



Red Hat JBoss Enterprise Application Platform 7.0

配置指南

适用于 Red Hat JBoss Enterprise Application Platform 7.0

Red Hat JBoss Enterprise Application Platform 7.0 配置指南

适用于 Red Hat JBoss Enterprise Application Platform 7.0

法律通告

Copyright © 2018 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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, OpenShift, 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

本文档为管理员配置 Red Hat JBoss Enterprise Application Platform 7.0 提供了实用指南。

目录

第 1 章 概述	17
第 2 章 启动和停止 JBOSS EAP	18
2.1. 启动 JBOSS EAP	18
将 JBoss EAP 作为独立服务器启动	18
在受管域里启动 JBoss EAP	18
2.2. 停止 JBOSS EAP	18
停止交互式的 JBoss EAP 实例	18
停止后台运行的 JBoss EAP 实例	19
2.3. 以 ADMIN-ONLY 模式运行 JBOSS EAP	19
以 Admin-Only 模式启动 JBoss EAP	19
检查 JBoss EAP 是否以 Admin-Only 模式运行	19
从管理 CLI 里以其他模式重启	20
2.4. 优雅地挂起和关闭 JBOSS EAP	20
2.4.1. 挂起服务器	21
检查服务器的挂起状态	21
挂起	21
恢复	22
2.4.2. 优雅地关闭服务器	22
2.5. 启动和停止 JBOSS EAP (RPM 安装方式)	22
2.5.1. 启动 JBoss EAP (RPM 安装方式)	22
将 JBoss EAP 作为独立服务器启动 (RPM 安装方式)	22
在受管域里启动 JBoss EAP (RPM 安装方式)	23
配置 RPM 服务属性	23
2.5.2. 停止 JBoss EAP (RPM 安装方式)	24
停止作为独立服务器的 JBoss EAP (RPM 安装方式)	24
停止受管域里的 JBoss EAP (RPM 安装方式)	24
2.6. POWERSHELL 脚本	24
第 3 章 JBOSS EAP 的管理	26
3.1. 关于子系统、扩展和配置集	26
使用管理控制台或管理 CLI	26
3.2. 管理用户	26
3.2.1. 添加管理用户	26
3.2.2. 非交互式地运行 Add-User 工具	27
创建属于多个组的用户	27
指定替代的属性文件	27
3.2.3. 设置 Add-User 工具的密码限制	28
3.3. 管理接口	28
3.3.1. 管理 CLI	28
启动管理 CLI	28
连接到运行的服务器	29
显示帮助	29
退出管理 CLI	29
查看系统设置	29
更新系统设置	29
启动服务器	29
3.3.2. 管理控制台	29
3.3.2.1. 启用/禁用管理控制台	30
3.3.2.2. 修改管理控制台的语言设置	31
要修改管理控制台的语言设置	31
3.4. 配置数据	31

3.4.1. 独立服务器的配置文件	31
3.4.2. 受管域配置文件	32
3.4.3. 备份配置数据	32
3.4.4. 备份文件快照	32
生成快照	33
列出快照	33
删除快照	33
用快照启动服务器	33
3.4.5. 查看配置修改	34
3.4.6. 属性替换	35
嵌套的表达式	35
基于描述符的属性替换	36
3.5. 文件系统路径	37
3.5.1. 覆盖标准路径	38
覆盖受管域的标准路径	38
3.5.2. 添加自定义路径	39
3.5.3. 目录分组	39
按服务器进行目录分组	40
按类型进行目录分组	40
3.6. 系统属性	41
在启动脚本里传入系统属性	41
用管理 CLI 设置系统属性	41
用管理控制台设置系统属性	42
用 JAVA_OPTS 设置系统属性	42
3.7. 管理审计日志	42
独立服务器日志审计	43
管理域审计日志	43
3.7.1. 启用管理审计日志	44
启用独立服务器日志审计	44
启用受管域审计日志	44
3.7.2. 发送管理审计日志到 Syslog 服务器	44
3.7.3. 读取审计日志条目	45
第 4 章 网络和端口配置	47
4.1. 接口	47
4.1.1. 默认的接口配置	47
4.1.2. 配置接口	47
添加带有 NIC 值的接口	48
添加带有几个条件值的接口	48
更新接口属性	48
把接口添加到受管域里的一个服务器	48
4.2. 套接字绑定	49
4.2.1. 管理端口	49
4.2.2. 默认的套接字绑定	49
独立服务器	49
受管域	50
4.2.3. 配置套接字绑定	51
4.2.4. 端口偏移	51
4.3. IPV6 地址	52
根据 IPV6 地址配置 JVM 栈	52
根据 IPV6 地址更新接口声明	52
第 5 章 JBOSS EAP 的安全性	54

第 6 章 JBOSS EAP 的类加载	55
6.1. 模块	55
静态模块	55
动态模块	55
6.2. 模块依赖关系	56
可选的依赖关系	56
导出依赖关系	56
全局模块	56
6.3. 创建自定义模块	57
手工创建自定义模块	57
使用管理 CLI 创建自定义模块	57
添加依赖的模块	58
6.4. REMOVE A CUSTOM MODULE	58
Remove a Custom Module Manually	58
Remove a Custom Module Using the Management CLI	58
6.5. 定义全局模块	59
6.6. 配置子部署隔离	59
为所有部署启用子部署隔离	59
6.7. 定义外部的 JBOSS EAP 模块目录	60
6.8. 动态模块的命名规则	60
第 7 章 部署应用程序	62
7.1. 用管理 CLI 部署应用程序	62
7.1.1. 在一个独立服务器上部署应用程序	62
部署应用程序	62
卸载应用程序	62
列出部署	63
7.1.2. 在受管域里部署应用程序	63
部署应用程序	63
卸载应用程序	63
列出部署	64
7.2. 使用管理控制台部署应用程序	64
7.2.1. 在一个独立服务器上部署应用程序	64
部署应用程序	64
卸载应用程序	64
禁用应用程序	64
替换一个应用程序	65
7.2.2. 在受管域里部署应用程序	65
部署应用程序	65
为服务器组分配一个应用程序	65
从服务器组中取消分配的应用程序	65
卸载应用程序	65
禁用应用程序	65
替换一个应用程序	66
7.3. 用部署扫描器部署应用程序	66
7.3.1. 在一个独立服务器上部署应用程序	66
部署应用程序	66
卸载应用程序	67
重部署应用程序	67
7.3.2. 配置部署扫描器	67
禁用部署扫描器	67
修改扫描间隔	67
修改部署文件夹	67

启用展开内容的自动部署	67
禁用压缩内容的自动部署	67
禁用 XML 内容的自动部署	68
7.3.3. 定义自定义部署扫描器	68
7.4. 用 MAVEN 部署应用程序	68
7.4.1. 在一个独立服务器上部署应用程序	68
部署应用程序	68
卸载应用程序	69
7.4.2. 在受管域里部署应用程序	70
部署应用程序	70
卸载应用程序	70
7.5. 用 HTTP API 部署应用程序	71
部署应用程序	71
卸载应用程序	71
7.6. 自定义部署行为	71
7.6.1. 为部署内容定义自定义目录	71
为独立服务器定义自定义目录	72
为受管与定义自定义目录	72
7.6.2. 控制部署的顺序	72
7.6.3. 覆盖部署内容	72
7.6.4. 指定操作头	73
7.6.5. 使用 Rollout 计划	74
关于 Rollout 计划	74
Rollout 计划语法	75
用 Rollout 计划进行部署	75
用 Rollout 计划进行卸载	75
删除一个保存的 rollout 计划	76
默认的 Rollout 计划	76
第 8 章 域管理	77
8.1. 关于受管域	77
8.1.1. 关于域控制器	77
8.1.2. 关于主机控制器	78
8.1.3. 关于进程控制器	78
8.1.4. 关于服务器组	79
8.1.5. 关于服务器	79
8.2. 管理域配置	79
管理控制台	79
管理 CLI	80
8.3. 启动受管域	80
8.3.1. 启动受管域	80
启动域控制器	80
启动主机控制器	81
8.3.2. 配置域控制器	81
将主机配置为域控制器	81
8.3.3. 配置主机控制器	81
连接域控制器	81
8.3.3.1. 配置主机的名称	82
8.3.4. 域控制器的发现和故障转移	83
提升主机控制器为域控制器	84
8.4. 管理服务器	84
8.4.1. 配置服务器组	84
添加服务器组	84

更新服务器组	84
删除服务器组	85
服务器组属性	85
8.4.2. 配置服务器	85
添加服务器	86
更新服务器	86
删除服务器	86
服务器属性	86
8.4.3. 启动和停止服务器	86
启动服务器	86
停止服务器	87
重载服务器	87
8.5. 管理域设置	87
8.5.1. 在单台机器上设立受管域	87
8.5.2. 在两台机器上设立受管域	88
在两台机器上创建受管域	88
8.6. 管理 JBOSS EAP 配置集	89
8.6.1. 关于配置集	89
8.6.2. 克隆配置集	89
8.6.3. 创建分层的配置集	90
第 9 章 配置 JVM 设置	91
9.1. 配置独立服务器的 JVM 设置	91
9.2. 为受管域配置 JVM 设置	91
9.2.1. 在主机控制器上定义 JVM 设置	92
9.2.2. 应用 JVM 设置到服务器组	92
9.2.3. 应用 JVM 设置到单独的服务器	92
9.3. 显示 JVM 状态	93
9.4. 指定 32 位或 64 位 JVM 架构	93
为独立服务器指定 64 位架构	93
为受管域指定 64 位架构	93
第 10 章 MAIL 子系统	95
10.1. 配置 MAIL 子系统	95
配置应用程序里使用的 SMTP 服务器	95
10.2. 配置自定义传输	95
第 11 章 配置 WEB SERVICES	98
第 12 章 JBOSS EAP 的日志	99
12.1. 关于服务器日志	99
12.1.1. 服务器日志	99
12.1.2. 引导日志	99
12.1.2.1. 查看引导错误	99
查看服务器日志文件	100
通过管理 CLI 命令读取引导错误	100
12.1.3. 垃圾收集日志	101
12.1.4. 默认的日志文件位置	101
12.1.5. Set the Default Locale of the Server	102
Set the Language	102
Set the Language and Country	102
12.2. 查看日志文件	102
在管理控制台里查看日志	102
通过管理 CLI 查看日志	103

12.3. 关于日志子系统	104
12.3.1. Root Logger	104
12.3.2. 日志类别	104
12.3.3. 日志处理程序	104
日志处理程序类型	105
12.3.4. 日志级别	105
支持的日志级别	105
12.3.5. 日志格式器	106
日志格式器语法	107
12.3.6. 过滤器表达式	108
12.3.7. 隐性的日志依赖关系	109
12.4. 配置日志类别	110
添加日志类别	110
配置日志类别设置	110
分配处理程序	111
删除日志类别	111
12.5. 配置日志处理程序	111
12.5.1. 配置控制台日志处理程序	111
添加控制台日志处理程序	112
配置控制台日志处理程序设置	112
分配 Console 日志处理程序给 Logger	113
删除 Console 日志处理程序	113
12.5.2. 配置 File 日志处理程序	113
添加 File 日志处理程序	113
配置 File 日志处理程序设置	113
分配 File 日志处理程序给 Logger	114
删除 File 日志处理程序	115
12.5.3. 配置 Periodic Rotating 日志处理程序	115
添加 Periodic 日志处理程序	115
配置 Periodic 日志处理程序设置	115
分配 Periodic 日志处理程序给 Logger	116
删除 Periodic 日志处理程序	117
12.5.4. 配置 Size Rotating 日志处理程序	117
添加 Size 日志处理程序	117
配置 Size 日志处理程序设置	117
分配 Size 日志处理程序给 Logger	119
删除 Size 日志处理程序	119
12.5.5. 配置 Periodic Size Rotating 日志处理程序	119
添加 Periodic Size 日志处理程序	120
配置 Periodic Size 日志处理程序设置	120
分配 Periodic Size 日志处理程序给 Logger	121
删除 Periodic Size 日志处理程序	122
12.5.6. 配置 Syslog 处理程序	122
添加 Syslog 处理程序	122
配置 Syslog 处理程序设置	122
分配 Syslog 处理程序给 Logger	123
删除 Syslog 处理程序	123
12.5.7. 配置自定义日志处理程序	123
添加 Custom 日志处理程序	124
配置 Custom 日志处理程序设置	124
分配 Custom 日志处理程序给 Logger	125
删除 Custom 日志处理程序	125
12.5.8. 配置 Async 日志处理程序	125

添加 Async 日志处理程序	125
添加子处理程序	126
配置 Async 日志处理程序设置	126
分配 Async 日志处理程序给 Logger	126
删除 Async 日志处理程序	126
12.6. 配置 ROOT LOGGER	127
配置 Root Logger	127
12.7. 配置日志格式器	127
12.7.1. 配置命名模式格式器 (Named Pattern Formatter)	127
创建命名格式器	128
为日志处理程序分配命名格式器	128
12.7.2. 配置自定义日志格式器	128
配置自定义日志格式器	128
自定义 XML 格式器示例	129
12.8. 关于应用程序日志	129
12.8.1. Per-deployment 日志	130
12.8.1.1. 禁用 Per-deployment 日志	130
12.8.2. 日志配置集	130
12.8.2.1. 配置日志配置集	131
创建和配置日志配置集	131
12.8.2.2. 日志配置集示例	131
12.8.3. 查看部署日志配置	132
第 13 章 数据源管理	135
13.1. 关于 JBOSS EAP 数据源	135
关于 JDBC	135
支持的数据库	135
数据源类型	135
ExampleDS 数据源	135
13.2. JDBC 驱动	135
13.2.1. 将 JDBC 驱动安装为核心模块	135
13.2.2. 将 JDBC 驱动安装为 JAR 部署	137
将 JDBC 驱动 JAR 更新为兼容 JDBC 4	137
13.2.3. JDBC 驱动的下载位置	138
13.2.4. 访问厂商专有的类	138
使用 MANIFEST.MF 文件	139
使用 jboss-deployment-structure.xml 文件	139
13.3. 创建数据源	140
13.3.1. 创建非 XA 数据源	140
数据源参数	140
13.3.2. 创建 XA 数据源	141
数据源参数	141
13.4. 修改数据源	142
13.4.1. 修改非 XA 数据源	142
13.4.2. 修改 XA 数据源	142
13.5. 删除数据源	143
13.5.1. 删除非 XA 数据源	143
13.5.2. 删除 XA 数据源	143
13.6. 测试数据源连接	144
13.7. XA 数据源的恢复	144
13.7.1. 配置 XA 恢复	144
13.7.2. 厂商专有的 XA 恢复	145
厂商专有的配置	145

