to the Server's **Thread Monitoring** sub-tab again. You'll see that our **Execute Thread Total Count** value has changed to reflect the configured minimum size:



Settings for AdminServer

Configuration Protocols Logging Debug **Monitor**

General Health Channels Performance **Thread Monitoring**

Dump Thread Stacks

This page provides information on the thread activity for the AdminServer.

The first table provides general information about the state of the thread pool.

Note: The user ID displayed for individual threads is the user ID of the thread owner.

[Customize this table](#)

Self-Tuning Thread Pool (Filtered - More Columns)

Active Execute Threads	Execute Thread Total Count	Execute Thread Idle Count	Queue Length
16	16	15	0

How it works...

In order to process incoming application requests, SOA Suite leverages WebLogic's thread pool to acquire a thread resource. WebLogic utilizes a self-tuning thread pool that dynamically determines exactly how many threads are required to service the present load. This pool is shared among all hosted applications and services.

The server throughput is sampled at intervals of two minutes, based on internal algorithms. If the same throughput can be achieved with fewer threads, WebLogic will shrink the pool. Conversely, if throughput can be improved, it will increase the pool size.

By default, the pool size is unlimited. We can control the maximum thread pool size to prevent over-allocation of resources from the hosting machine to a WebLogic instance. We can also insulate against "cold starts" by preventing the thread pool from shrinking below a certain minimum size. This will ensure that the server will always have a buffer of resources to handle incoming requests.

There's more...

WebLogic has a powerful mechanism for managing its shared thread pool, called **Work Managers**. These can be configured to check on the number of threads to run an application or service with, or to add service-level agreements on things such as component response time.

Using JMS file persistence

JMS is a candidate SOA Suite endpoint destination, either for consumption or production of messages. WebLogic also provides JMS resources for SOA Suite Event Delivery Network, reliable Web Service Messaging, or for bridging to SOA Suite service hosting tiers such as Oracle Service Bus.

In this recipe, we'll configure WebLogic's JMS resources to use file-based persistence over database persistence.

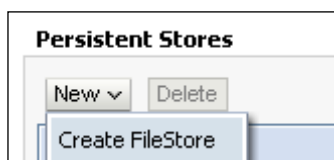
Getting ready

You will need to have Oracle SOA Suite 11g installed on a server as we'll be working in the area configured for the default JMS Persistence Stores used by SOA Suite.

How to do it...

The following steps will allow us to configure JMS persistence to use a file store:

1. Open the SOA Suite WebLogic administration console and log in as administrator. If you have issues locating the console, see steps 1 and 2 in the *Tuning global transactions* recipe in *Chapter 6, Platform Tuning*.
2. Navigate to the **Persistent Stores** section by performing either of the following actions:
 - ❑ Click on the **Persistent Stores** link under the **Services** drop-down list in the **Domain Structure** box located on the left of the screen
 - ❑ In the **Home Page** box in the center of the screen, under **Services** in **Domain Configurations**, select **Persistent Stores**
3. Click on **New** and select **Create FileStore**.



4. Now, select a location on disk for the JMS persistent store.
5. In the **Name** box, it is a good idea to give the store a descriptive name, with the targeted JMS Server as a part of it.
6. In the **Target** box, select the primary WebLogic Managed Server instance on which to host the JMS persistent store.
7. In the **Directory** box, enter a directory that exists on the target Managed Server host. You can omit an entry, and a folder named after the persistent store will be created dynamically under the domain folder in your %MIDDLEWARE_HOME% location, but we think it's best to explicitly state a location manually.
8. Finally, within the **Messaging** section, select a JMS Server to utilize our new file-based persistent store and use its **configuration** tab to select the store. This server must be targeted to the same Managed Server as the store.

Settings for SOAJMServer

Configuration | Logging | Targets | Monitoring | Control | Notes

General | Thresholds and Quotas | Session Pools

Save

JMS servers act as management containers for the queues and topics in JMS modules that are targeted to them. The persistent store is used for any persistent messages that arrive on the destinations, and to maintain the states.

Use this page to define the general configuration parameters for this JMS server.

Name: SOAJMServer

Persistent Store: SOAJMSFileStore

Paging Directory:

Paging File Locking Enabled

How it works...

JMS messages that are marked as being durable are stored before being delivered by WebLogic. Database message stores are slower than local disk-based stores for two reasons:

- ▶ A network hop is involved in streaming the message contents through a JDBC connection

- ▶ The RDBMS schema validation and record index updates in the database messaging table add overhead to the message-consumption process

By writing messages to file locally, we are omitting both of these overheads. WebLogic applies intelligent file-offset policies to enable it to track message locations, and also maintains a local index within the persistent store that allows rapid lookup and manipulation of messages within the store.

The File Store-write policy can have a big impact on messaging performance in high-throughput systems; Direct-Write-With-Cache leverages a local cache to help speed up the store's performance and will scale better than Direct-Write.

There's more...

JMS File Persistence stores can use WebLogic's native libraries to automatically apply values, which help speed up disk access on a host operating system. Where these libraries are not available, you can configure these values and other settings under the **Persistent Store Configuration** tab by clicking on the **Advanced** tab.

IO Buffer Size:	<input type="text" value="-1"/>
Maximum File Size:	<input type="text" value="1342177280"/>
Block Size:	<input type="text" value="-1"/>
Initial Size:	<input type="text" value="0"/>

Here, we can control a number of things:

- ▶ **IO Buffer Size:** This value changes depending on the store write policy. Typically, it should be greater than the largest store write value.
- ▶ **Maximum File Size:** This is not the size of the store, but the maximum size of individual files that comprise the store. A small number of large files is best; 1 GB is a good starting value.
- ▶ **Block Size:** There's lots of information on tuning this. Generally, it is best to have this larger than your message type so as to prevent splitting. Detailed information is available at http://docs.oracle.com/cd/E12839_01/web.1111/e13814/storetune.htm#CACDDHIB.
- ▶ **Initial Size:** The value in this field can help prevent file-expansion pauses at runtime, which can occasionally add small delays.

JMS resources in WebLogic can be configured to be migratable. This means that should a SOA cluster of WebLogic Server instances determine that a WebLogic-managed server is unreachable, then its JMS resources will be recreated in another existing instance of WebLogic.

A JMS persistent store must be designated as being migratable in order for this to succeed. This is achieved by targeting at a migratable WebLogic Managed Server. However, local persistent store messages cannot be streamed to the new hosting server as there may be a large archive of undelivered messages. In order for this to work effectively, the persistent store must be located on a slice of networked, shared storage. If this is correctly configured then the JMS Server will be migrated to a new host and message delivery will resume using the same JMS message store.

There is a potential performance trade-off in placing messaging stores on a network drive, however, as the native WebLogic file drivers need to be compatible with the drive location in order to leverage the faster access policies such as Direct-Write-With-Cache.

Tuning JMS connection factories

JMS is a candidate SOA Suite endpoint destination, used for either consumption or production of messages. WebLogic also provides JMS resources for SOA Suite Event Delivery Network, reliable Web Service Messaging, or for bridging to SOA Suite service hosting tiers such as Oracle Service Bus.

In this recipe, we'll change settings to tune the WebLogic JMS connection factories that dispense connections for our JMS destinations.

Getting ready

You will need to have Oracle SOA Suite 11g installed on a server as we'll be working in the area configured for the default JMS persistence stores used by SOA Suite.

How to do it...

Follow the steps given here to tune our JMS connection factory:

1. Open the SOA Suite WebLogic administration console and log in as administrator. If you have issues locating the console, see steps 1 and 2 in the *Tuning global transactions* recipe in *Chapter 6, Platform Tuning*.
2. Navigate to the JMS Modules section by performing either of the following actions:
 - ❑ Click on the **JMS Modules** link under the **Messaging** drop-down list under **Services** in the **Domain Structure** box located on the left of the screen
 - ❑ In the **Home Page** box in the center of the screen, under **Services** in **Domain Configurations**, select **JMS Modules**

- Choose the **SOAJMSModule** JMS module. In the list of resources, choose a JMS connection factory to tune. In the following screenshot, we've chosen **B2BEventQueueConnectionFactory**:

Settings for SOAJMSModule

Configuration Subdeployments Targets Security Notes

This page displays general information about a JMS system module and its resources. It also allows you to configure the module.

Name: SOAJMSModule

Descriptor File Name: jms/soajmsmodule-jms.xml

This page summarizes the JMS resources that have been created for this JMS system module, including queue and connection factory resources.

[Customize this table](#)

Summary of Resources

New Delete

<input type="checkbox"/>	Name	Type
<input type="checkbox"/>	B2BEventQueueConnectionFactory	Connection Factory

- Navigate to the **Default Delivery** tab. Set the **Default Redelivery Delay:** field to 10,000 milliseconds. Set the **Send Timeout:** field to 5,000 milliseconds.

Settings for B2BEventQueueConnectionFactory

Configuration Subdeployment Notes

General **Default Delivery** Client Transactions Flow Control Load Balance Security

Save

Use this page to define the default delivery configuration parameters for this JMS connection factory, such as the default delivery mode, default time to live, and default redelivery delay.

Default Priority: 4

Default Time-to-Live: 0

Default Time-to-Deliver: 0

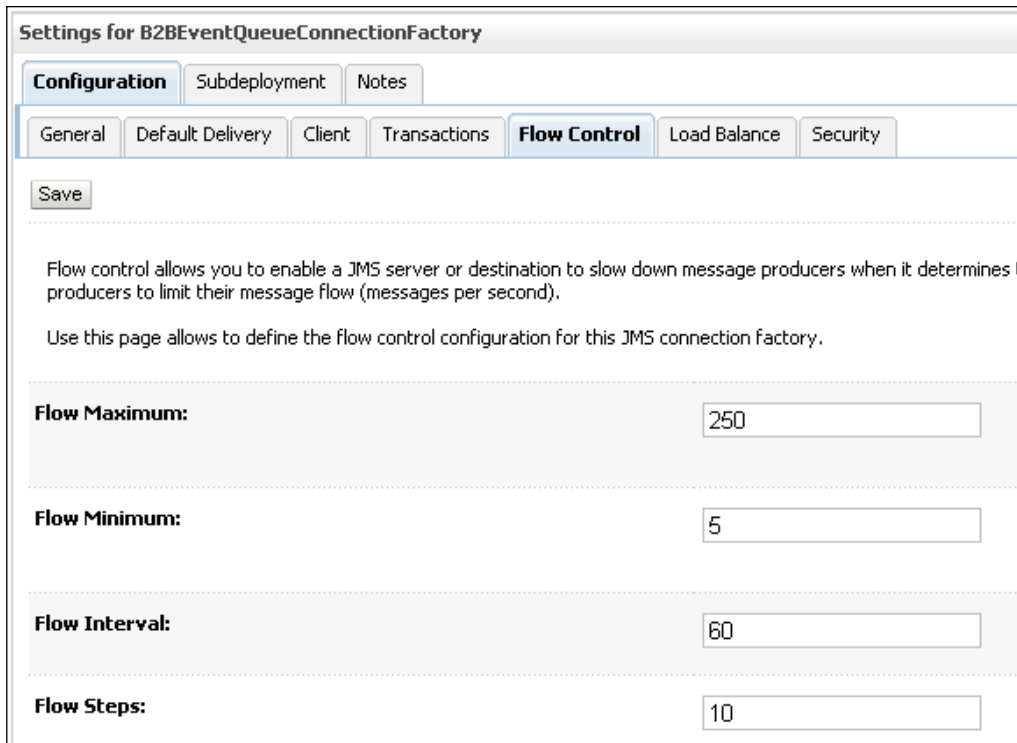
Default Delivery Mode: Persistent

Default Redelivery Delay: 0

Default Compression Threshold: 2147483647

Send Timeout: 10

5. Click on **Save**.
6. Navigate to the **Flow Control** tab and set the **Flow Maximum:** messages setting to **250**.



Settings for B2BEventQueueConnectionFactory

Configuration Subdeployment Notes

General Default Delivery Client Transactions **Flow Control** Load Balance Security

Save

Flow control allows you to enable a JMS server or destination to slow down message producers when it determines that producers are producing messages too fast. It allows producers to limit their message flow (messages per second).

Use this page to define the flow control configuration for this JMS connection factory.

Flow Maximum: 250

Flow Minimum: 5

Flow Interval: 60

Flow Steps: 10

7. Set the **Flow Minimum:** field's value to **5**.
8. Click on **Save**.

How it works...

The default connection factories provided by WebLogic JMS are not tunable. However, working with custom factories in our JMS modules opens up a host of tuning options.

We can set default delivery parameters (such as flow control) and message properties, which clients can override. Transaction, load balancing, and security can also be configured for connections used by inbound clients.

In this recipe, we have configured some of the default delivery options. Setting the **Send Timeout:** field to a higher value allows us to give a JMS Destination that is over a specific size) or a resource quota of 5,000 milliseconds to free up space to accommodate our message before an error is thrown. This allows for less message redelivery at the cost of some slight blocking on connections. Setting a redelivery delay can further help this process by waiting a certain period of time before resuming the delivery process for this message.

Furthermore, we set the flow control options that the connections from SOA Suite JMS clients are bound to obey; this instructs them to rate-limit their messages on demand from the SOA Suite WebLogic Managed Servers. JMS Destinations (Queues, Topics, and Distributed Queues) can be configured to expose Quotas and high/low watermarks, which when breached engage flow control to limit messaging throughput (which we have not explored here). Understanding these with our application requirements can help to control and optimize the system throughput. Our changes now require that when flow control conditions are triggered, no more than 250 messages can be sent per producer per second. If a producer is sending less than 5 messages a second, it will not be controlled.

There's more...

If you have lots of large messages that are not bytes or serialized objects, you may gain performance by setting a low KB value for the message compression value. This will reduce the amount of upstream and downstream network transmission on message delivery.

If you have a cluster of SOA Suite instances you can gain significant performance improvements by using distributed JMS destinations to spread the processing load.

JMS Destinations (Queues, Topics, Distributed Queues) also, by default, contain a notion of batching messages into groups, which can reduce the overall server throughput as separate per-message sends are not required. The trade-off with doing so is that pauses are introduced while building up a collection of messages to send, and retries involve larger number of messages.

8

BPEL and BPMN Engine Tuning

In this chapter, we will look at the following recipes:

- ▶ Tuning the dispatcher invoke threads
- ▶ Tuning the dispatcher engine threads
- ▶ Tuning the dispatcher system threads
- ▶ Disabling BPEL monitors and sensors
- ▶ Reducing the audit level
- ▶ Changing a BPEL process to be transient
- ▶ Reducing the completion persist level
- ▶ Disabling persistence on invocation
- ▶ Setting the audit store policy
- ▶ Reducing audit trail size threshold
- ▶ Increasing large document threshold
- ▶ Reducing SOA infra log levels
- ▶ Purging data from the BPEL store
- ▶ Ensuring automatic database statistics gathering is enabled
- ▶ Tuning Oracle database parameters
- ▶ Tuning the Oracle database

Introduction

The BPEL and BPMN engines are the heart of most Oracle SOA Suite 11g applications, and so the tuning suggestions here are relevant to most of the users of SOA Suite. While tuning lower down, the stack is focused on identifying and removing bottlenecks; tuning the BPEL and BPMN engines is much more of a trade-off between performance and reliability or the fault-tolerance of the system. So, these tuning recommendations should not be applied without carefully considering the consequences.

If you do some online research into the tuning options for SOA Suite, you may see references to tuning the internal business process engine. These engines are the runtime components that deal with executing the business processes in response to the user input. However, in our experience, this advice tends towards cases of setting parameter X to value Y and hoping for the best.

In this chapter, we want to expose the tuning options for SOA Suite's execution engines. In addition to showing you how to configure them, we'll talk about the motivating scenarios for doing so, and what outcomes you can expect.

We have found that the process engine tuning typically only comes into people's minds in production scenarios, where heavy load processing is required and throughput is not high enough; in other words, composites are not being as performant as they need to be. Gaining benefit from tuning the process engines really is dependent on the types of services you want to offer and the loads you expect to generate on them. Hopefully, by the end of this chapter, you'll have a good grasp on the things that can be controlled and how this feeds back into composite design.

The recipes in this chapter are directly used in *Chapter 11, SOA Application Design*, where we think about composite design and bring a number of the recipes together to suggest performant configurations.

Whilst this chapter's title references the BPEL and BPMN engines separately, SOA Suite 11g actually has largely merged the runtimes of these engines. Most of the recipes in this chapter affect the engine that executes both runtimes. Where a recipe only affects one runtime, we will mention that this is the case.

When trying the recipes in this chapter, we heavily recommend having a test environment where you can experiment with load in a controlled fashion and monitor the results.

Tuning the dispatcher invoke threads

The runtime SOA engine deals with composite execution. In this recipe, we'll tune one of its internal components—the BPEL Dispatcher Invoke thread pool size.

This recipe applies to the BPMN engine too; settings need to be applied to each engine individually. See the *There's more...* section at the end of this recipe.

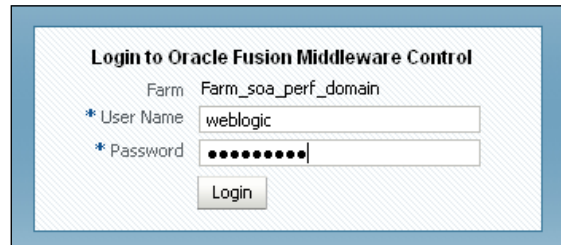
Getting ready

You will need to install SOA Suite with the Enterprise Manager Fusion Middleware Control component to follow this recipe.

How to do it...

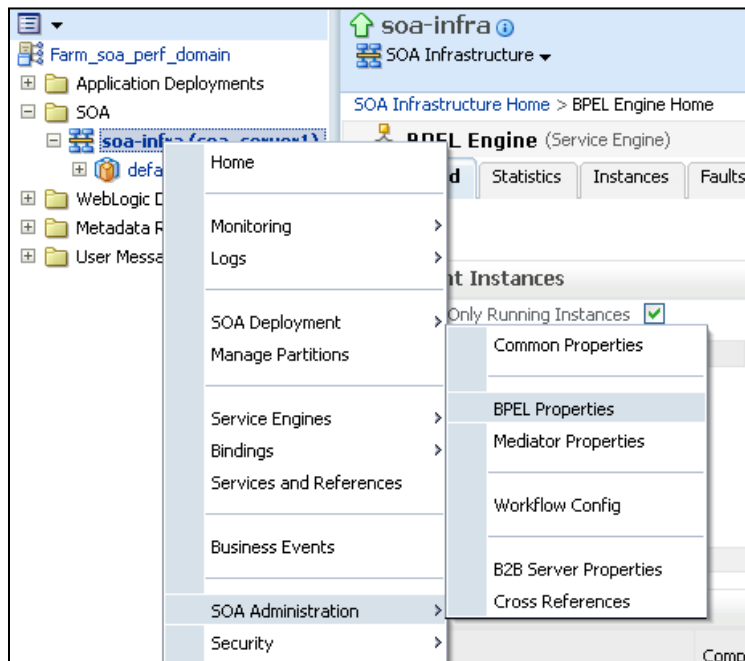
Follow these steps to set the Dispatcher Invoke Thread pool size:

1. Navigate to the **Enterprise Manager Fusion Middleware Control** homepage by pointing your browser to the URL hosted on the IP of the SOA Suite domain administration server, on the admin server's listen port below `http://SOA_SUITE_ADMIN_SERVER_IP:7001/em`.
2. Log in with the administration server credentials. These are the same as supplied for the admin user on the SOA Suite administration server when installing the domain. If you're unsure what these are, consult your domain administrator.



The screenshot shows the 'Login to Oracle Fusion Middleware Control' page. It includes a 'Farm' label with the value 'Farm_soa_perf_domain'. Below this are fields for '* User Name' (containing 'weblogic') and '* Password' (masked with dots). A 'Login' button is positioned at the bottom of the form.

- Expand the **SOA** folder in the left-hand pane and right-clicking on the **soa-infra** instance, then select **SOA Administration** and **BPEL Properties**. If you don't see a **SOA** folder, try checking if you're managed SOA servers are enabled and visible by the admin server.



- In **Dispatcher Invoke Threads** box, enter a new value. The default is 20. If your application sees high numbers of composites created, then increasing the number of threads can provide improvement, a good starting value is in the region of 50 to 100 threads.

SOA Infrastructure Home > BPEL Properties

BPEL Service Engine Properties ?

Properties

Edit property values and click Apply.

* Audit Level	Inherit	
* Audit Trail Threshold (Byte)	50000	
* Large Document Threshold (Byte)	100000	
* Dispatcher System Threads	2	
* Dispatcher Invoke Threads	20	Enter a number greater than or equal to 1. Dispatcher Invoke Threads Count
* Dispatcher Engine Threads	30	

5. Click on **Apply** in the top-right corner to persist these changes to the SOA domain.

How it works...

We started with this **BPEL/BPMN Engine** tuning option as it makes sense when you consider the lifecycle of a composite's execution. The **Dispatcher Invoke** thread pool is the entry point to SOA composite execution. In response to the requests from external systems to your composite, internal in-memory messages will be generated and consumed by threads in this pool. In turn, these threads will spawn instances of the required composites, to deal with the inbound requests.

The default of 20 threads means that we are limited to creating at the most 20 new composite instances at any one time. Typically, this is ample, but in scenarios where we have a lot of composite instantiations (especially those that execute only in memory and return synchronously to the client), we can increase the throughput on the server instance by setting a higher number of threads in this pool.

You will need to be mindful of the memory allocated to the SOA-managed server instances in your domain, and the thread stack size in the JVM settings, to ensure that you have enough memory for the required number of threads. If the SOA Server is executing lots of threads concurrently, then the CPU on the host may be forced to switch thread contexts a lot, which will degrade performance.

There's more...

Settings that tune the BPEL/BPMN engines are a trade-off between different types of composite execution. We cannot always guarantee how external systems will send requests into our domain, and thus that the runtime engine will be asked to run the same type of composite in one style of execution (synchronous, asynchronous, and so on). Depending upon the types of composites deployed, this may not be an issue.

It may be the case that the best way to tune your engine is to actually fragment the engines and host them across multiple domains, with each one tuned for the style of throughput required. This gets into the realm of architecture and design, however, which is beyond the scope of this book, but is worth considering if single engine performance is not enough for your needs.

To change the settings for the BPMN engine, you must select the properties from by right-clicking on the **Administration** dialog box on the `soa-infra` application. By default, this does not appear without a BPMN application, but it can still be accessed by configuring SOA system MBeans from the Enterprise Manager application:

1. From the **BPEL Service Engine Properties** page in step 2, select **More BPEL Configuration Properties**.

SOA Infrastructure Home > BPEL Properties

BPEL Service Engine Properties ?

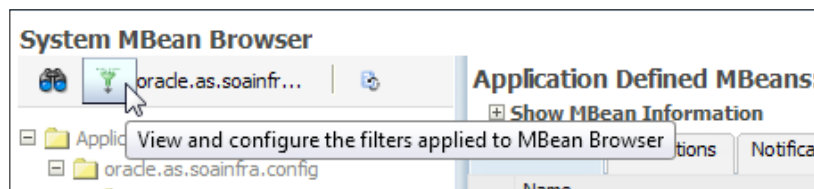
Properties

Edit property values and click Apply.

