# JBoss 企业级 SOA 平台 4.3
# 管理指南

管理 JBoss 企业级 SOA 平台 4.3.CP05 的指南
版 4.3.5

Red Hat Documentation Group

# JBoss 企业级 SOA 平台 4.3 管理指南

---

## 管理 JBoss 企业级 SOA 平台 4.3.CP05 的指南
## 版 4.3.5

Red Hat Documentation Group

**法律通告**

Copyright © 2011 Red Hat, Inc..

This document is licensed by Red Hat under the Creative Commons Attribution-ShareAlike 3.0 Unported License. If you distribute this document, or a modified version of it, you must provide attribution to Red Hat, Inc. and provide a link to the original. If the document is modified, all Red Hat trademarks must be removed.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, MetaMatrix, Fedora, the Infinity Logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack Logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

**摘要**

《管理指南》包括配置和管理 JBoss SOA 平台的重要信息。

# 目录

**序言** ..... **4**
1. 文档约定 4
1.1. 排版约定 4
1.2. 抬升式引用约定 5
1.3. 备注及警告 5

**第 1 章 配置** ..... **7**
1.1. 基本概念介绍 7
1.2. 独立服务器 7
1.3. 群集的 ESB 服务 7
1.4. JMS 提供者 9
1.4.1. 每个连接的最大会话数 10
1.4.2. JBossMessaging 11
1.4.3. Apache ActiveMQ 11
1.4.4. IBM Websphere MQ Series 6.0 11
1.4.4.1. XA 连接 12
1.4.4.2. JCA 适配器 12
1.4.4.3. 验证 14
1.4.5. Oracle 高级队列（Advanced Queuing） 14
1.4.6. Tibco 企业级消息服务 15
1.4.7. 扩展属性 16
1.5. 数据库配置 16
1.5.1. 手动切换数据库 16
1.6. 使用 JSR-170 消息库 18
1.7. 消息跟踪 18
1.8. 对群集和失效切换的支持 19
1.9. 使用 OpenSSO 20
1.9.1. 在 Tomcat 里安装和配置 OpenSSO 20
1.9.2. 为 JBoss SOA 平台配置 OpenSSO 21

**第 2 章 注册表** ..... **23**

**第 3 章 配置 Web 服务的集成** ..... **24**

**第 4 章 缺省的 "ReplyTo" 端点引用** ..... **25**

**第 5 章 ServiceBinding 管理者** ..... **26**

**第 6 章 监控和管理** ..... **27**
6.1. JMX MBeans 27
6.2. 监控和管理控制台 27
6.2.1. 安装企业级服务总线控制台 28
6.2.2. 使用其他数据库 28
6.2.3. 采集时间间隔 29
6.2.4. 服务 29
6.2.5. 消息计数器 30
6.2.6. Smooks Transformations 30
6.2.7. Dead Letter 服务 30
6.3. 消息警告 31
6.4. JON for SOA 32
6.4.1. 把 JBoss SOA 平台服务器添加到 JON 库存里。 32
6.4.2. JBoss SOA ESB 统计信息 32
6.4.3. 管理部署的 ESB 归档文件 35

6.4.4. 自动服务发现（Automatic Service Discovery） 35

**第 7 章 热部署** ..... **37**
7.1. 服务器模式 37
7.2. 独立（引导）模式 37

**第 8 章 发布合约信息（Contract Publishing）** ..... **39**
8.1. Contract 应用程序 39
8.2. 用 Action 发布合约 39

**第 9 章 JBoss 商业过程管理者** ..... **41**
9.1. jBPM 控制台 41
9.2. jBPM 的 Message 和 Scheduler 服务 41

**第 10 章 性能调优** ..... **42**
10.1. 概述 42
10.2. InVM 传输 42
10.3. Transport 线程 42
10.4. 消息过滤器（Message Filter） 43
10.5. 引用传递（Passing By Reference） 43
10.6. HTTP 路由器 43
10.7. HTTP 连接器 43
10.8. 日志 44
10.8.1. HTTP 连接器 44

**修订历史记录** ..... **46**

# 序言

## 1. 文档约定

本手册使用几个约定来突出某些用词和短语以及信息的某些片段。

在 PDF 版本以及纸版中，本手册使用在 Liberation 字体套件中选出的字体。如果您在您的系统中安装了 Liberation 字体套件，它还可用于 HTML 版本。如果没有安装，则会显示可替换的类似字体。请注意：红帽企业 Linux 5 以及其后的版本默认包含 Liberation 字体套件。

### 1.1. 排版约定

我们使用四种排版约定突出特定用词和短语。这些约定及其使用环境如下。

**单行粗体**

用来突出系统输入，其中包括 shell 命令、文件名以及路径。还可用来突出按键以及组合键。例如：

> 要看到文件您当前工作目录中文件 **my_next_bestselling_novel** 的内容，请在 shell 提示符后输入 **cat my_next_bestselling_novel** 命令并按 **Enter** 键执行该命令。

以上内容包括一个文件名，一个 shell 命令以及一个按键，它们都以固定粗体形式出现，且全部与上下文有所区别。

按键组合与单独按键之间的区别是按键组合是使用加号将各个按键连在一起。例如：

> 按 **Enter** 执行该命令。
>
> 按 **Ctrl+Alt+F2** 切换到虚拟终端。

第一个示例突出的是要按的特定按键。第二个示例突出了按键组合：一组要同时按下的三个按键。

如果讨论的是源码、等级名称、方法、功能、变量名称以及在段落中提到的返回的数值，那么都会以上述形式出现，即**固定粗体**。例如：

> 与文件相关的等级包括用于文件系统的 **filesystem**、用于文件的 **file** 以及用于目录的 **dir**。每个等级都有其自身相关的权限。

**比例粗体**

这是指在系统中遇到的文字或者短语，其中包括应用程序名称、对话框文本、标记的按钮、复选框以及单选按钮标签、菜单标题以及子菜单标题。例如：

> 在主菜单条中选择**「系统」** → **「首选项」** → **「鼠标」**启动 **鼠标首选项**。在**「按钮」**标签中点击**「惯用左手鼠标」** 复选框并点击 **关闭**切换到主鼠标按钮从左向右（让鼠标适合左手使用）。
>
> 要在 **gedit** 文件中插入特殊字符，请在主菜单栏中选择**「应用程序」** → **「附件」** → **「字符映射表」**。接下来选择从 **Character Map** 菜单中选择**Search** → **「查找.....」**，在**「搜索」**字段输入字符名称并点击**「下一个」**按钮。此时会在**「字符映射表」**中突出您搜索的字符。双击突出的字符将其放在**「要复制的文本」**字段中，然后点击**「复制」**按钮。现在返回您的文档，并选择 **gedit** 菜单中的**「编辑」** → **「粘贴」**。

以上文本包括应用程序名称、系统范围菜单名称及项目、应用程序特定菜单名称以及按钮和 GUI 界面中的文本，所有都以比例粗体出现并与上下文区别。

***固定粗斜体*** 或者 ***比例粗斜体***

无论固定粗体或者比例粗体，附加的斜体表示是可替换或者变量文本。斜体表示那些不直接输入的文本或者那些根据环境改变的文本。例如：

> 要使用 ssh 连接到远程机器，请在 shell 提示符后输入 **ssh *username@domain.name***。如果远程机器是 **example.com** 且您在该其机器中的用户名为 john，请输入 **ssh john@example.com**。
>
> **mount -o remount *file-system*** 命令会重新挂载命名的文件系统。例如：要重新挂载 **/home** 文件系统，则命令为 **mount -o remount /home**。
>
> 要查看目前安装的软件包版本，请使用 **rpm -q *package*** 命令。它会返回以下结果：***package-version-release***。

请注意上述使用黑斜体的文字 -- username、domain.name、file-system、package、version 和 release。每个字都是一个站位符，可用作您执行命令时输入的文本，也可作为该系统显示的文本。

不考虑工作中显示标题的标准用法，斜体表示第一次使用某个新且重要的用语。例如：

> Publican 是一个 *DocBook* 发布系统。

## 1.2. 抬升式引用约定

终端输出和源代码列表要与周围文本明显分开。

将发送到终端的输出设定为 **Mono-spaced Roman** 并显示为：

```
books        Desktop   documentation  drafts  mss    photos   stuff  svn
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

源码列表也设为 **Mono-spaced Roman**，但添加下面突出的语法：

```
package org.jboss.book.jca.ex1;

import javax.naming.InitialContext;

public class ExClient
{
   public static void main(String args[])
       throws Exception
   {
      InitialContext iniCtx = new InitialContext();
      Object         ref    = iniCtx.lookup("EchoBean");
      EchoHome       home   = (EchoHome) ref;
      Echo           echo   = home.create();

      System.out.println("Created Echo");