已知问题	146
13.8. 数据库连接检验	147
13.9. 数据源的安全性	149
用安全域保护数据源	149
用密码库保护数据源	150
13.10. 数据源统计	150
启用数据源统计	150
查看数据源统计	150
13.11. 容量策略	151
MaxPoolSize Incrementer 策略	152
Size Incrementer 策略	152
Watermark Incrementer 策略	152
MinPoolSize Decrementer 策略	152
Size Decrementer 策略	153
TimedOut Decrementer 策略	153
TimedOut/FIFO Decrementer 策略	153
Watermark Decrementer 策略	153
13.12. 登记跟踪	153
13.13. 数据源配置示例	154
13.13.1. MySQL 数据源示例	154
MySQL 数据源配置示例	154
MySQL JDBC Driver module.xml 文件示例	155
管理 CLI 命令示例	155
13.13.2. MySQL XA 数据源示例	155
MySQL XA 数据源配置示例	156
MySQL JDBC Driver module.xml 文件示例	156
管理 CLI 命令示例	156
13.13.3. PostgreSQL 数据源示例	157
PostgreSQL 数据源配置示例	157
PostgreSQL JDBC 驱动的 module.xml 文件示例	158
管理 CLI 命令示例	158
13.13.4. PostgreSQL XA 数据源示例	159
PostgreSQL XA 数据源配置示例	159
PostgreSQL JDBC 驱动的 module.xml 文件示例	159
管理 CLI 命令示例	159
13.13.5. Oracle 数据源示例	160
Oracle 数据源配置示例	160
Oracle JDBC 驱动的 module.xml 文件示例	161
管理 CLI 命令示例	161
13.13.6. Oracle XA 数据源示例	162
Oracle XA 数据源配置示例	162
Oracle JDBC 驱动的 module.xml 文件示例	163
管理 CLI 命令示例	163
13.13.7. Microsoft SQL Server 数据源示例	164
Microsoft SQL Server 数据源配置示例	164
Microsoft SQL Server JDBC 驱动的 module.xml 文件示例	164
管理 CLI 命令示例	164
13.13.8. Microsoft SQL Server XA 数据源示例	165
Microsoft SQL Server XA 数据源配置示例	165
Microsoft SQL Server JDBC 驱动的 module.xml 文件示例	166
管理 CLI 命令示例	166
13.13.9. IBM DB2 数据源示例	167
IBM DB2 数据源配置示例	167

IBM DB2 JDBC 驱动的 module.xml 文件示例	167
管理 CLI 命令示例	168
13.13.10. IBM DB2 XA 数据源示例	168
IBM DB2 XA 数据源配置示例	168
IBM DB2 JDBC 驱动的 module.xml 文件示例	169
管理 CLI 命令示例	170
13.13.11. Sybase 数据源示例	170
Sybase 数据源配置示例	171
Sybase JDBC 驱动的 module.xml 文件示例	171
管理 CLI 命令示例	171
13.13.12. Sybase XA 数据源示例	172
Sybase XA 数据源配置示例	172
Sybase JDBC 驱动的 module.xml 文件示例	173
管理 CLI 命令示例	173
13.13.13. MariaDB 数据源示例	174
MariaDB 数据源配置示例	174
MariaDB JDBC 驱动的 module.xml 文件示例	174
管理 CLI 命令示例	175
13.13.14. MariaDB XA 数据源示例	175
MariaDB XA 数据源配置示例	175
MariaDB JDBC 驱动的 module.xml 文件示例	176
管理 CLI 命令示例	176
第 14 章 配置事务	178
14.1. 事务子系统配置	178
14.1.1. 配置事务管理者	178
用管理控制台配置事务管理者	178
用管理 CLI 配置事务管理者	178
14.1.2. 配置数据源来使用 JTA	178
先决条件	178
配置数据源使用 JTA	178
14.1.3. 关于事务日志消息	179
14.1.4. 配置事务子系统的日志	179
用管理控制台配置事务 Logger	179
用管理 CLI 配置事务 Logger	180
14.2. 事务管理	180
14.2.1. 浏览和管理事务	180
刷新日志库	180
查看所有预备的事务	180
14.2.1.1. 管理事务	181
查看事务的属性	181
查看事务的参与者	181
删除事务	181
恢复事务	182
刷新需要恢复事务的状态	182
14.2.2. 查看事务统计信息	182
14.2.3. 事务对象库	183
将 JDBC 数据源用作事务对象库	183
第 15 章 ORB 配置	185
15.1. 关于公共对象请求代理架构 (CORBA)	185
15.2. 配置 JTS 事务的 ORB	185
用管理 CLI 配置 ORB	185

启用安全拦截器	185
启用 IIOP 子系统里的事务	185
在 Transactions 子系统中启用 JTS	185
用管理控制台来配置 ORB。	185
第 16 章 JAVA 连接器架构 (JCA) 管理	187
16.1. 关于 JAVA 连接器架构 (JCA)	187
16.2. 关于资源适配器	187
16.3. 配置 JCA 子系统	187
归档校验	188
Bean 校验	188
工作管理者	189
引导上下文	190
缓存的连接管理者	190
16.4. 配置资源适配器	190
16.4.1. 部署资源适配器	190
用管理 CLI 部署资源适配器	190
用管理控制台部署资源适配器	190
用部署扫描器部署资源适配器。	191
16.4.2. 配置资源适配器	191
添加资源适配器配置	191
配置资源适配器设置	191
激活资源适配器	192
16.4.3. 部署和配置 Websphere MQ 资源适配器	192
16.4.4. 部署和配置通用的 JMS 资源适配器	192
16.5. 配置受管连接池	192
16.6. 查看连接统计信息	193
第 17 章 配置 WEB 服务器 (UNDERTOW)	194
17.1. UNDERTOW 子系统概述	194
默认的 Undertow 子系统配置	194
17.2. 配置缓冲缓存	195
默认的 Undertow 子系统配置	195
更新现有的缓冲缓存	195
创建新的缓冲缓存	195
删除缓冲缓存	195
17.3. 配置服务器	196
默认的 Undertow 子系统配置	196
更新现有的服务器	196
创建新的服务器	196
删除服务器	196
17.4. 配置 SERVLET 容器	197
默认的 Undertow 子系统配置	197
更新现有的 Servlet 容器	197
创建新的 Servlet 容器	197
删除 Servlet 容器	197
17.5. 配置处理程序	198
默认的 Undertow 子系统配置	198
对静态资源使用 WebDAV	198
更新现有的文件处理程序	198
创建新的文件处理程序	198
删除文件处理程序	199
17.6. 配置过滤器	199

默认的 Undertow 子系统配置	199
更新现有的过滤器	200
创建新的过滤器	200
删除过滤器	200
17.7. 配置默认的 WELCOME WEB 应用程序	200
默认的 Undertow 子系统配置	200
修改 welcome-content 文件处理程序	201
修改 default-web-module	201
禁用默认的 Welcome Web 应用程序	201
17.8. 配置 HTTPS	202
17.9. 配置 HTTP 会话超时	202
配置默认的会话超时	202
17.10. 配置 HTTP-ONLY 会话管理 COOKIE	202
为 Servlet 容器会话 Cookie 配置 host-only 属性	203
为主机单点登录配置 host-only	203
17.11. 配置 HTTP/2	203
17.11.1. 配置 Undertow 来使用 HTTP/2	203
配置 Undertow 使用 HTTPS	203
下载 ALPN JAR	204
把 ALPN JAR 添加到 boot Classpath	204
在 HTTPS Listener 启用 HTTP/2	204
检验 HTTP/2 是否被使用	204
17.12. 配置 REQUESTDUMPING 处理程序	204
17.12.1. 配置服务器上的 RequestDumping 处理程序	205
用 RequestDumping 处理程序创建一个新的表达式过滤器	205
在 Undertow Web Server 里启用表达式过滤器	205
为特定的 URL 配置 RequestDumping 处理程序	205
17.12.2. 在应用程序里配置 RequestDumping 处理程序	205
第 18 章 配置 REMOTING	207
18.1. 关于 REMOTING 子系统	207
默认的 Remoting 子系统配置	207
Remoting 端点	207
连接器	207
出站连接	207
额外配置	207
18.2. 配置端点	207
更新现有的端点配置	208
创建新的端点配置	208
删除端点配置	208
18.3. 配置连接器	208
更新现有的连接器配置	208
创建新的连接器	208
删除连接器	208
18.4. 配置 HTTP 连接器	209
更新现有的 HTTP 连接器配置	209
创建新的 HTTP 连接器	209
删除 HTTP 连接器	209
18.5. 配置出站连接	209
更新现有的出站连接	209
创建新的出站连接	209
删除出站连接	209
18.6. 配置远程出站连接	210

18.7. 配置本地出站连接	210
更新现有的本地出站连接	210
创建新的本地出站连接	210
删除本地出站连接	210
18.8. 额外的远程配置	210
第 19 章 配置 IO 子系统	212
19.1. IO 子系统概述	212
默认的 IO 子系统配置	212
19.2. 配置工作节点	212
更新现有的工作节点	212
创建新的工作节点	212
删除工作节点	212
19.3. 配置缓冲池	212
更新现有的缓冲池	213
创建缓冲池	213
删除缓冲池	213
第 20 章 配置批应用程序	214
20.1. 配置批任务	214
20.1.1. 配置批任务仓库	214
添加 In-memory 任务仓库	214
添加 JDBC 任务仓库	214
设置默认的任务仓库	214
20.1.2. 配置批线程池	215
配置线程池	215
使用线程工厂	215
设置默认的线程池	216
查看线程池统计信息	216
20.2. 管理批任务	216
重启批任务	216
启动批任务	217
停止批任务	217
查看批任务的执行细节	217
第 21 章 配置 NAMING 子系统	218
21.1. 关于 NAMING 子系统	218
21.2. 配置全局绑定	218
配置简单绑定	218
绑定对象工厂	219
绑定外部上下文	220
绑定查找别名	221
21.3. 配置远程 JNDI 接口	221
第 22 章 配置高可用性	223
22.1. 高可用性介绍	223
22.2. 与 JGROUPS 的集群通讯	223
22.2.1. 关于 JGroups	223
22.2.2. 切换默认的 JGroups 频道以使用 TCP	224
22.2.3. 配置 TCPPING	224
22.2.4. 配置 TCPGOSSIP	226
22.2.5. 绑定 JGroups 到网络接口	227
22.2.6. 保护集群	227
配置验证	227

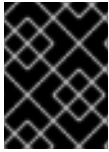
配置加密	228
使用对称加密	228
使用不对称加密	229
22.2.7. 配置 JGroups 线程池	230
22.2.8. 配置 JGroups Send 和 Receive 缓冲	230
解决缓冲区大小警告	230
配置 JGroups 缓冲的大小	231
22.2.9. JGroups 的故障排除	231
22.2.9.1. 节点没有组成一个集群	231
22.2.9.2. 在故障监测中导致丢失“心跳”的原因	232
22.3. INFINISPAN	232
22.3.1. 关于 Infinispan	232
22.3.2. 缓存容器	233
22.3.2.1. 配置缓存容器	234
用管理控制台配置缓存	234
用管理 CLI 配置缓存	234
22.3.3. 集群模式	235
缓存模式	235
同步和异步复制	235
22.3.3.1. 配置缓存模式	236
修改复制缓存模式	236
修改为分布式缓存模式	236
22.3.3.2. 缓存策略性能	237
22.3.4. 配置 Infinispan 线程池	237
22.3.5. Infinispan 统计	238
22.3.5.1. 启用 Infinispan 统计	238
22.3.6. Infinispan 分区处理	239
22.3.6.1. 裂脑	239
22.3.6.2. 配置分区处理	240
22.3.7. 外部化 HTTP 会话到 JBoss Data Grid	240
22.4. 配置 JBOSS EAP 作为一个前端负载均衡器	242
22.4.1. 配置 Undertow 作为一个使用 mod_cluster 的负载均衡器	242
配置 mod_cluster 前端负载平衡器	243
22.4.2. 配置 Undertow 作为一个静态负载均衡器	244
22.5. 使用一个外部 WEB 服务器作为一个代理服务器	245
22.5.1. HTTP 连接器概述	245
22.5.2. Apache HTTP Server	246
22.5.2.1. 安装 Apache HTTP Server	246
22.5.3. 接受外部 web 服务器的请求	246
更新 JBoss EAP 配置	247
22.6. MOD_CLUSTER HTTP 连接器	247
22.6.1. 在 Apache HTTP Server 中配置 mod_cluster	248
配置 mod_cluster	248
22.6.2. 为 mod_cluster 禁用广告	249
22.6.3. 配置一个 mod_cluster Worker 节点	250
配置一个 worker 节点	250
22.6.4. 配置 mod_cluster fail_on_status 参数	254
22.6.5. 迁移集群间的通讯	255
集群的升级过程 - 负载均衡组	255
默认的 Load-Balancing 组	256
使用管理 CLI	257
22.7. APACHE MOD_JK HTTP 连接器	257
22.7.1. 配置 Apache HTTP 服务器里的 mod_jk	257