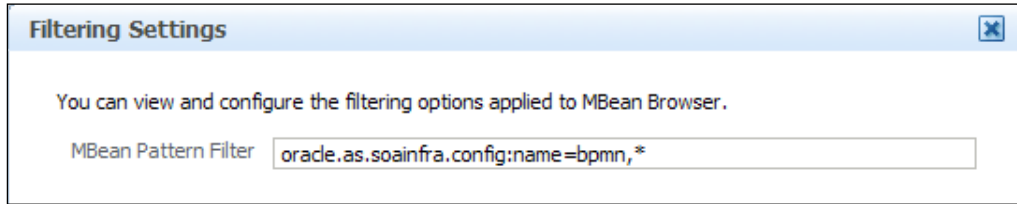
- * Audit Level: Inherit
- * Audit Trail Threshold (Byte): 50000
- * Large Document Threshold (Byte): 100000
- * Dispatcher System Threads: 2
- * Dispatcher Invoke Threads: 50
- * Dispatcher Engine Threads: 80
- * Payload Validation: ☐
- * Disable BPEL Monitors and Sensors: ☐

[More BPEL Configuration Properties...](#)

2. Click on the green filter icon in the following screenshot:



3. Change the MBean name to `name=bpmn`.



Filtering Settings

You can view and configure the filtering options applied to MBean Browser.

MBean Pattern Filter

4. You can now set the **Dispatcher Invoke Threads** value by using the Web form.

See also

- ▶ The *Tuning the dispatcher engine threads and Tuning the dispatcher system threads* recipes
- ▶ The *Setting the thread stack size and Increasing the JVM Heap size* recipes in Chapter 4, *JVM Memory*

Tuning the dispatcher engine threads

The runtime SOA engine deals with composite execution. In this recipe, we'll tune one of its internal components—the BPEL Dispatcher Engine thread pool size.

This recipe applies to the BPMN engine too; settings need to be applied to each engine individually. See the *There's more...* section at the end of this recipe.

Getting ready

You will need to install SOA Suite with the Enterprise Manager Fusion Middleware Control component to use this recipe.

How to do it...

Follow these steps to set the Dispatcher Engine Thread pool size:

1. Follow steps 1 to 3 in the *Tuning SOA engine dispatcher invoke threads* recipe, to get to the **BPEL Service Engine** properties page.
2. Set **Dispatcher Engine Threads** to a value for your system depending upon the number of asynchronous callbacks you process. We recommend starting at a value in the range of 80 to 150.

SOA Infrastructure Home > BPEL Properties
BPEL Service Engine Properties ?

Properties
Edit property values and click Apply.

* Audit Level	Inherit
* Audit Trail Threshold (Byte)	50000
* Large Document Threshold (Byte)	100000
* Dispatcher System Threads	2
* Dispatcher Invoke Threads	50
* Dispatcher Engine Threads	30

Enter a number greater than or equal to 1.
Dispatcher Engine Threads Count

3. Click on **Apply** in the top-right corner to persist these changes to the SOA Domain.

How it works...

In this recipe, we changed the BPEL Engine configuration for the Dispatcher Engine thread pool. This pool of threads is used to process a composite when it receives an asynchronous callback. When a composite calls to an external system using an asynchronous call, it is dehydrated to the SOA database to save its state for future use. When this composite needs to be loaded into memory for processing, either because a response from the external system has been received, an alarm is triggered (`onAlarm`) or a message is sent to it (`onMessage`), then an internal in-memory message is created in the SOA engine targeted at this composite. This thread pool processes these messages.

The default of 30 threads means we are limited to rehydrating at most 30 composites instances for execution at any one time. This may not be high enough, especially in systems that have a "bursty" style of execution, or for composites that have many dehydration points. We can increase the throughput on the server instance by setting a higher number of threads in this pool. If there is one engine parameter that usually gets the biggest increase, it is this one.

You will need to be mindful of the memory allocated to the SOA-managed server instances in your domain, and the thread stack size in the JVM settings, to ensure that you have enough memory for the requisite number of threads. If the SOA server is executing lots of threads concurrently, then the CPU on the host may be forced to switch thread contexts a lot, which will degrade the performance. These are the trade-offs that need to be considered when increasing the threads for the engine parameters.

There's more...

To change the settings for the BPMN engine, you must select the properties by right-clicking on the **Administration** dialog box on the `soa-infra` application. By default, this does not appear without a BPMN application, but it can still be accessed by configuring SOA system MBeans from the Enterprise Manager application.

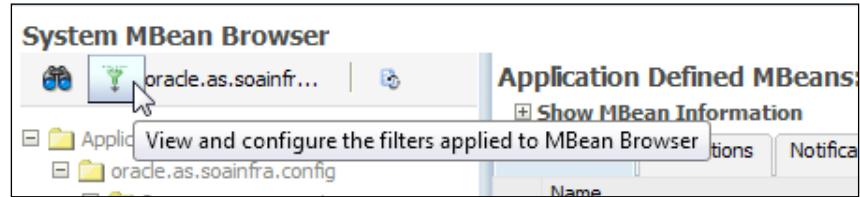
1. From the **BPEL Service Engine Properties** page in step 2, select **More BPEL Configuration Properties**.

The screenshot shows the 'BPEL Service Engine Properties' page. At the top, there is a breadcrumb 'SOA Infrastructure Home > BPEL Properties' and a title 'BPEL Service Engine Properties' with a help icon. Below the title is a section labeled 'Properties' with the instruction 'Edit property values and click Apply.' The properties are listed in a table-like format with labels and values. The 'Audit Level' is set to 'Inherit' via a dropdown menu. Other properties include 'Audit Trail Threshold (Byte)' at 50000, 'Large Document Threshold (Byte)' at 100000, 'Dispatcher System Threads' at 2, 'Dispatcher Invoke Threads' at 50, and 'Dispatcher Engine Threads' at 80. There are also two unchecked checkboxes for 'Payload Validation' and 'Disable BPEL Monitors and Sensors'. At the bottom, there is a link 'More BPEL Configuration Properties...'.

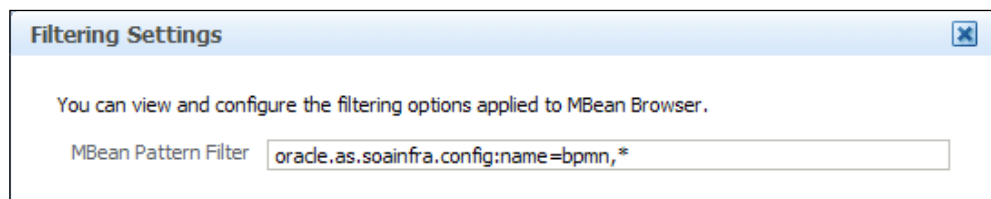
Property	Value
* Audit Level	Inherit
* Audit Trail Threshold (Byte)	50000
* Large Document Threshold (Byte)	100000
* Dispatcher System Threads	2
* Dispatcher Invoke Threads	50
* Dispatcher Engine Threads	80
* Payload Validation	<input type="checkbox"/>
* Disable BPEL Monitors and Sensors	<input type="checkbox"/>

[More BPEL Configuration Properties...](#)

2. Click on the green filter icon in the following screenshot:



3. Change the MBean name to name=bpmn.



4. You can now set the **Dispatcher Engine Threads** value by using the Web form.

See also

- ▶ The *Tuning the dispatcher invoke threads and Tuning the dispatcher system threads* recipes
- ▶ The *Setting the thread stack size and Increasing the JVM Heap size* recipes in *Chapter 4, JVM Memory*

Tuning the dispatcher system threads

The runtime SOA engine deals with composite execution. In this recipe, we'll tune one of its internal components—the BPEL Dispatcher System thread pool size.

This recipe applies to the BPMN engine too; settings need to be applied to each engine individually. See the *There's more...* section at the end of this recipe.

Getting ready

You will need to install SOA Suite with the Enterprise Manager Fusion Middleware Control component to use this recipe.

How to do it...

Follow these steps to set the Dispatcher System thread pool size:

1. Follow steps 1 to 3 in the *Tuning SOA engine dispatcher invoke threads* recipe, to get to the **BPEL Service Engine** properties page.
2. Set **Dispatcher System Threads** to a higher value for your system. We recommend starting at a value of 5.

BPEL Service Engine Properties ?

Properties

Edit property values and click Apply.

* Audit Level	Inherit
* Audit Trail Threshold (Byte)	50000
* Large Document Threshold (Byte)	100000
* Dispatcher System Threads	2
* Dispatcher Invoke Threads	50
* Dispatcher Engine Threads	80

Enter a number greater than or equal to 1.
Dispatcher System Threads Count

3. Click on **Apply** in the top-right corner to persist these changes to the SOA Domain.

How it works...

In this recipe, we changed the BPEL Engine configuration for the Dispatcher System thread pool. This pool of threads is of lower importance in improving the process throughput as compared to the other two Dispatcher pools mentioned in the previous recipes. It is used to process in-memory messages that instruct when to perform internal housekeeping tasks on the stateful components, such as EJBs, that provide the SOA Suite functionality. Increasing the size of the Invoke and Engine thread pools means there will potentially be more load on the system, so we increase the maintenance pool to help clean up resources as they are freed.

There's more...

To change the settings for the BPMN engine, you must select the properties from by right-clicking on the **Administration** dialog box on the `soa-infra` application. By default, this does not appear without a BPMN application, but it can still be accessed by configuring the SOA system MBeans from the Enterprise Manager application.

1. From the **BPEL Service Engine Properties** page in step 2, select **More BPEL Configuration Properties**.

SOA Infrastructure Home > BPEL Properties

BPEL Service Engine Properties ?

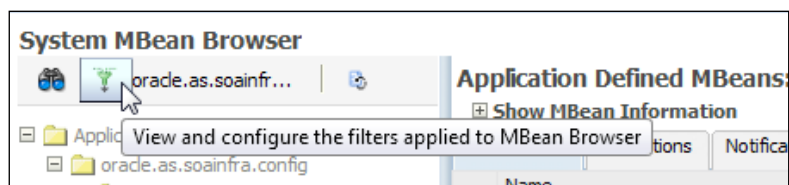
Properties

Edit property values and click Apply.

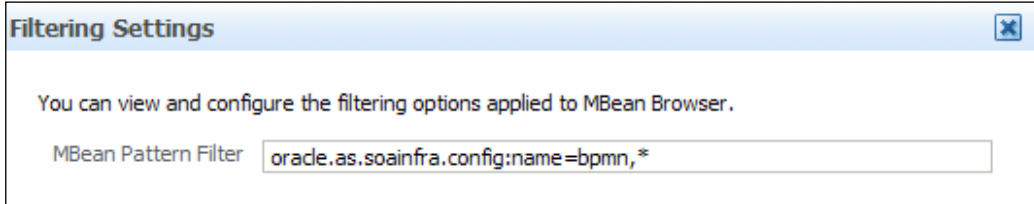
* Audit Level	Inherit
* Audit Trail Threshold (Byte)	50000
* Large Document Threshold (Byte)	100000
* Dispatcher System Threads	2
* Dispatcher Invoke Threads	50
* Dispatcher Engine Threads	80
* Payload Validation	<input type="checkbox"/>
* Disable BPEL Monitors and Sensors	<input type="checkbox"/>

[More BPEL Configuration Properties...](#)

2. Click on the green filter icon in the following screenshot:



3. Change the MBean name to `name=bpmn`.



Filtering Settings

You can view and configure the filtering options applied to MBean Browser.

MBean Pattern Filter

4. You can now set the **Dispatcher System Threads** value by using the Web form.

See also

- ▶ The *Tuning the dispatcher engine threads and Tuning the dispatcher invoke threads* recipes
- ▶ The *Setting the thread stack size and Increasing the JVM Heap size* recipes in Chapter 4, JVM Memory

Disabling BPEL monitors and sensors

In this recipe, we will change a global control to control monitoring components that can affect the throughput.

Getting ready

You will need to install SOA Suite with the Enterprise Manager Fusion Middleware Control component to use this recipe.

How to do it...

Follow these steps to control the BPEL sensors and monitors:

1. Follow steps from 1 through 3 in the *Tuning SOA engine dispatcher invoke threads* recipe to get to the **BPEL Service Engine** properties page.
2. Select the **Disable BPEL Monitors and Sensors** checkbox as shown in the following screenshot:

The screenshot shows the 'BPEL Service Engine Properties' page in a web console. The page title is 'SOA Infrastructure Home > BPEL Properties' and 'BPEL Service Engine Properties ?'. Under the 'Properties' section, it says 'Edit property values and click Apply.' The properties listed are:

- * Audit Level: Inherit (dropdown)
- * Audit Trail Threshold (Byte): 50000 (text input)
- * Large Document Threshold (Byte): 100000 (text input)
- * Dispatcher System Threads: 2 (text input)
- * Dispatcher Invoke Threads: 50 (text input)
- * Dispatcher Engine Threads: (empty text input)
- * Payload Validation: ☐
- * Disable BPEL Monitors and Sensors: ☒

A tooltip box points to the 'Disable BPEL Monitors and Sensors' checkbox with the text: 'Select to force disable all monitors and sensors in the BPEL service engine. Monitors and sensors cannot be enabled at the composite level when this is selected.'

3. Click on **Apply** in the top-right corner to persist these changes to the SOA domain.

How it works...

BPEL sensors are a part of the deployed composites that are evaluated at runtime by the BPEL engine. They are targeted at sections of the composite and can be fired around or within the composite in a number of ways, depending upon how the composite is behaving. When fired, they can trigger the further publishing of events and actions that add extra processing burden and database load to our production deployment. It is not uncommon for BPEL sensors, designed for development functional diagnosis, to be deployed unmodified to production.

In this recipe, we altered a global setting that disables processing of BPEL sensors completely. This can be useful for helping improve the throughput in environments, where there are high numbers of composites deployed, but is a big change that may impact monitoring and integration with external monitoring systems.

There's more...

It is possible to disable sensors on a per-composite basis if you know which composite you wish to make this change to. Use the **Enterprise Manager** console to view a composite, and click on the **Settings** button to make the change.

It is also possible to disable the BPMN sensors via the properties page or engine MBeans; see the *There's more...* section in the *Tuning Dispatcher Invoke Threads* recipe, to see how to get to the MBean tree for the BPMN engine.

Reducing the audit level

In this recipe, we'll look at controlling the level of auditing performed by the BPEL engine when executing our processes.

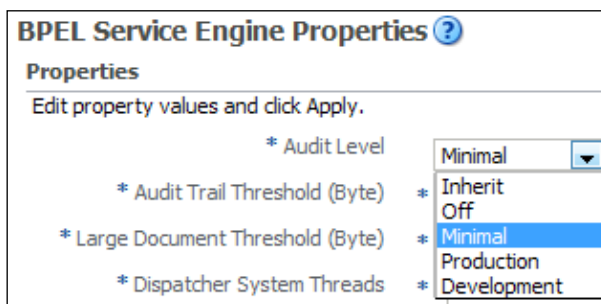
Getting ready

You will need to install SOA Suite with the Enterprise Manager Fusion Middleware Control component to use this recipe.

How to do it...

Follow these steps to control the audit level:

1. Follow steps 1 to 3 in the *Tuning SOA engine dispatcher invoke threads* recipe, to get to the *BPEL Service Engine properties* page.
2. Set **Audit Level** to a lower value, such as **Minimal**:



3. Click on **Apply** in the top-right corner to persist these changes to the SOA domain.

How it works...

BPEL processes that execute in the runtime engine generate logging events that allow inspection in the **Enterprise Manager** console; these writes to the database are synchronous and blocking. Typically, it is rare that the operations teams will want to comb through every execution of a composite in production, if it is running successfully. This information is only made available in the **Enterprise Manager** console, so it is a candidate for removal if you are not using this facility.

The audit level controls the level of details collected, as the composite works through its flows. This directly affects the number of items in the `AUDIT_TRAIL` database table backing SOA Suite. The levels of note are as follows:

- ▶ **Off:** No information is collected. This is likely to be the setting, which will really improve the concurrent BPEL performance, as no database audit logging is performed.
- ▶ **Minimal:** Logging of the composite flow still occurs, but no request/response (payload) information is sent to the database. This can help improve performance if the payloads involved are large, or many stages are involved in the composite flow.
- ▶ **Production:** All audit information is collected including payloads at all stages of the composite flows, with the exception of BPEL process "assign" activities. This is the default value.
- ▶ **Development:** We don't recommend running this in production. It captures the state of the composite at every flow stage.

Unless you are experiencing issues in production we recommend tuning the engine to the minimal setting. This will help reduce the overhead involved in communication with the database at the cost of detailed flow execution data.

There's more...

The **Payload Validation** setting on the BPEL engine properties page will add extra processing to the composite execution. The **Capture Composite Instance State** value will also require more database storage operations for the composite execution. These are both off by default, as shown in the following screenshot, but may be enabled when composites are under development:



The screenshot shows the 'SOA Infrastructure Common Properties' configuration page. At the top, there is a breadcrumb 'SOA Infrastructure Home > Common Properties' and a title 'SOA Infrastructure Common Properties' with a help icon. Below the title is a note: 'The properties set at this level will impact all deployed composites'. The configuration includes three settings: 'Audit Level' set to 'Production' via a dropdown menu, 'Capture Composite Instance State' with an unchecked checkbox, and 'Payload Validation' with an unchecked checkbox.

Combining reducing audit logging with changing the completion persist policy and `inMemoryOptimization` settings can reduce the simultaneous number of database round trips required by executing composites. This can have a big impact on the performance at the cost of composite execution insight.

See also

- ▶ The *Disabling persistence on invocation* and *Changing a BPEL process to be transient* recipes

Changing a BPEL process to be transient

In this recipe, we'll disable some of the persistence that occurs by default on processes executed in the BPEL engine.

Getting ready

You'll also need a composite loaded into JDeveloper for this recipe. We have provided one with this chapter's source code available from the book's website.

How to do it...

Follow these steps to make a BPEL process transient:

1. Load the composite into JDeveloper and open the `composite.xml` file.
2. Locate the `<component>` section and add a property element, as follows:

```
<property name="bpel.config.inMemoryOptimization">true</property>
```

3. The file should look similar to the following output:

```
<component name="BPELProcess1" version="1.1">
  <implementation.bpel src="BPELProcess1.bpel"/>
  <property name="bpel.config.transaction" type="xs:string" many="false">required</property>
  <property name="bpel.config.inMemoryOptimization">true</property>
</component>
```

How it works...

BPEL processes are, by default, persisted to the underlying SOA database at a number of stages during the flow execution; this process is known as **dehydration**. We call these processes **durable**, as they are resilient against outages in the server runtime. However, certain types of processes can be configured in such a way that the BPEL engine will execute them entirely in-memory, making them transient. This has the potential to speed up the execution of our processes. We have to satisfy some conditions to execute entirely in memory, however:

- ▶ A process must not contain a receive point (beyond the initial invocation)
- ▶ The process must be of a synchronous type
- ▶ onMessage items but not be present
- ▶ There must be no wait items

Any of the preceding conditions enforces dehydration as a part of the BPEL engine's internal logic – there's no way to optimize it presently. For this reason short-lived, synchronous processes are good candidates for setting this memory optimization flag. This is a good option that we recommend setting if your requirements are compatible.

If set, the completion persist level is used to determine at which points process dehydration occurs. See the next recipe for options on setting this flag.

There's more...

BPEL processes will still be saved on initial invocation; see the *Disabling persistence on invocation* recipe, to prevent this and improve the throughput even more.

See also

- ▶ The *Disabling persistence on invocation* and *Reducing the audit level for messages written to disk* recipes

Reducing the completion persist level

In this recipe, we'll change conditions that can govern when dehydration is performed by the BPEL engine upon process completion.

Getting ready

You'll also need a composite loaded into JDeveloper for this recipe. We have provided one with this chapter's source code, available from the book's website.

You'll also need to have set the `inMemoryOptimization` flag; see the *Changing a BPEL process to be transient* recipe for instructions on doing this.

How to do it...

Follow these steps to reduce the BPEL completion persist level:

1. Load the composite into JDeveloper and open the `composite.xml` file.
2. Locate the `<component>` section, and add a `property` element, as follows:

```
<property name="bpel.config.completionPersistPolicy">faulted</property>
```
3. The file should look similar to the following output:

```
<component name="BPELProcess1" version="1.1">
  <implementation.bpel src="BPELProcess1.bpel"/>
  <property name="bpel.config.transaction" type="xs:string" many="false">required</property>
  <property name="bpel.config.inMemoryOptimization">true</property>
  <property name="bpel.config.completionPersistPolicy">faulted</property>
```

How it works...

When we mark a process as being transient (executed totally in-memory), we can control when the completed process data is persisted to the database, if ever. Our options for this setting are:

- ▶ **On:** This is the default value. The completed instance is saved as a part of the process lifecycle. This is a synchronous operation, so it adds to the total processing time.
- ▶ **Deferred:** The process state is saved asynchronously by a separate thread to the one which ran the process. This offers a small performance increase, as the process thread can return quicker in exchange for a small risk.

- ▶ **Faulted:** This is a recommended setting. The instance state is only saved when the process errors. If most instances are expected to succeed, this will have a lot of impact on overall database utilization.
- ▶ **Off:** No instance data is saved. Whilst the fastest, we also lose any state information.

If your non-functional requirements allow it, for faster execution of BPEL processes, we recommend a faulted completion persist policy alongside a lower audit message level.

See also

- ▶ The *Changing a BPEL process to be transient*, *Disabling persistence on invocation*, and *Reducing the audit level for messages written to disk* recipes

Disabling persistence on invocation

In this recipe, we'll look at how we can speed up the BPEL execution by using memory queuing.

Getting ready

You'll also need a composite loaded into JDeveloper for this recipe. We have provided one with this chapter's source code available from the book's website.

How to do it...

Follow these steps to disable the BPEL process persistence when invoked:

1. Load the composite into JDeveloper, and open the `composite.xml` file.
2. Locate the `<component>` section and add a property element with the name `bpel.config.oneWayDeliveryPolicy` and value `async.cache` as follows:

```
<property name="bpel.config.oneWayDeliveryPolicy">async.cache</property>
```
3. The file should look similar to the following output:

```
<component name="BPELProcess1" version="1.1">
  <implementation.bpel src="BPELProcess1.bpel"/>
  <property name="bpel.config.transaction" type="xs:string" many="false">required</property>
  <property name="bpel.config.oneWayDeliveryPolicy">async.cache</property>
```

How it works...

The BPEL engine, by default, saves inbound requests to a database table named `dlv_message` (delivered messages). From this table, the threads in the BPEL engine populate the in-memory queue used to instantiate processes. This default setting is represented by a value of `sync` on the property `bpel.config.oneWayDeliveryPolicy`.

By setting the value to `async.cache`, we remove the intermediary call to the database, placing invocations only in a memory queue ready to be run as BPEL processes. This will prevent us from being able to restart the process at a later date, but removes a database roundtrip. Furthermore, if we have too many inbound requests, the queue can fill up causing server memory errors; so exercise caution, if enabling this setting. A monitoring solution is recommended for keeping a track of request counts, if you change this setting.

Another possible value for this property is `sync`, which will remove the invoke queue completely from the processing chain; threads that accept calls will synchronously invoke the process to completion. This removes the asynchronous decoupling, and can improve the throughput in some use cases, where we have fast-executing, synchronous processes. Coupling this with `inMemoryOptimization` can give big improvements. However, if high loads are possible within a short space of time, this increases the risk that the system will not have sufficient resources to service inbound requests and that calling components will be made to wait.

The optimum tuned setting depends upon the types of things in your composite, but usually a setting of `async.cache` will help the throughput in environments that make heavy use of the database.

There's more...

If database I/O is causing contention, see the *Changing a BPEL process to be transient* and *Reducing the audit level* recipes, which can optimize a lot of the dehydration persistence out of the composite execution.

See also

- ▶ The *Changing a BPEL process to be transient* and *Reducing the completion persist level* recipes

Setting the audit store policy

Audit records are normally written to the database synchronously, which imposes a significant performance overhead on the application. By switching to writing audit information asynchronously, we can get a large performance improvement.