      System.out.println("Echo.echo('Hello') = " + echo.echo("Hello"));
   }
}
```

## 1.3. 备注及警告

最后，我们使用三种视觉形式来突出那些可能被忽视的信息。

**注意**

备注是对手头任务的提示、捷径或者备选的解决方法。忽略提示不会造成负面后果，但您可能会错过一个更省事的诀窍。

**重要**

重要框中的内容是那些容易错过的事情：配置更改只可用于当前会话，或者在应用更新前要重启的服务。忽略'重要'框中的内容不会造成数据丢失但可能会让您抓狂。

**警告**

警告是不应被忽略的。忽略警告信息很可能导致数据丢失。

# 第 1 章 配置

本书指导系统管理员配置 **JBoss SOA 平台**企业级总线以用于金融机构、银行、电信公司以及其他大型企业。本章解释了在读者完成*《起步指南》*里的初始安装后如何配置软件。

## 1.1. 基本概念介绍

*面向服务架构（Service Oriented Architecture，SOA）*不是一个单一的程序或技术。相反，我们应该将其视作软件架构或设计模式。硬件的*总线*是一个物理连接器，它连接多个系统和子系统。它不是在两个系统间存在的大量点对点连接器，而是简单地将每个系统连接到总线上。*企业级服务总线（Enterprise Service Bus，ESB）*在软件上实现了相同的功能。

不是通过连接（称为*端点*）在总线上传输数据，ESB 在逻辑上位于消息系统的架构层里。这个消息系统允许服务间通过 ESB 异步通讯。事实上，当你使用 ESB 时，任何东西从概念上来说都是*服务（Service）*（在这个上下文里是你的应用程序软件）或者在服务间传送的*消息（Message）*。

请注意，"服务" 并不一定是 Web Service。其他类型的应用程序，使用不同的传输协议如 FTP 和 JMS，都可以称作 "服务"。

**注意**

现在，你可能会想知道 ESB 是否和 SOA 是同一个东西。答案是，“不完全是”。这是因为 ESB 实际上并不组成一个自身的 SOA。相反，它提供可以构建 SOA 的许多工具。在实践中，它利用了 SOA 所需要的*松耦合（loose-coupling）*和*异步消息传送（asynchronous message passing）*。请不要仅仅将 SOA 视作软件：它是一系列的原则、模式和最佳实践。

**JBoss 企业级服务总线**是红帽提供的一个开源 ESB。它支持：

- 多个传输协议
- *listener-action 模型*（用于松耦合服务）
- *基于内容的路由（content-based routing）*（通过 **JBoss Rules**）
- 和 **JBoss 商业过程管理者**集成。（以使用 *service orchestration*）

## 1.2. 独立服务器

如果你想在同一台机器上运行 JBoss 应用服务器（AS）和 **JBoss 企业级 SOA 平台**，你需要阅读 http://www.jboss.org/community/wiki/ConfiguringMultipleJBossInstancesOnOnemachine。

## 1.3. 群集的 ESB 服务

要设立群集的企业级服务总线环境，请以下列两种方式之一配置节点：

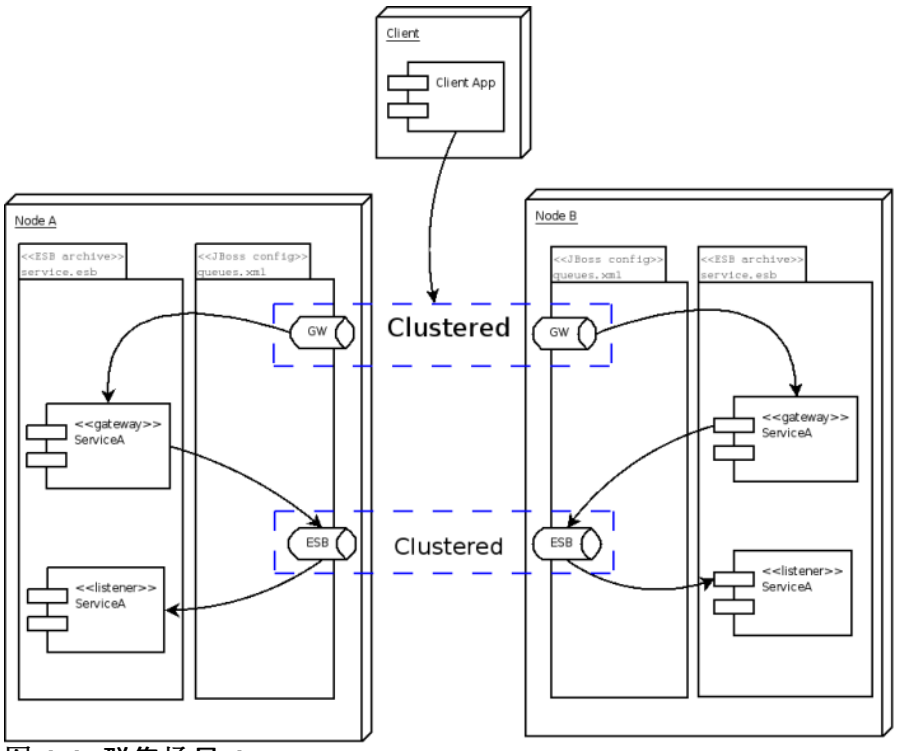



图 **1.1. 群集场景 1**

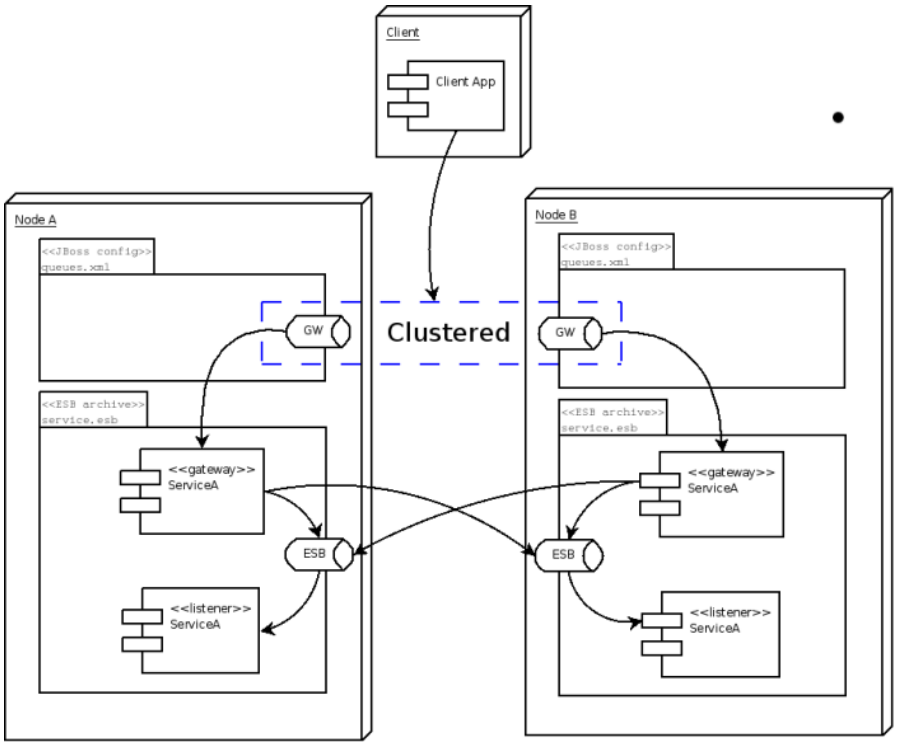



图 **1.2. 群集场景 2**

这些方法的区别是可识别企业级服务总线的网关在第一种情况下是*群集的*，而在第二种情况下是*复制的*。

其他需要记住的两点是：

- 每个群集队列在任何时候都必须部署在每个群集节点上。

- 客户必须在 **jndi.properties** 文件里列出所有的节点。这个示例代码展示了如何进行：

```
java.naming.provider.url=jnp://jawa01:1099
java.naming.provider.url=jnp://jawa02:1099
```

## 1.4. JMS 提供者

JBoss 企业级 SOA 平台目前支持下列*消息服务（Java Message Service，JMS）*提供者：

- **JBoss Messaging**
- **IBM Websphere MQ Series 6.0**
- **Tibco EMS**

**重要**

我们推荐 **JBoss Messaging** 并把它包括在缺省的配置里。
任何兼容 *JSR-914*（http://jcp.org/en/jsr/detail?id=914）的 JMS 实现都应该能够运行，如 **Apache ActiveMQ** 和**OracleAQ**。然而，其他 JMS 提供者还没有进行完备的测试，所以目前还不被支持。
如果你想尝试其他 JMS 提供者，你应该和参考相关文档。

**警告**

本节内容并不打算替代 JMS 实现的配置文档。对于高级的功能，如 *群集（clustering）*和*管理（management）*，你应该参考相关文档，。

### 我怎样才能进行配置？

你可以配置 **JMS Listeners** 和 **JMS Gateways**，通过在其配置文件（**jbossesb-listener.xml** 和 **jbossesb-gateway.xml**）里指定下面的参数使它们侦听*队列*和*主题*：

- jndi-URL
- jndi-context-factory
- jndi-pkg-prefix
- connection-factory
- destination-type
- destination-name

**重要**

你需要确保在 classpat 里包含 JMS 提供者的客户端 **JAR** 文件。

**重要**

在下面的章节里，我们将假设：

- 你的 JMS 提供者运行在 localhost 上。
- connection-factory 是 'ConnectionFactory'
- destination-type 是 'queue'
- destination-name 是 'queue/A'

在阅读本手册后面的内容时请记住这些假设。

**注意**

你可以配置每个 JMSList**`JMSListener`** 和 **`JMSGateway`** 都使用自己的 JMS 提供者，所以你可以在部署里使用多个提供者。

在使用 JMS 时，SOA 平台利用一个*连接池（Connection Pool）*来提高性能。这个池的默认大小是 **`20`**。要修改这个值，请设置 ESB 配置文件里的 **`transports`** 部分的 org.jboss.soa.esb.jms.connectionPool 属性。

如果不能获得初始的会话，这个服务将在 30 秒内不停地尝试。（这个超时时间可以通过 org.jboss.soa.esb.jms.sessionSleep 属性进行配置。）

### 1.4.1. 每个连接的最大会话数

顾名思义，**`JmsConnectionPool`** 池化所有 JMS 组件（如 JMS Listener、Courier 和 Router）使用的 JMS 会话。

请注意，有些 JMS 提供者限制了每个连接的会话数量。此时，对于每个 **`JmsConnectionPool`** 实例，你可以指定每个 JMS 连接创建的最大会话数目。要实现这一点，你可以在 JMS 组件的 JNDI 配置文件里指定一个或两个下列参数：

**`org.jboss.esb.jms.max.sessions.per.connection`**
: 这个是每个连接允许的会话的最大数目（包括 XA 和非 XA 会话实例）。缺省值是 **`JmsConnectionPool`** 所允许的最大 JMS 会话数目，通常是 **`20`**（如 **`jbossesb-properties.xml`** 里所配置的）。

**`org.jboss.esb.jms.max.xa.sessions.per.connection`**
: 这个每个连接所允许的最大 XA 会话数目，它的缺省值是 **`org.jboss.esb.jms.max.sessions.per.connection`**。

如果上面两个参数都没有配置，**`JmsConnectionPool`** 将创建一个 JMS 连接并用它来创建所有的 JMS 会话。

通过 JMS 提供者配置上的普通属性来修改配置，如下面的例子所示：

```
<jms-provider ...>
    <property
        name="org.jboss.esb.jms.max.sessions.per.connection" value="5" />
    <property
        name="org.jboss.esb.jms.max.xa.sessions.per.connection" value="1" />
    <!-- And add providers.... -->
</jms-provider>
```

### 1.4.2. JBossMessaging

**JBoss Messaging** 是 **JBoss SOA 平台**的缺省 JMS 提供者。

设置它的参数以反映下面的配置：

```
jndi-URL="localhost”
jndi-context-factory="org.jnp.interfaces.NamingContextFactory"
connection-factory="ConnectionFactory"
destination-type="queue"
destination-name="queue/A"
```

Jar 文件 **`jboss-messaging-client.jar`** 必须包含在你的 classpath 里。

**注意**

配置群集的 **JBoss Messaging** 可提供 JMS 的负载平衡和实效切换功能。因为在不同的 **JBoss Messaging** 版本里，这个功能已经有所变化，你应该参考 JBoss Messaging 文档以获取更多细节。

### 1.4.3. Apache ActiveMQ

**警告**

**Apache ActiveMQ** 还没有进行完备的测试，所以它还不被支持。

如要使用，请设置下列参数：

```
jndi-URL="tcp://localhost:61616”
jndi-context-factory="org.apache.activemq.jndi.ActiveMQInitialContextFactory"
connection-factory="ConnectionFactory"
destination-type="queue"
destination-name="queue/A"
```

然后，确保 class-path 包含：

- **`activemq-core-4.x`**
- **`backport-util-concurrent-2.1.jar`**

这些文件都位于 **`lib/ext/jms/activemq`** 子目录下。

### 1.4.4. IBM Websphere MQ Series 6.0

使用下列 JNDI 参数来配置任何 ESB 的 JMS 组件上的 **IBM Websphere MQ** series (WMQ) ：

```
jndi-URL="localhost:1414/SYSTEM.DEF.SVRCONN”
jndi-context-factory="com.ibm.mq.jms.context.WMQInitialContextFactory"
connection-factory="WMQQueueManager"
destination-type="queue"
destination-name="QUEUEA"
```

### 其他 WMQ JAR 文件

请注意，connection-factory 设置应该引用 WMQ 队列管理者的名称。（这必须位于配置 JMS 目的地的 WMQ 服务器上）

为了从 **JBoss Enterprise Service Bus** 接

下列文件应该包含在 class-path 里：

- **`com.ibm.mq.pcf.jar`**
- **`mqcontext.jar`**
- **`com.ibm.mq.jar`**（客户端 jar 文件）
- **`com.ibm.mqjms.jar`** (客户端 jar 文件)

**注意**

这些 JAR 文件可以在 WMQ 的 **`Java/lib`** 目录里找到。请注意，MQ 5.3 和 MQ 6.0 的客户端 Jar 文件不同，但 6.0 的 Jar 文件应该向后兼容。这些 Jar 文件不是开源的也不是我们提供的。你需要从 Websphere AS 和 MQ 安装里获取它们。

#### 1.4.4.1. XA 连接

要在使用 Websphere MQ 的 **JBoss 企业级服务总线**上管理任何 *XA 连接资源*，请在 ***`$SOA_ROOT`*`/server/`*`$PROFILE`*`/lib/`** 目录里安装 **`WMQ`** **扩展客户端** JAR 文件。这个 JAR 文件名为 **`com.ibm.mqetclient.jar`** 且必须从 IBM 的合作伙伴或代理那里获取。

一旦安装了这个库，Websphere MQ 将每个 JMS 连接的 JMS 服务会话数据限制为 1。这个限制应用于 XA 和非 XA 连接。

然后，配置 **`JmsConnectionPool`** 的 org.jboss.esb.jms.max.sessions.per.connection 属性。简单地将这个属性值设置为 **`1`** 就可以了。（更多信息请参考本文档的 "Max Sessions Per Connection" 章节）

**注意**

设置这个属性暗示了 **`JmsConnectionPool`** 将消费 WMQ 提供者上的多个 JMS 连接。
请注意，这个设置不需要为任何基于 Websphere MQ 的 *J2EE 连接器架构（JCA）* (JCA) 提供者配置进行调整。

#### 1.4.4.2. JCA 适配器

**注意**

**IBM Websphere MQ** 的缺省安装没有包含 JCA 适配器。

过程 1.1. 配置 JCA 适配器的步骤

1. **更新 IBM Websphere MQ**

   首先，更新 **IBM Websphere MQ** 的版本到最新的 6.0.2.1 以获得 J2EE 连接器架构适配器。

2. **在 JBoss 企业级 SOA 平台服务器上部署适配器**

   WMQ JCA 适配器 **`wmq.jmsra.rar`**，可以在 WebSphereMQ 的 **`Java/lib/jca/`** 下找到。要部署这个适配器，复制这个文件到 SOA 平台服务器的 **`${SOA_ROOT}/server/${CONFIG}/deploy/`** 目录里。

3. **创建一个 JCA 连接工厂的配置**

   通过创建 **JCA 连接工厂**配置文件来配置 SOA 平台服务器以使用 WMQ JCA 适配器。（这个文件名无关紧要，但应该具有描述性。）然后复制这个文件到 SOA 平台服务器的 **`${SOA_ROOT}/server/{CONFIG}/deploy/`** 目录下。

   **例 1.1. JCA 连接工厂的配置**