22.7.2. 配置 JBoss EAP 与 mod_jk 通讯	260
22.8. APACHE MOD_PROXY HTTP 连接器	260
22.8.1. 配置 Apache HTTP 里的 mod_proxy	261
添加 Non-load-balancing 代理	261
添加 Load-balancing 代理	261
启用粘性会话	262
22.8.2. 配置 JBoss EAP 与 mod_proxy 通讯	262
22.9. MICROSOFT ISAPI CONNECTOR	262
22.9.1. 配置 Microsoft IIS 使用 ISAPI Connector	263
22.9.2. 配置 ISAPI Connector 发送客户请求至 JBoss EAP	264
创建属性文件并设立重定向	264
22.9.3. 配置 ISAPI Connector 在多个 JBoss EAP 服务器间平衡客户请求	266
平衡多个服务器间的客户请求	266
22.10. ORACLE NSAPI CONNECTOR	268
22.10.1. 配置 Oracle iPlanet Web Server 使用 NSAPI Connector	269
22.10.2. 配置 NSAPI Connector 发送客户请求至 JBoss EAP	270
设置基本的 HTTP Connector	270
22.10.3. 配置 NSAPI Connector 在多个 JBoss EAP 服务器间平衡客户请求	272
配置连接器的负载平衡	272
附录 A. 参考资料	274
A.1. 服务器运行时参数	274
A.2. RPM 服务配置文件	276
A.3. RPM 服务配置文件属性	276
A.4. JBOSS EAP 子系统概述	278
A.5. ADD-USER 工具参数	280
A.6. 管理审计日志属性	281
A.7. 接口属性	283
A.8. 套接字绑定属性	284
A.9. 默认的套接字绑定	285
A.10. 部署扫描器的 MARKER 文件	286
A.11. 部署扫描器属性	287
A.12. ROOT LOGGER 属性	288
A.13. 日志类别属性	288
A.14. 日志处理程序属性	289
A.15. 数据源连接 URL	295
A.16. 数据源参数	295
A.17. 数据源统计	301
A.18. 事务管理者的配置选项	304
A.19. 资源适配器属性	306
A.20. 资源适配器统计	310
A.21. UNDERTOW 子系统属性	310
缓冲缓存属性	311
Servlet 容器属性	311
servlet-container 属性	311
mime-mapping 属性	312
welcome-file 属性	312
crawler-session-management 属性	312
jsp 属性	313
persistent-sessions 属性	314
session-cookie Attributes	315
websockets 属性	315
过滤器属性	316

custom-filter 过滤器	316
error-page 过滤器	316
expression-filter 过滤器	316
gzip 过滤器	316
mod-cluster 过滤器	316
request-limit 过滤器	319
response-header 过滤器	320
rewrite 过滤器	320
Handler 属性	320
file 属性	320
对静态资源使用 WebDAV	320
reverse-proxy 属性	321
服务器属性	321
server 属性	322
http-listener 属性	322
https-listener 属性	325
ajp-listener 属性	328
host 属性	330
filter-ref 属性	330
access-log 属性	331
single-sign-on 属性	332
location 属性	332
A.22. HTTP 方法的默认行为	332
A.23. IO 子系统属性	333
A.24. REMOTING 子系统属性	333
连接器属性	335
HTTP 连接器属性	337
出站连接属性	338
远程出站连接	338
本地出站连接属性	339
A.25. APACHE HTTP SERVER 的 MOD_CLUSTER 指令	339
A.26. MODCLUSTER 子系统属性	342
A.27. MOD_JK 工作节点属性	346
A.28. 安全管理者子系统属性	348

第 1 章 概述

本指南涵盖了设置和维护 JBoss EAP，以及运行应用程序和其他服务所需的配置任务。在使用这个指南配置 JBoss EAP 之前，我们假定您已下载并安装了 JBoss EAP 的最新版本。关于安装说明，请参考 [《安装指南》](#)。



重要

不同主机上 JBoss EAP 的安装位置会有不同，本指南将安装位置统称为 **EAP_HOME**。在执行管理任务时，您应该使用 JBoss EAP 的实际安装位置而不是 **EAP_HOME**。

第 2 章 启动和停止 JBOSS EAP

2.1. 启动 JBOSS EAP

JBoss EAP 有两种操作模式：以一个独立服务器运行或位于一个受管域中。它被以下平台支持：Red Hat Enterprise Linux、Windows Server、Oracle Solaris 和 Hewlett-Packard HP-UX。

启动 JBoss EAP 的命令取决于底层的平台和期望的操作模式。

将 JBoss EAP 作为独立服务器启动

```
$ EAP_HOME/bin/standalone.sh
```



注意

对于 Windows 服务器，请使用 **EAP_HOME\bin\standalone.bat** 脚本。

这个启动脚本使用 **EAP_HOME/bin/standalone.conf** 文件（对于 Windows 服务器，使用 **standalone.conf.bat**）来设置某些默认首选项，如 JVM 选项。您可以在这个文件里自定义设置。

JBoss EAP 默认使用 **standalone.xml** 配置文件，但也可以用其他配置启动。关于可用的独立服务器配置文件及如何使用它们的细节，请参考[独立服务器的配置文件](#)一节。

如需获得启动脚本参数的完整列表，请使用 **--help** 参数或参阅[服务器运行时参数](#)一节。

在受管域里启动 JBoss EAP

域控制器必须在域中所有服务器组中的服务器之前启动。使用这个脚本首先启动域控制器，然后启动每个关联的主机控制器。

```
$ EAP_HOME/bin/domain.sh
```



注意

对于 Windows 服务器，请使用 **EAP_HOME\bin\domain.bat** 脚本。

这个启动脚本使用 **EAP_HOME/bin/domain.conf** 文件（对于 Windows 服务器，使用 **domain.conf.bat**）来设置某些默认首选项，如 JVM 选项。您可以在这个文件里自定义设置。

JBoss EAP 默认使用 **host.xml** 主机配置文件，但也可以用其他配置启动。关于可用的受管域配置文件及如何使用它们的细节，请参考[受管域配置文件](#)一节。

当设立受管域时，也需要将其他参数传入启动脚本。关于可用启动脚本参数和用途的完整列表，请使用 **-help** 参数或参阅[服务器运行时参数](#)一节。

2.2. 停止 JBOSS EAP

停止 JBoss EAP 的方式取决于您启动它的方式。

停止交互式的 JBoss EAP 实例

在启动 JBoss EAP 的终端窗口中按 **Ctrl+C**。

停止后台运行的 JBoss EAP 实例

使用管理 CLI 连接至运行实例并关闭服务器。

1. 启动管理 CLI。

```
$ EAP_HOME/bin/jboss-cli.sh --connect
```

2. 执行 **shutdown** 命令。

```
shutdown
```



注意

当运行在受管域时，您必须用 **shutdown** 命令和 **--host** 参数指定要关闭的主机名。

2.3. 以 ADMIN-ONLY 模式运行 JBOSS EAP

JBoss EAP 可以用 *admin-only* 模式启动。这可以让 JBoss EAP 运行并接受管理请求，而无需启动其他运行时服务或接受最终用户请求。Admin-only 模式对于独立服务器或受管域都可用。在受管域里，如果域控制器以 Admin-only 模式启动，它不会从主机控制器接受转入连接。

要以 Admin-only 模式启动 JBoss EAP 实例，请使用 **--admin-only** 运行时参数。



注意

显示的管理 CLI 命令假定您运行的是 JBoss EAP 独立服务器。关于在 JBoss EAP 受管域使用管理 CLI 的更多细节，请参考 [JBoss EAP Management CLI Guide](#)。

以 Admin-Only 模式启动 JBoss EAP

```
$ EAP_HOME/bin/standalone.sh --admin-only
```

检查 JBoss EAP 是否以 Admin-Only 模式运行

检查 JBoss EAP 是否以 Admin-Only 模式运行：



注意

显示的管理 CLI 命令假定您运行的是 JBoss EAP 独立服务器。关于在 JBoss EAP 受管域使用管理 CLI 的更多细节，请参考 [JBoss EAP Management CLI Guide](#)。

```
:read-attribute(name=running-mode)
```

如果 JBoss EAP 是以 Admin-Only 模式运行，结果将是：

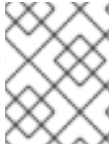
```
{
  "outcome" => "success",
  "result"  => "ADMIN_ONLY"
}
```

否则，结果将是：

```
{
  "outcome" => "success",
  "result" => "NORMAL"
}
```

从管理 CLI 里以其他模式重启

除了用不同的运行时参数停止和启动 JBoss EAP 实例之外，管理 CLI 也可以用于重载服务器并以不同的模式启动。要以 admin-only 模式重载 JBoss EAP 实例：



注意

显示的管理 CLI 命令假定您运行的是 JBoss EAP 独立服务器。关于在 JBoss EAP 受管域使用管理 CLI 的更多细节，请参考 [JBoss EAP Management CLI Guide](#)。

```
reload --admin-only=true
```

要重载 JBoss EAP 以普通模式启动：

```
reload --admin-only=false
```



注意

除了当前的运行模式，您也可以用下列命令检查初始的运行模式：**/core-service=server-environment:read-attribute(name=initial-running-mode)**。这个命令和 **:read-attribute(name=running-mode)** 不同，它显示了 JBoss EAP 启动时的运行模式而不是当前的运行模式。

2.4. 优雅地挂起和关闭 JBOSS EAP

JBoss EAP 可以优雅地挂起或关闭。这允许活动的请求正常完成而不会接受任何新的请求。超时时间指定了挂起或关闭操作等待活动请求完成的时间。当服务器被挂起时，管理请求仍在被处理。

优雅关闭是在服务器范围里进行协调的，多数集中在请求到达服务器的入口点。下面的子系统支持优雅关闭：

Undertow

undertow 子系统将等待所有请求完成。

mod_cluster

modcluster 子系统将通知负载均衡器服务器在 **PRE_SUSPEND** 阶段挂起。

EJB

ejb3 子系统将等待所有的远程 EJB 请求和 MDB 消息递送完成。MDB 递送停止在 **PRE_SUSPEND** 阶段。EJB 定时器被挂起，漏掉的定时器在服务器恢复时被激活。

EE Concurrency

服务器将等待所有活动任务完成。所有队列中的任务将被忽略。目前，因为 EE Concurrency 不具有持久性，所以被忽略的队列中的任务将会丢失。

当服务器处于挂起状态时，已调度的任务会在为它们调度的时间点上继续执行，但会抛出一个 **java.lang.IllegalStateException** 异常。一旦服务器被重新恢复运行，已调度的任务会继续正常运行，在多数情况下，不需要对它们重新进行调度。

Batch

服务器将在超时时间内停止所有运行的任务并推迟所有预定的任务。



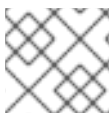
注意

优雅关闭目前不会拒绝转入的远程分布式事务或新转入的 JMS 消息。inflight 活动调度的 EE 批处理任务和 EE 并发任务当前允许继续进行。然而，如果当前提交的 EE 并发任务已超过超时窗口，则会在执行时报错。

request-controller 子系统被用来对请求进行跟踪。如果没有这个子系统，挂起和重新恢复的功能将会受到限制，服务器在进行挂起或关机前将不会等待请求完成。但是，如果您不需要这些功能，则可以不使用 **request-controller** 子系统，这可以对性能有一点提高。

2.4.1. 挂起服务器

JBoss EAP 7 引入了 *suspend* 模式，它优雅地挂起服务器操作。这允许所有活动请求正常地完成，但不会接受新的请求。一旦服务器被挂起，它可以关闭、返回运行状态或保持在挂起状态以进行维护。



注意

管理接口不受挂起的服务器影响。

服务器可以用管理控制台或 CLI 挂起或恢复运行。

检查服务器的挂起状态

服务器的挂起状态可以用下列管理 CLI 命令查看。结果将是 **RUNNING**、**PRE_SUSPEND**、**SUSPENDING** 或 **SUSPENDED** 中的一个。

- 检查独立服务器的挂起状态

```
:read-attribute(name=suspend-state)
```

- 查看受管域里服务器的挂起状态。

```
/host=master/server=server-one:read-attribute(name=suspend-state)
```

挂起

使用下列管理 CLI 命令来挂起服务器，指定服务器等待活动请求完成的超时时间（秒）。其默认值是 **0**，表示将立即挂起。**-1** 将让服务器无限期地等待直至所有活动请求完成为止。

以下示例中，在挂起前最多等待 60 秒让请求完成。

- 挂起独立服务器。

```
:suspend(timeout=60)
```

- 挂起受管域里的所有服务器。

```
:suspend-servers(timeout=60)
```

- 挂起受管域里的单个服务器。

```
/host=master/server-config=server-one:suspend(timeout=60)
```

- 挂起服务器组里的所有服务器。

```
/server-group=main-server-group:suspend-servers(timeout=60)
```

恢复

在相应级别（服务器、服务器组、整个域）中执行 **resume** 命令，服务器可以返回正常的运行状态来接受新的请求。例如：

```
:resume
```

2.4.2. 优雅地关闭服务器

在停止服务器时如果指定了合适的超时时间，服务器将被优雅地关闭。一旦执行了这个命令，服务器将被挂起并等待指定的一段时间让所有请求在服务器关闭前完成。

使用下列管理 CLI 命令来关闭服务器，指定服务器等待活动请求完成的超时时间（秒）。其默认值是 **0**，表示将立即关闭。**-1** 将让服务器在关闭前将无限期地等待直至所有活动请求完成为止。

以下示例中，服务器在关闭之前最多等待 60 秒让请求完成。

- 优雅地关闭独立服务器。

```
:shutdown(timeout=60)
```

- 优雅地停止受管域里的所有服务器。

```
:stop-servers(timeout=60)
```

- 优雅地停止受管域里的单个服务器。

```
/host=master/server-config=server-one:stop(timeout=60)
```

- 优雅地停止服务器组里的所有服务器。

```
/server-group=main-server-group:stop-servers(timeout=60)
```

2.5. 启动和停止 JBOSS EAP（RPM 安装方式）

对于 RPM 安装和 ZIP 安装，启动和停止 JBoss EAP 是不一样的。

2.5.1. 启动 JBoss EAP（RPM 安装方式）

启动 JBoss EAP 的 RPM 安装的命令取决于您要启动的模式（独立服务器或受管域），以及您运行的 Red Hat Enterprise Linux 的版本。

将 **JBoss EAP** 作为独立服务器启动（RPM 安装方式）

- 对于 Red Hat Enterprise Linux 6：

```
$ service eap7-standalone start
```

- 对于 Red Hat Enterprise Linux 7：

```
$ systemctl start eap7-standalone.service
```

在默认情况下，这将用 **standalone.xml** 配置文件启动 JBoss EAP。您可以通过[RPM 服务配置文件](#)里的属性用不同的[独立服务器配置文件](#)启动 JBoss EAP。详情请参考下面的[配置 RPM 服务属性](#)。

在受管域里启动 **JBoss EAP**（RPM 安装方式）

- 对于 Red Hat Enterprise Linux 6：

```
$ service eap7-domain start
```

- 对于 Red Hat Enterprise Linux 7：

```
$ systemctl start eap7-domain.service
```

在默认情况下，这将用 **host.xml** 配置文件启动 JBoss EAP。您可以通过[RPM 服务配置文件](#)里的属性用不同的[受管域配置文件](#)启动 JBoss EAP。详情请参考下面的[配置 RPM 服务属性](#)。

配置 RPM 服务属性

本节向您展示如何配置 RPM 服务属性和其他的启动选项。请注意，在进行修改之前，我们建议您先备份配置文件。

关于 RPM 安装的所有可用的启动选项，请参考 [RPM 服务配置属性](#) 章节。

- 指定服务器配置文件。
在启动独立服务器时，默认是使用 **standalone.xml** 文件。当运行在受管域里时，默认使用 **host.xml** 文件。您可以通过指定合适的 [RPM 配置文件](#) 里的 **WILDFLY_SERVER_CONFIG** 属性用不同的配置文件启动 JBoss EAP，例如 **eap7-standalone.conf**。

```
WILDFLY_SERVER_CONFIG=standalone-full.xml
```

- 绑定到专门的 IP 地址。
在默认的情况下，JBoss EAP RPM 安装绑定 **0.0.0.0**。您可以通过合适的 [RPM 配置文件](#) 里的 **WILDFLY_BIND** 属性绑定 JBoss EAP 到专门的 IP 地址，例如 **eap7-standalone.conf**。

```
WILDFLY_BIND=192.168.0.1
```