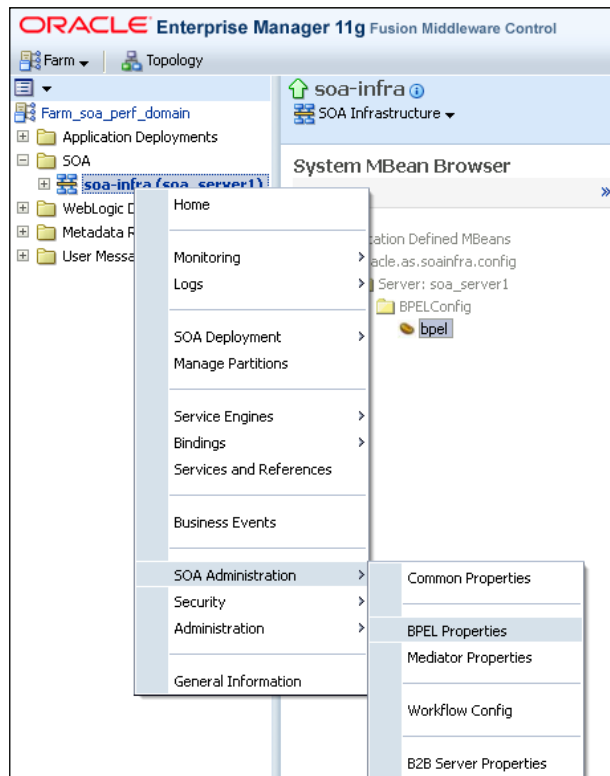
Getting ready

You will need to install SOA Suite with the Enterprise Manager Fusion Middleware Control component to use this recipe.

How to do it...

Follow these steps to set the Audit storage policy:

1. Follow steps from 1 through 3 in the *Tuning SOA engine dispatcher invoke threads* recipe to get to the **BPEL Service Engine properties** page.



- This will open the BPEL properties page; click on the **More BPEL Configuration Properties** link to see the advanced properties:

- Locate the **Audit Store Policy** property, and set its value to `async`, as shown.

8	AuditStorePolicy	Flag to choose the audit store strategy	RW	<input type="text" value="async"/>
---	----------------------------------	---	----	------------------------------------

- Save the changes.

How it works...

The audit store policy determines how audit messages are persisted to the database. The default is for audit messages to be persisted to the database in the same transaction as the `s` instance data. By persisting messages asynchronously on a different thread, we can improve the performance, with a slight risk that the audit record might not get written if the system fails at exactly the wrong time.

In addition to the default value of `syncSingleWrite` and `async`, there is another option, which is `syncMultipleWrite`. This option will use the same thread, but write the audit message in a different transaction. This has the advantage that if the process instance transaction is rolled back for any reason, the audit record is still written.

There's more...

When using the `async` audit store policy, there are two additional parameters that we can tune, which affect how frequently the asynchronous audit events are written to the database. These are `AuditFlushByteThreshold`, which is the number of bytes after which the audit messages are flushed, and `AuditFlushEventThreshold`, which is the number of audit events. By tuning these, it is possible to further reduce the number of times that SOA Suite talks to the database, but with the additional risk of possible data loss of audit events.

See also

- ▶ The *Reducing audit trail size threshold* recipe

Reducing audit trail size threshold

The bodies of messages are normally saved to the `AUDIT_TRAIL` table in the database, with larger messages being stored in `AUDIT_DETAILS`. By decreasing the threshold for messages to be saved in `AUDIT_DETAILS`, we can prevent a large amount of payload data from being loaded in the `AUDIT_TRAIL` table, which can improve performance.

Getting ready

You will need to install SOA Suite with the Enterprise Manager Fusion Middleware Control component to use this recipe.

How to do it...

Follow these steps to set the Audit trail size threshold:

1. Follow steps from 1 through 2 of the *Setting the audit store policy* recipe to navigate to the **Advanced BPEL Configuration** page.
2. Locate the **AuditDetailThreshold** property, and decrease its value.

4	<code>AuditDetailThreshold</code>	The maximum size (in bytes) an audit trail details string can be before it is stored separately from the audit trail	RW	<input type="text" value="10000"/>
---	-----------------------------------	--	----	------------------------------------

3. Save your changes.

How it works...

The audit trail size threshold determines how large audit messages have to be before their content is stored in the `AUDIT_DETAILS` table rather than in the `AUDIT_TRAIL` table. By decreasing this value, we can make processing of the data in `AUDIT_TRAIL` faster when audit details are viewed in the Enterprise Manager console, because it does not need to load as many large audit message bodies. The larger details are still available in `AUDIT_DETAILS`, if required.

See also

- ▶ The *Setting the audit store policy* and *Increasing large document threshold* recipes

Increasing large document threshold

By default, documents larger than 100 k are saved to the database, and the key for the document is passed around the server. If we have sufficient memory in our application, then increasing this threshold above 100 k will prevent frequent trips to the database.

Getting ready

You will need to install the SOA Suite with the Enterprise Manager Fusion Middleware Control component to use this recipe..

How to do it...

1. Follow steps from 1 through 2 of the *Setting the audit store policy* recipe to navigate to the **Advanced BPEL Configuration** page.
2. Locate the **LargeDocumentThreshold** property, and increase its value.

24	LargeDocumentThreshold	The maximum size (in bytes) a BPEL variable can be before it is stored in a separate location from the rest of the instance scope data	RW	<input type="text" value="100000"/>
----	------------------------	--	----	-------------------------------------

3. Save your changes.

How it works...

The large document threshold is the size above which the document body is saved in the database, and a key to the body is passed between process instance components. If you have sufficient memory, you can increase this value to allow larger document bodies to be passed around in-memory, saving trips to the database, and increasing performance.

You should note, however, that if your BPEL process has any dehydration points (as discussed in the *Changing a BPEL process to be transient* recipe), then the message will be stored regardless of this setting when the process is dehydrated. It is possible to make the process transient (in-memory only), which speeds up the processing at the cost of recovery from system failure.

See also

- ▶ The *Reducing the audit trail size threshold* and *Changing a BPEL process to be transient* recipes

Reducing SOA infra log levels

Oracle SOA Suite 11g writes log information to log files, and by reducing the amount of file I/O that logging performs, we can improve the performance of the entire SOA Suite 11g infrastructure.

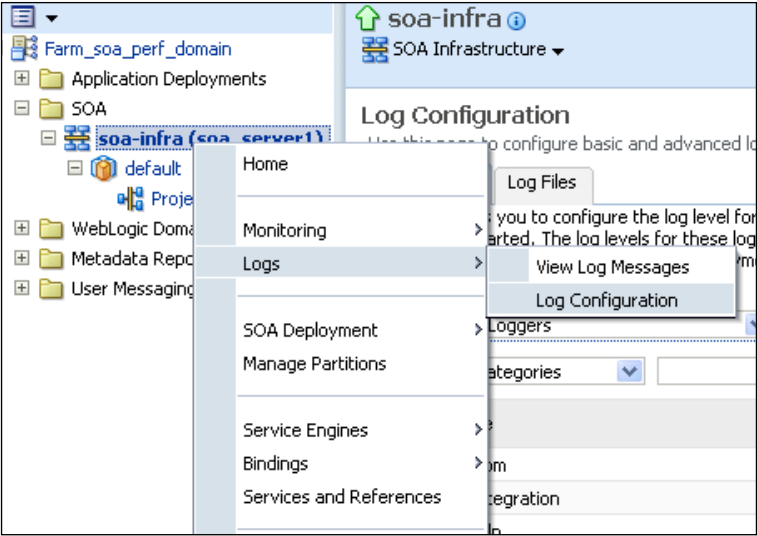
Getting ready

You will need to install the SOA Suite with the Enterprise Manager Fusion Middleware Control component to use this recipe.

How to do it...

Follow these steps to configure the SOA Suite SoaInfra application's log levels:

1. Follow steps from 1 through 2 in the *Tuning SOA Engine dispatcher invoke threads* recipe to get to the **BPEL Service Engine properties** page.
2. Navigate to **Logs** and **Log Configuration**, as shown in the following screenshot:



3. Set all the log levels to **SEVERE**.

Log Configuration
Use this page to configure basic and advanced log configuration settings.

Log Levels | Log Files

This page allows you to configure the log level for both persistent loggers and active runtime loggers. Persistent loggers are loggers that are created when the component is started. The log levels for these loggers are persisted across component restarts. Runtime loggers are automatically created when a component is exercised. For example, `oracle.j2ee.ejb.deployment.Logger` is a runtime logger that becomes active when an EJB module is deployed.

View: Runtime Loggers

Search: All Categories

Logger Name	Oracle Diagnostic Logging Level (Java Level)	Log File
oracle.bpm	ERROR:1 (SEVERE)	odl-handler
oracle.integration	ERROR:1 (SEVERE)	odl-handler
oracle.sdp	ERROR:1 (SEVERE)	odl-handler
oracle.sdpinternal	ERROR:1 (SEVERE)	odl-handler
oracle.soa	ERROR:1 (SEVERE)	odl-handler
oracle.sysman	ERROR:1 (SEVERE)	em-log-handler em-trc-handler
oracle.wsm	ERROR:1 (SEVERE)	odl-handler

How it works...

By reducing the amount of file I/O that occurs from logging, it is possible to improve the performance of the application, but with the trade-off that the log information is not available if you have a problem. In a stable system, where you do not expect frequent outages, it is worth setting the log levels so that only critical errors are reported in the log files.

There's more...

In this recipe, we set all the log levels to **SEVERE**. If you don't want to turn off logging for all the components, you can selectively set the log levels for different components and subsystems. The + icons next to each category can be expanded to allow the levels to be set at a finer level, or to allow specific components to continue to log less severe messages if required.

See also

- ▶ The *Reducing the audit threshold* recipe
- ▶ The *Reducing server logging level* recipe in *Chapter 6, Platform Tuning*

Purging data from the BPEL store

In this recipe we will remove old BPEL dehydration data and state from the SOA infrastructure database.

Getting ready

You will need to have access to the database on which the `SOA_INFRA` schema is hosted. In this recipe, we'll be using a command line local to the host on which we installed the database.

You'll also need access to the SQL scripts bundled with SOA Suite. If you have SOA Suite installed on the host running the database, you can find them under `MW_HOME/SOA_ORACLE_HOME/rcu/integration/soainfra/sql/soa_purge`.

If the database is running on a separate host, you can simply copy the `soa_purge` directory from the WebLogic administration server to a directory on the database host; we'll be using `e:\soa_purge` for this purpose.

How to do it...

Follow these steps to run the `soa_purge` scripts:

1. First, log in to `sqlplus` as a user with `sysdba` privileges, and grant the following permissions to the `dev_soainfra` user, then exit the shell:

```
sqlplus / as sysdba
SQL> GRANT EXECUTE ON DBMS_LOCK TO DEV_SOAINFRA;
SQL> GRANT CREATE ANY JOB TO DEV_SOAINFRA;
SQL> exit
```

2. In the command line, it's easiest to change the directory to the location of the SOA Suite `purge_sql` folder. So, do that now and then log in as the `dev_soainfra` user:

```
cd e:\soa_purge
SQL> sqlplus DEV_SOAINFRA/your_soainfra_password
Now execute the SQL script, this will load generate lots of
output. If successful this will end with 'procedure completed
successfully'.
SQL> @soa_purge_scripts.sql
...
Procedure created.
Type created.
Type body created.
PL/SQL procedure successfully completed.
```

3. Enter the following, substituting the creation dates as appropriate:

```
SQL> DECLARE
2
3     max_creation_date timestamp;
4     min_creation_date timestamp;
5     retention_period timestamp;
6 BEGIN
7
8     min_creation_date := to_timestamp('2012-10-01','YYYY-MM-
DD');
9     max_creation_date := to_timestamp('2013-01-30','YYYY-MM-
DD');
10    retention_period := to_timestamp('2010-02-01','YYYY-MM-
DD');
11
12    soa.delete_instances_in_parallel(
13        min_creation_date => min_creation_date,
14        max_creation_date => max_creation_date,
15        batch_size => 10000,
```

```
16      max_runtime => 60,  
17      retention_period => retention_period,  
18      DOP => 3,  
19      max_count => 1000000,  
20      purge_partitioned_component => false);  
21  
22  END;  
23  /
```

PL/SQL procedure successfully completed.

How it works...

SOA Suite comes bundled with maintenance scripts that allow us to perform housekeeping on the underlying database tables. As these tables are used for each invocation, dehydration and audit tasks over time can fill up to large sizes, which hamper the database performance.

In this recipe, we copied the SOA Suite SQL procedures within the WebLogic Admin server referenced by `MW_HOME/SOA_ORACLE_HOME` (on our host, this was `E:\Oracle\Oracle_SOA1`) to the database host containing the SOA schema. We granted suitable privileges to the `dev_soainfra` user to allow the user to run the task, and then set some parameters for the scripts. We set a start and end date for composite creation of `min_creation_date` and `max_creation_date`, and set a cutoff date for the last modification time of a composite (`retention_period`). Any composite that was created in the min/max date bounds, and was last modified before `retention_period`, will be deleted by this script invocation.

We instantiated the delete instances procedure (`soa.delete_instances_in_parallel`) that runs in parallel. This will speed up the deletion of composite data. The `DOPS` parameter determines how many parallel invocations will be started. A looping script is also available.

A number of parameters exist; we recommend setting `max_runtime` to prevent the script from hanging when running, or from users thinking that a long running process has hung. The full list of parameters is available at http://docs.oracle.com/cd/E29505_01/admin.1111/e10226/soadmin_partition.htm.

There's more...

Out-of-the-box, there are limitations on the purge scripts, such as not purging the LOB segments in the `SOA_DEV_SOAINFRA` schema. As such, these scripts can be used as a starting point to build out purge scripts targeting your environment.

The purge script can be changed to target only certain composites, or alternatively other SOA Suite repositories, such as BAM and Mediator.

Separately, the SOA Suite database tables support partitioning to allow for the DEV_ SOAINFRA schemas to be hosted across multiple databases, splitting the storage costs and reducing the processing times. Partitioning databases is beyond the scope of this book and the expected skills of the reader; a database administrator can consult the Oracle Fusion Middleware documentation for more information.

See also

- The *Tuning Oracle database parameters* and *Tuning the Oracle database* recipes

Ensuring automatic database statistics gathering is enabled

As with many SOA Suite components, the BPEL and BPMN engines make a lot of use of the database. Ensuring that the database statistics are updated regularly, will ensure that the optimizer picks the correct plan for accessing the underlying database tables.

Getting ready

You will need database administration access to the database being used for your Oracle SOA Suite 11g applications.

How to do it...

Follow these steps to ensure the database statistics generation task is enabled:

1. Log in to the database, and open an SQL prompt.
2. Execute the following SQL query:

```
exec dbms_auto_task_admin.enable;
```

How it works...

The database optimizer uses the most recent database statistics to determine its plan (index lookup, table scan, and so on) for each SQL statement that is executed. Having up-to-date statistics is therefore vital to ensure that the correct plan is selected, and that the database operates as fast as possible. Oracle recommends that automatic statistics gathering is used for databases that run SOA Suite 11g applications. By default, automatic statistics gathering is enabled, but the preceding steps will enable it if it has been disabled for any reason.

The automatic statistics generation looks for tables or schemas where the content has changed by more than 10 percent, and reruns the statistics generation procedures over these. You can see the database statistics in the `USER_TAB_MODIFICATIONS` table.

There's more...

It is possible to manually update the database statistics, and to disable automatic statistics updates. This might be appropriate if the usage of the database changes drastically over the course of a day. To manually gather statistics, you should first disable automatic statistics gathering, by executing the following query:

```
exec dbms_atuo_task_admin.disable;
```

Then, schedule tasks to execute the `GATHER_SCHEMA_STATS` and `GATHER_TABLE_STATS` procedures on a periodic basis.

See also

- ▶ The *Purging data from the BPEL store* recipe in this chapter
- ▶ The *Tips on tuning data sources* recipe in *Chapter 7, Data Sources and JMS*

Tuning Oracle database parameters

In this recipe, we'll tune the database schema that backs our SOA Suite processes. We'll also make some changes to the global database parameters.

Getting ready

We'll need access to the backing SOA Suite Oracle database, which needs to be Version 11g r2 or higher. To execute commands, we'll need to be able to log in with the `SYSDBA` role.

How to do it...

Follow these steps to set global database parameters. Before proceeding, ensure that the database is not processing production load:

1. Navigate to the database install location (known as `$ORACLE_HOME`), and log in to the database:

```
bin/sqlplus / as sysdba
```

2. You will see an output similar to the following code snippet:

```
SQL*Plus: Release 11.2.0.1.0 Production on Sun Jan 6 12:38:37 2013
Copyright (c) 1982, 2009, Oracle. All rights reserved.
Connected to:

Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit
Production

With the Partitioning, OLAP, Data Mining and Real Application
Testing options
```

3. Run the following commands:

```
ALTER SYSTEM SET SGA_TARGET=15G SCOPE=SPFILE;
ALTER SYSTEM SET SGA_MAX_SIZE=15G SCOPE=SPFILE;
ALTER SYSTEM SET PGA_AGGREGATE_TARGET=5G SCOPE=SPFILE;

ALTER SYSTEM SET PROCESSES=2500 SCOPE=SPFILE;
ALTER SYSTEM SET SESSIONS=3772 SCOPE=SPFILE;
ALTER SYSTEM SET TRANSACTIONS=4150 SCOPE=SPFILE;

ALTER SYSTEM SET OPEN_CURSORS=1500 SCOPE=SPFILE;
ALTER SYSTEM SET CURSOR_SHARING='SIMILAR' SCOPE=SPFILE;

ALTER SYSTEM SET SHARED_POOL_SIZE=300 SCOPE=SPFILE;
ALTER SYSTEM SET SESSION_CACHED_CURSORS=750 SCOPE=SPFILE;

ALTER SYSTEM SET AQ_TM_PROCESSES=1 SCOPE=SPFILE;
```

How it works...

This recipe is focused on fundamental changes to the whole database, which is quite a far-reaching change. Our aim is simply to ensure that there is enough memory and network connections available to service all of the concurrent threads within WebLogic that represent our SOA Suite processes executing database queries, as each one will require a session in the database.

SGA_TARGET represents all buffered memory available to the system for caching, streaming of data, connections to Java middleware, and so on. It is recommended to set this to a high value to support the concurrent nature of SOA Suite components. A good starting value is around 45 percent of system memory. On the 32-GB test system used in this recipe, it is around the 15-GB mark.

The `SGA_MAX_SIZE` value should be set to the same value as `SGA_TARGET`, so we set it to match the 45 percent of available system memory to start with.

`PGA_AGGREGATE_TARGET` is a parameter that tells the database memory manager how much memory across all running processes should be shared. This helps the database to keep only so much memory private to each running process. A good starting value is a third of the `SGA_TARGET` (5 GB), but check with the database administrator before changing this further!

The `PROCESSES` value is changed to a higher number, as this correlates to the number of processes that can connect to the database. We change this value to 2500 as a starting value, but note that there will need to be sufficient memory to create these on demand. Changing the value of `PROCESSES` alters the default value of `SESSIONS` and `TRANSACTIONS` also.

`SESSIONS` correlates directly to the number of users that can log in at one time. It is derived from the number of `PROCESSES`. The default value is a good starting value—it is calculated as the value of `PROCESSES` multiplied by 1.6 added to 22. So, we set it to 3772. Along with `PROCESSES`, a higher value consumes more SGA memory.

`TRANSACTIONS` should be modified to reflect the expected number of in-flight concurrent database transactions. By default, this is the number of `SESSIONS` multiplied by 1.1—as this allows some overhead for nested transactions. We leave this value to tally with the default, which calculates to be 4150, but if you expect a large number of transactions, this is a candidate to increase. Note that this will consume memory from the SGA area.

The `OPEN_CURSORS` parameter determines how many active SQL statements can be active at any one time. If there are calls using the EIS DB adapter, AQ message processing, BPEL processes, or component trace or logging, then each of these will require a cursor. We changed the default from 300 to 1500, which is quite an increase. This will allow a significant number of concurrent connections; tally this number with the connections allowed in the adaptors and data source connection pools.

Setting `CURSOR_SHARING` to `similar` from the default of `exact` allows SQL statements to re-use similar cursors rather than having to match exactly. This reduces cursor contention, allowing more in-memory at one time. The trade-off for this more efficient memory usage is that there will be greater CPU utilization on the database, as it spends more time marshaling data in and out of the cursors that are being re-used. Observe whether the application is CPU bound at the database before making this change.

`SHARED_POOL_SIZE` controls the amount of memory used for storing the Oracle library cache; it is important for caching stored procedures and in systems that make high use of shared cursors. Increasing this can help SOA Suite if composites repeat the same SQL often in different threads. A good starting value is to increase the default from 84 MB to 300 MB.

Increasing `SESSION_CACHED_CURSORS` from 50 to 750 will also help composites that repeatedly call the same SQL, as the database cursor that represents the parsed query will more likely be held in-memory. Note that this will require sufficient SGA memory.

Finally, if you are using AQ messaging, it is important to force that the parameter `AQ_TM_PROCESSES` is enabled by setting it to 1. By default, this is set to 1 when using AQ streams, but setting the initialization value from the default of 0 prevents confusion, as this value will always be honored. This parameter allows the database to auto-tune the number of processes that monitor and control the processing of AQ messages.

It is important to determine a suitable amount of resource that is required at the database level to support the peak loads of each set of deployed SOA Suite components. A good approach is to measure the database memory usage during load testing and revise your settings by using the method in this recipe. Do not take the values in this tip as applicable to all situations!

We do need to caveat this tip; Oracle database tuning is an intricate and broad subject. These changes should be supplemented with knowledge and expertise from your DBA. Do not simply apply them without measuring for improvement!

Tuning the Oracle database

This recipe looks at tuning advanced database options that affect its global behavior. We'll cover changing two items, namely undo space and redo logs.

Getting Ready

You will need access to the backing SOA Suite Oracle database, which needs to be Version 11g r2 or higher. To execute commands, you'll need to be able to log in with the SYSDBA role.

How to do it...

This recipe is split into two parts. First, follow these steps to alter the undo space. Before proceeding, ensure that the database is not processing the production load:

1. Navigate to the database install location (known as `$ORACLE_HOME`) and log in to the database:

```
bin/sqlplus / as sysdba
```

2. You will see an output similar to the following code snippet:

```
SQL*Plus: Release 11.2.0.1.0 Production on Sun Jan 6 12:38:37 2013
Copyright (c) 1982, 2009, Oracle. All rights reserved.
Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit
Production
With the Partitioning, OLAP, Data Mining and Real Application
Testing options
```

3. Determine the maximum time a transaction has taken to complete:

```
SQL> select max(maxquerylen) from v$undostat;
```

```
MAX (MAXQUERYLEN)
-----
942
```

4. Set this to be the UNDO_RETENTION period, adding on a buffer for headroom, we'll use 1000 as a starting value. Restart the database for this to take effect:

```
SQL> alter system set UNDO_RETENTION = 1000 scope = spfile;
SQL> shutdown immediate
SQL> startup
```

5. Before changing the undo space size for reference, check the current undo space file size and utilization:

```
SQL> select data_file.tablespace_name, sum(data_file.bytes)/
(1024*10240) "Allocation (MB)", sum(s.bytes)/(1024*10240) "Usage
(MB)" from dba_data_files data_file, dba_segments s where data_
file.tablespace_name like 'UNDO%' and data_file.tablespace_name =
s.tablespace_name group by data_file.tablespace_name;
TABLESPACE_NAME      Allocation (MB) Usage (MB)
-----
UNDOTBS1              470          5.64375
```

6. Change undo space to be an extending tablespace with an initial size of 2 GB, which grows in increments of 256 MB and has a fixed ceiling value of 20 GB:

```
SQL>create undo tablespace UNDOTBS2 datafile '/opt/oracle/
database/oradata/orcl/undotbs2_01.dbf' size 2048m autoextend on
next 256m maxsize 20480m;
Tablespace created.
```

7. Swap undo space to the new file:

```
SQL> alter system set undo_tablespace= UNDOTBS2 scope=both;
System altered.
```

8. Remove the old undo tablespace file:

```
SQL> drop tablespace UNDOTBS1 including contents;
Tablespace dropped.
```

If this command fails, you will need to wait until transactions running in the old undo tablespace are removed.