```
<?xml version="1.0" encoding="UTF-8"?>
<connection-factories>

<mbean code="org.jboss.jms.jndi.JMSProviderLoader"
    name=":service=JMSProviderLoader,name=WSMQJmsProvider">
    <attribute name="ProviderName">WSMQProvider</attribute>
    <attribute name="ProviderAdapterClass">
        org.jboss.jms.jndi.JNDIProviderAdapter
    </attribute>
    <attribute name="QueueFactoryRef">ConnectionFactory</attribute>
    <attribute name="TopicFactoryRef">ConnectionFactory</attribute>
    <attribute name="FactoryRef">ConnectionFactory</attribute>
    <attribute name="Properties">
java.naming.factory.initial=com.ibm.mq.jms.context.WMQInitialContextFactory
java.naming.provider.url=mqserver.domain.com:1414/SYSTEM.DEF.SVRCONN
    </attribute>
</mbean>

<tx-connection-factory>
    <jndi-name>WSMQJmsXA</jndi-name>
    <xa-transaction/>
    <rar-name>jms-ra.rar</rar-name>
    <connection-definition>
        org.jboss.resource.adapter.jms.JmsConnectionFactory
    </connection-definition>
    <config-property name="SessionDefaultType" type="java.lang.String">
        javax.jms.Queue
    </config-property>
    <config-property name="JmsProviderAdapterJNDI"
type="java.lang.String">
        java:/WSMQProvider
    </config-property>
    <max-pool-size>20</max-pool-size>
    <security-domain-and-application>
        JmsXARealm
    </security-domain-and-application>
</tx-connection-factory>

</connection-factories>
```

J2EE 连接器架构适配器现在可用了。你可以通过配置好的连接工厂访问它。

连接工厂现在可以使用了。你可以通过 JCA 提供者配置提供事务性上下文来实现。（下面的例子在 **${SOA_ROOT}/server/${CONFIG}/deploy/jbpm.esb/META-INF/jboss-esb.xml** 里进行配置）

```
<?xml version="1.0" encoding="UTF-8"?>
<jbossesb xmlns="http://anonsvn.labs.jboss.com/labs/jbossesb/trunk/product/
etc/schemas/xml/jbossesb-1.0.1.xsd" parameterReloadSecs="5">

<providers>
  <jms-jca-provider connection-factory="ConnectionFactory"
    jndi-URL="mqserver.domain.com:1414/SYSTEM.DEF.SVRCONN"
    jndi-context-factory="com.ibm.mq.jms.context.WMQInitialContextFactory"
    name="CallbackQueue-JMS-Provider"
    providerAdapterJNDI="java:/WSMQProvider">
    <jms-bus busid="jBPMCallbackBus">
      <jms-message-filter dest-name="queue/CallbackQueue"
        dest-type="QUEUE"/>
    </jms-bus>
  </jms-jca-provider>
</providers>

<services>
  <service category="JBossESB-Internal"
    description="Service which makes Callbacks into jBPM"
    name="JBpmCallbackService">
    <listeners>
      <jms-listener busidref="jBPMCallbackBus" maxThreads="1"
        name="JMS-DCQListener"/>
    </listeners>
    <actions mep="OneWay">
      <action
        class="org.jboss.soa.esb.services.jbpm.actions.JBpmCallback"
        name="action"/>
    </actions>
  </service>
</services>

</jbossesb>
```

#### 1.4.4.3. 验证

**警告**

如果正在使用 **Websphere MQ**，下面的异常消息可能会出现。通过添加运行 **JBoss 企业级 SOA 平台** 到用户的名字到 **mqm** 组可以解决这个问题。

```
Message: Unable to get a MQ series Queue Manager or Queue Connection. Reason:
failed to
create connection -javax.jms.JMSSecurityException: MQJMS2013: invalid
security
authentication supplied for MQQueueManager
```

访问 MQ 的用户必须是 **mqm** 组的成员。

**警告**

Oracle 高级队列 还未进行充分测试，它不是被支持的 JMS 实现。

对于 Oracle AQ，你应该这样设置参数：

```
connection-factory=”QueueConnectionFactory”
```

并使用下列的属性：

```
<property name="java.naming.factory.initial"
    value="org.jboss.soa.esb.oracle.aq.AQInitialContextFactory"/>
<property name="java.naming.oracle.aq.user" value="<user>"/>
<property name="java.naming.oracle.aq.password" value="<pw>"/>
<property name="java.naming.oracle.aq.server" value="<server>"/>
<property name="java.naming.oracle.aq.instance" value="<instance>"/>
<property name="java.naming.oracle.aq.schema" value="<schema>"/>
<property name="java.naming.oracle.aq.port" value="1521"/>
<property name="java.naming.oracle.aq.driver" value="thin"/>
```

**注意**

你可能注意到了对 **InitialContext** 工厂的引用。只有你希望避免 Oracle 高级队列注册 LDAP 服务器时，它才是必需的。
**AqinitialContextFactory** 引用 **org.jboss.soa.esb.oracle.aq-4.2.jar** 里的代码。这个 jar 文件位于 **plugins/org.jboss.soa.esb.oracle.aq**，你需要把它部署在 **jbossesb.sar/lib** 目录里。

当创建队列时，请确保选择 **SYS AQ$_JMS_MESSAGE** 的 payload 类型。

**注意**

你可以在 ***${SOA_ROOT}*/samples/quickstarts/helloworld_action/oracle-aq/** 目录里找到一个 **jboss-esb.xml** 配置文件示例。

### 1.4.6. Tibco 企业级消息服务

要使用 Tibco 企业级消息服务，请设置下列参数：

```
jndi-URL="tcp://localhost:7222”
jndi-context-factory=”com.tibco.tibjms.naming.TibjmsInitialContextFactory"
connection-factory="QueueConnectionFactory"
destination-type="queue"
destination-name="<queue-name>"
```

然后，检查 **Tibco 企业级消息服务**附带的客户端 JAR 是否包含在 classpath 里。（这些文件位于 **tibco/ems/clients/java** 目录里）

- **jaxp.jar**

- **jndi.jar**
- **tibcrypt.jar**
- **tibjmsapps.jar**
- **tibrvjms.jar**
- **jms.jar**
- **jta-spec1_0_1.jar**
- **tibjmsadmin.jar**
- **tibjms.jar**

**注意**

TibcoEMS 4.4.1 和 5.0 已经在 JBoss 企业级 SOA 平台下进行了测试。

### 1.4.7. 扩展属性

在缺省情况下，用于获取 JMS 资源的 JNDI 配置将继承所有名称以 "java.naming" 前缀的属性。一些 JMS 提供者可以指定使用不同前缀的属性。

**注意**

某些 Java 消息服务提供者指定了不同的命名前缀。为了支持这些不同的前缀，红帽提供了可以为每个提供者指定单独属性前缀的功能。
要使用这个功能，配置相关 jms-provider 元素的 jms-provider 属性来配置这些前缀，它是一个逗号隔开的列表。（扩展属性也可以在相同的位置进行配置。）

```
<jms-provider name="JMS" connection-factory="ConnectionFactory">
    <property name="jndi-prefixes" value="test.prefix."/>
    <property name="test.prefix.extension1" value="extension1"/>
    <property name="test.prefix.extension2" value="extension2"/>
</jms-provider>
```

## 1.5. 数据库配置

SOA 平台使用数据库来持久化**服务注册表** 和**消息**库（**Message-Store**）。

数据库配置是由**数据**库**配置脚本**处理的。更多信息请参考*《SOA 起步指南》*。**数据**库**配置脚本**是我们支持的用于数据库配置的机制。本节剩余的内容仅供参考。

**注意**

用于消息存储数据库配置的 SQL 脚本可以在 **PROFILE/deploy/jbossesb.esb/message-store-sql** 目录里找到。服务注册表，**jUDDI v3**，使用 Hibernate 来持久化数据，因此对于 jUDDI v3 来说没有 SQL 脚本。

### 1.5.1. 手动切换数据库

本节解释了如何从缺省的数据库(**Hypersonic**) 移植到 **PostgreSQL**。这些步骤对于其他数据库也是几乎一

样的。

**警告**

本节仅可用于演示目的。数据库配置不应该手动完成。本书包含了这些说明以展示 **Database 配置脚本**如何工作且只应作为参考使用。手动配置可能使 **Database 配置脚本**无法运行。如果要手动配置数据库，请联系红帽支持以寻求帮助。

1. 删除 **deploy/hsqldb-ds.xml** 并在 **deploy/postgres-ds.xml** 里添加下面一行：

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
  <local-tx-datasource>
    <jndi-name>DefaultDS</jndi-name>
    <connection-url>jdbc:postgresql://host:port/database</connection-url>
    <driver-class>org.postgresql.Driver</driver-class>
    <user-name>username</user-name>
    <password>password</password>
      <metadata>
         <type-mapping>PostgreSQL 8.3</type-mapping>
      </metadata>
    <check-valid-connection-sql>select count(*) from jbm_user</check-valid-
connection-sql>
  </local-tx-datasource>
</datasources>
```

   你可以根据需要修改连接参数。然而，DS 名称必须是 **DefaultDS**。

2. Replace the contents of **deploy/jbossesb-registry.sar/juddi-ds.xml** with the same configuration as that created in the previous step (change the database name if need be.) Ensure that jndi-name (**juddiDB**) is kept.

3. Replace the contents of **deploy/jbossesb.esb/message-store-ds.xml** with the same configuration that was created in step one (change the database name if need be.) Again, ensure that jndi-name (**JBossESBDS**) is kept.