注意

如果您想将管理接口绑定到专门的 IP 地址，可以在 JBoss EAP 的启动配置文件里进行配置，如下所示。

- 设置 JVM 选项或 Java 属性。
您可以编辑启动配置文件，指定 JVM 选项或 Java 属性来传入 JBoss EAP 启动脚本。这个文件是 **EAP_HOME/bin/standalone.conf**（独立服务器）或 **EAP_HOME/bin/domain.conf**（受管域）。下面的例子配置了堆（heap）大小，并将 JBoss EAP 管理接口绑定到专门的 IP 地址。

```
JAVA_OPTS="$JAVA_OPTS -Xms2048m -Xmx2048m"
JAVA_OPTS="$JAVA_OPTS -Djboss.bind.address.management=192.168.0.1"
```

**注意**

如有需要，您必须用 **WILDFLY_BIND** 属性而不是这里的 **jboss.bind.address** 标准属性来配置 JBoss EAP 的绑定地址。

**注意**

如果某个属性在 RPM 服务配置文件（如 `/etc/sysconfig/eap7-standalone`）和 JBoss EAP 启动配置文件（如 `EAP_HOME/bin/standalone.conf`）都有相同的名称，那么 JBoss EAP 启动配置文件里的值将优先使用。**JAVA_HOME** 就是这类属性中的一个。

2.5.2. 停止 JBoss EAP（RPM 安装方式）

停止 JBoss EAP 的 RPM 安装的命令取决于您启动的操作模式（独立服务器或受管域），以及您运行的 Red Hat Enterprise Linux 的版本。

停止作为独立服务器的 **JBoss EAP**（RPM 安装方式）

- 对于 Red Hat Enterprise Linux 6：

```
$ service eap7-standalone stop
```

- 对于 Red Hat Enterprise Linux 7：

```
$ systemctl stop eap7-standalone.service
```

停止受管域里的 **JBoss EAP**（RPM 安装方式）

- 对于 Red Hat Enterprise Linux 6：

```
$ service eap7-domain stop
```

- 对于 Red Hat Enterprise Linux 7：

```
$ systemctl stop eap7-domain.service
```

关于 RPM 安装的所有可用的启动选项，请参考 [RPM 服务配置文件](#) 章节。

2.6. POWERSHELL 脚本

PowerShell 目前只是技术预览且不受支持。PowerShell 脚本的目的是让版本 2 和更高版本能正确运行，因为 Windows Server 2008 R2 Enterprisex86_64 和 Windows Server 2012 R2 Standard x86_64 都支持 JBoss EAP。

- 在默认情况下，Windows Server 2008 R2 Enterprisex86_64 使用 PowerShell 2。更高的版本也可以被安装。
- 在默认情况下，Windows Server 2012 R2 Standard x86_64 使用 PowerShell 4。更高的版本也可以被安装。

PowerShell 脚本的路径是 'EAP_HOME/bin'，当您希望在 Windows 上使用现代的脚本语言时，您可以使用这些脚本。



注意

脚本的参数应该用引号括起来。例如：

```
PS C:\Users\Administrator\7.0.0.ER2\jboss-eap-7.0\bin>
.\standalone.ps1 "-c=standalone-full.xml"
```

第 3 章 JBOSS EAP 的管理

JBoss EAP 使用了简化的配置，每个独立服务器或受管域对应一个配置文件。独立服务器的默认配置保存在 `EAP_HOME/standalone/configuration/standalone.xml` 文件，而受管域的默认配置保存在 `EAP_HOME/domain/configuration/domain.xml` 文件里。此外，主机控制器的默认配置保存在 `EAP_HOME/domain/configuration/host.xml` 文件里。

JBoss EAP 可以用命令行管理 CLI 或基于 Web 的管理控制台来配置。用这些管理界面进行的修改将自动持久化，XML 配置文件将被管理 API 重写。我们不推荐手动地编辑 XML 配置文件。

3.1. 关于子系统、扩展和配置集

JBoss EAP 功能的不同方面会在不同的子系统中进行配置。例如，应用程序和服务器的日志功能（logging）在 **logging** 子系统中配置。

子系统 (subsystem) 为一个特定的扩展提供配置选项。一个 **扩展 (extension)** 就是一个模块，它被用来扩展服务器的核心功能。扩展会在需要时被加载，并在不需要时被卸载。

子系统配置的一组集合组成了一个 **配置集 (profile)**，它被配置来满足服务器的需要。一个独立的服务器有一个单独的、没有命名的配置集。在一个受管域中可以定义多个 **配置集** 以供域中的服务器组使用。

如需了解更多与子系统相关的信息，请参阅 [JBoss EAP 子系统概述](#)。

使用管理控制台或管理 CLI

管理控制台和管理 CLI 都是有效的、受支持的更新 JBoss EAP 实例配置的方式。使用哪一个取决于您的喜好。喜欢使用图形化的、基于 Web 的界面的用户应该使用管理控制台。而偏好使用命令行的用户可以选择使用管理 CLI。

3.2. 管理用户

默认的 JBoss EAP 配置提供了本地用户身份验证功能，因此，一个用户可以在不需要请求验证的情况下访问本地主机上的管理 CLI。

但是，如果您需要远程访问 CLI 或使用管理控制台（使用管理控制台被认为是一个远程访问，即使它的网络数据是由本地产生的），则需要添加一个管理用户。如果您在添加管理用户前试图访问管理控制台，则会收到一个错误信息。

如果使用图形化安装程序安装 JBoss EAP，则在安装过程中会创建一个管理用户。

本指南涵盖了使用 **add-user** 脚本进行简单的 JBoss EAP 用户管理，这个脚本添加可立即验证的新用户至属性文件里。关于高级验证和授权选项，如 LDAP 或基于角色的访问控制（Role-Based Access Control，RBAC），请参考 [《JBoss EAP 安全架构》](#) 里的 **核心管理验证** 章节。

3.2.1. 添加管理用户

1. 运行 **add-user** 工具脚本并按提示进行。

```
$ EAP_HOME/bin/add-user.sh
```



注意

对于 Windows 服务器，请使用 `EAP_HOME\bin\add-user.bat` 脚本。

2. 按 **ENTER** 选择默认的选项 **a** 来添加管理用户。
这个用户将会添加至 *ManagementRealm*，并被授权通过管理控制台或管理 CLI 执行管理操作。
另外一个选项 (**b**) 添加用户至 *ApplicationRealm*，它用于应用程序，并且没有特定的权限。
3. 输入想要的用户名和密码，您会被提示确认密码。
在默认情况下，JBoss EAP 允许弱密码，但会发出警告。关于修改这个默认行为的细节，请参考[设置 Add-User 工具的密码限制](#)。
4. 输入用户所属的用逗号隔开的组列表。如果不希望这个用户属于任何组，请按 **ENTER** 留空。
5. 复查信息并输入 **yes** 确认。
6. 确定这个用户是否代表远程的 JBoss EAP 服务器实例。关于基本的管理用户，请输入 **no**。
可能需要添加至 *ManagementRealm* 里的一种用户是代表其他 JBoss EAP 实例的用户，它能够
通过验证以作为集群成员加入。如果是这样，请输入 **yes**，您将得到一个代表用户密码的哈希密
码值，它将需要被添加到不同的配置文件里。

您也可以传入参数到 **add-user** 脚本非交互式地创建用户。我们不推荐在共享系统上使用这个方法，因为在日志和历史记录文件里可以看到密码。详情请参考[非交互式地运行 Add-User 工具](#)。

3.2.2. 非交互式地运行 Add-User 工具

您可以在命令行传入参数，非交互式地运行 **add-user** 脚本。但最少您得提供用户名和密码。



警告

我们不推荐在共享系统上使用这个方法，因为在日志和历史记录文件里可以看到密码。

创建属于多个组的用户

下列命令添加了一个管理用户 (**mgmtuser1**) 和 **guest**、**mgmtgroup** 组。

```
$ EAP_HOME/bin/add-user.sh -u 'mgmtuser1' -p 'password1!' -g
'guest,mgmtgroup'
```

指定替代的属性文件

在默认情况下，用 **add-user** 脚本创建的用户和组信息保存在位于服务器 **configuration** 目录下的属性文件里。

用户信息保存在下列属性文件里：

- **EAP_HOME/standalone/configuration/mgmt-users.properties**
- **EAP_HOME/domain/configuration/mgmt-users.properties**

组信息保存在下列属性文件里：

- **EAP_HOME/standalone/configuration/mgmt-groups.properties**
- **EAP_HOME/domain/configuration/mgmt-groups.properties**

这些默认的目录和属性文件名可以被覆盖。下列命令添加了一个新的用户，指定不同的名称和用户属性文件的位置。

```
$ EAP_HOME/bin/add-user.sh -u 'mgmtuser2' -p 'password1!' -sc
'/path/to/standaloneconfig/' -dc '/path/to/domainconfig/' -up
'newname.properties'
```

新用户现在被添加到 `/path/to/standaloneconfig/newname.properties` 和 `/path/to/domainconfig/newname.properties` 用户属性文件里。请注意，这些必须已经存在，否则您会看到错误信息。

如需获得 `add-user` 的完整参数列表，请使用 `--help` 参数，或参阅 [ADD-USER 工具参数](#) 一节。

3.2.3. 设置 Add-User 工具的密码限制

`add-user` 工具脚本的密码限制可以用 `EAP_HOME/bin/add-user.properties` 文件进行配置。

在默认情况下，JBoss EAP 允许弱密码，但会发出警告。要拒绝不符合指定的最低要求的密码，请将 `password.restriction` 属性设置为 `REJECT`。

在 `EAP_HOME/bin/add-user.properties` 文件里可以配置的其他密码要求：

- 最小长度
- 最少的字母字符
- 最小数字
- 最小符号
- 禁止的密码列表（如 `admin`）
- 是否允许密码和用户名相同

3.3. 管理接口

3.3.1. 管理 CLI

管理命令行接口（Management Command-line Interface，CLI）是一个 JBoss EAP 命令行管理工具。

使用管理 CLI 来启动和停止服务器、部署和卸载应用程序、配置系统设置和执行其他管理性任务。操作可以批模式进行，允许将多个任务以组运行。

有许多常用的命令，例如 `ls`、`cd` 和 `pwd`。管理 CLI 也支持 Tab 自动完成。

关于使用管理 CLI 的详情，包括命令和操作、语法和批模式运行，请参考 [《JBoss EAP 管理 CLI 指南》](#)。

启动管理 CLI

```
$ EAP_HOME/bin/jboss-cli.sh
```




注意

对于 Windows 服务器, 请使用 **EAP_HOME\bin\jboss-cli.bat** 脚本。

连接到运行的服务器

```
connect
```

或者您可以通过 **EAP_HOME/bin/jboss-cli.sh --connect** 命令启动管理 CLI 并进行连接。

显示帮助

使用下列命令获得帮助。

```
help
```

使用下列命令获得关于某个命令的帮助。

```
deploy --help
```

退出管理 CLI

```
quit
```

查看系统设置

下列命令使用 **read-attribute** 操作显示是否启用了示例数据源。

```
/subsystem=datasources/data-source=ExampleDS:read-attribute(name=enabled)
{
  "outcome" => "success",
  "result" => true
}
```

当运行在受管域时, 您必须在命令前面使用 **/profile=PROFILE_NAME** 指定要更新的配置集。

```
/profile=default/subsystem=datasources/data-source=ExampleDS:read-attribute(name=enabled)
```

更新系统设置

下列命令使用 **write-attribute** 操作来禁用示例数据源。

```
/subsystem=datasources/data-source=ExampleDS:write-attribute(name=enabled,value=false)
```

启动服务器

当运行在受管域里时, 管理 CLI 也可以用于启动和停止服务器。

```
/host=HOST_NAME/server-config=server-one:start
```

3.3.2. 管理控制台

管理控制台是一个基于 Web 的 JBoss EAP 管理工具。

使用管理控制台来启动和停止服务器、部署和卸载应用程序、调优系统设置、永久地修改服务器配置文件。管理控制台也可以执行管理性任务，在当前用户进行的修改需要服务器实例重启或重载时发出即时通知。

在受管域里，相同域里的服务器实例和服务器组都可以通过域控制器的管理控制台集中管理。

对于在使用默认管理端口的本地主机上运行的 JBoss EAP 实例，管理控制台可以通过浏览器来访问：<http://localhost:9990/console/App.html>。您将需要用具有访问管理控制台权限的用户来进行验证。

管理控制台提供下列标签页来导航和管理您的 JBoss EAP 独立服务器或受管域。

主页

学习如何完成几个常见的配置和管理任务。体验控制台以熟悉它的使用。

部署

添加、删除和启用部署。在受管域里，分配部署到服务器组。

配置

配置可用的子系统，它提供 Web Service、消息或高可用性功能。在受管域里，管理包含不同子系统配置的配置集。

运行时

查看运行时信息，如服务器状态、JVM 使用、服务器日志。在受管域里，管理您的主机、服务器组和服务器。

访问控制

当使用基于角色的访问控制时为用户和组分配角色。

补丁

应用补丁到您的 JBoss EAP 实例。



注意

要体验更新的管理控制台，请点击管理控制台主页上的 **Take a Tour**。

要查看表单字段的细节，请点击 **Need Help?** 链接。

要查看您已执行的配置操作的历史记录，请点击管理控制台右上角的 **Messages** 链接。

3.3.2.1. 启用/禁用管理控制台

您可以通过 `/core-service=management/management-interface=http-interface` 资源里的 `console-enabled` 布尔值启用或禁用管理控制台。对于域模式里的主主机，是 `/host=master/core-service=management/management-interface=http-interface`。

例如，要启用控制台：

```
/core-service=management/management-interface=http-interface:write-attribute(name=console-enabled,value=true)
```

例如，要禁用控制台：

```
/core-service=management/management-interface=http-interface:write-attribute(name=console-enabled,value=false)
```

3.3.2.2. 修改管理控制台的语言设置

管理控制台的默认语言是英语。但您也可以选择下列语言：

- 德语 (de)
- 简体中文 (zh-Hans)
- 巴西葡萄牙语 (pt-BR)
- 法语 (fr)
- 西班牙语 (es)
- 日语 (ja)

要修改管理控制台的语言设置

1. 登录管理控制台。
2. 点击管理控制台右下角的 **Settings** 链接。
3. 从 **Locale** 选择框里选择想要的语言。
4. 选择 **Save**。确认框会通知您需要重新加载应用程序。
5. 点击 **Confirm**。系统会自动刷新您的浏览器以使用所选的语言。

3.4. 配置数据

3.4.1. 独立服务器的配置文件

The standalone configuration files are located in the **EAP_HOME/standalone/configuration/** directory. A separate file exists for each of the four predefined profiles (*default*, *ha*, *full*, *full-ha*).