9. Observe the change:

```
SQL> select data_file.tablespace_name, sum(data_file.bytes)/
(1024*10240) "Allocation (MB)", sum(s.bytes)/(1024*10240) "Usage
(MB)" from dba_data_files data_file, dba_segments s where data_
file.tablespace_name like 'UNDO%' and data_file.tablespace_name =
s.tablespace_name group by data_file.tablespace_name;
```

TABLESPACE_NAME	Allocation (MB)	Usage (MB)
-----	-----	-----
2048	.125	UNDOTBS2

Secondly, perform the following steps to set the redo log size. Before proceeding, ensure that the database is not processing the production load:

1. Investigate the number of redo logs presently on the system. On the following system, there are three; others will likely have a different amount:

```
SQL> select GROUP#, member from v$logfile;
```

GROUP#	MEMBER
-----	-----
4	/opt/oracle/database/oradata/orcl/redo04.log
5	/opt/oracle/database/oradata/orcl/redo05.log
6	/opt/oracle/database/oradata/orcl/redo06.log

2. In this step, I have another disk on mount point `/mnt/t1`; we'll use this to host my new redo logs. If your system does not have an external disk, you can use a local folder location:

```
SQL>alter database add logfile group 1 '/mnt/t1/oradata/orcl/
redo01.log' size 500M reuse;
```

Database altered.

```
SQL>alter database add logfile group 2 '/mnt/t1/oradata/orcl/
redo02.log' size 500M reuse;
```

Database altered.

```
SQL>alter database add logfile group 3 '/mnt/t1/oradata/orcl/
redo03.log' size 500M reuse;
```

Database altered.

```
SQL>alter database add logfile group 7 '/mnt/t1/oradata/orcl/
redo07.log' size 500M reuse;
```

Database altered.

3. Check that they are added correctly:

```
SQL> select GROUP#, member from v$logfile;
```

GROUP#	MEMBER

	/mnt/t1/oradata/orcl/redo01.log
4	/opt/oracle/database/oradata/orcl/redo04.log
5	/opt/oracle/database/oradata/orcl/redo05.log
6	/opt/oracle/database/oradata/orcl/redo06.log
	/mnt/t1/oradata/orcl/redo02.log
3	/mnt/t1/oradata/orcl/redo03.log
7	/mnt/t1/oradata/orcl/redo07.log

4. Check if they are accessible (the group numbers match the entries in v\$logfile):

```
SQL>select GROUP#, BYTES, ARCHIVED, STATUS from v$log;
```

GROUP#	BYTES	ARC	STATUS

524288000	YES	UNUSED	
2	524288000	YES	UNUSED
3	524288000	YES	UNUSED
4	367001600	NO	INACTIVE
5	367001600	NO	CURRENT
6	367001600	NO	INACTIVE
7	524288000	YES	UNUSED

5. Rotate the logs four times to bring the unused ones online (by making them active):

```
SQL>alter system switch logfile;
System altered.
SQL>alter system switch logfile;
System altered.
SQL>alter system switch logfile;
System altered.
SQL>alter system switch logfile;
System altered.
```

6. This is shown by repeating the select on the log view:

```
SQL>select GROUP#, BYTES, ARCHIVED, STATUS from v$log;
```

GROUP#	BYTES	ARC	STATUS
-----	-----	---	-----
1	524288000	NO	ACTIVE
2	524288000	NO	ACTIVE
3	524288000	NO	ACTIVE
4	367001600	NO	INACTIVE
367001600	NO		ACTIVE
6	367001600	NO	INACTIVE
7	524288000	NO	CURRENT

7. Force a checkpoint to write all contents to data files:

```
SQL>alter system checkpoint;
```

8. Drop the old log files:

```
SQL>alter database drop logfile group 4;
```

```
SQL>alter database drop logfile group 5;
```

```
SQL>alter database drop logfile group 6;
```

9. Check the changes:

```
SQL> select GROUP#, BYTES, ARCHIVED, STATUS from v$log;
```

GROUP#	BYTES	ARC	STATUS
-----	-----	---	-----
1	524288000	NO	INACTIVE
2	524288000	NO	INACTIVE
3	524288000	NO	INACTIVE
7	524288000	NO	CURRENT

How it works...

An Oracle database will use a marked area of the database for ensuring data consistency. This is important in SOA Suite, where we use multiple concurrent transactions with the potential for timeouts and rollbacks in communications with disparate systems. The database calls this area undo space (it is actually a tablespace for those who want to know more about Oracle naming conventions). In this area is stored all of the information required for recovery and rollback of database transactions (it is also for the Flashback feature of the Oracle database, but this is beyond the scope of this book). Data is retained in the undo space for long enough to "undo" inconsistencies, or until the undo data expires. In the first half of this recipe, we altered this to be much bigger for SOA Suite.

Dependent on the load that we wish to apply in terms of concurrent processing, we might need a larger undo space to store all of this concurrent information (synchronous BPEL with logging is particularly heavy on database writes, therefore has more transactions). Since Oracle database version 11g, undo space is, by default, automatically managed for us; if undo space is too small, it will automatically extend but will never automatically shrink! The `V$UNDOSTAT` database view contains the statistics on undo space that serve as a guide to automatic tuning, and we can manually access this information. Typically, to ensure data consistency, the retention time for undo data is set to be the longest transaction time. As this will be applied to all undo data, we can imagine that if even only one of our transactions is long-running, we will unfortunately need a large amount of space to cope with lots of concurrent transactions.

Sizing undo space manually prevents slowdown should we run out of undo space. By default, undo space will automatically increase in size. We also apply a maximum size to prevent it from growing larger than our system can handle (which could prevent logins to the database host). Further to that, we increased the increment to use when growing, to prevent lots of small increases in the undo space file size. A good starting point for undo space is to start with a large value and shrink it down to a suitable value. This requires that we monitor the undo space utilization to ensure that the sizes are relevant to your production SOA Suite load. We can now monitor the utilization by using the query in this recipe.

In the second part of this recipe, we altered the redo log sizes. Redo logs are used by the Oracle database to capture all the changes that occur in our data in-flight. This allows the database to rebuild our data content in the event of a system crash.

Typically, the database writes to a list of log files, completely filling a log file marked as `current`, before moving on to the next file in the list of "active": files. When the active list is exhausted, it returns to the first active element, always marking the current log as the one currently in use (note that it is possible to have inactive logs in the list, but they will not be used). When logs are rotated, a checkpointing system is engaged to ensure data is not lost; this can be costly in terms of time for applications, causing timeouts and retries of user inputs. For these reasons, there are two items of importance when ensuring there is adequate redo log capacity for our system—file size and number of redo log files.

In this recipe, we placed our redo logs on a separate disk; this can greatly help with concurrent disk I/O on busy systems, but may not be applicable in all environments. Do this if possible, or just use a local disk. We also added a series of files that were both larger in size and larger in number than the previous list of redo logs. Larger files will help with system capacity, as more can now happen in between checkpoint operations.

There's more...

Redo logs can be set to a checkpoint and rotated on a time-based frequency. Enlist a DBA to check this as it can incur extra overhead when switching between logs.

If you're working with an Oracle RAC instance, then ensure you check with an Oracle DBA how to configure the steps in the recipe across all the nodes. Bear in mind the following points:

- ▶ Each node has its own redo log (typically referred to as a **redo thread**). These work in much the same way as in a RAC instance, even when you only have one database being presented by all the nodes. This model allows for the removal of a single point of contention that would exist if we only had one redo log.
- ▶ Each node will also have its own undo space.

There are a lot of other changes that can be made to improve the performance of SOA Suite, such as indexing the content and compressing/purging old data. An Oracle DBA will be able to aid us with these settings on standalone instances and RAC nodes.

See also

- ▶ The *Ensuring automatic database statistics gathering is enabled* and *Tuning Oracle database parameters* recipes
- ▶ The *Tuning database XA timeouts* recipe in *Chapter 7, Data Sources and JMS*

9

Mediator and BAM

In this chapter, we will look at the following recipes:

- ▶ Setting Mediator Parallel Metrics Level
- ▶ Setting Mediator Parallel Worker Threads
- ▶ Setting Mediator Parallel Maximum Rows Retrieved
- ▶ Setting Parallel Locker Thread Sleep
- ▶ Using batched delivery in the BAM Adapter

Introduction

The **Mediator** engine and the **Business Activity Monitoring (BAM)** connector are used less frequently than the BPMN and BPEL engines, but there are still performance considerations when using these technologies. The Mediator engine is a relatively simple component, yet it works in quite a complex way, giving us only a few options for tuning it. These options are all focused on the execution of parallel routing rules, as sequential routing rules run on a single thread, and keep that thread until the rule has completed executing. Parallel rules, on the other hand, are executed by retrieving a list of requests from the database, and then processing each request using a set of worker threads. These threads allow the requests to be processed in parallel.

The BAM Adapter does not have many properties that can be tuned, but we do look at one improvement here that can improve performance.

Setting Mediator Parallel Metrics Level

If you make use of DMS (discussed in *Chapter 2, Monitoring Oracle SOA Suite*), then collecting the DMS metrics from the Mediator engine adds an unnecessary overhead.

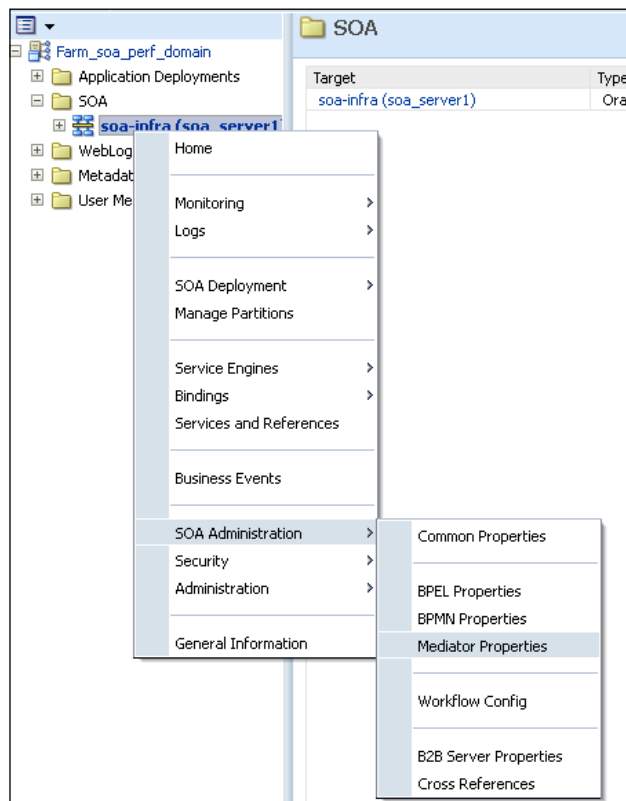
Getting ready

You will need to know the administration credentials for your SOA Suite 11g domain, and the URL for the Enterprise Manager console.

How to do it...

To set the `Parallel Metrics Level` property, perform the following steps:

1. Log in to the Enterprise Manager.
2. Expand the **SOA** section in the left-hand pane, and right-click on **soa-infra**.
3. Now, select **SOA Administration | Mediator Properties**.



4. You will now see the SOA properties page.

[SOA Infrastructure Home](#) > Mediator Properties

Mediator Service Engine Properties ?

Properties

Edit property values and click Apply to save the changes.

Audit Level	<input type="text" value="Inherit"/>
Metrics Level	<input type="text" value="Enabled"/>
Parallel Worker Threads	<input type="text" value="4"/>
Parallel Maximum Rows Retrieved	<input type="text" value="200"/>
Parallel Locker Thread Sleep(sec)	<input type="text" value="2"/>
Error Locker Thread Sleep(sec)	<input type="text" value="5"/>
Parameters	<input type="text"/>
Container ID Refresh Time(sec)	<input type="text" value="60"/>
Container ID Lease Timeout(sec)	<input type="text" value="300"/>
Resequencer Locker Thread Sleep(sec)	<input type="text" value="10"/>
Resequencer Maximum Groups Locked	<input type="text" value="4"/>
Resequencer Worker Threads	<input type="text" value="4"/>

[More Mediator Configuration Properties...](#)

5. Set the **Metrics Level** property to **Disabled**.

How it works...

By default, the Mediator engine will collect metrics from the execution of composite instances, and make them available via the DMS. If you do not make use of these metrics, then there is little point collecting them, and by removing them, performance can be improved.

See also

- The *Monitoring the system using the DMS servlet* recipe in *Chapter 2, Monitoring Oracle SOA Suite*

Setting Mediator Parallel Worker Threads

By increasing the number of parallel worker threads, environments with large workloads can process more requests at the same time.

Getting ready

For more details, refer to the *Getting ready* section of the *Setting Mediator Parallel Routing Rules* recipe.

How to do it...

To set the `Parallel Worker Threads` property, perform the following steps:

1. Follow the steps from 1 through 3 from the *Setting Mediator Parallel Metrics Level* recipe.
2. Set the property `Parallel Worker Threads` to a larger value, such as 10.

How it works...

The `Parallel Worker Threads` property determines the number of threads available for processing parallel routing rules. If you have many parallel routing rules defined in your application, then the default value of 4 may not be sufficient; so you can increase the number of threads to allow more parallel rule requests to be processed concurrently.

See also

- ▶ The *Setting Parallel Locker Thread Sleep* recipe
- ▶ The *Setting Mediator Parallel Maximum Rows Retrieved* recipe

Setting Mediator Parallel Maximum Rows Retrieved

By increasing the number of rows retrieved from the Mediator store table, performance can be improved in high demand systems.

Getting ready

For more details, refer to the *Getting ready* section of the *Setting Mediator Parallel Metrics Level* recipe.

How to do it...

The following steps will allow us to set the `Parallel Maximum Rows Retrieved` property:

1. Follow the steps from 1 through 3 from the *Set Mediator Parallel Metrics Level* recipe.
2. Set the property `Parallel Maximum Rows Retrieved` to a larger value, such as 500.

How it works...

The `Parallel Maximum Rows Retrieved` property determines how many rows are read from the Mediator store database when loading pending parallel rule execution requests. By loading more rows at a time, the number of trips to the database can be reduced, improving performance; however, this will only have an effect if there are usually multiple requests waiting to be processed. The default is 200 rows, so you will need to have a busy system with lots of parallel Mediator rules, in order to see any improvement from increasing this value.

See also

- ▶ The *Setting Parallel Locker Thread Sleep* recipe
- ▶ The *Setting Mediator Parallel Worker Threads* recipe

Setting Parallel Locker Thread Sleep

By reducing the amount of time that the engine sleeps between checking for parallel composite instances that need executing, performance can be improved.

Getting ready

For more details, refer to the *Getting ready* section of the *Setting Mediator Parallel Metrics Level* recipe.

How to do it...

To set the `Parallel Locker Thread Sleep` time, perform the following steps:

1. Follow the steps from 1 through 3 from the *Set Mediator Parallel Metrics Level* recipe.
2. Set the `Parallel Locker Thread Sleep` property to 1.

How it works...

While parallel routing rules do allow for truly asynchronous processing in the Mediator, the way that they are processed is not intuitive. A daemon thread is used to check the Mediator store database for requests to be processed, and checks each parallel rule in turn. If there are requests to be processed for a rule, then exactly one request will be processed for that rule using a Mediator worker thread. After it has checked all the parallel rules (and executed one request for each that has pending requests), the thread will sleep for a period of seconds defined by the `Parallel Locker Thread Sleep` time. By default this value is 2 seconds, but performance can be improved if you are using parallel routing rules by decreasing this to 1 second (the lowest value allowed). Note that the daemon thread does not execute the requests; it is just used to check which rules have pending requests, and assign them to worker threads.

This seemingly strange approach is used to prevent a very busy parallel rule from hogging all of the worker threads, thus starving other rules of the threads they need, and potentially causing performance bottlenecks.

See also

- ▶ *The Setting Parallel Mediator Maximum Rows Retrieved recipe*
- ▶ *The Setting Mediator Parallel Worker Threads recipe*

Using batched delivery in the BAM Adapter

By altering the batching settings for the BAM Adapter, you can make fewer calls out to the Active Data Cache Server, which can save bandwidth and improve performance.

Getting ready

This recipe is only useful if your domain makes use of the BAM Adapter. You will need write access to the files in your SOA Suite domain for this recipe.

The configuration files are not created until the server is first started, so you will need to start the server before you are able to edit the files.

How to do it...

The following steps will help us enable batched delivery for the BAM Adapter:

1. Edit the following file:
 ORACLE_HOME/user_projects/domains/\$DOMAIN_NAME/config/
 fmwconfig/servers/bam_server1/applications/oracle-bam_11.1.1/
 config/BAMCommonConfig.xml
2. Locate the BAM batching properties at the top of the file, and increase their values.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<BAMCommon>
  <Adapter_SOAP_Batching_FlushOnDemand_Limit>1000</Adapter_SOAP_Batching_FlushOnDemand_Limit>
  <Adapter_SOAP_Batching_Limit_Lower>1000</Adapter_SOAP_Batching_Limit_Lower>
  <Adapter_SOAP_Batching_Limit_Upper>5000</Adapter_SOAP_Batching_Limit_Upper>
  <Adapter_SOAP_Batching_Timeout>50</Adapter_SOAP_Batching_Timeout>
  <ApplicationURL>http://DEFAULT:0000</ApplicationURL>
  <ChannelName>OracleBAM</ChannelName>
  <Clustered>false</Clustered>
  <GroupName>default</GroupName>
</BAMCommon>
```

We suggest trying the following values:

Property	Default Value	Tuned Value
Batching_Limit_Lower	1000	2500
Batching_Limit_Upper	5000	10000
Batching_Timeout	50	100

3. Save the file, and restart your BAM server.

How it works...

The BAM Adapter sends data to the Active Data Cache server in batches. If you are making heavy use of the BAM Adapter in your SOA Suite application, you can improve performance by increasing the size of these batches. The three properties we tune here are the key ones used to determine the batch size. They can be explained as follows:

- ▶ **Batching_Limit_Lower:** It determines the minimum number of messages in the batch.
- ▶ **Batching_Limit_Upper:** It determines the maximum number of messages in the batch.
- ▶ **Batching_Timeout:** It determines how long we have to wait before sending the batch if we have not reached either of the other two limits.

There's more...

The BAM Adapter batching system has retry logic built into it that will attempt to resend batches of messages if the Active Data Cache server is down, or some other failure occurs when sending a batch of messages. These properties can also be tweaked in order to configure this retry behavior, including changing the time between retries, and determining the number of pending batches before calls to the BAM Adapter start blocking. These properties are discussed in more detail in the Oracle BAM Adapter configuration guide at http://docs.oracle.com/cd/E23943_01/admin.1111/e10226/bam_config.htm#SOAAG9931. As the retry logic is focused primarily on resilience rather than performance, we will not discuss it in detail here.

10

Rules and Human Workflow

In this chapter, we will cover the following recipes:

- ▶ Disabling automatic release timers globally
- ▶ Choosing the correct workflow service client
- ▶ Preventing looping and inefficient rule executions
- ▶ Tuning rule execution

Introduction

The rules and the human workflow components of the SOA Suite provide fewer options for tuning than some of the other components such as BPEL and BPMN, but there are still a number of things to be aware of to ensure that you get the best performance possible. This chapter looks at some of the options available for these components.

Both the human workflow component and the rules component of the Oracle SOA Suite are built on top of the WebLogic server application stack, and the Java virtual machine. This means that for general performance problems, your first step should be to diagnose and resolve any problems that are occurring at the JVM or WebLogic layers, before looking at the recipes in this chapter that focus on tuning the human workflow and the rules components.

The workflow components in particular can make a lot of use of the underlying database schema, and as such, the recipes in *Chapter 8, BPEL and BPMN Engine Tuning*, relating to the schema tuning also apply here.

As with the other recipes that deal with the higher level SOA Suite services, performance improvements in the workflow and the rules engines often come with a trade-off, so care should be taken when deciding how to use them. A number of the recipes in this section reference specific features of the SOA Suite, or provide guidance on how to best use a specific feature, and these recipes are obviously useful only if your application already makes use of these features.

Disabling automatic release timers globally

Automatic release timers release workflow items assigned to groups or roles. This can impose an overhead for your application if they are used heavily. Globally disabling them can provide a performance improvement.

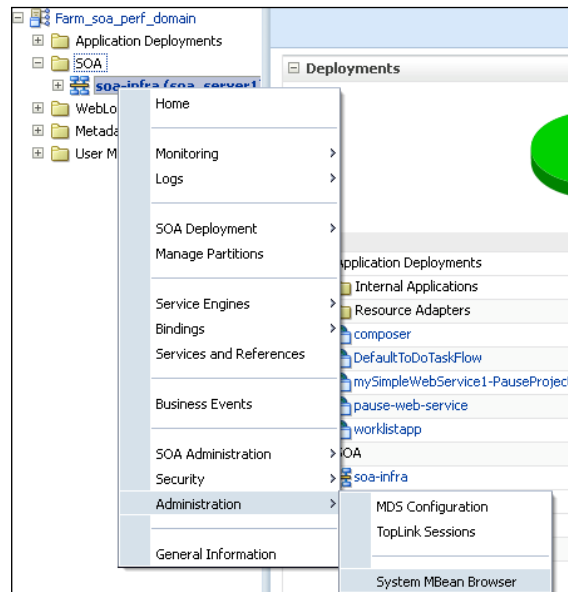
Getting ready

You will need to know the administration credentials for your Oracle SOA Suite WebLogic domain, and have access to the Oracle Enterprise Manager console.

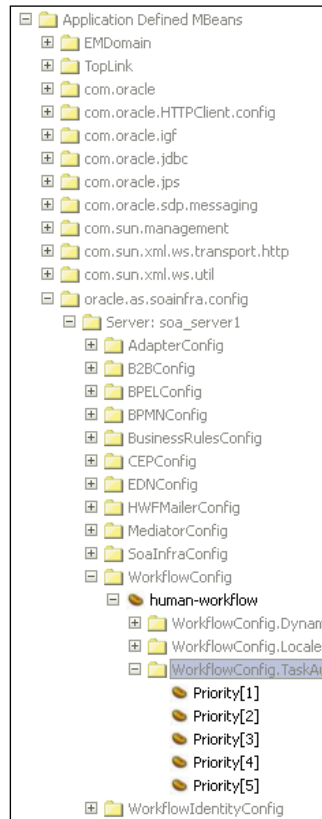
How to do it...

To disable the automatic release timers, perform the following steps:

1. Log in to the Enterprise Manager.
2. Open the **SOA** tab, right-click on **soa-infra**, and select **Administration | System MBean Browser**.



- This will open the **System MBean Browser** window in the main panel. Expand the tree to find **WorkflowConfig.TaskAutoReleaseConfiguration** in **Application Defined MBeans | oracle.as.soainfra.config | Server: soa_server1 | WorkflowConfig | human-workflow**.



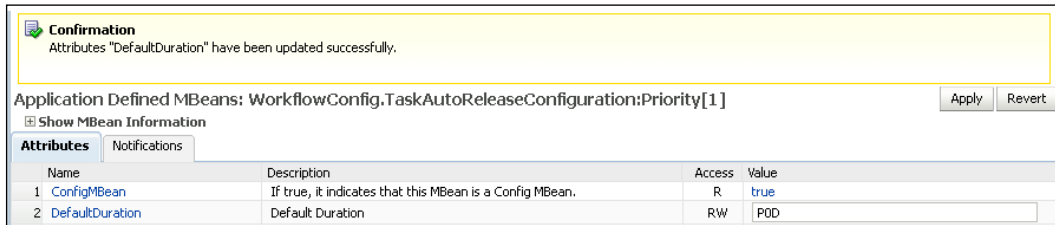
- In the main panel, change the value of **DefaultDuration** to P0D, which represents 0 days, or off.

Application Defined MBeans: WorkflowConfig.TaskAutoReleaseConfiguration:Priority[1] Apply Revert

Show MBean Information

Attributes		Notifications
Name	Description	Access Value
1 ConfigMBean	If true, it indicates that this MBean is a Config MBean.	R true
2 DefaultDuration	Default Duration	RW P1D
3 eventProvider	If true, it indicates that this MBean is an event provider as defined by JSR-77.	R true
4 eventTypes	All the event's types emitted by this MBean.	R jmx.attribute.change
5 objectName	The MBean's unique JMX name	R oracle.as.soainfra.config:name=Priority[1],type=Wor
6 PercentageOfExpiration	Percentage of Expiration	RW 30
7 Priority	Priority	R 1
8 ReadOnly	If true, it indicates that this MBean is a read only MBean.	R false
9 RestartNeeded	Indicates whether a restart is needed.	R false
10 SystemMBean	If true, it indicates that this MBean is a System MBean.	R false

- Click on **Apply**, and a message should indicate that the change was made successfully, as shown in the following screenshot:



How it works...

Each claimed task has a timer within which it must be completed before it is released back to the group. If you have many of these tasks, the checking and firing of these timers can create an overhead in both the database and the application server, so disabling them can improve performance, but this means that the tasks are not automatically released.

Priority for each task can be set independently by entering a value of `PxD`, where `x` is the number of days after which you wish the task to be released if it has not been completed. Setting a value of `P0D`, which represents zero days, will disable the timer.

Choosing the correct workflow service client

There are a number of options available when choosing how your client code makes calls to the human workflow service. This recipe discusses some of the trade-offs between them.

How to do it...

The following guidelines are used when selecting a strategy for creating a client that interacts with the workflow service:

- If an application is not implemented in Java, or connects over a WAN or a LAN, then using the SOAP client will be your only option.
- If you are connecting using Java, either from within the same or from a remote JVM, then using the Remote EJB client will be the best option.

How it works...

The workflow service can be accessed using one of three clients: the SOAP client, the remote EJB client, and the local EJB client. The SOAP client uses SOAP over HTTP as its access protocol, so it is perfect for situations where you are talking to a non-Java client, or where you need a protocol that will work through a firewall. The local EJB client can be used when you are using the workflow service from within the same JVM, but cannot be used from an external application. Because of the call optimizations in the remote EJB layer, there is no performance improvement when using the local client above the performance available in the remote EJB client, so we recommend using the remote client. This will mean that if you ever decide to split the client code into a separate application, it will still work correctly.

Any application that queries the workflow for management purposes can have its performance improved by purging obsolete runtime data from the underlying database tables `WF_ITEMS`, `WF_ITEM_ATTRIBUTE_VALUES`, `WF_ITEM_ACTIVITY_STATUSES`, and `WF_NOTIFICATION_ATTRIBUTES`. Oracle provides scripts and further guidance on doing this in a note on workflow purging in a metalink. Furthermore, grouping workflows under roles and users efficiently will allow for less work to be done when retrieving active workflow lists, helping to reduce the impact when the number of active items increases.

Preventing looping and inefficient rule execution

This recipe gives guidance on how to avoid rule executions that will loop, potentially indefinitely! We'll use an inbound XML and a local RL (Rule Language) fact as an example.

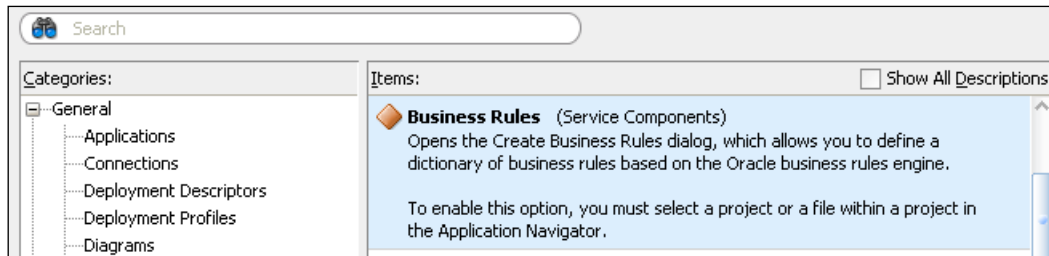
Getting ready

You'll need access to an SOA composite containing an Oracle Business Rules component in JDeveloper to apply this recipe. We'll assume that you have an XSD schema with an input *RequestInput* containing two strings called *input* and *bonus*, and an output *ResponseOutput* containing a string called *output*. These aren't efficient, but will serve as an example. We'll step through adding a rule to a composite and creating an RL fact.

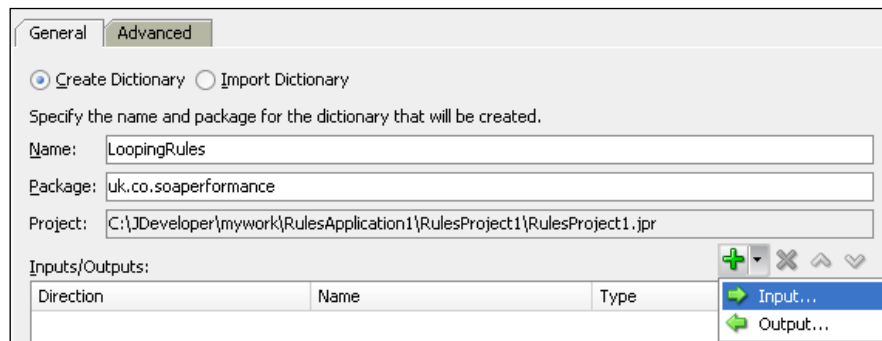
How to do it...

To remove potential circular references, perform the following steps:

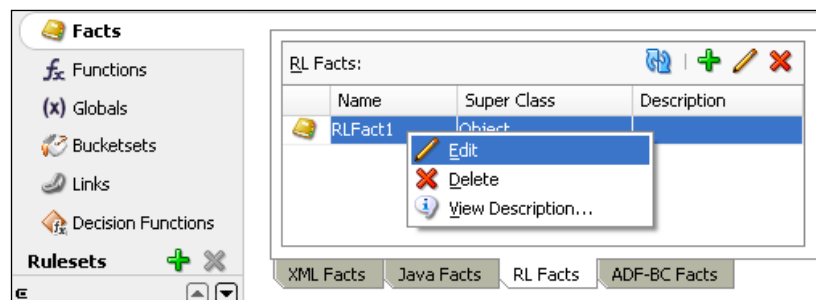
1. Open the SOA composite. Right-click on the project, and select **Business Rules (Service Components)**; use the search box if it is not immediately available.



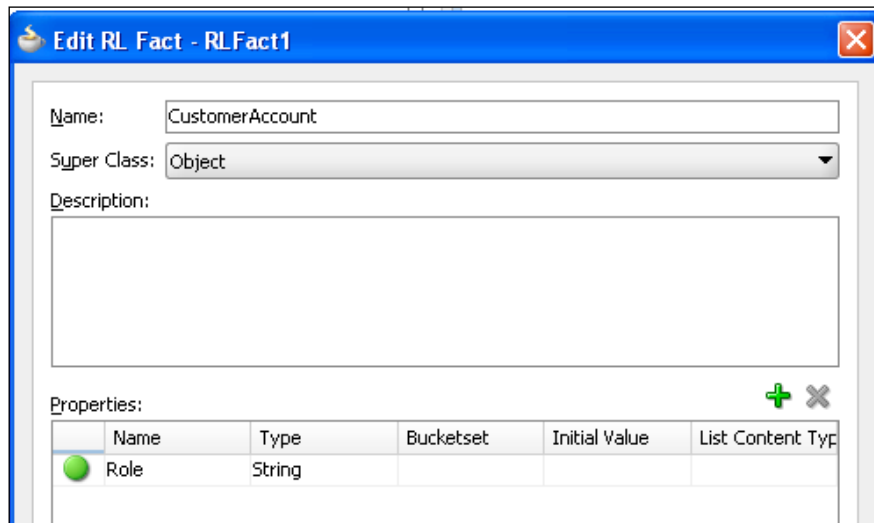
2. Give the rule a name, and click on the green plus (+) icon to add Request Input to the **Input** and ResponseOutput to the **Output** types.



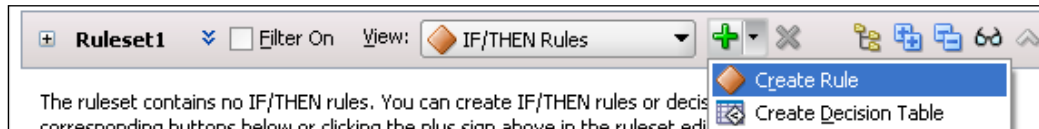
3. In the rules designer, select **Facts**, and then select the **RL Facts** tab. Click on the green plus (+) icon, then right-click on **RLFact1** and select **Edit**.



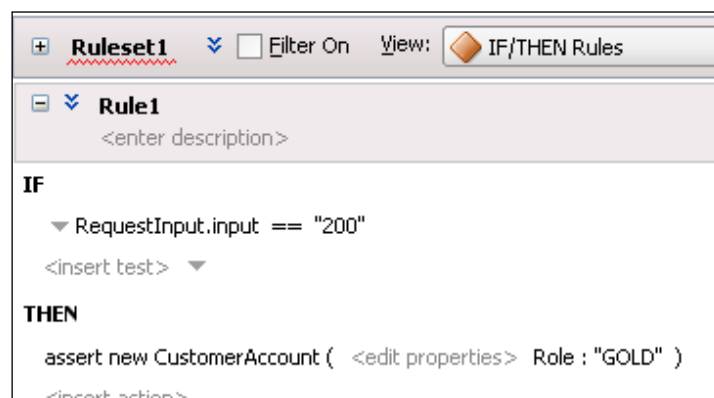
- Name the RL fact `CustomerAccount`, and click on the green plus (+) icon to add a property. Name the property `Role`, and set its type to `String`. Add another property called `Reward` of type `int`.



- Select **Ruleset1** and click on the green plus (+) icon to create a rule.



- In the IF test, set the condition to select `RequestInput.input >= "200"`.
- Set the THEN action to assert `new CustomerAccount (Role: "GOLD")`.



8. Click on the green plus (+) icon in the ruleset to add a second rule. In the IF condition set `CustomerAccount.Role=="GOLD"`.
9. In the IF action, set `modify CustomerAccount.Reward=150`.
10. In the THEN action, set `modify CustomerAccount.Role="SILVER"`.
11. To complete the ruleset so that it will compile, add a third rule to assign a high value to the output of `Customer.Account` (if it is GOLD), and a fourth rule to set it to low value otherwise.

How it works...

In this example, we created a ruleset with rules that checked input XML facts and modified RL facts appropriately before assigning a response to an output XML fact.

While the assignment in step 11 may seem odd, it illustrates an important facet of the rule execution. When actions alter a value that is checked in another condition, in this case `CustomerAccount.Role`, then a rule loop occurs. These loops can occur across a single rule and a decision table, or multiple rules and decision tables in the same ruleset. It would be easy for a business user to inadvertently create a logic pattern that caused this looping by applying an inefficient pattern such as a reward scheme in our example. The loops created by these patterns have the potential to continue indefinitely, ultimately preventing further composites from being able to execute in the rule context.

To avoid this scenario, we should construct our rule logic to try and prevent these scenarios from occurring. Runtime monitoring is helpful in identifying these items when testing.

There's more...

Due to the way in which the rules engine builds its plan and executes rules at runtime, rules may be executed more than once. As such we should avoid performing operations in actions that we don't want to be executed more than once. This particularly holds true if using Java Beans as facts in which it is easy to perform such operations, for instance, calls that perform blocking operations (on database, or disk activity) should be avoided.

See also

- ▶ The *Configuring Hyperic to monitor Oracle SOA Suite* recipe in *Chapter 2, Monitoring Oracle SOA Suite*
- ▶ The *Designing Advanced Load Tests* recipe in *Chapter 3, Performance Testing*

Tuning rule execution

We can control rule ordering execution, which can help improve rule evaluation efficiency or reduce memory.

Getting ready

You'll need access to an SOA composite containing an Oracle Business Rules component in JDeveloper to apply this recipe. For simplicity, we'll presume that the component is empty for this recipe.

How to do it...

To add some rules and tune their configuration, perform the following steps:

1. Open the rules component for editing.
2. Next to the **Rulesets** title on the left, click on the green plus icon (+) to add a new ruleset. Name it `Ruleset2`. This will be the second ruleset in the stack.



3. In `Ruleset2`, add two rules by clicking on the green plus (+) icon, as shown in the following screenshot. You can leave the rule logic blank for this recipe.



- Click on the blue double down arrow icon to show the advanced settings for each rule. In Rule2, set the **Priority** in the selection box by typing in an integer value of 1001, as shown in the following screenshot:



- In Rule1, set the **Priority** to **Medium**.

How it works...

Rules belong to a unit of execution called a **ruleset**. Multiple rulesets are executed in the order determined by the stacking of the rulesets, which is called the rule flow. Finally, within a ruleset, rules fire according to the priority associated with each individual rule. Rules with the same priority can fire in an indeterminate order.

In this recipe we created two rulesets: Ruleset1 and Ruleset2 (note that Ruleset1 will fire first due to the stack ordering). For rules in Ruleset2, the priority setting of 1001 will cause Ruleset2 to fire before Ruleset1. This is because **Medium** priority has a value of 0, and higher integer priorities will be evaluated sooner by the engine (the full range of numerical priority equivalents is in the SOA Suite documentation, and at the time of writing, the highest was 3000, and stepped down in increments of 1000, to the lowest at -3000).

When thinking about the ordering of your rules, there are two main things to consider, as follows:

- ▶ If we order rules such that facts that do not change frequently are evaluated first, then we can have the greatest potential for fast rule execution.
- ▶ If we order rules such that the most restrictive sets of facts are matched first, then this can reduce the memory used during rule evaluation, and in some cases improve throughput.

There's more...

These guidelines also apply to the tests and fact evaluations within individual rules, so careful consideration of what you're evaluating is the key to performance gains here.

In our experience, there is not a great deal of mileage gained by trying to write compact decision tables over rulesets that contain multiple if/else clauses, as these are executed in the memory. The main improvement with choosing a decision table is in its readability.

11

SOA Application Design