4. 将 **deploy/jbossesb.esb/jbossesb-service.xml** 文件里的 **message-store-sql** 元素的数据库名替换为：

```
<?xml version="1.0" encoding="UTF-8"?>

<server>
   <mbean code="org.jboss.internal.soa.esb.dependencies.DatabaseInitializer"
       name="jboss.esb:service=MessageStoreDatabaseInitializer">
      <attribute name="Datasource">java:/JBossESBDS</attribute>
      <attribute name="ExistsSql">select * from message</attribute>
      <attribute name="SqlFiles">
         message-store-sql/postgresql/create_database.sql
      </attribute>
      <depends>jboss.jca:service=DataSourceBinding,name=JBossESBDS</depends>
   </mbean>
</server>
```

5. 编辑 **jbossesb-registry.sar/META-INF/persistence.xml** 文件。hibernate.dialect 属性必须设置为将用作数据源的数据库的类型。（例如，它可能是 **org.hibernate.dialect.PostgreSQLDialect**）

6. 用你正运行的 JBM 里的 **postgres-persistence-service.xml** 替换 **deploy/jboss-messaging/hsqldb-persistence-service.xml**。

   请注意，版本号需要匹配，如果不同可能无法工作。这些文件可以在 JBM 的 **src/etc/server/default/deploy** 里找到。
7. 把数据库驱动复制到服务器的 **lib** 目录并启动服务器。
8. 启动服务器。

## 1.6. 使用 JSR-170 消息库

多条消息存储可以通过 JBoss SOA 平台的插件架构来实现。其中一个选择是用 JSR-170 兼容的 *Java content repository* (JCR) 替换缺省的数据库。这个实现被称为 **Apache Jackrabbit**。

要启用它，在 ***${SOA_ROOT}*/server/*${CONFIG}*/deploy/jbossesb.sar/jbossesb-properties.xml** 的 "Core" 部分添加下列属性：

```
<property name="org.jboss.soa.esb.persistence.base.plugin.jcr" value=
"org.jboss.internal.soa.esb.persistence.format.jcr.JCRMessageStorePlugin"
/>
```

这会在可用的消息库列表里添加 JCR 插件。JCR 消息库可以通过 JNDI 使用现有的库或着在本地创建独立的实例。下面的属性列表应该添加到 **jbossesb-properties.xml** 里的 "dbstore" 部分来配置对库的访问：

```
<property name="org.jboss.soa.esb.persistence.jcr.jndi.path" value="jcr"/>
<property name="org.jboss.soa.esb.persistence.jcr.username"
  value="username"/>
<property name="org.jboss.soa.esb.persistence.jcr.password"
  value="password"/>
<property name="org.jboss.soa.esb.persistence.jcr.root.node.path"
  value="JBossESB/MessageStore"/>
```

- jcr.jndi.path - 寻找库的可选 JNDI 路径。如果没有指定，新的库将根据 ***${SOA_ROOT}*/server/*${CONFIG}*/deploy/jbossesb.esb/repository.xml** 创建，库数据存储在 ***${SOA_ROOT}*/server/*${CONFIG}*/data/repository/** 目录里。
- jcr.username - 获取库会话的用户名
- jcr.password - 获取库会话的密码
- jcr.root.node.path - 相对消息存储的库的根目录的路径

要快速的测试你的 JCR 消息库是否配置正确，可以在现有服务里添加 **org.jboss.soa.esb.actions.persistence.StoreJCRMessage** action。这个 action 将试图当前的消息存储在 JCR 库里。

## 1.7. 消息跟踪

跟踪所有通过 JBoss SOA 平台发送的消息是有可能的。基于许多原因，如审计和调试，这经常是必需的。消息必须用消息头部信息的 MessageID 字段唯一标识以便跟踪。这会在 *《程序员指南》* 里提到。

所有 JBoss SOA 组件和消息的交互都会被登记。日志消息将包含和消息相关的头部信息、启用多个 SOA 平台实例间的相互关系。在输出里寻找下列信息可以识别这些消息：

```
header: [ To: EPR: PortReference < <wsa:Address ftp://foo.bar/> >,
From: null, ReplyTo: EPR: PortReference < <wsa:Address http://bar.
foo/> >, FaultTo: null, Action: urn:dowork, MessageID: urn:foo/bar
/1234, RelatesTo: null ]
```

而且，你可以启用一个日志*元数据过滤器（MetaData Filter）*，它的唯一作用是在消息输入到 SOA 平台组件或者从中输出时发布日志报告。这个过滤器，**`org.jboss.internal.soa.esb.message.filter.TraceFilter`**，可以和其他过滤器一起添加到 **JBossESB** 配置文件的 Filter 部分，因为它对消息的输入或输出并没有影响。

每当消息通过这个过滤器时，你可以看到下列 **`Information`** 级别的日志报告：

```
TraceFilter.onOutput ( header: [ To: EPR: PortReference < <wsa:Add
ress ftp://foo.bar/> >, From: null, ReplyTo: EPR: PortReference <
<wsa:Address http://bar.foo/> >, FaultTo: null, Action: urn:dowork
, MessageID: urn:foo/bar/1234, RelatesTo: null ] )
```

```
TraceFilter.onInput ( header: [ To: EPR: PortReference < <wsa:Addr
ess ftp://foo.bar/> >, From: null, ReplyTo: EPR: PortReference < <
wsa:Address http://bar.foo/> >, FaultTo: null, Action: urn:dowork,
MessageID: urn:foo/bar/1234, RelatesTo: null ] )
```

只有属性 org.jboss.soa.esb.messagetrace 被设置为 **`off/OFF`** 时，TraceFilter 才会记录日志。其缺省色设置是 **`off/OFF`**。如果启用，它将为所有通过它的消息记录日志。然而你可以对哪些消息记录日志或者忽略来启用更细颗粒度的控制，为此你需要把这个属性设置为 **`off/OFF`**。
org.jboss.soa.esb.message.unloggable 属性为 **`yes/YES`** 的消息将被此过滤器所忽略。

## 1.8. 对群集和失效切换的支持

**JBoss SOA 平台**支持 *stateless* 服务的*失效切换*。详情请参考*《程序员指南》*，但请注意下列要点：

- 因为 **`ServiceInvoker`** 对用户隐藏了大部分的失效切换的复杂性，它只能用于原生的 SOA 平台消息。而且，不是所有的网关都进行了修改以适应 ServiceInvoker，所以转入的 SOA-unaware 的消息并不总能利用服务的失效切换。
- 当 ServiceInvoker 试图递送消息时，它可以选择多个端点引用（Endpoint References，EPR）。为了帮助选择端点，你可以配置一个策略。在 **`jbossesb-properties.xml`** 里，你可以选择 org.jboss.soa.esb.loadbalancer.policy。现在系统提供了 3 个策略，你也可以创建自己的策略。
  1. First Available：如果找到了健康的**服务绑定**，它将被使用，直至其终止。然后将使用列表里的下一个端点引用。

     使用这个策略时，两个服务实例间没有负载平衡。
  2. Round Robin：典型的负载平衡策略，每个端点引用（End-point Reference）都会按照列表里的顺序进行选择。
  3. Random Robin：它和前一策略类似，但随机进行选择。
- 策略使用的端点引用列表可能会越来越短，因为终止运行的 EPR 会逐渐从列表里删除。当这个列表为空或者超过了列表缓存的 time-to-live 时，ServiceInvoker 将从注册表里获取一个新的 EPR 列表。org.jboss.soa.esb.registry.cache.life 属性缺省为 60000 毫秒，但你可以在 **`jbossesb-properties`** 文件里对其进行设置。
- 如果端点引用都不能工作，则可以使用消息重递送服务（Message Redelivery Service）。
- 如果在群集里的多个节点上运行相同的服务，在服务可以完全工作在群集环境之前，你需要等待服务注册表缓存重新验证。你可以在 ***`${SOA_ROOT}`*`/server/`*`${CONFIG}`*`/deploy/jbossesb.sar/jbossesb-properties.xml`** 里设

置这个缓存重验证的超时时间。

```
<properties name="core">
  <!-- 60 seconds is the default -->
  <property name="org.jboss.soa.esb.registry.cache.life" value="60000"/>
</properties>
```

- 如果你把 org.jboss.soa.esb.failure.detect.removeDeadEPR 设置为 **true**，每当 **Service Invoker** 怀疑端点引用已经失败，它就会把该 EPR 从注册表里删除。它使用要特别小心，所以缺省的设置是 **false**。对于超载的服务，也可能错误地从注册表删除其 EPR。这些 "orphaned" 服务将不会再进行任何交互且必须重新启动才可以使用。

## 1.9. 使用 OpenSSO

**JBoss SOA 平台**包含 *Open Web 单点登录*服务软件（**OpenSSO**）。使用它可以减缓透明服务的实现。

**注意**

关于 **OpenSSO** 的更多信息，请访问：http://opensso.dev.java.net。

### 1.9.1. 在 Tomcat 里安装和配置 OpenSSO

在 JBoss SOA 平台里部署 **OpenSSO** 有一个已知的问题，但它也可以部署到 SOA 平台里使用的其他 Web 容器里。

**注意**

你可以在 https://jira.jboss.org/jira/browse/SOA-731 里找到关于部署问题的细节。

这里提供的说明用于在 **Tomcat** 里部署 **OpenSSO**。关于在其他 We 容器里使用 OpenSSO 的信息，你可以在 https://opensso.dev.java.net/public/use/docs/fampdf/index.html 里找到。

**过程 1.2. 在 Tomcat 里部署 OpenSSO**

1. 从 Apache 网站 http://tomcat.apache.org 里下载 **Tomcat**。

   **注意**

   红帽客户可以从软件库获取并安装 **Tomcat** 而不需要从 Apache 网站上去下载。

2. 解压到某个目录。下面的例子假设这个目录是 **/opt/tomcat**。
3. 编辑 **/opt/tomcat/bin/catalina.sh** (**catalina.bat** 用于 Windows 部署）并添加 ***-Xmx1G*** 到 JAVA_OPTS 属性。这指定了 JVM 实例的最大堆大小为 1G 字节。

   **例 1.2. 在 JAVA_OPTS 里指定堆的最大值。**

   ```
   JAVA_OPTS="$JAVA_OPTS -Xmx1G -
   Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager"
   ```

4. 从 **OpenSSO** 网站 https://opensso.dev.java.net/public/use/index.html 里下载 **opensso.zip**（build 4.5）。
5. 解压 **opensso.zip** 并把 **opensso.war** 从 **deployable-war/** 复制到 **/opt/tomcat/webapps/**。
6. 如果你想在相同的主机上部署 **JBoss 企业级 SOA 平台**和 Tomcat，你可以在 **$tomcat/server.xml** 里更新 **Tomcat** 的端口：

   **例 1.3. 更新 Tomcat 的端口**

   ```
   <Connector port="8090" protocol="HTTP/1.1">
   <Connector port="8099" protocol="AJP/1.3" redirectPort="8443"/>
   ```

7. 运行 **/opt/tomcat/bin/startup.sh**（Windows 里是 **startup.bat**）启动 **Tomcat**。
8. 在浏览器里打开 http://localhost:8090/opensso 。
9. 点击 **Create Default Configuration**。这会使 OpenSSO 用缺省值对自己进行配置。
10. 为缺省用户和代理帐号输入密码。这两个帐号不能使用相同的密码。
11. 用前一步骤提供的凭证登录 http://localhost:8090/opensso。

### 1.9.2. 为 JBoss SOA 平台配置 OpenSSO

**openssoclientsdk.jar** 里找到的 **AuthContext** 类执行验证。下面的步骤描述了启用这种集成所要求的配置。

过程 **1.3. 配置 OpenSSO 集成**

1. 编辑 ***${SOA_ROOT}*/server/*${CONFIG}*/conf/login-config.xml**。

   **login-config.xml** 里需要下面的配置来集成 OpenSSO。orgName 和moduleName 属性必须和你在 OpenSSO 系统里配置的值相同。最后的属性显示了加载 **AMConfig.properties** 的位置。

   **例 1.4. 编辑 login-config.xml 里的 OpenSSO 设置**

   ```
   <application-policy name="OpenSSOLogin">
    <authentication>
      <login-module
          code="org.jboss.soa.security.opensso.OpenSSOLoginModule"
          flag="required">
        <module-option name="orgName">opensso</module-option>
        <module-option name="moduleName">DataStore</module-option>
        <module-option name="amPropertiesFile">
          props/AMConfig.properties
        </module-option>
      </login-module>
    </authentication>
   </application-policy>
   ```