表 3.1. 独立配置文件

配置文件	目的
standalone.xml	这个独立配置文件是您启动独立服务器时使用的默认配置。它包含所有关于服务器的信息，如子系统、网络、部署、套接字绑定和其他可配置细节。但它没有提供消息或高可用性所必需的子系统。
standalone-ha.xml	这个独立配置文件包括所有的默认子系统并添加了用于高可用性的 modcluster 和 jgroups 子系统。它没有提供消息所必需的子系统。
standalone-full.xml	这个独立配置文件包括所有的默认子系统并添加了用于高可用性的 messaging-activemq 和 iiop-openjdk 子系统。它没有提供高可用性所必需的子系统。
standalone-full-ha.xml	这个独立配置文件包括对任何子系统的支持，且也包括消息和高可用性所必需的子系统。

在默认情况下，将 JBoss EAP 作为独立服务器启动会使用 **standalone.xml** 文件。要用不同的配置启动 JBoss EAP，请使用 **--server-config** 参数。例如：

```
$ EAP_HOME/bin/standalone.sh --server-config=standalone-full.xml
```

3.4.2. 受管域配置文件

受管域配置文件位于 **EAP_HOME/domain/configuration/** 目录。

表 3.2. 受管域配置文件

配置文件	目的
domain.xml	这是用于受管域的主要配置文件。只有域主控制器会读取这个文件。它包含所有配置集的配置 (<i>default</i> 、 <i>ha</i> 、 <i>full</i> 、 <i>full-ha</i>) 。
host.xml	这个文件包括受管域里的物理主机所专有的配置细节，如网络接口、套接字绑定、主机名称和其他主机专有的细节。如下所示， host.xml 文件包括 host-master.xml 和 host-slave.xml 的所有功能。
host-master.xml	这个文件只包括了一个服务器作为主域控制器运行所需的配置详情。
host-slave.xml	这个文件只包含将服务器作为受管域主机控制器运行所必需的配置细节。

在默认情况下，在受管域里启动 JBoss EAP 会使用 **host.xml** 文件。要用不同的配置启动 JBoss EAP，请使用 **--host-config** 参数。例如：

```
$ EAP_HOME/bin/domain.sh --host-config=host-master.xml
```

3.4.3. 备份配置数据

为了在以后恢复 JBoss EAP 服务器配置，您应该备份下列位置里的内容：

- **EAP_HOME/standalone/configuration/**
 - 备份整个目录来保存独立服务器的用户数据、服务器配置和日志设置。
- **EAP_HOME/domain/configuration/**
 - 备份整个目录来保存受管域的用户和配置集数据、域和主机配置及日志设置。
- **EAP_HOME/modules/**
 - 备份任何自定义模块。
- **EAP_HOME/welcome-content/**
 - 备份任何自定义欢迎内容。
- **EAP_HOME/bin/**
 - 备份任何自定义脚本或启动配置文件。

3.4.4. 备份文件快照

为了协助服务器的维护和管理，JBoss EAP 在启动时创建了一个原始配置文件的带时间戳的版本。管理操作进行的任何额外的配置修改都将导致原始文件被自动备份，且保留实例的一个工作备份以供引用和回滚。此外，您也可以生成当前服务器配置的时间点拷贝，也就是配置快照。管理员可以保存和加载这些快照。

下面的例子使用了 **standalone.xml** 文件，但相同的过程也适用于 **domain.xml** 和 **host.xml** 文件。

生成快照

使用管理 CLI 来生成当前配置的快照。

```
:take-snapshot
{
  "outcome" => "success",
  "result" =>
    "EAP_HOME/standalone/configuration/standalone_xml_history/snapshot/20151022-133109702standalone.xml"
}
```

列出快照

使用管理 CLI 列出所有已生成的快照。

```
:list-snapshots
{
  "outcome" => "success",
  "result" => {
    "directory" =>
      "EAP_HOME/standalone/configuration/standalone_xml_history/snapshot",
    "names" => [
      "20151022-133109702standalone.xml",
      "20151022-132715958standalone.xml"
    ]
  }
}
```

删除快照

使用管理 CLI 删除快照。

```
:delete-snapshot(name=20151022-133109702standalone.xml)
```

用快照启动服务器

服务器可以用快照或自动保存的配置启动。

1. 进入 **EAP_HOME/standalone/configuration/standalone_xml_history** 目录并确定要加载的快照或配置文件。
2. 启动服务器并指向所选的配置文件。传入相对于配置目录 (**EAP_HOME/standalone/configuration/**) 的相对路径。

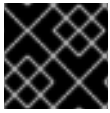
```
$ EAP_HOME/bin/standalone.sh --server-
config=standalone_xml_history/snapshot/20151022-
133109702standalone.xml
```

**注意**

当运行在受管域时，请使用 **--host-config** 参数来指定配置文件。

3.4.5. 查看配置修改

JBoss EAP 7 提供了对运行系统的配置修改进行跟踪的功能。这允许管理员查看其他授权用户进行的配置修改的历史记录。

**重要**

修改保存在内存里且在服务器重启时会丢失。这个功能不是对[管理审计日志](#)的替代。

要启用配置修改的跟踪，请使用下列管理 CLI 命令。您可以用 **max-history** 指定保存多少个条目。

```
/core-service=management/service=configuration-changes:add(max-history=10)
```

要查看最近的配置修改列表，请使用下列管理 CLI 命令。

```
/core-service=management/service=configuration-changes:list-changes
```

这将列出每个配置修改，包括日期、来源、结果和操作细节。例如，**list-changes** 命令的下列输出显示了配置修改，首先是最近的修改。

```
{
  "outcome" => "success",
  "result" => [
    {
      "operation-date" => "2016-02-12T18:37:00.354Z",
      "access-mechanism" => "NATIVE",
      "remote-address" => "127.0.0.1/127.0.0.1",
      "outcome" => "success",
      "operations" => [{
        "address" => [],
        "operation" => "reload",
        "operation-headers" => {
          "caller-type" => "user",
          "access-mechanism" => "NATIVE"
        }
      }]
    },
    {
      "operation-date" => "2016-02-12T18:34:16.859Z",
      "access-mechanism" => "NATIVE",
      "remote-address" => "127.0.0.1/127.0.0.1",
      "outcome" => "success",
      "operations" => [{
        "address" => [
          ("subsystem" => "datasources"),
          ("data-source" => "ExampleDS")
        ],
        "operation" => "write-attribute",
        "name" => "enabled",
        "value" => false,
```

```

        "operation-headers" => {
            "caller-type" => "user",
            "access-mechanism" => "NATIVE"
        }
    }
},
{
    "operation-date" => "2016-02-12T18:24:11.670Z",
    "access-mechanism" => "HTTP",
    "remote-address" => "127.0.0.1/127.0.0.1",
    "outcome" => "success",
    "operations" => [{
        "operation" => "remove",
        "address" => [
            ("subsystem" => "messaging-activemq"),
            ("server" => "default"),
            ("jms-queue" => "ExpiryQueue")
        ],
        "operation-headers" => {"access-mechanism" => "HTTP"}
    ]
}
]
}

```

这个例子列出影响配置的三个操作的细节：

- 从管理 CLI 重载服务器。
- 从管理 CLI 禁用 **ExampleDS** 数据源。
- 从管理控制台删除 **ExpiryQueue** 队列。

3.4.6. 属性替换

JBoss EAP 允许您使用表达式来定义可替换属性来替换配置里的值。表达式使用格式 **\${PARAMETER:DEFAULT_VALUE}**。如果设置了指定的参数，参数的值将被使用。否则将使用默认值。

解析表达式所支持的来源是系统属性、环境变量和库。对于部署，来源可以是 **META-INF/jboss.properties** 文件里列出的属性。对于支持子部署的部署类型，如果属性文件位于外部部署（如 EAR），解析的作用域是全部子部署。如果属性文件位于某个子部署里，那么解析的作用域只是该子部署。

下面的 **standalone.xml** 配置文件将 **public** 接口的 **inet-address** 设置为 **127.0.0.1**，除非设置了 **jboss.bind.address**。

```

<interface name="public">
    <inet-address value="${jboss.bind.address:127.0.0.1}"/>
</interface>

```

用下列命令将 JBoss EAP 作为独立服务器启动时可以使用 **jboss.bind.address** 参数：

```
$ EAP_HOME/bin/standalone.sh -Djboss.bind.address=IP_ADDRESS
```

嵌套的表达式

表达式可以嵌套，这允许用更灵活的表达式替换固定值。嵌套表达式的格式和普通表达式类似，唯一不同的是，一个表达式可以嵌套在另外的表达式中。例如：

```
${SYSTEM_VALUE_1}${SYSTEM_VALUE_2}}
```

嵌套表达式是递归地进行求值的，所以里层的表达式会先进行求值，然后才是外层的表达式。表达式也可以是递归的，此时表达式会先解析为另外一个表达式，然后再进行解析。嵌套表达式在任何可以使用表达式的地方都可以使用，但管理 CLI 命令除外。

例如，如果要隐藏在数据源定义里使用的密码，您就可以使用嵌套表达式。数据源的配置可能有下列内容：

```
<password>${VAULT::ds_ExampleDS::password::1}</password>
```

ds_ExampleDS 的值可以用使用嵌套表达式的系统属性 (**datasource_name**) 替换。数据源的配置可以有下列内容：

```
<password>${VAULT::${datasource_name}::password::1}</password>
```

JBoss EAP 会首先对表达式 **\${datasource_name}** 求值，然后将结果放入更大的表达式并对结果表达式求值。这样配置的优势是数据源的名称可以独立于固定的配置。

基于描述符的属性替换

应用程序配置（如数据源连接参数）在部署、测试和产品环境里通常是不一样的。这种变化有时是通过构建系统脚本来处理，因为 Java EE 规格没有包含外部化这些配置的方法。在 JBoss EAP 里，您可以使用基于描述符的属性替换在外部管理配置。

基于描述符的属性替换通过描述符文件进行属性的替换，允许您从应用程序和构建链删除关于环境的假设。环境专有的配置可以通过部署描述符而不是注解或构建系统脚本来指定。您可以在文件里或通过命令行参数进行配置。

ee 子系统里有几个标记 (flag) 用来控制是否应用属性替换。

JBoss 专有的描述符替换是由 **jboss-descriptor-property-replacement** 标记控制的，它默认为 *enabled*。在启用后，属性可以在下列部署描述符里进行替换：

- **jboss-ejb3.xml**
- **jboss-app.xml**
- **jboss-web.xml**
- ***-jms.xml**
- ***-ds.xml**

下列管理 CLI 命令可以用来启用或禁用 JBoss 专有描述符里的属性替换：

```
/subsystem=ee:write-attribute(name="jboss-descriptor-property-replacement",value=VALUE)
```

Java EE 描述符替换是由 **spec-descriptor-property-replacement** 标记控制的，它默认为 *enabled*。在启用后，属性可以在下列部署描述符里进行替换：

- **ejb-jar.xml**

- persistence.xml
- application.xml
- web.xml

下列管理 CLI 命令可以用来启用或禁用 Java EE 描述符里的属性替换：

```
/subsystem=ee:write-attribute(name="spec-descriptor-property-replacement",value=VALUE)
```

3.5. 文件系统路径

JBoss EAP 将逻辑名称用于文件系统路径。配置的其他部分然后通过逻辑名称引用这些路径，这避免了对每个实例使用绝对路径，也允许将专有的主机配置解析为统一的逻辑名称。

例如，默认的 logging 子系统配置将 `jboss.server.log.dir` 声明为服务器日志目录的逻辑名称。

示例：用于服务器日志目录的相对路径

```
<file relative-to="jboss.server.log.dir" path="server.log"/>
```

JBoss EAP 自动提供大量的标准路径而无需用户在配置文件里进行配置。

表 3.3. 标准路径

属性	描述
java.ext.dirs	Java 开发工具包扩展的目录。
java.home	Java 安装目录
jboss.controller.temp.dir	独立服务器和受管域的共用别名。它是用于临时文件存储的目录。等同于受管域里的 <code>jboss.domain.temp.dir</code> 和独立服务器上的 <code>jboss.server.temp.dir</code> 。
jboss.domain.base.dir	域内容的基础目录。
jboss.domain.config.dir	包含域配置的目录。
jboss.domain.data.dir	域用来存储持久性数据文件的目录。
jboss.domain.log.dir	域用来存储持久性日志文件的目录。
jboss.domain.temp.dir	域用来存储临时文件的目录。
jboss.domain.deployment.dir	域用来存储部署内容的目录。
jboss.domain.servers.dir	域用来存储受管域实例输出的目录。

属性	描述
jboss.home.dir	JBoss EAP 的根目录。
jboss.server.base.dir	独立服务器内容的基础目录。
jboss.server.config.dir	包含独立服务器配置的目录。
jboss.server.data.dir	独立服务器用来存储数据文件的目录。
jboss.server.log.dir	独立服务器用来存储日志文件的目录。
jboss.server.temp.dir	独立服务器用于临时文件存储的目录。
jboss.server.deploy.dir	独立服务器用来存储部署内容的目录。
user.dir	用户的当前工作目录。
user.home	用户的主目录。

您可以[覆盖标准路径](#)或[添加自定义路径](#)。

3.5.1. 覆盖标准路径

您可以覆盖以 **jboss.server.*** 或 **jboss.domain.*** 开头的标准路径的默认位置。这可用两种方式来完成：

- 启动服务器时传入命令行参数。例如：

```
$ EAP_HOME/bin/standalone.sh -Djboss.server.log.dir=/var/log
```

- 修改服务器配置文件里的 **JAVA_OPTS** 变量 (**standalone.conf** 或 **domain.conf**)。例如：

```
JAVA_OPTS="$JAVA_OPTS -Djboss.server.log.dir=/var/log"
```

覆盖受管域的标准路径

这个例子的目标是将域文件存储到 **/opt/jboss_eap/domain_data** 目录，并自定义顶层目录的名称。这里使用默认的目录分组 **by-server**。

- 日志文件将保存在 **all_logs** 子目录里。
- 数据文件将保存在 **all_data** 子目录里。
- 临时文件将保存在 **all_temp** 子目录里。
- 服务器的文件将保存在 **all_servers** 子目录里。

要实现这个配置，您可以在启动 JBoss EAP 时覆盖几个系统属性。

```
$ EAP_HOME/bin/domain.sh -
```

```
Djboss.domain.temp.dir=/opt/jboss_eap/domain_data/all_temp -
Djboss.domain.log.dir=/opt/jboss_eap/domain_data/all_logs -
Djboss.domain.data.dir=/opt/jboss_eap/domain_data/all_data -
Djboss.domain.servers.dir=/opt/jboss_eap/domain_data/all_servers
```