This chapter looks at the ways in which you can design your application for high performance. The topics that will be covered are:

- ▶ Using BPEL process parallelization
- ▶ Using non-blocking service invocations in BPEL flows
- ▶ Turning off payload validation and composite state monitoring
- ▶ Designing BPEL processes to reduce persistence
- ▶ Using parallel routing rules
- ▶ Setting HTTP timeouts for external HTTP services
- ▶ Tuning BPEL adapter properties

Introduction

In this chapter, we'll focus on recipes for designing high performance SOA Suite 11g applications. These recipes look at how you can design your applications for high performance and scalability, where high performance is defined as providing low response times even under load, and scalability is defined as the ability to expand to cope with large numbers of requests.

While many of the recipes in other chapters can be applied after the application has been designed and written, those in this chapter need to be applied while the application is being written, and may require that your application is implemented in a certain way. Designing an application with performance as a requirement from the start is much easier than trying to add performance to an application that is already live. So, the recipes in this chapter provide some of the best value for money in terms of getting the most performance out of your SOA Suite infrastructure. However, while this book focuses on decisions that should be made during the design stages of a development process, this chapter is not a list of general SOA Suite design patterns.

As for many of the recipes in other chapters, a lot of the focus in this chapter is on reducing the amount of time your application spends waiting on external services and the SOA Suite database tables.

There are many aspects to the performance of a SOA Suite application, and the design guidelines depend very much on the particular business problems that your application is designed to solve. Factors such as payload size, number of external systems being orchestrated, data transformation complexity, and persistence requirements, all have an impact on the performance of your application. Performance is a relative term, with each application and use-case having its own requirements, but there are a number of basic principles that can help ensure that your application will have a good chance of meeting its goals.

- ▶ Design for peak loads, not average loads. Average loads can be very misleading; there are many situations in which the average load of a system is not a good indicator of the expected load. A good example of this would be a tax return system, where the usage for most of the year is very low, building into a peak in 30 or so days before people's tax returns are due.
- ▶ Smaller payloads are faster. When designing your application, try and limit the amount of payload data that goes through your composites and processes. It is often better to store the data in a database and send the key and metadata through the processes, only going to retrieve data when required.
- ▶ Understand your transaction boundaries. Many applications suffer performance problems because their transactions boundaries are in the wrong places, causing work to be redone unnecessarily when failures happen, or leaving data in an inconsistent state.
- ▶ Understand what causes your application to access the database, and why. Much of the performance overhead of Oracle SOA Suite applications is in repeated trips to the database. These trips add value by persisting state between steps or within processes, but the overuse of steps that cause database persistence is a common cause of performance problems.
- ▶ Follow standard web service design patterns, such as using asynchronous callbacks and stateless invocations, where you are using web services.

Using BPEL process parallelization

By having your BPEL process execute steps in parallel when there are no dependencies, you can increase the performance by spending less time waiting for external systems to complete.

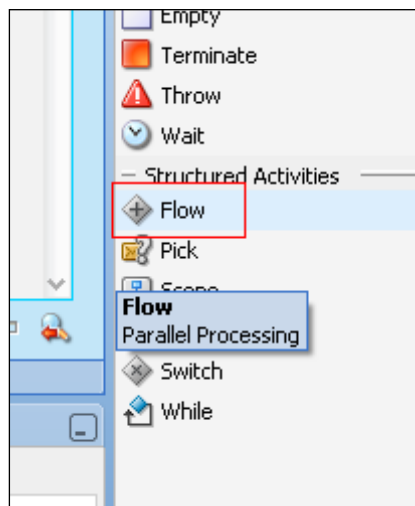
Getting ready

You will need JDeveloper installed, and have an open BPEL project.

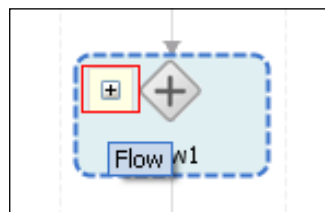
How to do it...

Follow these steps to use BPEL process parallelization:

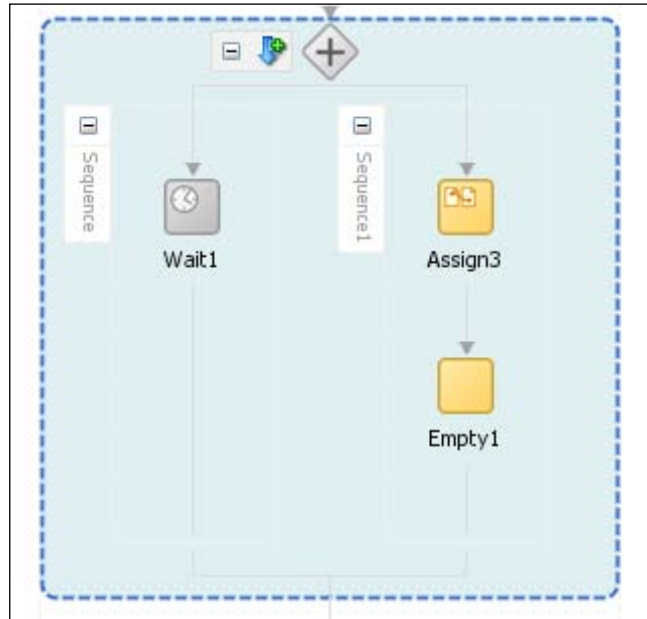
1. Expand the **BPEL Constructs** section in the component palette.
2. Drag **Flow** from the palette onto the process.



3. Click on the + icon next to the flow to expand it.



4. Populate the flow with the process steps.



How it works...

If you have a number of tasks that do not have dependencies on each other, you can improve performance by executing the preceding tasks in parallel. This is most effective with partner links, where you know you are waiting on an external system to produce a response. The default behaviors of these flows is still to use a single thread to execute the branches if external systems are invoked. See the *Using non-blocking service invocations in BPEL* recipe to learn how to execute flows that contain partner links in parallel.

There's more...

It is possible to include a limited amount of synchronization between branches of a flow, so that tasks on one branch will wait for tasks on another branch to complete before proceeding. This is best used with caution, but it can provide benefits, and allow tasks that would not otherwise easily lend themselves to parallelization to be run in parallel.

See also

- The *Using non-blocking service invocations in BPEL flows* recipe

Using non-blocking service invocations in BPEL flows

We can reduce the latency of forked external service invocations in a BPEL process to the longest flow's execution time if we assign a thread to each flow, making it multi-threaded.

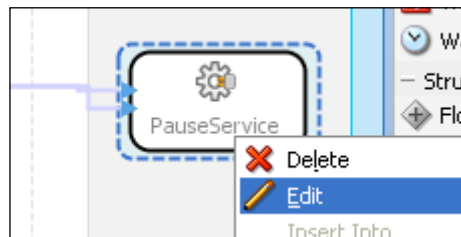
Getting ready

You'll need a composite loaded in JDeveloper to execute this recipe. This composite will need a flow that makes calls to a partner link external service.

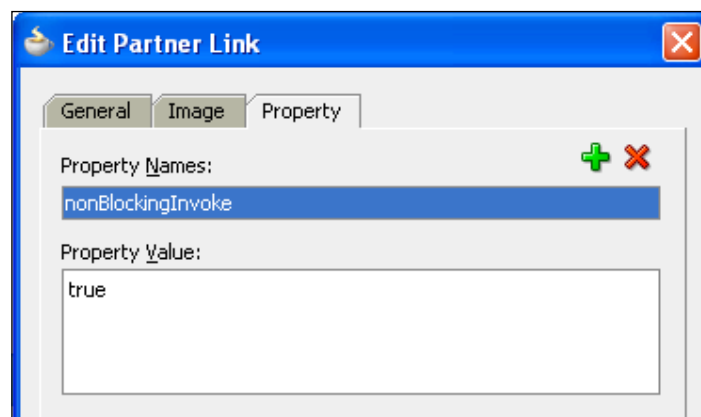
How to do it...

Follow these steps to use non-blocking service invocations:

1. Right-click on each partner link that is being executed in your BPEL process flow, and select **Edit**.



2. In the **Property** tab, select the green + icon and add **nonBlockingInvoke** as a property name. In the **Value** box at the bottom, enter `true`.



How it works...

This recipe causes flow branches to be executed in parallel, with a new thread to be used for each branch flow.

For multiple service invocations that each have a high latency, this can greatly improve the total BPEL execution time. For example, assume we have a BPEL process that calls two web services, one that takes four seconds to execute, and one that takes six seconds to execute. Applying this change will prevent the BPEL process making the calls serially, which would take 10 seconds in total, and enforce parallel service calls in separate threads, reducing the execution time to just over six seconds, or the latency of the longest call plus time to collate the results in the main BPEL process execution thread.

While it may sound like a silver bullet performance improvement, this recipe is actually not necessarily going to improve the execution time of our BPEL process! Consider that we may now be at the mercy of greater thread context switching in the CPU; for every invocation of our process, we now have a larger number of threads that will be spawned. If each service invocation has a low latency, the overhead of creating threads and collating callbacks might actually be greater than the cost of invoking the services in a single thread. Our example in this explanation is contrived, so ensure to test the response time of your composite and the profile of your application, when placed under operational load (which may result in lots of threads spawning), as these may well be different with the configuration applied.

There's more...

This recipe used an alternative way of setting property values to that which we've used elsewhere in the book. Previously, we've edited composite files directly; here, we used the JDeveloper BPEL graphical editor to achieve the same end result. If you check the `composite.xml` source, you'll see a property added with a name, such as `partnerLink.[your service name].nonBlockingInvoke` for each service added.

See also

- ▶ The *Using BPEL process parallelization* recipe
- ▶ The *Changing a BPEL process to be transient* recipe in *Chapter 8, BPEL and BPMN Engine Tuning*

Turning off payload validation and composite state monitoring

Payload validation checks all inbound and outbound message data thus adding an overhead, especially for large message types. Composite state monitoring allows for administrators to view the results of all instance invocations. We can disable these to improve performance.

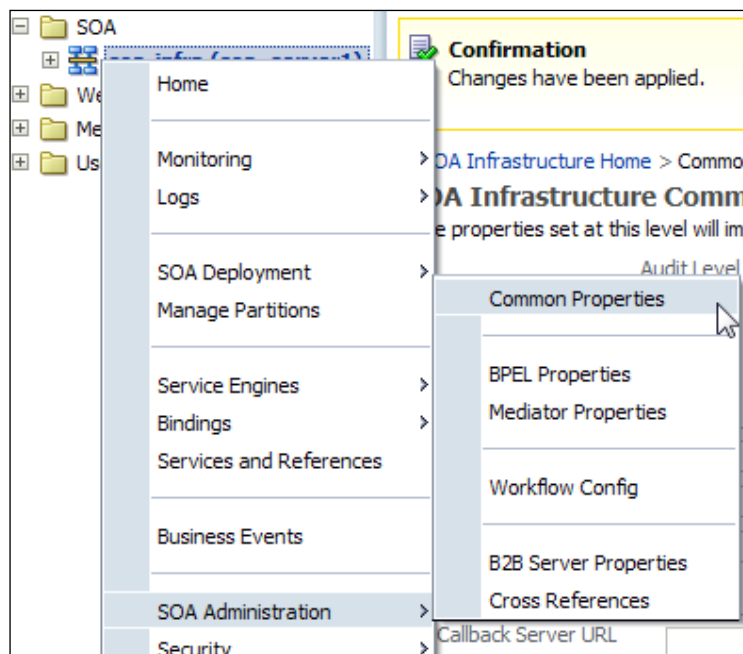
Getting ready

You will need to know the administration credentials for your Oracle SOA Suite WebLogic domain, and have access to the Oracle Enterprise Manager console.

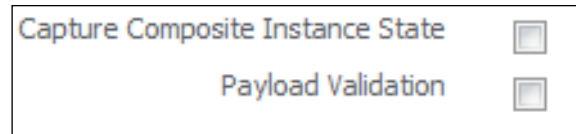
How to do it...

By following these steps, we can turn off payload validation:

1. Log in to Enterprise Manager.
2. Open the **SOA** tab, and right-click on **soa_infra**, select **SOA Administration** and **Common Properties**.



3. Un-tick the checkbox for **Payload Validation** to disable this feature.
4. Un-tick the checkbox for **Capture Composite Instance State**.



How it works...

In this recipe, we globally disabled payload validation. This instructs SOA Suite to not check the inbound and outbound message payloads against the schemas associated with our services. This can be particularly useful, not only if the payload is coming from a trusted source, but even if the source is untrusted. A common alternative to payload validation is to add steps to manually validate the payloads at the point that we first receive the request, while not validating those that have come from internal or trusted sources.

There are a number of levels of granularity for payload validation; it can be applied at the SOA Engine (BPEL) and composite levels to allow for fine-grained application of this property. You can access these properties via the enterprise manager console right-click menu on the SOA engines and deployed composites. For performance, I would recommend disabling this in all environments above development.

Composite state management is responsible for tracking and representing the health of our running composites. This is a powerful administration feature, but costs a lot in terms of performance. Anecdotal testing shows that this can be responsible for up to 30 percent of processing time. As such, for high throughput applications, the value of this feature should be considered.

There's more...

See the recipes on audit logging to further control composite recording activities at runtime.

Ensure that you check the payload validation at the Engine and Composite levels to ensure that they meet your performance requirements.

Designing BPEL processes to reduce persistence

Every step in a BPEL process adds an overhead. In this recipe we'll suggest some design options to consider when deciding how to construct your processes.

Getting ready

You will need an understanding of SOA Suite programming concepts.

How to do it...

The following steps cover some of the techniques for reducing process persistence:

1. If you have lots of variable assignment steps, consider a call out to a Business Rules component to check and set multiple values in response to the user input.
2. If complex logic is required, consider embedding a call to a Java class in the composite if this can reduce the number of steps.

How it works...

By default, the BPEL composite design will consist of adding steps to a process until the business logic can be satisfied. This can make it easy to end up with monolithic processes that have too many steps. For example, processes which make decisions and then fork on the results, require an increasing number of steps to deal with varying logic flows.

While it sounds easy to recommend simply reducing the number of steps, in practice this requires careful analysis of the composite requirements and consideration of where process dehydration will occur. SOA Suite offers two powerful options that can be leveraged to reduce the persistence between BPEL process steps.

Using business rules will transfer the process execution to an in-memory evaluation of the business logic. This can help with speeding up, forming and mutating the composite output payload.

Embedding calls to Java programs can help deal with replacing forking processes, as we can reduce multiple BPEL process steps to a number of if/else blocks in a program method. Note that to keep the execution speed high, we should resist the urge to perform slow external logic in the method, such as blocking calls to external databases.

See also

- ▶ The *Disabling BPEL monitors and sensors*, *Changing a BPEL process to be transient* and *Reducing the completion persist level* recipes in *Chapter 8, BPEL and BPMN Engine Tuning*

Using parallel routing rules in Mediator components

Mediator routing rules can be set to parallel or sequential execution. Using parallel execution can improve performance.

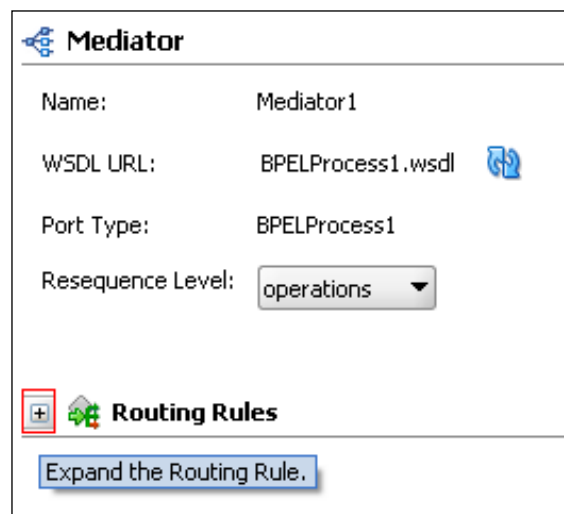
Getting ready

You will need to have JDeveloper, and have a good understanding of SOA Suite programming concepts.

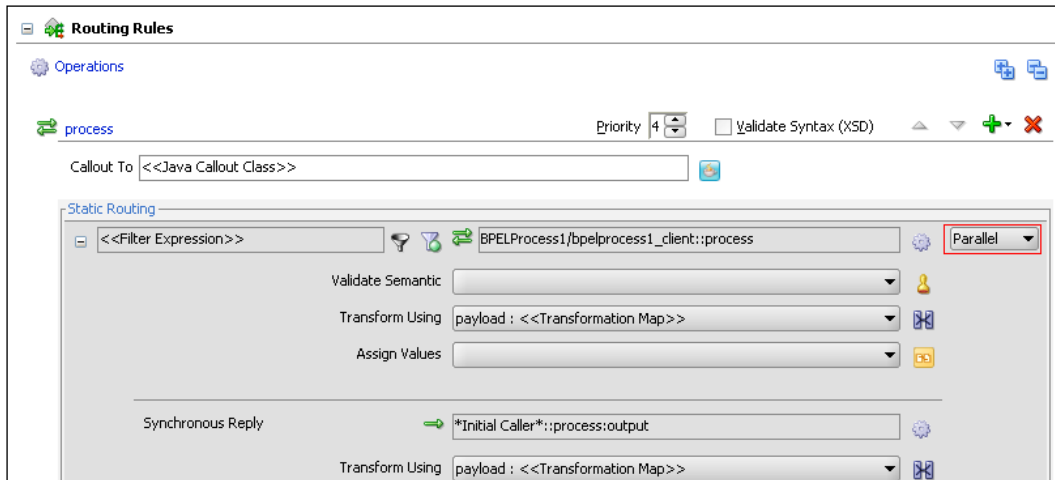
How to do it...

These steps show us how to configure parallel routing rules:

1. Using JDeveloper, open the Mediator file that contains the rules that you wish to execute in parallel.
2. If the routing rules are not visible, click on + to expand them:



- From the drop-down list, select **Parallel**.



- Save the file.

How it works...

The Mediator can execute routing rules either in parallel or sequentially. Parallel rules are executed at the same time in separate threads, using an algorithm that ensures that no rule can use up all of the threads, and starve another rule from executing. Each parallel rule is initiated in a new transaction, and that transaction is committed (or rolled back) by the Mediator process, once the rule has executed.

There's more...

You can set the priority of a parallel routing rule to a value between zero and nine. Higher priority rules will take precedence when threads are checking whether any parallel routing rules need executing.

The Oracle SOA Suite Mediator component is the old **Oracle Enterprise Service Bus (OESB)**, which is being gradually replaced by **Oracle Service Bus (OSB)**. OSB provides many more tuning options, and is often more efficient than the Mediator; so, for new SOA applications, we would recommend using OSB to perform the process mediation role, rather than the SOA Suite Mediator component.

See also

- ▶ The *Setting Mediator Parallel Metrics Level*, *Setting Mediator Parallel Worker Threads*, *Setting Mediator Parallel Maximum Rows Retrieved*, and *Setting Parallel Locker Thread Sleep* recipes in *Chapter 9, Mediator and BAM*

Setting HTTP timeouts for external services

Composite services often call out to external HTTP web services. If these external services are not available, or are slow to respond, then it can impact the application performance. By tuning the timeout to a lower or higher value, performance can be improved.

Getting ready

You will need to be familiar with SOA application development principals for this recipe.

How to do it...

These steps will set the timeout for an external HTTP service:

1. Open the `composite.xml` file in JDeveloper.
2. Select the source view at the bottom of the main pane.



3. Locate the `<reference>` section for the external service that you want to set the timeout for.

4. Inside the `<binding>` tags, add the property `oracle.webservices.httpReadTimeout` of type `xs:string` with a value high enough to allow the service to respond, such as `60000` (60 seconds).
5. Add the property `oracle.webservices.httpConnTimeout` with a type `xs:string`, and a low value, such as `5000` (5 seconds).