2. 配置 ***${SOA_ROOT}*/server/*${CONFIG}*/conf/props/AMConfig.properties**

   在缺省情况下，这个文件包含主机 **localhost**、端口 **8080**、上下文路径为 **opensso** 等信息。你可以手动修改为这些设置，但我们建议你使用**setup.sh** （Windows 部署使用 **setup.bat**）来完成配置。

   setup 脚本位于 ***${SOA_ROOT}*/samples/quickstarts/opensso/opensso-sdk/scripts**。你必须在 **opensso-sdk** 目录里运行这个脚本以确保 classpath 正确。

```
[opensso-sdk]$ ./scripts/setup.sh
Debug directory (make sure this directory exists): /var/tmp
Password of the server application: password
Protocol of the server: http
Host name of the server: gatekeeper.company.com
Port of the server: 8080
Server's deployment URI: opensso
Naming URL (hit enter to accept default value,
http://gatekeeper.company.com:8080/opensso/namingservice):
[opensso-sdk]$
```

这个脚本将提示你输入它所需的值并在 **resources** 目录里创建新的 **AMConfig.properties** 文件。然后你需要复制这个新文件到 ***${SOA_ROOT}*/server/*${CONFIG}*/conf/props/** 并覆盖现有的文件。

完成了上述两个步骤以后，你可以把 **OpenSSOLogin** 模块作为一个 JAAS 插件提供者使用。

你可以把它作为一个标识提供者（identity provider）来设置 SOA 平台的安全性，如：

**例 1.5. 将 OpenSSOLogin 模块作为标识符提供者**

```
<service category="OpenSSO"
    name="SimpleListenerSecured" description="Hello World">
  <security moduleName="OpenSSOLogin" runAs="adminRole"/>
  <listeners>
    <jms-listener name="JMS-Gateway" busidref="quickstartGwChannel"
      maxThreads="1" is-gateway="true"/>
  </listeners>

  <actions mep="OneWay">
    <action name="debug" class="org.jboss.soa.esb.actions.SystemPrintln">
      <property name="printfull" value="false"/>
      <property name="message" value="In Service1"/>
    </action>
  </actions>
</service>
```

学习了本章后，现在你应该有信心配置 SOA 平台了。本书的剩余部分解释了日常管理方面的内容。

# 第 2 章 注册表

所有 **JBoss SOA 平台**的部署的核心是注册表。*《JBoss SOA 平台服务指南》*对其进行了完整的描述，并讨论了相关的配置信息。然而，下面的事情值得注意：

- 服务运行时，它们通常把可以访问的*端点引用（EndPointReference，EPR）*放入注册表里。正常情况下，服务终止时应该把这些 EPR 从注册表里删除。而且，在机器崩溃或未正确部署服务时，一些条目仍然会留在注册表里。这些陈旧的条目阻止随后部署的正确执行。此时，你可以手工地删除这些条目。然而，在进行删除之前确保系统处于非活动状态是很重要的。
- 有一个选项可以让 ESB 在添加新实例之前从注册表里删除所有现有的服务条目。要使用它，只要简单地设置端点引用的 **`remove-old-service`** 标签名为 **`true`**。

**警告**

请注意，这个选项要小心使用，因为整个服务将被删除，包括所有的端点引用。

# 第 3 章 配置 Web 服务的集成

**JBoss SOA** 平台用 *SOAPProcessor* 动作开放 *Webservice 端点*。这个动作把 JBoss Webservices v2.x 容器集成到 JBoss SOA 里，允许你调 JBoss SOA 支持的任何频道上调用 *JBossWS* 端点点

**重要**

**JBossWS 2.0.1.SP2** (native) 或更高版本必须安装在你的 **JBoss SOA 平台**服务器上以能够使用 **`SOAPProcessor`** 行为。

**注意**

更多的细节，你应该参考*《程序员指南》*。

# 第 4 章 缺省的 "ReplyTo" 端点引用

**JBoss SOA** 使用端点引用（Endpoint Reference，EPR）来对消息寻址。如《程序员指南》里所描述的，消息具有含有接收者地址、序列号（用于消息纠正）、可选的回复地址、纠错信息等的头部信息。因为所推荐的 JBoss SOA 的交互模式基于单程（One-way）消息交换，对消息的回复并不需要自动化：它取决于应用程序的发送者是否需要回复。

回复地址是头部路由信息里的可选部分，应用程序在有需要时应该设置它。当需要回复而 **`ReplyTo`** EPR 没有设置时，**JBoss SOA 平台**可以为每种传输使用相应的缺省值。其中一些 **`ReplyTo`** 缺省值需要系统管理员正确的配置 JBoss SOA。

- 对于 JMS 来说，我们使用和用于递送原始请求名称相同的队列，但它以 '_reply' 为前缀。
- 对于 JDBC，我们使用和用于递送原始请求相同的数据库的表名，但以 '_reply_table' 为前缀。而且，请记住新的表和请求表使用相同的字段。
- 对于本地和远程文件则不需要管理性的修改。回复被写入和请求相同的目录，它带有一个唯一的后缀以确保原始发送者可以正确获取回复。

# 第 5 章 ServiceBinding 管理者

如果你想在同一机器上运行多个 **JBoss SOA 平台**服务器，你可能希望使用 *ServiceBinding Manager* 管理者。这个绑定管理者允许你集中所有运行实例的端口配置。JBoss SOA 服务器带有一个简单的绑定配置文件 **`docs/examples/binding-manager/sample-bindings.xml`**。

**注意**

JBoss 应用服务器的文档包含配置 **`ServiceBinding Manager`** 的详细说明。

**注意**

如果 **`jboss-messaging`** 被用作 JMS 提供者，请确保其 **`ServiceBinding Manager`** 配置匹配 **`remoting-service.xml`** 文件里的内容。

# 第 6 章 监控和管理

JBoss 企业级 SOA 平台提供了几种方法来监控和管理服务器。请阅读本章来进行学习。

## 6.1. JMX MBeans

在 **jboss.esb** 域里，你会看到下列类型的 **M-Bean**：

**deployment=<ESB package name>**

**Deployments M-Bean** 显示所有已部署的 ESB 包状态并给出 XML 配置和当前状态的信息。

**listener-name=<Listener name>**

**M-Bean** 显示所有部署的 *listeners* 的信息，包括 XML 配置、启动时间、**maxThreads** 和状态。管理员可以初始化、启动、停止和销毁 listener。

**category=MessageCounter**

消息计数器显示为 **listener** 部署的所有服务、每个服务的独立动作和处理的消息总数、以及每条消息的处理时间。

**service-name=<Service name>**

这个 M-Bean 显示每个服务的统计信息，包括消息计数、状态、消息的平均大小和处理时间。消息计数可以被重置，服务可以被停止和启动。

**注意**

除了上面列出的 M-Bean 以外，JMS 域显示消息队列的统计信息，这对于调试和分析性能是很有用的。

## 6.2. 监控和管理控制台

JBoss 企业级 SOA 平台包含一个**监控和管理控制台**。它从企业级服务总线注册表里的每个节点请求和显示 M-Bean 信息。要访问控制台，在浏览器里输入：http://localhost:8080/jbossesb。

**监控和管理控制台**收集已部署的不同 ESB 服务的性能方面的信息并在一段时期内保留历史状态信息。

**监控和管理控制台**允许系统管理员根据服务、动作、节点、处理时间、失败消息、传输字节、最后成功或失败消息的时间等来获取消息计数。

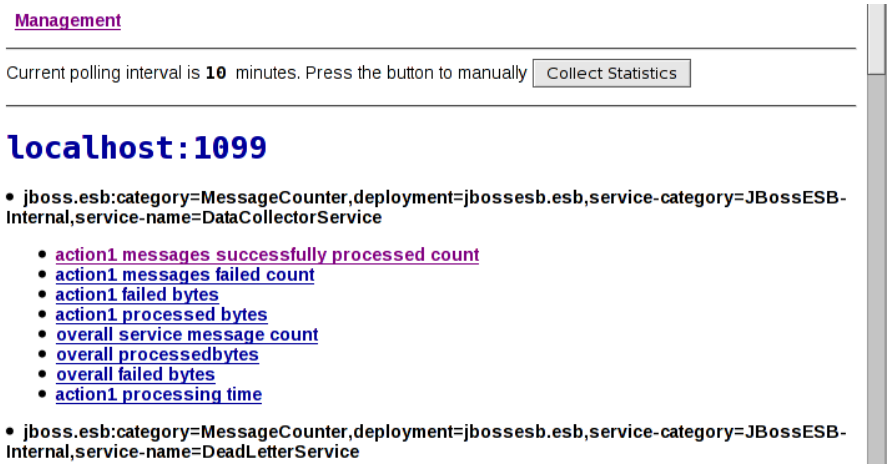



图 6.1. JBoss 企业级服务总线监控和管理控制台

**监控和管理控制台** 缺省安装在 **All** 和 **Production** 服务器配置里。如果需要它也可以轻易安装在其他配置集里。

### 6.2.1. 安装企业级服务总线控制台

在 ***${SOA_ROOT}*/tools/console/management-esb/** 目录里运行 **ant deploy** 来安装 **ESB 控制台**。这个脚本将检查 **jbossesb.sar** 的存在以自动决定是否部署到 **production** 还是 **default** 配置集里。**production** 配置具有优先权。如果两个配置都没找到，则会显示一个错误信息。

要部署不同的服务器配置，你必须首先编辑 **build.xml**。

**过程 6.1. 手动安装控制台**

1. 要覆盖缺省的服务器配置集，编辑 ***${SOA_ROOT}*/tools/console/management-esb/build.xml** 并添如下一行：

   ```
   <property name="org.jboss.esb.server.config" value="my_config"/>
   ```

   上面的 **value** 是你要部署的服务器配置的名称，在这个例子里是 **my_config**。将这一行和其他属性设置添加到文件的开头。
2. 如果控制台的后台数据库没有使用 **Hypersonic**，你会需要编辑 ***${SOA_ROOT}*/tools/console/management-esb/db.properties** 文件来定义正确的数据库设置。（详情请参考 第 6.2.2 节 “使用其他数据库”）
3. 进入 ***${SOA_ROOT}*/tools/console/management-esb** 并运行 **ant deploy**。

**例 6.1. 在 Linux/Unix 上部署企业级服务总线控制台。**

```
$ cd tools/console/management-esb
$ ant deploy
```

### 6.2.2. 使用其他数据库

**Hypersonic 数据库是管理和监控控制台**的缺省后台数据库。然而，红帽不推荐也不对在产品环境里使用 **Hypersonic** 提供支持。如要上线运行，请务必修改为其他配置。本节将描述如何完成这一点。

***${SOA_ROOT}*/tools/console/management-esb/db.properties** 文件定义了控制台的数据库设置

。在缺省情况下，这个文件包含了这样一行：**db=hsqldb**。这一行设置 **Hypersonic** 为要使用的数据库。要修改数据库，只要编辑这一行就可以了。你可以使用下面的设置：

- **hsqldb** - **Hypersonic**
- **mysql** - **MySQL**
- **oracle9i** - **Oracle 9i**
- **oracle10g** - **Oracle 10g**

编辑了这个文件后，按照 第 6.2.1 节 “安装企业级服务总线控制台” 里的说明部署控制台。

然后，添加 JDBC 驱动 JAR 文件到服务器的 ***${SOA_ROOT}*/server/*${CONFIG}*/lib** 目录里。（这个目录里有 JBoss 默认附带的 **hsqldb.jar** ）

对于 **MySQL** 用户，在部署前创建数据库统计信息可能是必要的。请查看 **management-esb/src/main/resources/*${DB}*/** 目录里用于数据库的 **management-ds.xml**。

### 6.2.3. 采集时间间隔

默认的数据采集的时间间隔（称为*轮询间隔*）是 10 分钟。这是在构建时指定的。这个属性是 pollMinuteFrequency，它可在 **management-esb/db.properties** 文件里指定。