结果路径结构将是这样：

```
/opt/jboss_eap/domain_data/
├── all_data
├── all_logs
├── all_servers
│   ├── server-one
│   │   ├── data
│   │   ├── log
│   │   └── tmp
│   └── server-two
│       ├── data
│       ├── log
│       └── tmp
└── all_temp
```

3.5.2. 添加自定义路径

您可以用管理 CLI 或管理控制台添加自定义文件系统路径。

- 在管理 CLI 里，您可以用下列命令添加新的路径。

```
/path=my.custom.path:add(path=/my/custom/path)
```

- 在管理控制台里，您可以进入 **Configuration** 标签页并选择 **Paths** 来配置文件系统路径。在这里，您可以添加、修改和删除路径。

然后您可以开始使用这个自定义路径了。例如，下面的日志处理程序在它的相对路径里使用了自定义路径。

```
<subsystem xmlns="urn:jboss:domain:logging:3.0">
  ...
  <periodic-rotating-file-handler name="FILE" autoflush="true">
    <formatter>
      <named-formatter name="PATTERN"/>
    </formatter>
    <file relative-to="my.custom.path" path="server.log"/>
    <suffix value=".yyyy-MM-dd"/>
    <append value="true"/>
  </periodic-rotating-file-handler>
  ...
</subsystem>
```

3.5.3. 目录分组

在受管域里，每个服务器的文件都保存在 **EAP_HOME/domain** 目录里。您可以用主机控制器的 **directory-grouping** 属性指定如何组织子目录。目录可以根据 *服务器* 或 *类型* 来分组。在默认情况下，目录是按 *服务器* 分组的。

按服务器进行目录分组

在默认情况下，目录是按服务器分组的。如果您的管理是**以服务器为中心 (server-centric)** 的，我们推荐使用这种配置。例如，它允许对每个服务器实例配置备份和日志文件处理。

如果 JBoss EAP 是用 ZIP 方式安装的，其默认的目录结构（按服务器分组）将是这样的：

```
EAP_HOME/domain
├── servers
│   ├── server-one
│   │   ├── data
│   │   ├── tmp
│   │   └── log
│   └── server-two
│       ├── data
│       ├── tmp
│       └── log
```

要按服务器对域目录进行分组，请使用下列管理 CLI 命令：

```
/host=HOST_NAME:write-attribute(name=directory-grouping,value=by-server)
```

这将更新主机控制器的 **host.xml** 配置文件：

```
<servers directory-grouping="by-server">
  <server name="server-one" group="main-server-group"/>
  <server name="server-two" group="main-server-group" auto-start="true">
    <socket-bindings port-offset="150"/>
  </server>
</servers>
```

按类型进行目录分组

除了按服务器分组，您也可以按文件类型分组。如果您的管理是**以文件类型为中心的 (file type-centric)**，我们推荐这种配置，它允许您轻易地只备份数据文件。

如果 JBoss EAP 是用 ZIP 方式安装且域的文件是按类型分组的，那它的目录结构将是这样的：

```
EAP_HOME/domain
├── data
│   └── servers
│       ├── server-one
│       └── server-two
├── log
│   └── servers
│       ├── server-one
│       └── server-two
└── tmp
    └── servers
        ├── server-one
        └── server-two
```

要按类型对域目录进行分组，请使用下列管理 CLI 命令：

```
/host=HOST_NAME:write-attribute(name=directory-grouping,value=by-type)
```

这将更新主机控制器的 `host.xml` 配置文件：

```
<servers directory-grouping="by-type">
  <server name="server-one" group="main-server-group"/>
  <server name="server-two" group="main-server-group" auto-start="true">
    <socket-bindings port-offset="150"/>
  </server>
</servers>
```

3.6. 系统属性

您可以使用 Java 系统属性来配置许多 JBoss EAP 选项，并设置应用服务器里使用的任何名称-值对。

系统属性可以用来覆盖 JBoss EAP 配置里的默认值。例如，下列公共接口绑定地址的 XML 配置展示了它可以通过 `jboss.bind.address` 系统属性设置，但如果没有提供系统属性，它将默认为 `127.0.0.1`。

```
<inet-address value="{jboss.bind.address:127.0.0.1}"/>
```

在 JBoss EAP 里您可以通过多种方式设置系统属性，包括：

- 传入系统属性到 JBoss EAP 启动脚本
- 使用管理 CLI
- 使用管理控制台
- 使用 `JAVA_OPTS` 环境变量

如果您使用 JBoss EAP 受管域，系统属性可以应用于整个域、专门的服务器组、某个主机及它的全部服务器实例、或者只是特定的服务器实例。和其他多数的 JBoss EAP 域设置一样，在更具体级别设置的系统属性将覆盖抽象的属性。详情请参考[域管理](#)章节。

在启动脚本里传入系统属性

您可以用 `-D` 参数将系统属性传入 JBoss EAP 启动脚本。例如：

```
$ EAP_HOME/bin/standalone.sh -Djboss.bind.address=192.168.1.2
```

对于在启动 JBoss EAP 之前需要设置的 JBoss EAP 选项，这种方法尤其有用。

用管理 CLI 设置系统属性

您可以用下列语法通过管理 CLI 设置系统属性：

```
/system-property=PROPERTY_NAME:add(value=PROPERTY_VALUE)
```

例如：

```
/system-property=jboss.bind.address:add(value=192.168.1.2)
```

当用管理 CLI 设置系统属性时，某些 JBoss EAP 选项，包括上面的 `jboss.bind.address` 示例，都只有在服务器重启后才生效。

对于受管域，上面的例子为整个域配置了一个系统属性。您也可以在更具体的级别上覆盖系统属性。

用管理控制台设置系统属性

- 对于独立服务器，您可以在管理控制台的 **Configuration** 标签页里配置系统属性。选择 **System Properties**，然后点击 **View** 按钮。
- 对于受管域：
 - 域级别的系统属性可以在 **Configuration** 标签页里设置。选择 **System Properties**，然后点击 **View** 按钮。
 - 服务器组和服务器级别的系统属性可以在 **Runtime** 标签页里设置。选择要配置的服务器组或服务器，点击旁边的 **View** 按钮，然后选择 **System Properties** 标签页。
 - 主机级别的系统属性可以在 **Runtime** 标签页里设置。选择要配置的主机，然后在主机名旁的下拉菜单里，选择 **Properties**。

用 JAVA_OPTS 设置系统属性

系统属性也可以用 **JAVA_OPTS** 环境变量来配置。修改 **JAVA_OPTS** 有许多方法，但 JBoss EAP 提供了一个配置文件来设置 JBoss EAP 进程使用的 **JAVA_OPTS**。

对于独立服务器，这个文件是 **EAP_HOME/bin/standalone.conf**，对于受管域则是 **EAP_HOME/bin/domain.conf**。而 Microsoft Windows 系统里这些文件的扩展名都是 **.bat**。

在相关的配置文件里添加您的 **JAVA_OPTS** 系统属性定义。下面的例子演示了在 Red Hat Enterprise Linux 系统里设置绑定地址。

- 对于 **standalone.conf**，在文件结尾添加您的 **JAVA_OPTS** 系统属性定义。例如：

```
...
# Set the bind address
JAVA_OPTS="$JAVA_OPTS -Djboss.bind.address=192.168.1.2"
```

- 对于 **domain.conf**，**JAVA_OPTS** 必须在进程控制器的 **JAVA_OPTS** 之前设置。例如：

```
...
# Set the bind address
JAVA_OPTS="$JAVA_OPTS -Djboss.bind.address=192.168.1.2"

# The ProcessController process uses its own set of java options
if [ "x$PROCESS_CONTROLLER_JAVA_OPTS" = "x" ]; then
...
```

3.7. 管理审计日志

您可以为管理接口启用审计日志，使用管理控制台、管理 CLI 或使用 Management API 的自定义应用程序执行的所有操作登记日志。审计日志条目以 JSON 格式保存。审计日志默认是禁用的。

您可以配置审计日志输出到[文件](#)或 [syslog 服务器](#)。



注意

Login 和 logout 事件不能被审计，因为 JBoss EAP 里没有验证的会话。相反，审计消息是在从用户接收操作时登记日志的。

独立服务器日志审计

虽然默认是禁用的，默认的审计日志配置是写入到一个文件。

```
<audit-log>
  <formatters>
    <json-formatter name="json-formatter"/>
  </formatters>
  <handlers>
    <file-handler name="file" formatter="json-formatter" path="audit-
log.log" relative-to="jboss.server.data.dir"/>
  </handlers>
  <logger log-boot="true" log-read-only="false" enabled="false">
    <handlers>
      <handler name="file"/>
    </handlers>
  </logger>
</audit-log>
```

这个配置可以用下列管理 CLI 命令读取。

```
/core-service=management/access=audit:read-resource(recursive=true)
```

关于启用代理服务器的审计日志，请参考[启用审计日志](#)。

管理域审计日志

虽然默认是禁用的，默认审计日志配置为每台主机和服务器写入到一个文件。

```
<audit-log>
  <formatters>
    <json-formatter name="json-formatter"/>
  </formatters>
  <handlers>
    <file-handler name="host-file" formatter="json-formatter"
relative-to="jboss.domain.data.dir" path="audit-log.log"/>
    <file-handler name="server-file" formatter="json-formatter"
relative-to="jboss.server.data.dir" path="audit-log.log"/>
  </handlers>
  <logger log-boot="true" log-read-only="false" enabled="false">
    <handlers>
      <handler name="host-file"/>
    </handlers>
  </logger>
  <server-logger log-boot="true" log-read-only="false" enabled="false">
    <handlers>
      <handler name="server-file"/>
    </handlers>
  </server-logger>
</audit-log>
```

这个配置可以用下列管理 CLI 命令读取。

```
/host=HOST_NAME/core-service=management/access=audit:read-
resource(recursive=true)
```

关于启用受管域的审计日志，请参考[启用审计日志](#)。

3.7.1. 启用管理审计日志

JBoss EAP 预配置了用于审计日志的文件处理程序，虽然审计日志默认是禁用的。启用审计日志的管理 CLI 命令取决于 JBoss EAP 是作为独立服务器还是受管域运行的。关于文件处理程序的属性，请参考[管理审计日志属性](#)。

要设立 syslog 审计日志，请参考[设立管理审计日志到 Syslog 服务器](#)。

启用独立服务器日志审计

您可以用下列命令启用审计日志。

```
/core-service=management/access=audit/logger=audit-log:write-attribute(name=enabled,value=true)
```

在默认情况下，这会将审计日志写入到 **EAP_HOME/standalone/data/audit-log.log**。

启用受管域审计日志

受管域的默认审计日志配置是预配置为对每个主机和每台服务器写入审计日志的。

您可以用下列命令为每个主机启用审计日志。

```
/host=HOST_NAME/core-service=management/access=audit/logger=audit-log:write-attribute(name=enabled,value=true)
```

在默认情况下，这会将审计日志写入到 **EAP_HOME/domain/data/audit-log.log**。

您可以用下列命令为每台服务器启用审计日志。

```
/host=HOST_NAME/core-service=management/access=audit/server-logger=audit-log:write-attribute(name=enabled,value=true)
```

在默认情况下，这会将审计日志写入到 **EAP_HOME/domain/servers/SERVER_NAME/data/audit-log.log**。

3.7.2. 发送管理审计日志到 Syslog 服务器

Syslog 处理程序指定发送日志条目至 Syslog 服务器使用的参数，特别是 Syslog 服务器侦听的主机名和端口。发送审计日志到 Syslog 服务器提供了一个比登录本地文件或 Syslog 服务器更安全的选项。您可以同时定义和激活多个 Syslog 处理程序。

在默认情况下，审计日志被预配置为在启用时输出到文件。使用下列步骤来设置并启用审计日志到 Syslog 服务器。关于 Syslog 处理程序的属性，请参考[管理审计日志属性](#)。

1. 添加 syslog 处理程序。

创建 Syslog 处理程序，指定 Syslog 服务器的主机和端口。在受管域里，您必须在 **/core-service** 命令之前使用 **/host=HOST_NAME**。

```
batch
/core-service=management/access=audit/syslog-
handler=SYSLOG_HANDLER_NAME:add(formatter=json-formatter)
/core-service=management/access=audit/syslog-
```



```
handler=SYSLOG_HANDLER_NAME/protocol=udp:add(host=HOST_NAME,port=PORT)
run-batch
```



注意

要传入的参数根据指定的协议的不同而不同。

要配置处理程序使用 TLS 来与 Syslog 服务器安全地通讯，您也必须配置验证，例如：

```
/core-service=management/access=audit/syslog-
handler=SYSLOG_HANDLER_NAME/protocol=tls/authentication=t
ruststore:add(keystore-path=PATH_TO_TRUSTSTORE,keystore-
password=TRUSTSTORE_PASSWORD)
```

- 2. 添加对 Syslog 处理程序的引用。
在受管域里，您必须在这个命令之前使用 `/host=HOST_NAME`。

```
/core-service=management/access=audit/logger=audit-
log/handler=SYSLOG_HANDLER_NAME:add
```

- 3. 启用审计日志。
关于如何启用审计日志，请参考[启用管理审计日志](#)。



重要

除非您也启用了操作系统里的日志，否则在 JBoss EAP 里启用审计日志至 Syslog 服务器无法成功。

关于 Red Hat Enterprise Linux 上 **rsyslog** 配置的更多信息，请参考<https://access.redhat.com/documentation/en/red-hat-enterprise-linux/>《Red Hat Enterprise Linux 系统管理员指南》的『Rsyslog 的基本配置』章节。

3.7.3. 读取审计日志条目

输出到文件的审计日志条目最好用文本查看器来查看，而输出到 Syslog 服务器的日志则最好使用 Syslog 查看器来查看。



注意

我们不推荐使用文件编辑器来查看日志文件，因为某些编辑器可能会阻止继续写入日志条目到日志文件里。

审计日志条目以 JSON 格式保存。每个日志条目都以可选的时间戳开始，后面跟着下表里的字段。

表 3.4. 管理审计日志的字段

字段名	描述
-----	----

字段名	描述
access	<p>它可以有这些值：</p> <ul style="list-style-type: none"> • NATIVE - 操作通过原生管理接口进行。 • HTTP - 操作通过域 HTTP 接口进行。 • JMX - 操作通过 jmx 子系统进行。
booting	如果在引导过程中执行操作则为 true ，如果在服务器已启动并运行时执行则为 false 。
domainUUID	当从域控制器传播到服务器、从主机控制器和从主机控制器服务器时，连接所有操作的 ID。
ops	正在执行的操作。这是序列换为 JSON 的操作列表。在引导时，这是解析 XML 所致的操作。引导后，这个列表通常包含单个条目。
r/o	如果操作没有修改管理模型，值为 true ，否则值为 false 。
remote-address	执行这个操作的客户的地址。
success	如果操作成功则为 true ，如果回滚则为 false 。
type	值可以为 core ，表示这是一个管理操作；或者为 jmx ，表示它来自 jmx 子系统。
user	已验证用户的用户名。如果使用管理 CLI 运行的操作在和运行服务器相同的主机上运行，特殊的用户 \$local 将被使用。
version	JBoss EAP 实例的版本号。