```

<reference name="PauseService"
  ui:wsdlLocation="http://127.0.0.1:7001/pause-service/HandleInputPort?WSDL">
  <interface.wsdl interface="http://c2b2.co.uk/soaperformance#wsdl.interface(HandleInput)"/>
  <binding.ws port="http://c2b2.co.uk/soaperformance#wsdl.endpoint(HandleInputService/HandleInputPort)"
    location="http://127.0.0.1:7001/pause-service/HandleInputPort?WSDL"
    soapVersion="1.1">
    <property name="weblogic.wsee.wsat.transaction.flowOption"
      type="xs:string" many="false">WSDLDriven</property>
    <property name="oracle.webservices.httpReadTimeout" type="xs:string" many="false">60000</property>
    <property name="oracle.webservices.httpConnTimeout" type="xs:string" many="false">5000</property>
  </binding.ws>
</reference>

```

6. Save the composite.
7. Deploy the application.

How it works...

When calling external services over HTTP, the application will need to wait for a response before it can continue. If the external service is slow to respond, and Oracle SOA Suite gives up too soon, then the request will fail and roll back to the most recent transactional save point. This is often compounded by retry logic, which will cause the request to be retried, increasing the load on an already busy system. By setting the read timeout to a high value, we can ensure that we give plenty of time for an external application to respond. In the preceding steps, we use a value of `60000` (60 seconds), but you can increase this value if you have a service that takes longer.

If the external service is not available, then waiting for a response is not going to work, and in fact we will usually be completely unable to establish the initial connection. By setting the connection timeout to a low value (we use `5000` or 5 seconds), we can return a fault quickly to the SOA Suite application and not wait for a response that will not arrive.

By tuning both of these parameters, it should be possible to come up with settings that will fail quickly if the server is not there, but will wait long enough for a response if the service is busy, while still timing out if the external service has stopped responding completely.

See also

- The *Increasing the HTTP accept backlog* recipe in *Chapter 6, Platform Tuning*

Tuning BPEL adapter properties

Each of the SOA Suite adapters has a number of properties that can be tuned depending upon your application requirements. This high-level recipe describes some of the options available, and how to tune them for your application.

Getting ready

You will need a good understanding of SOA Suite programming principles for this recipe.

How to do it...

The following steps explain how to tune the properties of the commonly used BPEL adapters.

1. Tune inbound FTP and file adapters to use a dedicated thread pool, by setting the `ThreadCount` property to a positive value, such as 10. If your file adapter is only being used to detect the arrival of a file, then setting `UseHeaders` to `true` can prevent the whole file payload being passed into the process. If the files do not arrive frequently, then `PollingFrequency` can be increased to above the default of one minute. If multiple, small files arrive, then `MaxRaiseSize` can be used to increase the number of files that are read in at a time.
2. Tune the outbound FTP and file adapters by setting the `ConcurrentThreshold` property to a higher value (for example, 50), and setting the `UseStaging` property to `false`.
3. Tune the JCA adapter by ensuring that it uses a connection pool to manage the connections, rather than setting up and tearing down a connection for each request.
4. For socket adapters, ensure that `KeepAlive` is set to `true`.
5. For JMS adapters, increase the value of `adapter.jms.receive.threads` property to 5.
6. For Oracle AQ adapters, set the `adapter.aq.dequeue.threads` property to 5.
7. Tune the Oracle MQ adapter by setting the `InboundThreadCount` property to 5.

How it works...

The preceding sets of recipes are the basics of configuring the adapters available in Oracle SOA Suite. There are many other properties that can be tuned, but their effectiveness is situational, so we have chosen to focus on the ones that we feel give improvement for the most projects.

The `ThreadCount` property on the FTP and file adapters controls the number of threads that are used to poll for new files and process them. It defaults to `-1`, which causes the default thread pool to be used, rather than a dedicated pool. This can lead to situations where the FTP and file adapters are starved of threads to process new files. It can also be set to `0`, which causes the behavior to be the same as the Single Thread Model, where files are not placed on an in-memory queue between processing steps.

For outbound FTP and file adapters, `ConcurrentThreshold` controls the maximum number of translation activities for a particular scenario that can be occurring at any one time. Because translation is CPU-intensive, having very large numbers of concurrent translations can use up all of the available CPU cycles. So it is sensible to limit the number that can occur, however the default of `20` is a bit low, given the increased parallelization available in modern CPUs.

The `UseStaging` property for file and FTP adapters determines whether an intermediate file is written between the translation of a file and later processing steps. The default is `true`, but if sufficient memory is available, this can be set to `false` to improve performance.

Setting `KeepAlive` to `true` on socket adapters will ensure that the TCP socket connections are kept open for multiple requests, rather than closing and reopening connections for each request. This can improve performance, as setting up and tearing down TCP connections is a high overhead.

The `adapter.jms.receive.threads`, `adapter.aq.dequeue.threads`, and `InboundThreadCount` properties set the number of threads available for processing JMS, AQ, and MQ messages. The default value for these is `5`, and performance can be improved by using more threads to dequeue messages from these providers.

There's more...

See http://docs.oracle.com/cd/E14571_01/core.1111/e10108/adapters.htm for more information on tuning the BPEL adapter properties.

12

High Performance Configuration

In this final chapter, we will look at some of the ways in which you can design your deployment architecture to improve performance. The topics that we will cover are:

- ▶ Configuring a cluster of SOA Suite servers
- ▶ Configuring an OHS load balancer
- ▶ Tuning for deployment on virtualized infrastructure
- ▶ Using distributed JMS destinations in a cluster
- ▶ Using JMS bridges to improve the enqueue speed
- ▶ Choosing deployment hardware

Introduction

While it is important to squeeze maximum performance out of every component in your application, true performance is an architectural concern. In this chapter, we are going to look at some of the ways in which you can design a deployment architecture that supports your performance requirements, and we are mostly going to do this by looking at the ways in which you can scale out your application.

As we discussed earlier in the book, performance limits are always caused by the limited availability of resources needed by the application, whether this is CPU cycles, storage (RAM or disk), network bandwidth, or availability of an external service, such as a database connection. Since it is not possible to keep adding more RAM or faster CPUs to your server for ever, there comes a time when the best way to increase your application's capacity is to add more instances of the application, and for a SOA Suite application, this normally means using the clustering features of the underlying WebLogic application server. We therefore focus most of our effort in this chapter on explaining how to configure WebLogic clustering and get the most out of it.

Configuring a cluster of SOA Suite servers

Configuring a SOA Suite cluster is the first step in allowing your application to scale out to support more users. This recipe explains how to create a new domain that contains a cluster.

Getting ready

This recipe creates a new domain with a cluster of SOA Suite servers. You will therefore need the Oracle SOA Suite software installed on all the servers that you wish the cluster to run on, and will need permissions to run the WebLogic configuration wizard on the server which will run the admin server.

How to do it...

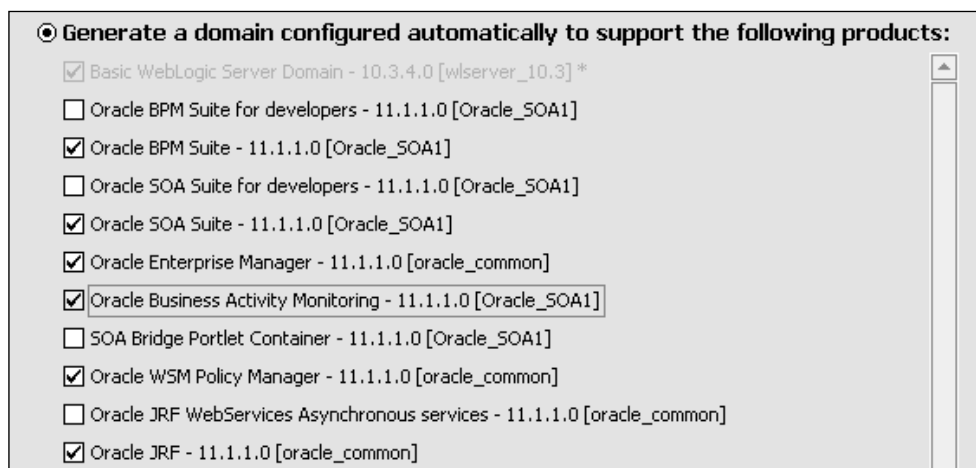
Follow these steps to configure a SOA Suite cluster:

1. Start the Oracle Fusion Middleware Configuration wizard, select the **Create a new WebLogic domain** option, and click on **Next**.

☉ Create a new WebLogic domain

Create a WebLogic domain in your projects directory.

2. Select the necessary components for a SOA Suite and BAM domain with Enterprise Manager, as shown in the following screenshot, then click on **Next**.



- Enter a name for the domain in the **Domain Name** box, and the location you want it to be installed to. This can be any path on a local system. Click on **Next**.
- Enter a name for the administrative user and a domain password, then click on **Next**.
- Select **Production Mode** for the domain and click on **Next**, leaving the default JVM choice if this is suitable.
- Obtain the database username and password from the database administrator, enter the database connection details, and click on **Next**.

Vendor: DBMS/Service:

Driver: Host Name:

Schema Owner: Port:

Schema Password:

☐ Configure selected component schemas as RAC multi data source schemas in the next panel.

	Component Schema	DBMS/Service	Host Name	Port	Schema Owner	Schema Password
<input type="checkbox"/>	BAM Schema	xe	192.168.106.128	1521	BOOK_ORABAM	*****
<input type="checkbox"/>	SOA Infrastructure	xe	192.168.106.128	1521	BOOK_SOAINFRA	*****
<input type="checkbox"/>	User Messaging Service	xe	192.168.106.128	1521	BOOK_ORASDPM	*****
<input type="checkbox"/>	OWSM MDS Schema	xe	192.168.106.128	1521	BOOK_MDS	*****
<input checked="" type="checkbox"/>	SOA MDS Schema	xe	192.168.106.128	1521	BOOK_MDS	*****

- The wizard will now test the connections. If the database is not up, or the repositories have not been created, the tests will fail, but the domain can still be created. Click on **Next**.

	Status	Component Schema	JDBC Connection URL
<input type="checkbox"/>	✓	BAM Schema	jdbc:oracle:thin:@192.168.106.128:1521/xe
<input type="checkbox"/>	✗	SOA Infrastructure	jdbc:oracle:thin:@192.168.106.128:1521/xe
<input type="checkbox"/>	✗	User Messaging Service	jdbc:oracle:thin:@192.168.106.128:1521/xe
<input type="checkbox"/>	✗	OWSM MDS Schema	jdbc:oracle:thin:@192.168.106.128:1521/xe
<input type="checkbox"/>	✗	SOA MDS Schema	jdbc:oracle:thin:@192.168.106.128:1521/xe

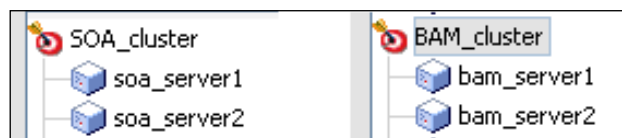
- Select that you wish to edit the managed servers, clusters, and machine configurations, and click on **Next**.
- Add additional managed servers, up to the number in your cluster design, give them the listen addresses and ports that you wish them to run on, and click on **Next**.

	Name*	Listen address*	Listen port	SSL listen port	SSL enabled
→ 1	bam_server1	All Local Addresses	9001	N/A	<input type="checkbox"/>
2	soa_server1	All Local Addresses	8001	N/A	<input type="checkbox"/>
3	bam_server2	All Local Addresses	9002	N/A	<input type="checkbox"/>
4	soa_server2	All Local Addresses	8002	N/A	<input type="checkbox"/>

10. Create two clusters—one for SOA servers and one for BAM servers. Select **multicast** as the cluster protocol (assuming your network supports it; if not then select **unicast**), and enter the cluster address as a comma-separated list of the hosts in the cluster. Click on **Next**.

	Name*	Cluster messaging mode	Multicast address	Multicast port	Cluster address
1	SOA_cluster	multicast	239.192.0.0	7001	soaserver
→ 2	BAM_cluster	multicast	239.192.0.0	7001	soaserver

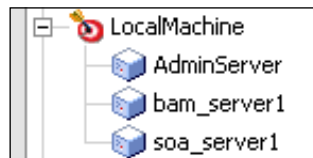
11. Allocate the servers to the clusters, then click on **Next**.



12. Create a machine for each host in your cluster, then click on **Next**.

	Name*	Node manager listen address	Node manager listen port
→ 1	LocalMachine	localhost	5556
2	machine2	localhost	5556

13. Allocate the WebLogic server instances to the machines that you wish them to run on, as in the following example screenshot, then click on **Next**.



14. Review the domain configuration, then click on **Next**.
15. Wait while the wizard creates the domain. Once it has finished, you can select the box if you wish to immediately start the admin server, then click on **Close**.

How it works...

It is worth starting by explaining the terminology used by Oracle for WebLogic clustering:

- ▶ A **domain** is a group of WebLogic servers that share the same configuration repository. The configuration repository is the `config.xml` file on the admin server. There can only be one admin server per domain.

- ▶ A **server** is an instance of the WebLogic server.
- ▶ An **admin server** is the WebLogic server instance that is used to administer the domain. It stores the master copy of the configuration files, and runs the WebLogic administration console application and Enterprise Manager SOA Suite component.
- ▶ A **managed server** is any WebLogic server instance that is not the admin server. Managed servers are used to run applications such as the SOA Suite components.
- ▶ A **cluster** is a group of WebLogic servers that all have the same configuration and application components on them. Clusters can share some state between the servers in the cluster, so some stateful operations can be performed on any cluster member. Resources can be targeted at a cluster, which will deploy them on each server in the cluster.
- ▶ A **machine** is the logical representation of a host, and is used by Oracle WebLogic to know which host a particular server should be started on.

When running the configuration wizard, we stepped through the normal process of creating a domain, then selected to customize the domain and add additional servers, clusters, and machines. These changes get written to the `config.xml` file that contains the domain's configuration. When we create the clusters and assign `soa_server1` and `bam_server1` to them, the configuration wizard is smart enough to know that it should deploy the relevant resources to the cluster, rather than to the individual servers.

In this domain setup, we also selected **multicast** as the cluster communication mechanism over **unicast**. Multicast is more efficient as updates are sent to a single address rather than distributed at the network tier, reducing network communication. Unicast on the contrary will create a connection between each cluster node, which is more work for the servers when sending updated cluster information. Unicast can sometimes be more efficient in very small clusters (for example, two to three nodes), because the networking infrastructure will often drop multicast packets first in order to preserve its QOS; unicast traffic can avoid this pitfall.

There's more...

It is possible to extend an existing domain to add the clustering functionality. There are a number of ways of doing this, including extending the domain with the configuration wizard, using the WebLogic scripting toolkit, or using the administration console. I recommend that you use the administration console (or the scripting toolkit **WLST**) rather than the configuration wizard, as it is less likely to lose any changes that you have made to your domain configuration.

See also

- ▶ *The Configuring an OHS load balancer, Choosing deployment hardware, and Tuning for deployment on a virtualized infrastructure recipes*

Configuring an OHS load balancer

Oracle HTTP server is an Apache httpd-based Web server that can be used to load balance requests across your SOA Suite cluster. This recipe looks at how to install it.

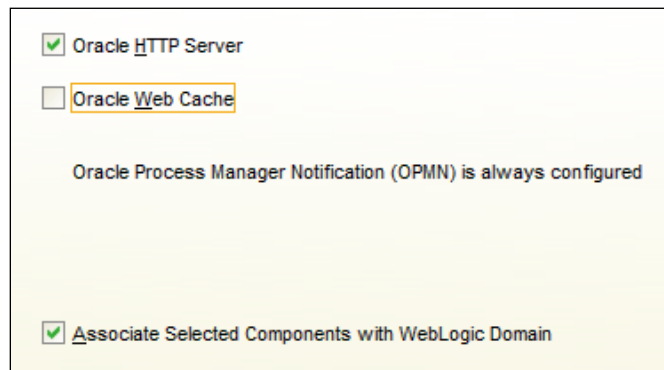
Getting ready

You will need to download the installer for the Oracle Fusion Middleware Web tier, and have the necessary filesystem permissions on the box on which you wish to install it. Your SOA Suite domain WebLogic admin server will need to be running, and you will need to know its administration credentials.

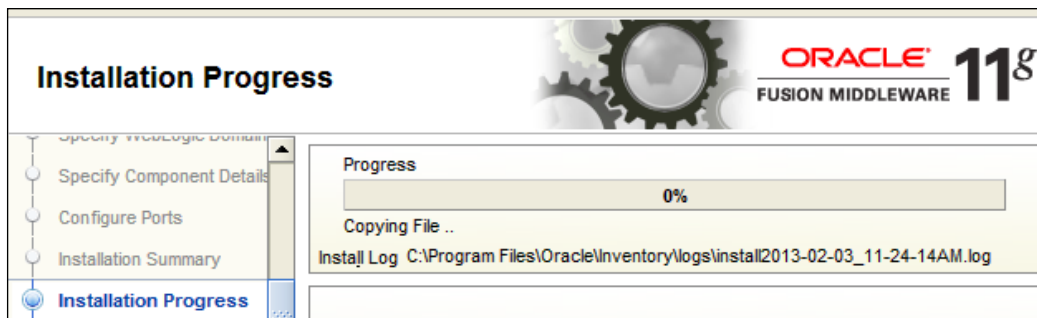
How to do it...

Follow these steps to install OHS on a target server:

1. Extract the installer archive, navigate to the `Disk1` folder, and execute the setup installer. It will open the welcome screen; click on **Next** to continue.
2. Select whether you wish to search for updates, and how you want them to be installed, then click on **Next** to continue.
3. Select that you wish to install and configure the server, then click on **Next** to continue.
4. The installer will now validate that your system meets the pre-requisites for the install; once it has done this, click on **Next** to continue.
5. Enter the details of the install location; we have selected `e:\Oracle\MiddlewareHome`, then click on **Next** to continue.
6. Select if you wish to receive security updates, then click on **Next** to continue.
7. Select that you only wish to install the HTTP server component, and that you wish to associate it with a WebLogic domain, then click on **Next** to continue.



8. Specify the connection details for the WebLogic administration server. In domain host name, set the IP of the administration server (if targeting the OHS server on the same server as the administration server, we can set `localhost`), for the domain port no. Set the administration server's port, then enter the administration user's credentials in username and password, then click on **Next** to continue.
9. Accept the default details for the Oracle HTTP Server instance, then click on **Next** to continue.
10. Accept the default port configuration, then click on **Next** to continue.
11. The installer will show a summary of what it is about to do; click on **Next** and it will start installing:



12. Once the installation is complete, the installer will configure the WebLogic domain and OHS instance. Click on **Next** to continue once this is done.
- The installer will display a summary screen, detailing all of the changes it has made.

How it works...

Oracle HTTP Server is an Apache httpd-based Web server that can be used to forward requests to a WebLogic server or cluster of servers. It contains a plugin that will proxy requests from OHS to the WebLogic servers, and since this plugin is specifically designed for Oracle WebLogic domains, it understands the nuances of clusters and how to balance load across them. It is dynamically updated by the cluster to contain an up-to-date topology as nodes leave and join.

As OHS will then be the client facing facet of a WebLogic domain, it is prudent to have more than one instance to prevent the introduction of a single point of failure. The worker counts should then be monitored to ensure that they are capable of providing enough throughput for your client traffic. It is a common practice to front a number of OHS instances with a hardware load balancer or component, such as Oracle Traffic Director. This helps to ensure that should an instance fail or become too busy, then traffic can be redirected to OHS nodes with capacity.

By using OHS, you can provide a common URL for your Web services that resolve to the OHS server, and the OHS server will then distribute the load across your servers based on session cookies. This allows you to scale your application outwards to meet your performance requirements.

See also

- ▶ *The Configuring a cluster of SOA Suite servers and Choosing deployment hardware recipes*

Tuning for deployment on a virtualized infrastructure

This recipe looks at some of the tuning you can perform on a virtualized infrastructure to improve the performance of your SOA Suite application.

Getting ready

This recipe assumes that you are using VMWare ESX as your virtualization environment, but the principles still apply if you are using another virtualization product. You will need the necessary permissions to make changes to the configuration of your virtual machine.

How to do it...

Consider the following items for your virtualized SOA Suite infrastructure:

1. We recommend that you run one instance of Oracle SOA Suite per virtual machine. We have found that this gives the best options for tuning each machine and the best scalability.
2. The second step is to select the number of vCPUs that will be allocated to each server. It is not best practice to allocate vCPU cycles that are not used, as then they cannot be used elsewhere. However, it is also not good to have insufficient vCPUs available to a machine. We suggest starting with two vCPUs per instance, and increasing this if you find that you are bottlenecking on CPU availability.
3. Next, we need to create a memory reservation for the server. You should create a memory reservation large enough to hold the total memory used by the JVM (not just the heap) plus 20 percent.
4. Finally, since virtual servers can suffer from significant clock drift, we recommend that you configure an NTP time source, to keep the servers in sync.

How it works...

Virtualization hypervisors have many significant optimizations and tricks that they use to allow over allocation of resources, however many of these do not work well with the way that application servers, such as WebLogic, are designed to work.

The largest issue is that of memory management. In general, it is considered best practice to over allocate the memory on a virtualized environment, on the theory that applications will not always need all of the memory allocated to them. However, this does not work well with Java applications, where the way garbage collection works causes large chunks of memory to become free at once, and then it all gets slowly used again until there is none free. When large chunks of memory become free, the VMWare balloon memory hypervisor assumes this is because an application just closed, so it will reclaim the memory from the virtual machine and allocate it to another server. When the JVM then wishes to use it again, the hypervisor has to find more memory to allocate to it, which can result in lengthy pauses. Creating a memory allocation stops this from occurring and ensures that the JVM always has sufficient memory.

The other large problem is clock drift. Since virtual servers only have a virtual hardware clock, its ticks are not as reliable as those of a physical server, resulting in significant clock drift on virtual servers. Since WebLogic clusters rely on timing information to maintain their consistency, and because many timeouts in SOA Suite components are measured in seconds, it is important to configure an NTP source to prevent this drift.

There's more...

In addition to these basic changes, VMWare has a number of recommendations for running Java application servers in a virtualized environment. See the VMWare documentation for more detail on these.

See also

- ▶ The *Choosing deployment hardware* and *Configuring an OHS load balancer* recipes
- ▶ The *Calculating the total memory used by your application* recipe in *Chapter 4, JVM Memory*

Using distributed JMS destinations in a cluster

A distributed JMS destination is a JMS queue or topic where the messages are spread across multiple servers in a cluster. They are used to have multiple JMS destinations across different servers that share a JNDI name.

Getting ready

You will need administrative access to the WebLogic domain that hosts your SOA Suite applications, and will need to have created a cluster already. If you already have a JMS queue using the JNDI name in your domain, you should delete this before creating the distributed destination.

How it works...

A distributed JMS queue allows you to have JMS queues with the same name on each node in a cluster. This allows the load to be distributed across multiple servers, with each server consuming off its own JMS queue and processing locally. A uniformly distributed destination is one in which the WebLogic Server creates the individual member queues on each server for you, rather than you having to do it yourself.

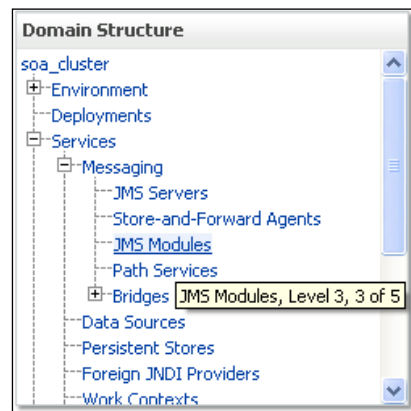
When configuring the producers and consumers for a distributed destination, the producers should usually be configured to produce to the distributed destination itself, and the local consumers should be configured to consume from the individual member queues created by WebLogic.

How to do it...

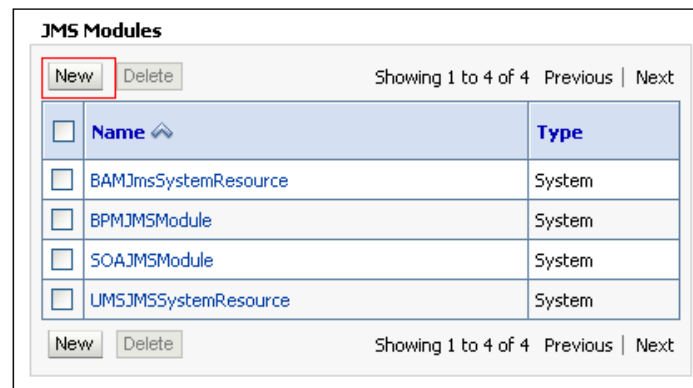
Follow these steps to configure a distributed JMS:

1. Connect to your WebLogic Server administration console, and log in with your administration credentials. By default, the administration console is available at `http://<adminserverhost>:<adminserverport>/console`.

2. In the left-hand navigation pane, expand **Services, Messaging**, and then **JMS Modules**.



3. In the main pane, click on **New** to create a new JMS module.



4. Enter a name for your JMS module, then click on **Next**.

Create JMS System Module

Back

Next

Finish

Cancel

The following properties will be used to identify your new module.

JMS system resources are configured and stored as modules similar to standard J2EE modules. Such resources include queues, topics, connection factories, templates, destination keys, quota, distributed queues, distributed topics, foreign servers, and JMS store-and-forward (SAF) parameters. You can administratively configure and manage JMS system modules as global system resources.

* Indicates required fields

What would you like to name your System Module?

* Name:

What would you like to name the descriptor file name? If you do not provide a name, a default will be assigned.

Descriptor File Name:

Where would like to place the descriptor for this System Module, relative to the jms configuration sub-directory of your domain?

Location In Domain:

Back

Next

Finish

Cancel

5. Target your JMS module to either the SOA or BAM server, depending upon where it is needed, then click on **Next**.

Targets :

Servers

☐ AdminServer

Clusters

☐ BAM_cluster

☐ All servers in the cluster
☐ Part of the cluster

☐ bam_server2
☐ bam_server3
☐ bam_server1

☒ SOA_cluster

☒ All servers in the cluster
☐ Part of the cluster

☐ soa_server3
☐ soa_server2
☐ soa_server1

6. The final screen will ask if you wish to create JMS resources for the new module; select that you do, then click on **Finish**.

Create JMS System Module

Back Next Finish Cancel

Add resources to this JMS system module

Use this page to indicate whether you want to immediately add resources to this JMS system module after it is created. JMS resources include queues, topics, connection factories, etc.

☒ **Would you like to add resources to this JMS system module?**

Back Next Finish Cancel

7. Click on the **New** button, as shown in the following screenshot, to create a new JMS resource.

Settings for MyAppJMSModule

Configuration Subdeployments Targets Security Notes

This page displays general information about a JMS system module and its resources. It also allows you to configure new resources and access existing resources.

Name:	MyAppJMSModule	The name of this JMS system module. More Info...
Descriptor File Name:	jms/myappjmsmodule-jms.xml	The name of the JMS module descriptor file. More Info...

This page summarizes the JMS resources that have been created for this JMS system module, including queue and topic destinations, connection factories, JMS templates, destination sort keys, destination quota, distributed destinations, foreign servers, and store-and-forward parameters.

[Customize this table](#)

Summary of Resources

New Delete Showing 0 to 0 of 0 Previous | Next

	Name	Type	JNDI Name	Subdeployment	Targets
There are no items to display					

New Delete Showing 0 to 0 of 0 Previous | Next

8. Select **Distributed Queue** as the type of resource, then click on **Next**.

<input type="radio"/> Topic	Defines a publish/subscribe destination type, which are used for asynchronous peer communications. A message delivered to a topic is distributed to all topic consumers. More Info...
<input checked="" type="radio"/> Distributed Queue	Defines a set of queues that are distributed on multiple JMS servers, but which are accessible as a single, logical queue to JMS clients. More Info...
<input type="radio"/> Distributed Topic	Defines a set of topics that are distributed on multiple JMS servers, but which are accessible as a single, logical topic to JMS clients. More Info...

9. Enter the name and JNDI name for your distributed queue, then click on **Finish**.

Create a New JMS System Module Resource

Back

Next

Finish

Cancel

JMS Distributed Destination Properties

The following properties will be used to identify your new Distributed Queue. The current module is MyAppJMSModule

* Indicates required fields

What would you like to name your new destination?

* Name:

MyAppQueue

What JNDI Name would you like to use to look up your new destination?

JNDI Name:

jms.myappqueue

Queue members may be either created uniformly from a common configuration, or created and weighted individually to fine tune performance. How would you like to create queue members?

Destination Type:

Uniform