*当前的*轮询间隔可以在运行时修改。为此，请使用**监控和管理控制台**或 **jmx-console** 来编辑 **jboss.esb:service=DataFilerScheduler** M-Bean 里的 pollMinuteFrequency 属性。

**注意**

控制台页面顶部的 **Collect Statistics** 按钮可以让用户强制立即采集统计数据。

### 6.2.4. 服务

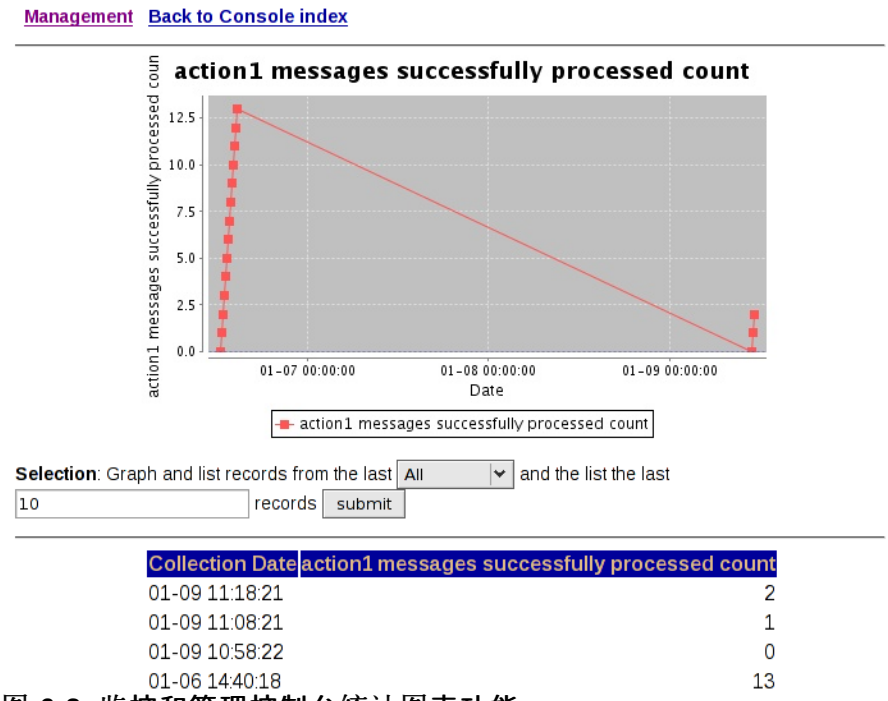
下面是每个 ESB 服务所显示的信息：

- the processing time per action
- processed count per action
- failed count per action
- overall message count per service

点击这些统计数据都会显示一个历史图表页面。

默认会显示最后的 10 条记录。要显示更多信息，请修改 **Display Records** 文本框里的值或图表周期。可用的图表周期是：

- 最后 5 分钟
- 小时
- 天
- 周
- 月
- 所有记录




| Collection Date | action1 messages successfully processed count |
|---|---|
| 01-09 11:18:21 | 2 |
| 01-09 11:08:21 | 1 |
| 01-09 10:58:22 | 0 |
| 01-06 14:40:18 | 13 |

图 **6.2.** 监控和管理控制台统计图表功能

## 6.2.5. 消息计数器

监控和管理控制台也提供一个计数器来为所有通过 ESB 的消息计数。MessageCounter 记录成功和失败的消息计数、以及时间和日期。

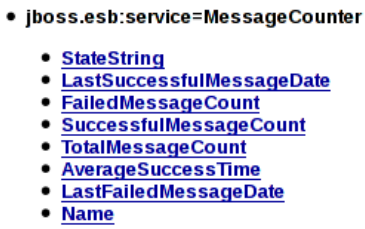



图 **6.3.** 监控和管理控制台消息计数器

## 6.2.6. Smooks Transformations

有一个 M-Bean 跟踪“已处理”的 *Smooks Transformations* 的计数和处理的时间。它也跟踪 *Transformation Chain* 的总计数。

图 **6.4.** 监控和管理控制台的 Transformations 功能

## 6.2.7. Dead Letter 服务

如《*程序员指南*》里提到的，*DeadLetterService*(DLQ) 可以用来存储不能递送的消息。这是一个可以被监控

和检查的 JBossESB 服务。然而，请注意，如果底层的传输具有原生支持（如 JMS），**DeadLetterService** 则不会被使用。此时你应该检查 JBossESB DLQ 以及任何传输专有的设置。

## 6.3. 消息警告

**JBoss Web 控制台**是 JBoss AS 和 JBoss ESB 服务器里的工具，它可以监控和发送基于 **JMX M-Bean** 属性的警告。你可以使用这个功能来接收 ESB 相关事件的警告 - 如 **Dead Letter Service Counter** 计数器达到某个极限值。

下面是配置所需的步骤：

1. 在 **./deploy/mail-service.xml** 里添加 SMTP 设置。
2. 修改 **./deploy/monitoring-service.xml** 取消 EmailAlertListener 部分的注释并添加合适的头部信息。
3. 创建一个 **./deploy** 文件来作为**监控 MBean**。

```
<?xml version="1.0" encoding="UTF-8"?>
<server>
 <mbean code="org.jboss.monitor.ThresholdMonitor"
  name="jboss.monitor:service=ESBDLQMonitor">
  <attribute name="MonitorName">
    ESB DeadLetterQueue Monitor
  </attribute>

  <attribute name="ObservedObject">
   jboss.esb:category=MessageCounter,
deployment=jbossesb.esb,service-name=DeadLetterService
  </attribute>

  <attribute name="ObservedAttribute">
    overall service message count
  </attribute>
  <attribute name="Threshold">4</attribute>
  <attribute name="CompareTo">-1</attribute>
  <attribute name="Period">1000</attribute>
  <attribute name="Enabled">true</attribute>
  <depends-list optional-attribute-name="AlertListeners">
    <depends-list-element>
      jboss.alerts:service=ConsoleAlertListener
    </depends-list-element>
    <depends-list-element>
      jboss.alerts:service=EmailAlertListener
    </depends-list-element>
  </depends-list>
  <depends>jboss.esb:deployment=jbossesb.esb</depends>
 </mbean>
</server>
```

这个 MBean 将作为一个监控程序，一旦 DeadLetterService 计数到 **5**，它将发送电子邮件到 **monitoring-service.xml** 里指定的地址。请注意，这个警告只被发送一次 - 在到达计数极限时。如果你想再次得到警告，你可以重置监控服务 MBean（这个例子里是 **jboss.monitor:service=ESBDLQMonitor**）上的 alerted 标记。

**注意**

关于使用 JBoss Web 控制台进行监控的更多细节，请参考 http://www.jboss.org/community/docs/DOC-12659。

## 6.4. JON for SOA

**JBoss Operations Network** (JON) 是系统管理员用来监控和管理 JBoss SOA 平台服务器的一个工具。

这个软件允许你处理库存、管理、监控、部署和更新任务。它通过带有可定制 Web-portal 界面的集中管理的模型来执行这些工作。

**注意**

如要了解 **JBoss Operations Network** 项目的更多信息，请访问其官方网站 http://www.jboss.com/products/jbosson。

*JON for SOA* 是 **JBoss Operations Network** 软件的一个独立版本，它包括特别为 JBoss SOA 平台设计的功能。本章提供这些功能的概述并假设读者已经具备 JBoss Operations Network 产品的基本知识。

**重要**

和其他的内嵌 JBoss SOA 平台控制台相反，访问 *JON 控制台*并不局限于本地服务器。这赋予了用户更大的灵活性，但也意味着无法依赖这种内在的限制来确保安全性。

### 6.4.1. 把 JBoss SOA 平台服务器添加到 JON 库存里。

你的 JBoss SOA 平台服务器将出现在 JON 里，其资源类型为 JBossAS Server，描述为 "JBoss Enterprise SOA Platform"。

**重要**

当你在 JON 里第一次访问 SOA 平台服务器时，你会看到一个错误信息，这是因为你还没有为合法的 SOA 平台用户提供验证信息。
修改 ***${SOA_ROOT}*/server/*${CONFIG}*/conf/props/soa-users.properties** 文件里的设置可以配置 SOA 用户信息。你把这些信息输入 Connection 属性的 Principal 和 Credentials（用户名 & 密码）里。选择服务器**INVENTORY** 标签页可以访问这些信息。错误信息也包含到连接属性页面的快捷链接。

### 6.4.2. JBoss SOA ESB 统计信息

正确配置了服务器使用 **JBoss Operations Network** 工具后，你应该可以在 **Resources** 菜单上面找到 **JBoss ESB Statistics**。

点击 **JBoss ESB Statistics** 链接以显示 ESB 实例的概述。要查看所有已部署的 ESB 软件包，点击 **JBoss ESB Deployment** 链接。

之后，你可以进入每个 ESB 软件包来查看且组件到详细统计信息。采集和显示的度量信息取决于你查看的

组件。

**JBoss ESB 统计信息**

| | |
|---|---|
| 消息计数（失败） | 最近失败消息的日期 |
| 最近成功消息的日期 | 已处理的字节 |
| 消息计数（成功） | 消息计数（总共） |

**ESB 部署**

| |
|---|
| 部署类型 |

**服务**

| | |
|---|---|
| 失败的字节总数 | 已处理的字节总数 |
| 字节总数 | 消息计数 |
| 消息计数（平均值） | 每分钟消息计数（平均值） |

**Listener 配置**

| | |
|---|---|
| 开始日期 | 线程的最大数目 |
| MEP | 服务类别 |
| 服务描述 | 服务名称 |

**动作**

| | |
|---|---|
| 成功处理的消息（平均值） | 消息计数（平均值） |
| 失败的消息 | 失败的消息（平均值） |
| 处理时间 | 已处理的字节总数 |
| 消息计数 | 成功处理的消息 |
| 字节总数 | 失败的字节总数 |
| 每分钟消息计数（平均值） | 每分钟失败的消息（平均值） |
| 每分钟成功处理的消息（平均值） | |

**注意**

上面所有的度量信息都是不需加以说明的，除了两个例外：

1. **`MEP`** 是一个字符串，它指明使用的消息交换协议（Message Exchange Protocol），其中一个例子是 "OneWay"。
2. **`Deployment Type`** 是一个字符串，它指明部署的 ESB 的类，其中一个例子是 "JBoss4ESBDeployment"。

**重要**

不管数据采集的间隔是多少，ESB 控制台用户界面里显示的消息计数值随着时间的推移会向上增长。例如，如果在 12:00 用户启动 JON 服务器和代理、部署 ESB 归档，并在 13:00 之前通过部署在 ESB 归档里的服务发送 1000 条消息，之后在 14:00 到 15:00 又指定一个数据“度量采集时间表”（期间没有处理任何消息），此时消息计数将仍然是 1000。
用户可以使消息计数器显示零值的唯一办法是将其重置。