第 4 章 网络和端口配置

4.1. 接口

在配置中，JBoss EAP 会指定已命名的接口。这将在配置中使用接口的逻辑名来指定独立的接口，而不需要在每次指定端口时都需要它们的完整详细信息。

这还使在一个管理域中的配置变得比较容易，因此网络接口的详细信息在多个机器中可能会有所不同。每个服务器实例都可以与一个逻辑名组相关联。

standalone.xml、**domain.xml** 和 **host.xml** 文件都包括了接口的声明。根据使用的默认配置的不同，可能会有几个不同的预配置的接口名。**management** 接口可用于所有需要管理层的组件和服务，包括 HTTP 管理端点。**public** 阶段可用于所有与应用程序相关的网络通讯。**unsecure** 接口用于标准配置中的 IIOP 套接字。**private** 接口用于标准配置中的 JGroups 套接字。

4.1.1. 默认的接口配置

```
<interfaces>
  <interface name="management">
    <inet-address value="${jboss.bind.address.management:127.0.0.1}"/>
  </interface>
  <interface name="public">
    <inet-address value="${jboss.bind.address:127.0.0.1}"/>
  </interface>
  <interface name="private">
    <inet-address value="${jboss.bind.address.private:127.0.0.1}"/>
  </interface>
  <interface name="unsecure">
    <inet-address value="${jboss.bind.address.unsecure:127.0.0.1}"/>
  </interface>
</interfaces>
```

在默认情况下，JBoss EAP 绑定这些接口到 **127.0.0.1**，但这些值可以通过设置合适的属性在运行时进行覆盖。例如，用下列命令将 JBoss EAP 作为独立服务器启动时，您可以设置 **public** 接口的 **inet-address**。

```
$ EAP_HOME/bin/standalone.sh -Djboss.bind.address=IP_ADDRESS
```

或者，您可以在命令行里使用 **-b** 参数。关于服务器启动选项的更多信息，请参考[服务器运行时参数](#)。



重要

如果您修改了 JBoss EAP 使用的默认网络接口或端口，您必须记得修改任何使用这些接口或端口的脚本，这包含 JBoss EAP 服务脚本，而且记得在访问管理控制台或管理 CLI 时指定正确的接口和端口。

4.1.2. 配置接口

网络接口通过指定一个逻辑名和物理接口选择条件进行声明。选择条件可以包括一个使用通配符的地址，也可以指定一组字符，只有包括这些字符的接口或地址才可以被有效匹配。如需了解选择条件的网站列表，请参阅[接口属性](#)项。

接口可以通过管理控制台或管理 CLI 进行配置。以下是添加和更新接口的一些示例。管理 CLI 命令被首先显示，接下来是相应的配置 XML。

添加带有 **NIC** 值的接口

添加具有 **eth0** NIC 值的新接口。

```
/interface=external:add(nic=eth0)
```

```
<interface name="external">
  <nic name="eth0"/>
</interface>
```

添加带有几个条件值的接口

添加一个新的网络接口，这个接口的匹配条件是：在特定子网中、已在线、支持多点传送，并且不是点对点的。

```
/interface=default:add(subnet-
match=192.168.0.0/16,up=true,multicast=true,not={point-to-point=true})
```

```
<interface name="default">
  <subnet-match value="192.168.0.0/16"/>
  <up/>
  <multicast/>
  <not>
    <point-to-point/>
  </not>
</interface>
```

更新接口属性

更新 **public** 接口的默认 **inet-address** 值，保留 **jboss.bind.address** 属性以允许在运行时设置这个值。

```
/interface=public:write-attribute(name=inet-
address,value="{jboss.bind.address:192.168.0.0}")
```

```
<interface name="public">
  <inet-address value="{jboss.bind.address:192.168.0.0}"/>
</interface>
```

把接口添加到受管域里的一个服务器

```
/host=master/server-config=SERVER_NAME/interface=INTERFACE_NAME:add(inet-
address=127.0.0.1)
```

```
<servers>
  <server name="SERVER_NAME" group="main-server-group">
    <interfaces>
      <interface name="INTERFACE_NAME">
        <inet-address value="127.0.0.1"/>
      </interface>
    </interfaces>
  </server>
</servers>
```

4.2. 套接字绑定

套接字绑定和套接字绑定组可以用来定义网络端口，以及它们与 JBoss EAP 配置所需的网络接口的关系。套接字绑定就是一个端口的已命名的配置。套接字绑定组是一组套接字绑定声明的集合，它们被分成组，并带有一个逻辑名。

这样，配置中的其它部分就可以使用逻辑名来指定套接字绑定，而不需要使用套接字配置的详细信息。

对于这些已命名配置的声明可以在 `standalone.xml` 和 `domain.xml` 配置文件中找到。一个独立的服务器只包括一个套接字绑定组，而一个管理的域可以包括多个组。您可以为管理域中的每个服务器组创建一个套接字绑定组，或在多个服务器组间共享一个套接字绑定组。

JBoss EAP 默认使用的端口取决于所使用的套接字绑定组以及部署的要求。

4.2.1. 管理端口

JBoss EAP 7 合并了管理端口。在默认情况下，JBoss EAP 7 对于原生管理（被管理 CLI 使用）和 HTTP 管理（被基于 Web 的管理控制台使用）使用端口 **9990**。在 JBoss EAP 6 里用于原生管理端口的 **9999** 不再使用，但仍可以在需要时启用。

如果管理控制台启用了 HTTPS，那默认情况下端口 **9993** 将被使用。

4.2.2. 默认的套接字绑定

JBoss EAP ships with a socket binding group for each of the four predefined profiles (*default*, *ha*, *full*, *full-ha*).

关于默认套接字绑定的详细信息，如默认端口和描述，请参考[默认套接字绑定](#)章节。



重要

如果您修改了 JBoss EAP 使用的默认网络接口或端口，您必须记得修改任何使用这些接口或端口的脚本，这包含 JBoss EAP 服务脚本，而且记得在访问管理控制台或管理 CLI 时指定正确的接口和端口。

独立服务器

在运行为独立服务器时，每个配置文件都只定义一个套接字绑定组。每个独立配置文件（`standalone.xml`、`standalone-ha.xml`、`standalone-full.xml`、`standalone-full-ha.xml`）都为它对应的配置集使用的技术定义了套接字绑定。

例如，默认的独立服务器配置文件（`standalone.xml`）指定了下面的套接字绑定。

```
<socket-binding-group name="standard-sockets" default-interface="public"
port-offset="${jboss.socket.binding.port-offset:0}">
  <socket-binding name="management-http" interface="management"
port="${jboss.management.http.port:9990}"/>
  <socket-binding name="management-https" interface="management"
port="${jboss.management.https.port:9993}"/>
  <socket-binding name="ajp" port="${jboss.ajp.port:8009}"/>
  <socket-binding name="http" port="${jboss.http.port:8080}"/>
  <socket-binding name="https" port="${jboss.https.port:8443}"/>
  <socket-binding name="txn-recovery-environment" port="4712"/>
  <socket-binding name="txn-status-manager" port="4713"/>
</socket-binding-group>
```

```

<outbound-socket-binding name="mail-smtp">
  <remote-destination host="localhost" port="25"/>
</outbound-socket-binding>
</socket-binding-group>

```

受管域

When running in a managed domain, all socket binding groups are defined in the **domain.xml** file. There are four predefined socket binding groups:

- **standard-sockets**
- **ha-sockets**
- **full-sockets**
- **full-ha-sockets**

每个套接字绑定组都为对应的配置集使用的技术指定了套接字绑定。例如，**full-ha-sockets** 套接字绑定组定义了几个高可用性的 **full-ha** 所使用的 **jgroups** 套接字绑定。

```

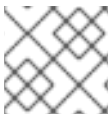
<socket-binding-groups>
  <socket-binding-group name="standard-sockets" default-
interface="public">
    <!-- Needed for server groups using the 'default' profile -->
    <socket-binding name="ajp" port="${jboss.ajp.port:8009}"/>
    <socket-binding name="http" port="${jboss.http.port:8080}"/>
    <socket-binding name="https" port="${jboss.https.port:8443}"/>
    <socket-binding name="txn-recovery-environment" port="4712"/>
    <socket-binding name="txn-status-manager" port="4713"/>
    <outbound-socket-binding name="mail-smtp">
      <remote-destination host="localhost" port="25"/>
    </outbound-socket-binding>
  </socket-binding-group>
  <socket-binding-group name="ha-sockets" default-interface="public">
    <!-- Needed for server groups using the 'ha' profile -->
    ...
  </socket-binding-group>
  <socket-binding-group name="full-sockets" default-interface="public">
    <!-- Needed for server groups using the 'full' profile -->
    ...
  </socket-binding-group>
  <socket-binding-group name="full-ha-sockets" default-
interface="public">
    <!-- Needed for server groups using the 'full-ha' profile -->
    <socket-binding name="ajp" port="${jboss.ajp.port:8009}"/>
    <socket-binding name="http" port="${jboss.http.port:8080}"/>
    <socket-binding name="https" port="${jboss.https.port:8443}"/>
    <socket-binding name="iiop" interface="unsecure" port="3528"/>
    <socket-binding name="iiop-ssl" interface="unsecure" port="3529"/>
    <socket-binding name="jgroups-mping" interface="private" port="0"
multicast-address="${jboss.default.multicast.address:230.0.0.4}"
multicast-port="45700"/>
    <socket-binding name="jgroups-tcp" interface="private"
port="7600"/>
    <socket-binding name="jgroups-tcp-fd" interface="private"
port="57600"/>

```

```

        <socket-binding name="jgroups-udp" interface="private"
port="55200" multicast-
address="${jboss.default.multicast.address:230.0.0.4}" multicast-
port="45688"/>
        <socket-binding name="jgroups-udp-fd" interface="private"
port="54200"/>
        <socket-binding name="modcluster" port="0" multicast-
address="224.0.1.105" multicast-port="23364"/>
        <socket-binding name="txn-recovery-environment" port="4712"/>
        <socket-binding name="txn-status-manager" port="4713"/>
        <outbound-socket-binding name="mail-smtp">
            <remote-destination host="localhost" port="25"/>
        </outbound-socket-binding>
    </socket-binding-group>
</socket-binding-groups>

```



注意

管理接口的套接字配置是在域控制器的 **host.xml** 文件里定义的。

4.2.3. 配置套接字绑定

在定义套接字绑定时，您可以配置 **port** 和 **interface** 属性以及多点传送设置，如 **multicast-address** 和 **multicast-port**。关于全部的可用套接字绑定属性，请参考[套接字绑定属性](#)章节。

套接字绑定可用管理控制台或管理 CLI 来配置。下面的步骤使用管理 CLI 来添加套接字绑定组、添加套接字绑定和配置套接字绑定设置。

1. 添加新的套接字绑定组。请注意，在作为独立服务器运行时，无法执行这个步骤。

```
/socket-binding-group=new-sockets:add(default-interface=public)
```

2. 添加套接字绑定。

```
/socket-binding-group=new-sockets/socket-binding=new-socket-
binding:add(port=1234)
```

3. 修改套接字绑定以使用默认接口之外的接口，这是通过套接字绑定组设置的。

```
/socket-binding-group=new-sockets/socket-binding=new-socket-
binding:write-attribute(name=interface,value=unsecure)
```

下面的例子展示了在完成上述步骤后的 XML 配置文件。

```

<socket-binding-groups>
    ...
    <socket-binding-group name="new-sockets" default-interface="public">
        <socket-binding name="new-socket-binding" interface="unsecure"
port="1234"/>
    </socket-binding-group>
</socket-binding-groups>

```

4.2.4. 端口偏移

端口偏移量是一个添加至所有在套接字绑定组里指定的端口值的数值。这允许该服务器继承套接字绑定组里定义的端口值，而偏移量确保了不会和相同主机上的其他服务器冲突。例如，如果套接字绑定组的 HTTP 端口是 **8080**，而服务器使用偏移量 **100**，那这个 HTTP 端口会是 **8180**。

下面是一个使用管理 CLI 为受管域的服务器设置端口偏移量 **250** 的例子。

```
/host=master/server-config=server-two/:write-attribute(name=socket-binding-port-offset,value=250)
```

端口偏移量可以用于受管域里的服务器，也可以用于在相同主机上运行的多个独立服务器。

当用 **jboss.socket.binding.port-offset** 属性启动独立服务器时您可以传入一个端口偏移量。

```
$ EAP_HOME/bin/standalone.sh -Djboss.socket.binding.port-offset=100
```

4.3. IPV6 地址

在默认情况下，JBoss EAP 是以 IPv4 地址运行的。下面的步骤展示了如何配置 JBoss EAP 以 IPv6 地址运行。

根据 IPv6 地址配置 JVM 栈
更新启动配置以首选 IPv6 地址。

1. 打开启动配置文件。
 - 当以独立服务器运行时，编辑 **EAP_HOME/bin/standalone.conf** 文件（或对于 Windows Server，编辑 **standalone.conf.bat**）。
 - 当运行在受管域里时，编辑 **EAP_HOME/bin/domain.conf** 文件（或对于 Windows Server，编辑 **domain.conf.bat**）。
2. 设置 **java.net.preferIPv4Stack** 属性为 **false**。

```
-Djava.net.preferIPv4Stack=false
```

3. 附加 **java.net.preferIPv6Addresses** 属性并设置它为 **true**。

```
-Djava.net.preferIPv6Addresses=true
```

下面的例子展示了在完成上述修改后启动配置里的 JVM 选项。

```
# Specify options to pass to the Java VM.
#
if [ "x$JAVA_OPTS" = "x" ]; then
    JAVA_OPTS="-Xms1303m -Xmx1303m -Djava.net.preferIPv4Stack=false"
    JAVA_OPTS="$JAVA_OPTS -
Djboss.modules.system.pkgs=$JBoss_MODULES_SYSTEM_PKGS -
Djava.awt.headless=true"
    JAVA_OPTS="$JAVA_OPTS -Djava.net.preferIPv6Addresses=true"
else
```