Templates provide an efficient means of defining multiple destinations with similar configuration values. Would you like to use a template for this destination?

Template:

None

Back

Next

Finish

Cancel

You can also create distributed topics, which work in exactly the same way. To do so, follow the instructions in this recipe, but select a unified, distributed topic rather than a unified, distributed queue.

Using JMS bridges to improve the enqueue speed

In situations where you have one WebLogic server enqueueing a JMS message to a JMS queue on a remote WebLogic server, you can improve both the performance and resilience, by creating a bridged JMS queue. This recipe explains how to do that.

Getting ready

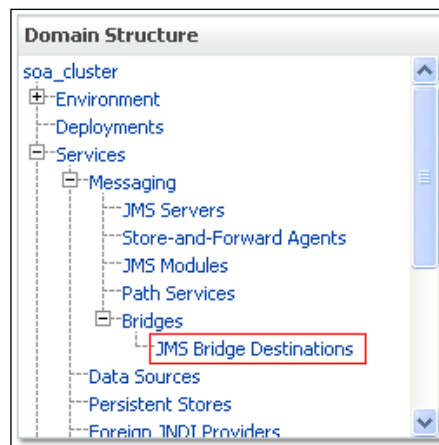
This recipe assumes that the JMS queue already exists on the remote JMS server, and that you wish to create a bridge to it from your local WebLogic server. You will need to know the administration credentials for your WebLogic server instance, to be able to create the bridges.

This recipe assumes you have created a local JMS queue with the same name as the remote JMS queue, and wish to bridge messages between them.

How to do it...

Follow these steps to create a JMS bridge:

1. Connect to your WebLogic Server administration console, and log in with your administration credentials. By default, the administration console is available at `http://<adminserverhost>:<adminserverport>/console`.
2. In the left-hand navigation pane, expand **Services, Messaging**, and then **Bridges**, as in the following screenshot:

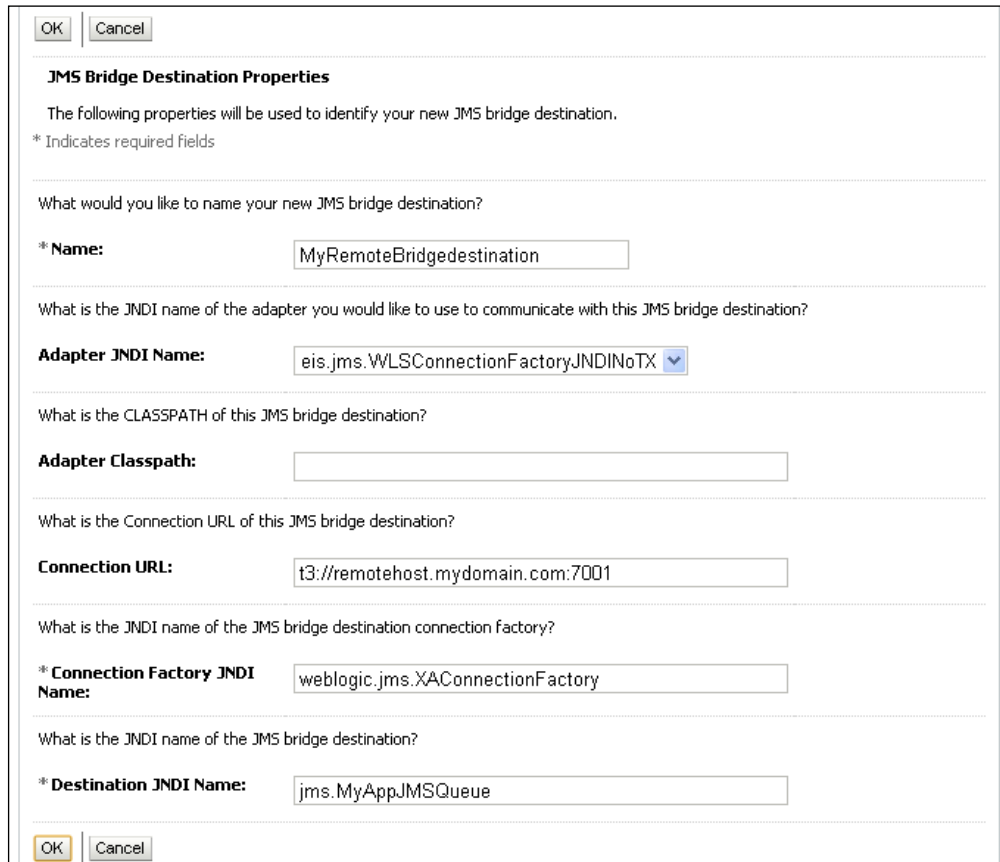


3. This will display a summary of any existing JMS bridge destinations. Click on the **New** button to create a new bridge destination.
4. This will be the local end of the JMS bridge; give it a descriptive name and enter the connection details such as the URL for the local server, whether you wish the bridge to use XA or be non-transactional, and the JNDI name of the JMS connection factory to use. Click on **Ok** when you are done.

The screenshot shows a dialog box titled "JMS Bridge Destination Properties". At the top are "OK" and "Cancel" buttons. Below the title, a note states: "The following properties will be used to identify your new JMS bridge destination." followed by an asterisk and "Indicates required fields". The dialog contains several labeled input fields: "Name:" with the value "MyQueueLocalDestination"; "Adapter JNDI Name:" with a dropdown menu showing "eis.jms.WLSCConnectionFactoryJNDIXA"; "Adapter Classpath:" with an empty text box; "Connection URL:" with the value "t3://localhost:7001"; "Connection Factory JNDI Name:" with the value "weblogic.jms.XAConnectionFactory"; and "Destination JNDI Name:" with the value "jms.MyAppJMSQueue". At the bottom are "OK" and "Cancel" buttons.

5. You will now be returned to the summary of bridge destinations, but should see your new bridge destination in the list. Click on **New** again, and create the remote end of the bridge.

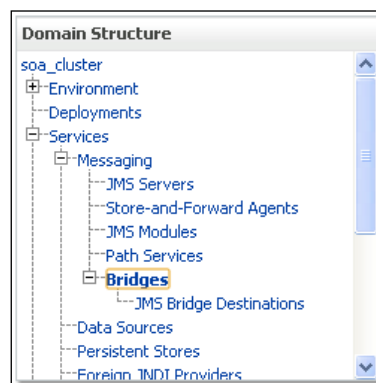
6. Enter the connection details for the remote end of the JMS bridge, then click on **Ok**.



The dialog box titled "JMS Bridge Destination Properties" contains the following fields and labels:

- OK** and **Cancel** buttons at the top.
- JMS Bridge Destination Properties** section header.
- Text: "The following properties will be used to identify your new JMS bridge destination."
- Text: "* Indicates required fields"
- Text: "What would you like to name your new JMS bridge destination?"
- * Name:**
- Text: "What is the JNDI name of the adapter you would like to use to communicate with this JMS bridge destination?"
- Adapter JNDI Name:**
- Text: "What is the CLASSPATH of this JMS bridge destination?"
- Adapter Classpath:**
- Text: "What is the Connection URL of this JMS bridge destination?"
- Connection URL:**
- Text: "What is the JNDI name of the JMS bridge destination connection factory?"
- * Connection Factory JNDI Name:**
- Text: "What is the JNDI name of the JMS bridge destination?"
- * Destination JNDI Name:**
- OK** and **Cancel** buttons at the bottom.

7. Navigate to the **Bridges** section of the **Messaging** services in the left-hand navigation pane:



8. This will give you the summary screen, showing you all of your existing JMS bridges, which will probably be none if you have not created any yet. Click on **New** to create the bridge.
9. Enter the properties for the JMS bridge including its name. Select the checkbox to ensure that the bridge is started when you have finished creating it, then click on **Next**.

The screenshot shows a dialog box titled "Create a New Bridge". At the top, there are four buttons: "Back", "Next", "Finish", and "Cancel". Below these is a section titled "Bridge Properties". A text label states: "The following properties will be used to identify your new messaging bridge." Below this is a note: "* Indicates required fields". The first question is "What would you like to name your new bridge?", followed by a text input field labeled "* Name:" containing the text "MyAppJMSBridge". The second question is "What would you like to name your message selector?", followed by a text input field labeled "Selector:". The third question is "What type of service quality would you like to select?", followed by a dropdown menu labeled "Quality Of Service:" with "Exactly-once" selected. The fourth question is "What initial state of this messaging bridge would you like to select?", followed by a checked checkbox labeled "Started". At the bottom, there are four buttons: "Back", "Next", "Finish", and "Cancel".

10. Select the JMS bridge destination that represents the local side of your JMS bridge, then click on **Next**.

The screenshot shows a dialog box titled "Create a New Bridge". At the top, there are four buttons: "Back", "Next", "Finish", and "Cancel". Below these is a section titled "Select or Create a Source Bridge Destination". A text label states: "You can select an existing source destination or create a new source destination." Below this is a question: "Which existing source destination would you like to select?". Below the question is a text input field labeled "Select an Existing Source Destination:" containing the text "MyQueueLocalDestination", followed by a dropdown arrow and a button labeled "New Destination". At the bottom, there are four buttons: "Back", "Next", "Finish", and "Cancel".

11. Select the adapter type, which should be `WebLogic Server 7` or higher, then click on **Next**.

Create a New Bridge

Back Next Finish Cancel

Select the Messaging Provider for the Source Destination.

Which messaging provider would you like to select?

Messaging Provider: WebLogic Server 7.0 or higher

Back Next Finish Cancel

12. Select the JMS bridge destination that represents the remote JMS destination, then click on **Next**.

Create a New Bridge

Back Next Finish Cancel

Select or Create a Target Bridge Destination

You can select an existing target destination or create a new target destination.

Which existing target destination would you like to select?

Select an Existing Target Destination: MyRemoteBridgedestination New Destination

Back Next Finish Cancel

13. Select the adapter type for the remote destination, which is probably **WebLogic Server 7** or higher, then click on **Next**.

Create a New Bridge

Back Next Finish Cancel

Select the Messaging Provider for the Target Destination.

Which messaging provider would you like to select?

Messaging Provider: WebLogic Server 7.0 or higher

Back Next Finish Cancel

14. Select the targets for the bridge, which should be the servers on which the JMS producers are running, then click on **Next**.

Create a New Bridge

Back Next Finish Cancel

Targeting this Messaging Bridge

You can select one or more targets to deploy your new bridge.

Servers

☐ AdminServer

Clusters

☐ BAM_cluster

☐ All servers in the cluster

☐ Part of the cluster

☐ bam_server1

☐ bam_server2

☐ bam_server3

☒ SOA_cluster

☒ All servers in the cluster

☐ Part of the cluster

☐ soa_server2

☐ soa_server1

☐ soa_server3

Migratable Targets

☐ bam_server1 (migratable)

☐ bam_server2 (migratable)

15. Review the information about the changes needed to remote domains, then click on **Finish**.

Create a New Bridge

Back Next Finish Cancel

Click Finish to Exit

When you click the Finish button, a messaging bridge will be created. However, you may need to manually implement some of the interoperability guidelines that apply when using the messaging bridge to access JMS destinations on different releases of WebLogic Server and in remote WebLogic domains. The console checks your configuration and lists any tasks you must perform manually below this message. Click the Help button on this panel for more information.

- If either the source or destination (or both) is in a remote Weblogic domain, the Guest Disabled check box must not be selected on that domain.
- The Enable Generated Credential check box must not be selected. Use the same password as the credential on all Weblogic domains that participate in the messaging bridge.
- The XA connection factory must be enabled for all WebLogic domains.

Back Next Finish Cancel

How it works...

A JMS bridge consumes messages from one JMS queue and enqueues them onto a different JMS queue. While it might sound like doing this would reduce the performance, it has a number of benefits. Firstly, it means that the component producing JMS messages does not need to make a network call to a remote server; it can instead enqueue the message locally with no network overhead, then continue processing. The bridge will ensure that the message reaches the intended JMS queue. Secondly, it means that poor performance on the remote JMS server (for example, if it is overloaded with messages) will not result in poor performance on the JMS producer. This decoupling from the external service can allow you to make performance guarantees for your services when you have external systems that you have little or no control over.

A second advantage of JMS bridges is resilience. If your SOA Suite application is enqueueing messages onto a remote JMS server, then if that remote server goes down and you are unable to enqueue messages, your application will fail. With a bridge between the two servers, you can still enqueue messages locally, and when the remote server comes back up, the bridge will reconnect and send any pending messages to the remote server.

There's more...

An alternative to using a JMS bridge where you want to enqueue a JMS message to a remote WebLogic server in the same domain is to use a store-and-forward JMS destination. These are WebLogic-specific features that use the internals of the WebLogic JMS infrastructure to store a JMS message until it can be transactionally enqueued onto the remote server.

Store and Forward Agents (SAF) are another JMS feature within WebLogic that offer a similar functionality to JMS bridges, however, in our experience, they are not as performant, are less reliable, and no easier to configure than a JMS bridge.

See also

- ▶ *The Configuring distributed JMS destinations recipe*

Choosing deployment hardware

Selection of deployment hardware is the key in ensuring that your application can support its required non-functional requirements.

Getting ready

You will need to have an understanding of computer hardware to follow this recipe, and be familiar with your expected user load and other non-functional requirements for your application.

How to do it...

Consider the following items when choosing the underlying deployment hardware:

1. The first decision is whether you are going to deploy into a physical or virtual environment. In our experience, Oracle SOA Suite runs better on physical servers, because it is quite demanding in terms of hardware performance, but we have seen many successful large-scale deployments on virtual servers. For this recipe, we will assume that you are deploying on physical hardware.
2. The second thing to decide is the number of application server instances that you wish to run on each machine, and how much memory you give to application server instances. As a starting point, consider running with two application servers per machine, with each application server running with a 4 GB heap. You can then calculate the approximate amount of memory needed on each machine with the following formula:

$$\text{Required memory (in GB)} = 2 + (n * (1.25 * m))$$

Here, n is number of instances, and m is their memory footprint.

3. To calculate the approximate number of CPUs that you need on the box (based on a standard Intel type x86 64-bit CPU), use this formula:

$$\text{Required CPUs} = (n * 8) / c$$

Here, n is the number of instances, and c is the number of cores per CPU.

4. Having determined the number of required CPUs and amount of memory, we just need to calculate how many machines we need. As we already have performance profiled our application, we know how many single users an application server can deal with and so we can use the following formula:

$$\text{Required machines} = (1.5 * e) / s$$

Here, e is the estimated number of users, and s is the number of users a single application server can deal with.

How it works...

In this recipe, we have seen three formulae, and they could do with a little more explanation. These formulae have all been derived based on the experience of our consultants working out in the field on SOA Suite applications; as such, they are guidelines rather than mathematically proven rules.

The first formula is used to derive the amount of memory needed on a machine. It assumes that each application server has a 25 percent overhead for memory usage (hence the $1.25 * m$), and that the operating system will require around 2 GB memory.

The second formula is used to derive the number of CPUs that you will use. The value of 8 comes from our experience that 8 cores per SOA Suite instance is about the minimum you can get away with in a live system.

The final formula calculates the number of machines needed, and uses a factor of 1.5 times the expected load, as it is important to plan in contingency to cope with failures or unexpected load increases.

These formulae together should allow you to calculate the approximate amount and size of hardware that you will need to host your live SOA Suite application.

There's more...

The preceding guidelines work equally as well for virtual servers, but there are a number of other concerns to think about. These are discussed in the *Tuning for deployment on a virtualized environment* recipe based on tuning for a virtualized environment. In a virtualized environment, it is normal to run with only one application server per machine.

It is also worth considering deployment on non-Intel x86 architectures, such as some of the Sparc platforms.

See also

- ▶ The *Tuning for deployment on a virtualized environment* recipe
- ▶ The *Calculating the total memory used by your application* recipe in Chapter 4, *JVM Memory*

Index

A

- admin server** 287
- advanced load tests, JMeter**
 - designing 94-96
 - working 97
- alerts, Hyperic**
 - configuring 69-72
- Amazon Elastic Cloud Compute (EC2)** 98
- Apache JMeter**
 - about 76
 - advanced load tests, designing 94
 - downloading 77
 - installing 77
 - performance tests, running from Cloud 98-102
 - responses, checking in tests 84-87
 - running, on multiple servers 81-84
 - used, for creating web service test 78-80
 - used, for recording user web sessions 92-94
 - working 78
- Application Monitoring tools** 36
- audit level**
 - levels of note 220
 - reducing 219, 220
- audit store policy**
 - setting 226, 227
 - working 227
- audit trail size threshold**
 - reducing 228, 229
- automatic release timers**
 - about 256
 - disabling 256-258

B

- BAM** 247
- BAM Adapter**
 - batched delivery, enabling 252-254
- batched delivery**
 - enabling, for BAM Adapter 252-254
- Batching_Limit_Lower** 253
- Batching_Limit_Upper** 253
- Batching_Timeout** 253
- boot.properties file**
 - creating 175, 176
- BPEL adapter properties**
 - tuning 280
- BPEL/BPMN engines**
 - about 206
 - audit level, reducing 219, 220
 - audit store policy, setting 226, 227
 - audit trail size threshold, reducing 228, 229
 - BPEL process transient, making 221, 222
 - BPEL sensors and monitors, disabling 217
 - completion persist level, reducing 223
 - data, purging 233
 - dispatcher engine threads, tuning 211, 212
 - dispatcher invoke threads, tuning 207, 208
 - dispatcher system threads, tuning 214
 - global database parameters, setting 236, 237
 - large document threshold, increasing 229, 230
 - Oracle database, tuning 239, 240
 - persistence, disabling on invocation 224, 225
 - SOA infra log levels, reducing 230
 - tuning 206

BPEL process

- changing, to transient 221, 222
- designing, for reducing process persistence 275

BPEL process parallelization

- using 269, 270

BPEL process persistence

- disabling, on invocation 224, 225

BPEL sensors and monitors

- disabling 218

Business Activity Monitoring. *See* **BAM**

C

Cloud 103

cluster 287

committed transactions, SOA Suite 11g

- monitoring 66, 68

completion persist level

- reducing 223

composite state

- monitoring 274

concurrent mark sweep 139

correct workflow service client

- selecting 258, 259

D

data

- purging, from BPEL store 232, 234

database parameters

- setting 236-239

database statistics generation task

- enabled, ensuring 235

database XA timeouts

- tuning 188-190

data source connection timeouts

- setting 187

data-source pooling 180

data source pool sizes

- setting 180-182

data sources grow

- configuring 185, 186

data sources shrink

- configuring 185, 186

data source testing

- configuring 183, 184

data source usage, SOA Suite 11g

- monitoring 60-62

dehydration 222

deployment hardware

- selecting 304

development mode, WebLogic domains 173

dispatcher engine threads

- tuning 211, 212

- working 212

Dispatcher Invoke thread pool 209

dispatcher invoke threads

- tuning 207-211

dispatcher system threads

- tuning 214

- working 215, 216

distributed JMS destination

- about 292

- configuring 292-296

- using, in cluster 292

DMS servlet

- about 74

- used, for monitoring system 72

- working 74

DMS Spy servlet 72

domain 286

E

EIS adapter connections

- tuning 191-194

explicit GC

- disabling 149, 150

F

file descriptors

- increasing, in Linux 162, 163

G

garbage collection

- about 133

- evaluating 144-147

- explicit GC, disabling 149, 150
- monitoring, jstat used 10, 11
- new size, configuring 134, 135
- RMI garbage collector, disabling 148
- survivor ratio, configuring 136, 137
- verbose garbage collection, enabling 142, 143

garbage-collection algorithm

- concurrent mark sweep 139
- garbage first 139
- parallel garbage collection 139
- selecting, in HotSpot 138
- selecting, in JRockit 140, 141

garbage first 139

GC. *See* **garbage collection**

GCviewer

- URL 11

global transaction timeouts

- about 153
- tuning 153, 154

H

HotRocket 160

HotSpot

- garbage-collection algorithm, selecting 138, 139

HTTP accept backlog

- about 155
- increasing 155, 156

HTTP timeouts

- setting, for external services 278, 279

HugePages 160

human workflow component 255

Hyperic

- about 36
- alerts, configuring 69-71

Hyperic agents

- about 42
- installing 42-44
- working 45

Hyperic HQ installer

- downloading 37

Hyperic server

- about 37
- configuring, for SOA Suite 11g monitoring 46-51

- installing 37-41
- working 41

J

Java 1.5 37

Java application

- locking, performing 14

Java management extensions (JMX) 128

Java Management eXtensions (JMX)

- interface 45

Java Runtime Environment (JRE) 41

Java thread stack size. *See* **thread stack size**

Java Virtual Machine. *See* **JVM**

JDBC connections

- monitoring, with WebLogic console 28-32

JMeter. *See* **Apache JMeter**

JMS bridge

- creating 297-302
- working 303

JMS connection factories

- tuning 200-203

JMS file persistence

- about 199
- configuring 197, 198

JMS persistent store 200

JMX 37

JRMC

- about 119
- features 123
- used, for identifying performance problems 21

JRockit

- garbage-collection algorithm, selecting 140
- used, for monitoring memory 119

JRockit flight recorder

- used, for identifying problems 24-27
- working 27

JRockit Mission Control. *See* **JRMC**

- about 21
- working 24

jstat

- about 7
- used, for identifying locking issues 12, 13
- used, for identifying performance problems 14-16

- used, for identifying permanent generation problems 9, 10
- used, for identifying problems 7
- used, for monitoring garbage collection 10, 11
- working 8

JVM

- about 7, 37, 105
- upgrading 167, 168

JVM free memory

- monitoring 57

JVM heap size

- increasing 106-110

JVM implementations

- finding out 159

JVM Just In Time (JIT) 127

JVM memory

- heap size, increasing 106-110
- memory usage, calculating 115-118
- memory usage, viewing using JRMCM for JRockit 119-122
- memory usage, viewing using VisualVM for HotSpot 124-127
- Permanent Generation heap size, configuring 113, 114
- thread stack size, configuring 130, 131
- Xmx and Xms parameters, setting to same value 111, 112

JVM memory usage

- monitoring 54-56

JVM Statistics Monitoring Tool Daemon (jstatd) 127

L

large document threshold

- increasing 229, 230

large pages

- using, in Linux 160, 161

Linux

- about 37
- file descriptors, increasing 162, 163
- large pages, using 160, 161

Linux kernel swappiness level

- setting to low 169, 170

locking issues

- identifying, jstat used 12, 13

M

machine 287

managed server 287

MBean 37

Mediator engine 247

Mediator Parallel Maximum Rows Retrieved

- setting 250, 251

Mediator Parallel Metrics Level

- setting 248, 249

Mediator Parallel Worker Threads

- setting 250

Memory Analyzer Toolkit 16

memory management 105

memory usage

- calculating 115-118
- viewing, JRMCM used for JRockit 119-122
- viewing, VisualVM used for HotSpot 124, 125

Mission Control 122

monitoring 36

N

new size, garbage collection

- configuring 134, 135

new size problems

- identifying, jstat used 7

non-blocking service invocations

- using, in BPEL flows 271, 272

O

OHS load balancer

- configuring 288
- installing 288, 289
- working 289

OPEN_CURSORS parameter 238

open sockets, SOA Suite 11g

- monitoring 63-65

Oracle database

- tuning 239-244

Oracle Dynamic Monitoring System

- monitoring, DMS servlet used 72-74

Oracle Enterprise Service Bus (OESB) 277

Oracle JRockit JVM

- using 171, 172

Oracle Service Bus (OSB) 277

Oracle SOA Suite application
 monitoring 36
 performance monitoring 36
Oracle SOA Suite Mediator component 277

P

parallel garbage collection 139
Parallel Locker Thread Sleep
 setting 251, 252
parallel routing rules, mediator components
 configuring 276, 277
pause time priority 141
payload validation
 disabling 273, 274
Payload Validation setting, BPEL engine 220
performance 6
performance bottlenecks, SOA Suite application
 about 6
 garbage collection, monitoring with jstat 10, 11
 JDBC connections, monitoring with WebLogic console 28-30
 locking issues, identifying with jstack 12, 13
 new size problems, identifying with jstat 7, 8
 performance problems, identifying using JRMCM on JRockit 21-24
 performance problems, identifying using VisualVM 16-20
 performance problems, identifying with jstack 14, 15
 permanent generation problems, identifying with jstat 9, 10
 problems, identifying using JRockit flight recorder 24-27
 slow running components, identifying with Enterprise Manager 33, 34
 slow running database queries, identifying 32, 33
performance problems
 identifying, JRMCM using on JRockit 21-24
 identifying, jstat used 14-16
 identifying, VisualVM using on HotSpot 16-20
performance tests, JMeter
 running 98-102
Permanent Generation. See PermGen

Permanent Generation heap
 size, configuring 113
permanent generation problems
 identifying, jstat used 9, 10
PermGen
 about 113, 114
 working 114
platform CPU usage, SOA Suite 11g
 monitoring 58, 59
potential circular references
 removing 260
production mode, WebLogic domains
 about 173
 working 174
Profiler 21

R

redo thread 245
remote Hotspot JVM
 connecting to 128
remote JRockit JVM
 connecting to 123
responses
 checking, in JMeter tests 84-87
RMI garbage collector
 about 148
 disabling 148
 working 148
rule execution
 looping, preventing 259-262
 tuning 263, 264
rules component 255
ruleset 264

S

server 287
server logging level
 reducing 157, 158
setSoaDomainEnv script 139, 141
SIGAR library 59
slow running components
 identifying, with Enterprise Manager 33, 34
slow running database queries
 identifying 32, 33
SOA infra log levels
 reducing 230-232

SOA Suite

monitoring, while testing 88-91

SOA Suite 11g

committed transactions, monitoring 66, 68

data source usage, monitoring 60-62

memory usage, monitoring 55, 56

open sockets, monitoring 63-65

platform CPU usage, monitoring 58, 59

SOA Suite application

performance bottlenecks 6

SOA Suite cluster

configuring 284-286

SOA Suite EJB timeouts

tuning 163-167

SOA Suite Java Virtual Machines 6

SOA Suite Server availability

monitoring 51-54

SOA Suite stack

layers 152

Solaris 37

startManagedWebLogic script 134

startWebLogic script 134

Store and Forward Agents (SAF) 303

survivor ratio, garbage collection

configuring 136, 137

T

thread stack size

configuring 130, 131

U

user web sessions

recording, with JMeter 92-94

V

verbose garbage collection

about 142

enabling 142

VFabric Hyperic HQ tool 36

virtualized SOA Suite infrastructure

deployment, tuning 290, 291

VisualVM

about 127

used, for identifying performance

problems 16-20

working 20, 127

W

WebLogic 45

WebLogic console

used, for monitoring JDBC connections 28-31

WebLogic domains

development mode 173

production mode 173

running, in production mode 173, 174

WebLogic server

monitoring 46

WebLogic thread pool

configuring 194-196

web service test

creating, JMeter used 78-81

Work Managers 197

X

Xmx and Xms parameters

setting, to same value 111, 112



Thank you for buying Oracle SOA Suite 11g Performance Tuning Cookbook

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution-based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: www.PacktPub.com.

About Packt Enterprise

In 2010, Packt launched two new brands, Packt Enterprise and Packt Open Source, in order to continue its focus on specialization. This book is part of the Packt Enterprise brand, home to books published on enterprise software – software created by major vendors, including (but not limited to) IBM, Microsoft and Oracle, often for use in other corporations. Its titles will offer information relevant to a range of users of this software, including administrators, developers, architects, and end users.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to author@packtpub.com. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.



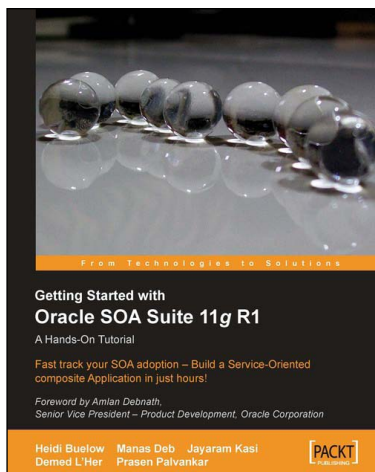
Oracle SOA Suite 11g R1 Developer's Guide

ISBN: 978-1-849680-18-9

Paperback: 720 pages

Develop Service-Oriented Architecture Solutions with the Oracle SOA Suite

1. A hands-on, best-practice guide to using and applying the Oracle SOA Suite in the delivery of real-world SOA applications
2. Detailed coverage of the Oracle Service Bus, BPEL PM, Rules, Human Workflow, Event Delivery Network, and Business Activity Monitoring
3. Master the best way to use and combine each of these different components in the implementation of a SOA solution



Getting Started With Oracle SOA Suite 11g R1 – A Hands-On Tutorial

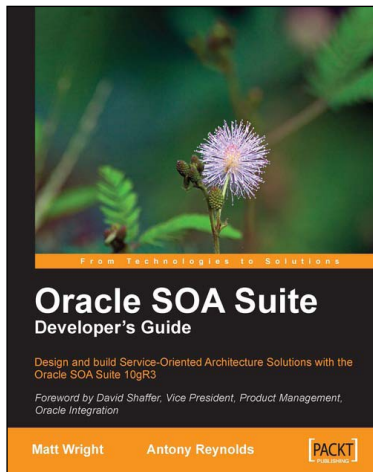
ISBN: 978-1-847199-78-2

Paperback: 482 pages

Fast track your SOA adoption — Build a Service-Oriented composite Application in just hours!

1. Offers an accelerated learning path for the much anticipated Oracle SOA Suite 11g release
2. Beginning with a discussion of the evolution of SOA, this book sets the stage for your SOA learning experience
3. Includes a comprehensive overview of the Oracle SOA Suite 11g Product Architecture
4. Explains how Oracle uses standards like Services Component Architecture (SCA) and Services Data Object (SDO) to simplify application development

Please check www.PacktPub.com for information on our titles



Oracle SOA Suite Developer's Guide

ISBN: 978-1-847193-55-1

Paperback: 652 pages

Design and build Service-Oriented Architecture Solutions with the Oracle SOA Suite 10gR3

1. A hands-on guide to using and applying the Oracle SOA Suite in the delivery of real-world SOA applications.
2. Detailed coverage of the Oracle Service Bus, BPEL Process Manager, Web Service Manager, Rules, Human Workflow, and Business Activity Monitoring.
3. Master the best way to combine / use each of these different components in the implementation of a SOA solution.



Oracle SOA Suite 11g Developer's Cookbook

ISBN: 978-1-849683-88-3

Paperback: 346 pages

Over 65 high-level recipes for extending your Oracle SOA applications and enhancing your skills with expert tips and tricks for developers

1. Extend and enhance the tricks in your Oracle SOA Suite developer arsenal with expert tips and best practices
2. Get to grips with Java integration, OSB message patterns, SOA Clusters and much more in this book and e-book
3. A practical Cookbook packed with recipes for achieving the most important SOA Suite tasks for developers

Please check www.PacktPub.com for information on our titles