所有标准的 **JBoss Operations Network** 功能如警告和图表都可以根据这些统计对每个 ESB 部署、服务或动作进行配置。

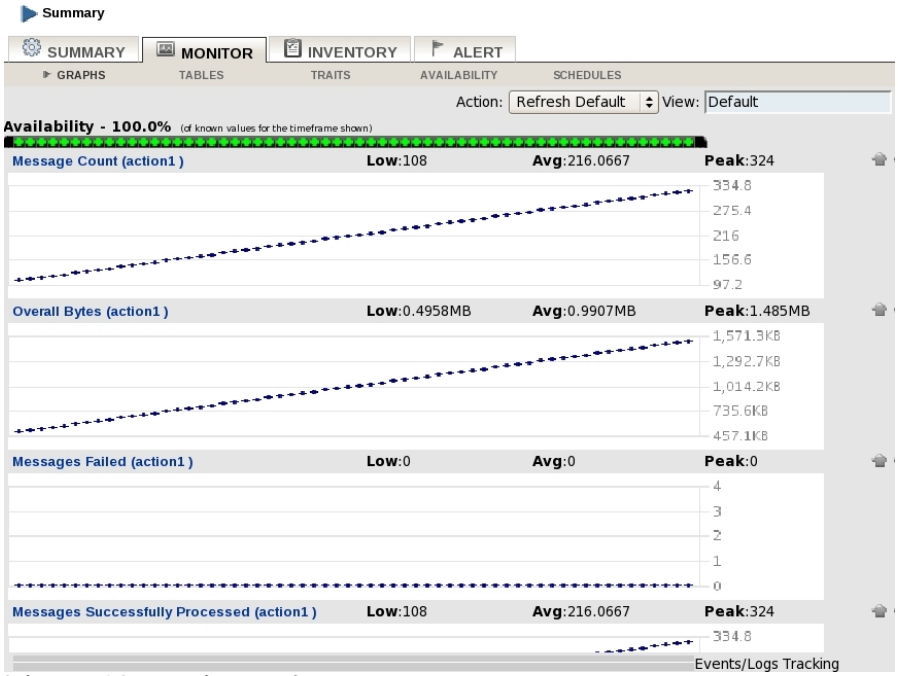



图 **6.5.** 显示的度量信息

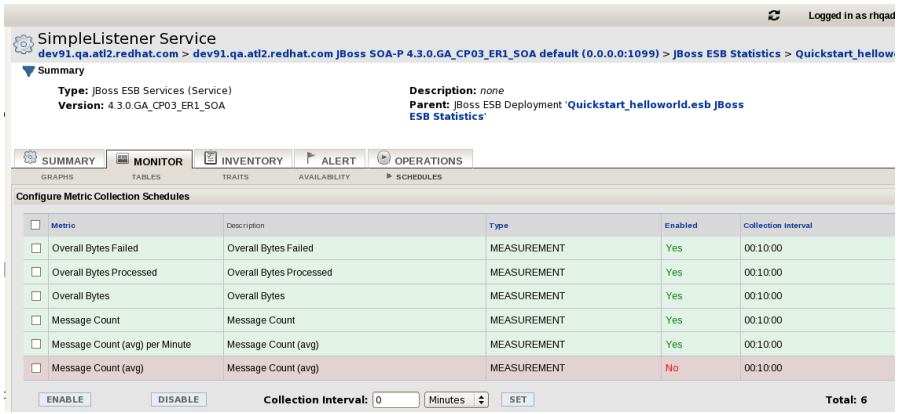



图 **6.6.** 服务度量缺省是禁用的

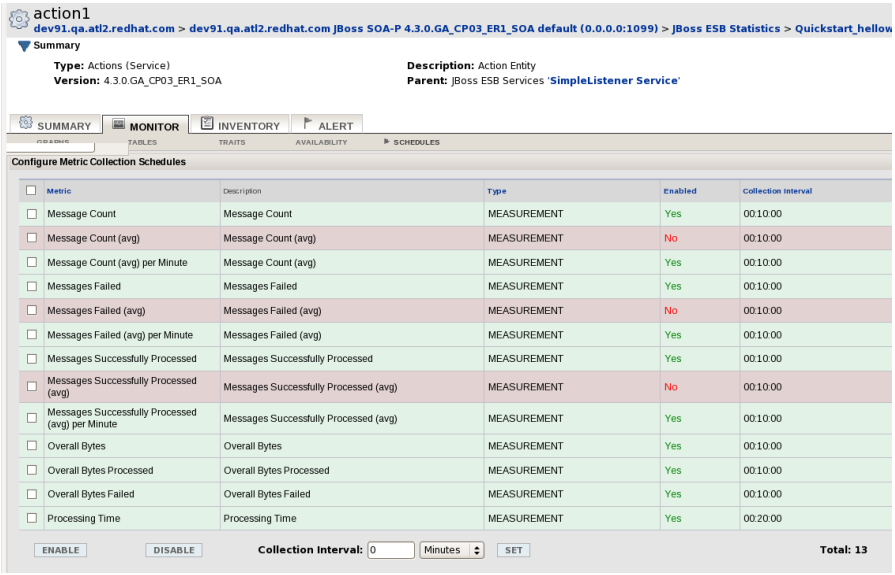



图 **6.7. 行为度量缺省是禁用的**

**重要**

红色显示的任何值都是禁用的。

### 6.4.3. 管理部署的 ESB 归档文件

**JON for SOA** 软件也允许你部署或删除 ESB 归档。要访问其功能，进入 **JBoss ESB Statistics** 屏幕并点击 **INVENTORY** 标签页。然后它会在 **Child Resources** 标签页里出现。

新的 ESB 归档可以通过选择 **Create New** 菜单里的 **JBoss ESB Deployment** 来进行部署。在 **Create New Resource** 页面里，你可以指定要部署的归档、部署的位置（通常是 **deploy** 目录）。只有压缩文件才可以上传，**Deploy Zipped** 选项指明是以压缩文件还是展开目录的形式部署。

现有的归档可以在 **Child Resources** 列表里点击 **DELETE** 来删除。

**注意**

这个部分也显示 ESB 部署和删除请求的历史记录。

### 6.4.4. 自动服务发现（Automatic Service Discovery）

*JBoss Operations Network Agent* 代理将自动检测到已经部署或删除的 ESB 归档。新部署的 ESB 归档将自动被添加至服务器库存，但被删除的归档不会从库存里删除。

缺省的代理配置每隔 24 小时执行一次”服务发现“。你有两个方法可以修改这个时间间隔。

1. 这个时间间隔也可以在代理配置文件 **conf/agent-configuration.xml** 里修改。要使修改生效，你需要重启代理。

```
<entry key="rhq.agent.plugins.service-discovery.period-secs" value="86400"/>
```

图 **6.8. conf/agent-configuration.xml 里的服务发现周期的设置：**

2. 通过 **JON 控制台**来编辑配置。

   JON 代理可以被添加至服务器资源的库存。之后，它们的配置可以象其他库存资源一样进行编辑。你可以编辑 **`CONFIGURE`** 标签页里的 **`Service Discovery Period`** 的值。这并不需要重启代理来使之生效。

和内嵌的 SOA 平台不同，你没有办法可以强制 **JBoss Operations Network Console** 控制台立即执行新数据的采集。**`Metric Data`** 标签页里的 **`Get Current Values`** 按钮只能更新显示以反映最近采集的数据。要进行立即更新，你可以把采集周期设置得非常短如 30 秒（之后再马上设置回原值）。

**重要**

出于性能考虑，红帽不推荐把采集周期设置得非常短。

# 第 7 章 热部署

## 7.1. 服务器模式

**JBoss SOA** 平台支持*热部署*。服务器定期地检查 **deploy** 目录里的新文件以进行部署。然而，它也检查已经部署的文件是否发生了特定的修改。当这些修改被检测到时，服务器卸载相关的文件并部署新的文件。这被称为 *热部署*。

根据包类型的不同，被监控的特定修改也不同。

1. **SAR** 文件

   **jbossesb.sar**是可热部署的。在下列情况下它将被重新部署：

   - 归档的时间戳改变了（如果 **SAR** 是压缩文件）。
   - 如果这个 **SAR** 是以展开目录的形式出现，而 **META-INF/jboss-service.xml** 的时间戳发生了变化。

2. **ESB** 文件

   在下列情况下，任何 ***.esb** 归档文件都将被重新部署：

   - 如果这个 **ESB** 是一个压缩归档文件，而时间戳发生了变化。
   - 如果 **ESB** 是以展开目录的形式出现，而 **META-INF/jboss-esb.xml** 的时间戳发生了变化。

   所有的动作（Action）都具有*生命周期的支持*。在进行热部署时，它将完成活动的请求且不接受任何转入消息，直至重新启动。所有这些都是自动完成的。如果你想更新一个动作，你可以用 **Groovy** 脚本在运行时进行修改（请参考 http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossESBQuickStart 上的 **Groovy** 快速启动指南）。

3. 规则（Rule）文件

   刷新规则文件有两个方法（**DRL** 或**DSL** 文件）：

   - 重部署 **jbrules.esb** 归档文件
   - 打开 *Action 配置*（请参考 http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossESBContentBasedRouting 里的 *JBossESBContentBasedRouting*）里的 **ruleReload**。

   如果 rule 文件发生了变化，它将被重新加载。

4. transformation 文件

   刷新 *transformation 文件*有两种方式：

   - 在 transformation 文件存在的目录里重新部署 **ESB** 归档文件。
   - 用 **ESB 控制台**通过 JMS（**topic**）发送通知消息。**Smooks** 处理程序将接收这个事件并重新加载。

5. 商业过程定义

   **JBoss 商业过程管理（JBoss Business Process Management，jBPM）工具**的*商业过程定义（Business Process Definition）*的新版本可以用 jBPM **Eclipse** 插件部署到 jBPM 数据库里。新的过程实例将使用新版本。现有的过程实例会遵循以前的定义完成其生命周期。更多的细节请参考《jBPM 用户指南》。

## 7.2. 独立（引导）模式

当使用 *Boot-Strapper* 模式时，ESB 归档不会被部署。每个节点只能有一个 **jboss-esb.xml** 配置文件。节点监控这个文件的时间戳并在有改动时重读其内容。为了更新规则，请使用 **ruleReload** 功能。

最后，为了更新商业过程定义，请使用和上面总结的相同的过程。

# 第 8 章 发布合约信息（Contract Publishing）

和某些 JBoss SOA 端点集成需要关于该端点以及它支持的操作的信息。特别是对于通过 **SOAPProcessor** action 开放 Webservice 端点而言更是如此（*《程序员指南》*里有相关介绍）。

## 8.1. Contract 应用程序

因此我们提供了 **Contract 程序**。你可以通过 http://127.0.0.1:8080/contract/ 访问它。

**重要**

这个程序只是作为技术预览出现，在以后的版本里它将被替换。

**注意**

**Contract 程序**也捆绑在 JBoss SOA 控制台里。如果部署了控制台，请首先从 SOA 平台服务器的 **deploy** 目录里删除 **contract.war** 文件。

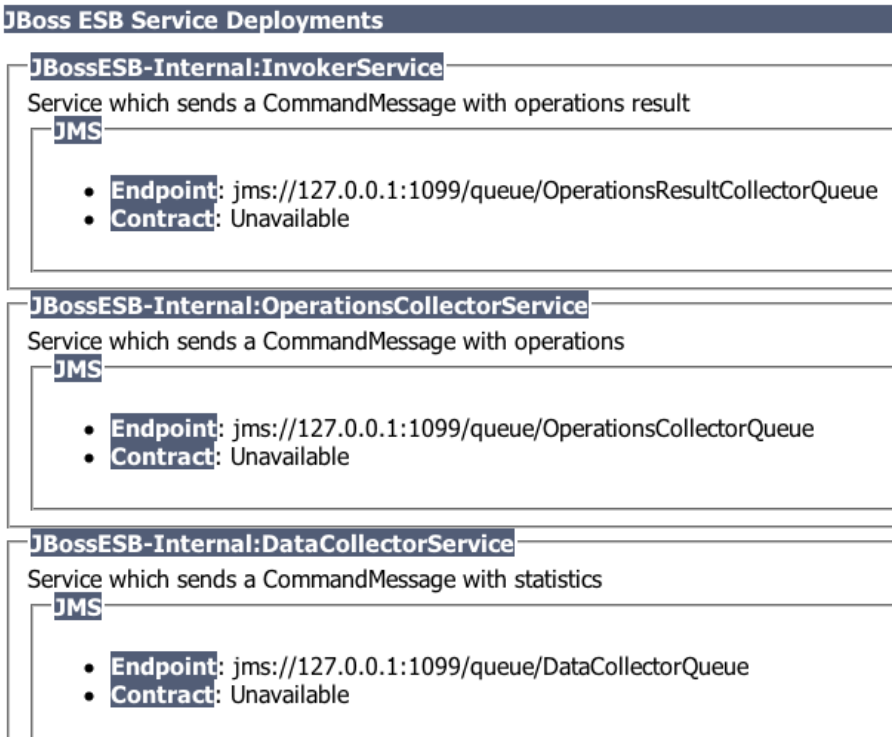



图 **8.1. JBoss ESB "Contract" 应用程序**

如你所看到的，它按照端点相关联的**服**务进行分组。这称为 *servicing*。

如你所看到的，它按照关联的服务将端点分组。你也将看到其中一些具有 "Contract" 超链接。它们是用于通过 **SOAPProcessor** 开放的 Webservice 端点的。这些超链接指向 WSDL。

## 8.2. 用 Action 发布合约

**JBoss 企业级 SOA 平台**根据服务所配置的 **action pipeline** 发现端点合约。它寻找管道里的发布合约信息的第一个 action。如果没有 action 发布合约信息，**Contract 程序**将对于该端点的合约显示

```
Unavailable on Contract
```

.

为了发布合约信息，Action 通过 **org.jboss.internal.soa.esb.publish.Publish** 注解来发布合约信息（以 **SOAPProcessor** 为例）：

```
@Publish(JBossWSWebserviceContractPublisher.class)
public class SOAPProcessor extends AbstractActionPipelineProcessor {
    //TODO: implement
}
```

**注意**

在这里你可找到一些 **SOAPProcessor** 示例：http://anonsvn.jboss.org/repos/labs/labs/jbossesb/trunk/product/services/soap/src/main/java/org/jboss/soa/esb/actions/soap/SOAPProcessor.java

然后你需要实现 **org.jboss.soa.esb.actions.soap.ContractPublisher**（它只要求实现一个方法）：

```
public ContractInfo getContractInfo(EPR epr);
```

**注意**

这里是一个描述 **JBossWSWebserviceContractPublisher** 的例子：http://anonsvn.jboss.org/repos/labs/labs/jbossesb/trunk/product/services/soap/src/main/java/org/jboss/soa/esb/actions/soap/JBossWSWebserviceContractPublisher.java。

# 第 9 章 JBoss 商业过程管理者

## 9.1. jBPM 控制台

*jBPM Web 控制台*缺省作为 **jbpm.esb** 的一部分进行部署，你可以通过 http://localhost:8080/jbpm-console/ 访问它。更多信息请参考*《jBPM 用户指南》*。

## 9.2. jBPM 的 Message 和 Scheduler 服务

商业过程管理者的缺省配置使用自己的 **JobExecutor** 和 **message** 和 **scheduler** 服务的数据库服务。

```
<service name="message"
    factory="org.jbpm.msg.db.DbMessageServiceFactory" />
<service name="scheduler"
    factory="org.jbpm.scheduler.db.DbSchedulerServiceFactory" />
<bean name="jbpm.job.executor" class="org.jbpm.job.executor.JobExecutor">
    ...
</bean>
```

JBoss 企业级 SOA 平台也允许你使用 ITS **message** 和 **scheduler** 服务来代替 jBPM 提供的原始服务。这些服务包含：

- 使用 JCA in-flow 的基于 JMS 的消息服务
- 基于 JBoss Messaging 的 Scheduling 服务

要使用基于 JMS 的 Message 服务和基于 JBoss Messaging 的 Scheduler 服务，只要简单地用 ***${SOA_ROOT}*/server/production/deploy/jbpm.esb/config/jmsscheduler/** 里的文件替换 ***${SOA_ROOT}*/server/production/deploy/jbpm.esb/** 里的缺省 jBPM 配置文件就可以了。

**例 9.1. 用 Linux 上的 BASH 终端替代 jBPM 配置。**

```
$ cd ${SOA_ROOT}/server/production/deploy/jbpm.esb/
$ cp -fb config/jmsscheduler/jbpm.cfg.xml.config jbpm.cfg.xml
$ cp -fb config/jmsscheduler/jbpm-service.xml.config jbpm-service.xml
$ cp -fb config/jmsscheduler/jbm-queue-service.xml.config jbm-queue-service.xml
```

**重要**

请注意，替代配置文件的名称以 **.config** 结尾，因此需要重新命名。

# 第 10 章 性能调优

## 10.1. 概述

请阅读本章来学习如何为特定环境优化 **JBoss 企业级服务总线（ESB）**的性能。在进行任何工作之前，请明白对于任何系统来说，性能和可靠性之间都得进行折中。（default 配置的目的是最大的可靠性和稳定性，而在某些环境里对性能则有负面影响。）

## 10.2. InVM 传输

*InVM 传输*（In Virtual Machine）通过相同虚拟机内部的 **`ServiceInvoker`** 调用服务。它对资源有着最小的影响，因为它没有导致任何网络或消息序列化方面的负荷。

**重要**

由于 InVM 队列的易变性，你可能无法达成所有的 ACID 模式，特别是当这个功能和其他事务性资源（如数据库）一起使用时。

关于这个主题的其他信息，请参考*《程序员指南》*的 "InVM Transport" 章节。

这段代码演示了如何用 InVM 传输功能配置服务：

```
<service category="HelloWorld" name="Service1"
                        description="Service 1" invmScope="GLOBAL">
    <listeners>
        <!-- So we just need to define a Gateway to the service... -->
        <jms-listener name="JMS-Gateway" busidref="quickstartGwChannel"
            is-gateway="true"/>
    </listeners>

    <actions>
        <action name="println"
            class="org.jboss.soa.esb.actions.SystemPrintln">
            <property name="message" value=" - > Service 1"/>
        </action>
        <!-- Route to the "Service 2" -->
        <action name="routeAction"
            class="org.jboss.soa.esb.actions.StaticRouter">
            <property name="destinations">
                <route-to service-category="HelloWorld"
                    service-name="Service2"/>
            </property>
        </action>
    </actions>

</service>
```

## 10.3. Transport 线程

几乎每个传输都能够配置处理请求的线程数目。增加线程的数目可以显著地提高性能。

下面是 InVM transport 的 maxThreads 属性的设置：

```
<service category="ServiceCategory" name="ServiceName"
description="..." invmScope="GLOBAL">
    <property name="maxThreads" value="100" />
    ...
</service>
```

下面是 JMS Listener 的 maxThreads 属性的设置：

```
<jms-listener name="GW" busidref="gwBus" maxThreads="100" is-gateway="true"/>
```

## 10.4. 消息过滤器（Message Filter）

*消息过滤器*用于动态地增加消息。例如，当消息流过 ESB 时它们可用于添加事务或安全信息。使用这些过滤器可能影响到系统性能。（这取决于 ESB 里配置的特定消息过滤器）关于这个主题的更多信息，请参考*《程序员指南》*的 "Meta-Data and Filters" 章节。

**注意**

关于整个主题的更多信息，请参考*《程序员指南》*的 "InVM Transport" 章节。

## 10.5. 引用传递（Passing By Reference）

在使用 InVM Transport 时，你可以通过值或*引用*传入消息。引用传递更快。然后，它并不适用于所有场合。在实施前请考虑所处的环境。

```
<service category="ServiceCategory" name="ServiceName"
description="..." invmScope="GLOBAL">
    <property name="inVMPassByValue" value="false" />
    ...
</service>
```

## 10.6. HTTP 路由器

HTTP 路由器可以保持多个连接。在 **`jboss-esb.xml`** 文件里设置下列配置选项（可在 HttpRouter 行为里找到）：

```
<http-client-property name="max-total-connections" value="100" />
<http-client-property name="max-connections-per-host" value="50" />
```

**注意**

类似的配置适用于其他在内部使用 HTTP 路由器的行为，如 **`SOAPProxy`**。

## 10.7. HTTP 连接器

*HTTP 服务器*基于 **Apache Tomcat** 且在 **`server/*/deploy/jbossweb.sar/server.xml`** 子目录里有一个类似的配置。红帽推荐对连接器配置更多的线程：

```
<Connector protocol="HTTP/1.1" port="8080" address="${jboss.bind.address}"
connectionTimeout="20000" redirectPort="8443" maxThreads="200" />
```

**注意**

关于配置连接器的更多信息，请参考 **Apache Tomcat** 文档。

## 10.8. 日志

### 10.8.1. HTTP 连接器

请注意 **log4j** 的 **INFO** 日志记录对于性能有影响。本节介绍了如何减少日志以避免降低性能的途径。

**org.milyn.util.ClassUtil** 的日志产生如下结果：

```
        ${date/time} INFO [ClassUtil] Loaded ${number} classes from ${number}
URLs through class list file
        ${file name}. Process took ${number}ms. Turn on debug logging for more
info.
```

在 **${jboss-as-home}/server/${configuration}/conf/jboss-log4j.xml** 文件里添加 *category limit* 以降低日志级别：

```
<category name="org.milyn.util.ClassUtil">
  <priority value="WARN" />
</category>
```

Logging from **org.milyn.delivery.ContentDeliveryConfigBuilder** produces the following:

```
        ${date/time} INFO  [ContentDeliveryConfigBuilder] All configured XML
        Element Content Handler resource configurations can be applied using the
        SAX or DOM Stream Filter. Defaulting to DOM Filter.
        Set 'global-parameters:stream.filter.type'.  Turn on debug logging for
more info.
```

在 **${jboss-as-home}/server/${configuration}/conf/jboss-log4j.xml** 文件里添加 *category limit* 以降低日志级别：

```
<category name="org.milyn.delivery.ContentDeliveryConfigBuilder">
  <priority value="WARN" />
</category>
```

Logging from **org.jboss.soa.esb.client.ServiceInvoker** produces the following:

```
        ${date/time} INFO  [ServiceInvoker] Badly formed EPR
        [EPR: PortReference wsa:Address ${address}] for Service
        [${service category}:${service name}] and Message [
           ${date/time} INFO  [ServiceInvoker] Invalid EPR for service
           (probably ESB-unaware): ignoring for message: header: [  ]
```

在 **${jboss-as-home}/server/${configuration}/conf/jboss-log4j.xml** 文件里添加

*category limit* 以降低日志级别：

```
<category name="org.jboss.soa.esb.client.ServiceInvoker">
  <priority value="WARN"/>
</category>
```

# 修订历史记录

| | | |
|---|---|---|
| **修订 4.3.5-3.400** | **2013-10-31** | **Rüdiger Landmann** |
| Rebuild with publican 4.0.0 | | |
| **修订 4.3.5-3** | **2012-07-18** | **Anthony Towns** |
| Rebuild for Publican 3.0 | | |
| **修订 15-0** | **Mon Mar 21 2011** | **David Le Sage** |
| 针对 4.3.CP05 进行更新 | | |
| **修订 1.4-0** | **Tue Apr 27 2010** | **David Le Sage** |
| 针对 SOA 4.3.CP04 进行更新<br>SOA-1943 - JON 消息计数器行为。6.4 章节 | | |
| **修订 1.3-0** | **Tue Apr 20 2010** | **David Le Sage** |
| 针对 SOA 4.3.CP03 进行更新 | | |
| **修订 1.2-0** | **Wed Sep 1 2009** | **Darrin Mison** |
| 针对 SOA 4.3.CP02 进行更新<br>SOA-1279 - 更新了数据库细节，请参考章节 1.4 的 Schema Tool<br>SOA-1310 - 添加了性能调优章节。第 10 章<br>SOA-1032 - 用其他的配置更新了 jBPM 章节。9.1 章节<br>SOA-1261 - 更新了 IBM WebSphere MQ JCA 适配器细节。章节 1.3.4.2 | | |
| **修订 1.1-0** | **Tue Feb 24 2009** | **Darrin Mison** |
| 针对 SOA 4.3.CP01 进行更新<br>添加了群集的 ESB 服务配置<br>更新了数据库模式工具说明<br>更新了 OpenSSO 安装说明<br>更新了 ESB 控制台安装说明 | | |
| **修订 1.0-0** | **Fri Sep 5 2008** | **Darrin Mison** |
| 创建 | | |