根据 IPv6 地址更新接口声明

配置里的默认接口值可改为 IPv6 地址。例如，下面的管理 CLI 命令将 **management** 接口设置为 IPv6 loopback 地址 (::1)。

```
/interface=management:write-attribute(name=inet-  
address,value="${jboss.bind.address.management:::1}")
```

下面的例子展示了在执行上述命令后的 XML 配置文件。

```
<interfaces>  
  <interface name="management">  
    <inet-address value="${jboss.bind.address.management:::1}"/>  
  </interface>  
  ....  
</interfaces>
```

第 5 章 JBOSS EAP 的安全性

JBoss EAP 可以为它自己的接口和服务进行安全配置，也可以为运行在它上面的应用程序进行安全配置。

- 如需了解一般性的安全概念，以及针对于 JBoss EAP 的安全概念，请参阅 [Security Architecture](#)。
- 如需了解对 JBoss EAP 本身进行安全设置的信息，请参阅 [How to Configure Server Security](#)。
- 如需了解如何为部署在 JBoss EAP 上的应用程序提供安全性的信息，请参阅 [How to Configure Identity Management](#)。
- 如需了解如何使用 Kerberos 来为 JBoss EAP 配置单点登录的信息，请参阅 [How to Set Up SSO with Kerberos](#)。
- 如需了解如何使用 SAML v2 为 JBoss EAP 配置单点登录的信息，请参阅 [How To Set Up SSO with SAML v2](#)。

第 6 章 JBOSS EAP 的类加载

JBoss EAP 使用一个模块类加载系统来控制所部署的应用程序的类路径。这个系统和传统的分级结构类加载程序系统相比，可以提供更多的灵活性以及控制功能。开发人员可以在他们的应用程序对使用的类进行“细颗粒”控制，并可以把部署配置为使用自己的类而不是使用应用程序服务器所提供的类。

模块化类加载程序把所有 Java 类分离为不同的逻辑组，这些逻辑组被称为模块。每个模块都可以定义它的依赖模块，从而把依赖模块中的类加到自己的模块路径中。因为每个部署的 JAR 和 WAR 文件都被看作作为一个模块，因此开发人员可以通过把模块配置添加到应用程序中来对应用程序类的内容进行控制。

6.1. 模块

模块是类的一个逻辑组，它被用来进行类加载以及对依赖模块的管理。JBoss EAP 有两种模块：*静态*（*static*）和*动态*（*dynamic*）。这两种模块的主要区别是如何进行打包。

静态模块

静态模块在应用程序服务器的 **EAP_HOME/modules/** 目录中定义。每个模块都以一个子目录的形式存在（例如 **EAP_HOME/modules/com/mysql/**）。每个模块目录包括一个子目录，默认为 **main** 并包括 **module.xml** 配置文件和所需的 JAR 文件。所有由应用程序服务器提供的 API（包括 Java EE API 以及其它 API）都以静态模块的形式提供。

MySQL JDBC Driver module.xml 文件示例

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.mysql">
  <resources>
    <resource-root path="mysql-connector-java-5.1.36-bin.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

模块名（**com.mysql**）必须匹配模块的目录结构，除了 Slot 名称（**main**）。

如果许多应用程序部署在一个使用相同第三程序库的服务器上时，创建自定义的静态模块就会非常有用。系统管理员可以创建并安装一个包括了这些程序库的模块，而不需要为每个应用程序单独绑定这些程序库。然后，应用程序就可以声明显性依赖（explicit dependency）于这个静态模块。

JBoss EAP 所提供的模块包括在 **EAP_HOME/modules** 目录下的 **system** 目录中。这将使这些模块和第三方所提供的模块相分离。所有红帽所提供的、位于 JBoss EAP 之上的产品都会在 **system** 目录中安装这些模块。

用户需要保证自定义的模块被安装在 **EAP_HOME/modules** 目录中，每个模块使用一个目录。这可以确保存在于 **system** 目录中的自定义版本模块被加载，而不是加载原来的标准版本。这样，用户提供的模块会在系统模块之前使用。

如果使用 **JBOSS_MODULEPATH** 环境变量修改 JBoss EAP 搜索模块的位置，则产品会在一个指定的位置中查找 **system** 子目录结构。一个 **system** 结构需要存在于 **JBOSS_MODULEPATH** 指定的位置中。

动态模块

动态模块由应用程序服务器为每个 JAR 或 WAR 部署（或一个 EAR 中的子部署）创建并加载。动态模块的名字由部署的归档名获得。因为部署会作为模块加载，所以它们可以配置依赖关系并可用作其它部署的依赖模块。

只在需要时才加载模块。这通常在部署具有显性或隐性依赖关系的应用程序时才会发生。

6.2. 模块依赖关系

一个模块依赖关系就是一个声明，它指定了一个模块需要一个或多个其它模块的类才可以正常工作。当 JBoss EAP 加载一个模块时，模块化类加载程序会解析这个模块的依赖关系，并把每个依赖模块的类添加到它的类路径中。如果一个指定的依赖模块没有找到，则模块加载将失败。



注意

关于模块及模块类加载系统的完整细节，请参考[模块](#)章节。

部署的应用程序（JAR 或 WAR）被加载为动态模块并利用依赖关系来访问 JBoss EAP 提供的 API。

有两种依赖关系：*显性*（*explicit*）和*隐性*（*implicit*）。

显性依赖关系

显性依赖关系由程序员在一个配置文件中声明。一个静态模块可以在它的 `module.xml` 文件中声明依赖关系；而一个动态模块可以在部署的 `MANIFEST.MF` 或 `jboss-deployment-structure.xml` 部署描述符（descriptor）中声明依赖关系。

隐性依赖关系

当在部署里找到某些条件或元数据时，JBoss EAP 会自动添加隐性依赖关系。JBoss EAP 提供的 Java EE7 API 是通过检测部署里的隐性依赖关系而添加的模块示例。

使用 `jboss-deployment-structure.xml` 部署描述符文件可以把部署配置为排除特定的隐性依赖关系。当一个应用程序绑定了某个特定版本的库，JBoss EAP 将试图加载这个库作为一个隐性依赖时非常有用。

可选的依赖关系

显性依赖关系可指定为可选的。加载可选依赖关系失败不会导致模块加载失败。然而，如果依赖关系之后变成可用的，它将不会被添加到模块的类路径里。加载模块时依赖关系必须是可用的。

导出依赖关系

一个模块的类路径只包括它自己的类，以及它直接的依赖关系。一个模块无法访问它的依赖模块的依赖模块的库。但是，一个模块可以指定输出一个显性依赖关系，所有依赖于这个模块的模块都可以使用这个输出的依赖关系。

例如，模块 A 依赖于模块 B，模块 B 依赖于模块 C。模块 A 可以访问模块 B 的类，模块 B 可以访问模块 C 的类。除非满足以下条件，模块 A 将无法访问模块 C 的类：

- 模块 A 声明了对模块 C 的显性依赖关系，或
- 模块 B 导出它对模块 C 的依赖关系。

全局模块

全局模块是 JBoss EAP 为每个应用程序作为依赖关系提供的模块。通过加入 JBoss EAP 的全局模块列表，任何模块都可以称为全局的。它不需要修改模块。

详情请参考[定义全局模块](#)章节。

6.3. 创建自定义模块

通过添加自定义模块，可以使资源对运行在 JBoss EAP 上的部署有效。您可以[手工创建模块](#)或[使用管理 CLI 创建模块](#)。

在创建模块后，为了使资源对应用程序有效，您需要[作为依赖关系添加模块](#)。

手工创建自定义模块

您可以通过以下步骤手工创建自定义模块。

1. 在 `EAP_HOME/modules/` 目录中创建适当的目录结构。

示例：创建 MySQL JDBC 驱动目录结构

```
$ cd EAP_HOME/modules/
$ mkdir -p com/mysql/main
```

2. 把 JAR 文件或其它所需资源复制到 `main/` 子目录中。

示例：复制 MySQL JDBC 驱动 JAR

```
$ cp /path/to/mysql-connector-java-5.1.36-bin.jar
EAP_HOME/modules/com/mysql/main/
```

3. 在 `main/` 子目录中创建 `module.xml` 文件，在文件中指定适当的资源和依赖程序。

示例：MySQL JDBC 驱动的 `module.xml` 文件

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.mysql">
  <resources>
    <resource-root path="mysql-connector-java-5.1.36-bin.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

使用管理 CLI 创建自定义模块

You can create a custom module using the **module add** management CLI command.



重要

Using the **module** management CLI command to add and remove modules is provided as [technology preview](#) only. This command is not appropriate for use in a managed domain or when connecting to the management CLI remotely. Modules should be [added](#) and [removed](#) manually in a production environment.

1. 启动 JBoss EAP 服务器。
2. 启动管理 CLI，但请不要使用 **--connect** 或 **-c** 参数来连接运行的实例。

```
$ EAP_HOME/bin/jboss-cli.sh
```

3. 使用 **module add** 管理 CLI 命令来添加新的核心模块。

```
module add --name=MODULE_NAME --resources=PATH_TO_RESOURCE --
dependencies=DEPENDENCIES
```

关于使用这个命令添加或删除模块的详情，请执行 **module --help**。

添加依赖的模块

为了让您的应用程序能够访问这个模块的资源，需要将模块添加为依赖关系。

- 如需了解使用部署描述符为特定应用添加依赖关系，请参阅 JBoss EAP *Development Guide* 中的 [Add an Explicit Module Dependency to a Deployment](#) 一节。
- 关于将模块作为依赖关系添加至所有应用程序的说明，请参考[定义全局模块](#)章节。

作为一个示例，以下步骤会添加一个包括了多个属性的 JAR 文件作为一个模块，并定义一个全局模块。然后，一个应用程序就可以加载这些属性。

1. 将 JAR 文件添加为核心模块。

```
module add --name=myprops --resources=/path/to/properties.jar
```

2. 将这个模块定义为全局模块，使其为所有部署可用。

```
/subsystem=ee:write-attribute(name=global-modules,value=
[ {name=myprops} ])
```

3. 应用程序然后可以从 JAR 文件包含的其中一个属性文件里获取属性。

```
Thread.currentThread().getContextClassLoader().getResource("my.prope
rties");
```

6.4. REMOVE A CUSTOM MODULE

Custom static modules can be removed [manually](#) or by [using the management CLI](#).

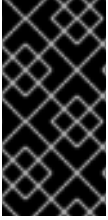
Remove a Custom Module Manually

Before manually removing a module, ensure that it is not required by deployed applications or elsewhere in the server configuration, such as by a datasource.

To remove a custom module, remove the module's directory under **EAP_HOME/modules/**, which includes its **module.xml** file and associated JAR files or other resources. For example, remove the **EAP_HOME/modules/com/mysql/main/** directory to remove a custom MySQL JDBC driver module in the **main** slot.

Remove a Custom Module Using the Management CLI

You can remove a custom module using the **module remove** management CLI command.



重要

Using the **module** management CLI command to add and remove modules is provided as [technology preview](#) only. This command is not appropriate for use in a managed domain or when connecting to the management CLI remotely. Modules should be [added](#) and [removed](#) manually in a production environment.

1. 启动 JBoss EAP 服务器。
2. 启动管理 CLI，但请不要使用 **--connect** 或 **-c** 参数来连接运行的实例。

```
$ EAP_HOME/bin/jboss-cli.sh
```

3. Use the **module remove** management CLI command to remove the custom module.

```
module remove --name=MODULE_NAME
```

- Use the **--slot** argument if the module to remove is in a slot other than **main**.

Example: Remove a MySQL Module

```
module remove --name=com.mysql
```

关于使用这个命令添加或删除模块的详情，请执行 **module --help**。

6.5. 定义全局模块

您可以为 JBoss EAP 定义一个全局模块列表，它将模块作为依赖关系添加至所有的部署。



注意

您必须知道将被配置为全局模块的模块名。关于部署里的模块命名规则，请参考[动态模块命名](#)章节。

使用下列管理 CLI 命令来定义全局模块列表。

```
/subsystem=ee:write-attribute(name=global-modules,value=[{name=MODULE_NAME_1},{name=MODULE_NAME_2}])
```

全局模块可以通过管理控制台的 **Configuration** 标签页里的 **EE** 子系统并选择 **Global Modules** 来添加或删除。

6.6. 配置子部署隔离

一个 Enterprise Archive (EAR) 中的每个子部署都是一个带有自己的类加载程序的动态模块。子部署都会在父模块中有一个隐性依赖关系，从而使它们可以访问 **EAR/lib** 中的类。在默认情况下，子部署可以访问那个 EAR 中的其它子部署的资源。

如果您不允许子部署访问属于其他子部署的类，在 JBoss EAP 里您可以启用严格的子部署隔离。这个设置将影响到所有的部署。

为**所有部署**启用**子部署隔离**

子部署隔离（subdeployment isolation）功能可以通过 **ee** 子系统的管理控制台或管理 CLI 启用或禁用。在默认情况下，子部署隔离被设置为“false（禁用）”，这将允许子部署访问一个 EAR 部署中的其它子部署的资源。

使用下列管理 CLI 来启用 EAR 子部署隔离。

```
/subsystem=ee:write-attribute(name=ear-subdeployments-isolated,value=true)
```

EAR 里的子部署不再可以访问其他子部署里的资源。

6.7. 定义外部的 JBOSS EAP 模块目录

JBoss EAP 模块的默认目录是 **EAP_HOME/modules**。您可以用 **JBOSS_MODULEPATH** 为 JBoss EAP 模块指定不同的目录。以下步骤在 JBoss EAP 启动配置文件里设置这个变量。



注意

您也可以把 **JBOSS_MODULEPATH** 设置为一个环境变量，而不是在 JBoss EAP 启动配置文件里设置它。

1. 编辑启动配置文件。

- 当以独立服务器运行时，编辑 **EAP_HOME/bin/standalone.conf** 文件（或对于 Windows Server，编辑 **standalone.conf.bat**）。
- 当运行在受管域里时，编辑 **EAP_HOME/bin/domain.conf** 文件（或对于 Windows Server，编辑 **domain.conf.bat**）。

2. 设置 **JBOSS_MODULEPATH** 变量，例如：

```
JBOSS_MODULEPATH="/path/to/modules/directory/"
```

要指定一个目录列表，请使用冒号（:）来分隔目录列表。



注意

对于 Windows Server，使用下列语法来设置 **JBOSS_MODULEPATH** 变量：

```
set "JBOSS_MODULEPATH /path/to/modules/directory/"
```

要指定一个目录列表，请使用分号（;）来分隔目录列表。

6.8. 动态模块的命名规则

JBoss EAP 将所有部署加载为模块，它们是根据下列规则进行命名的。

- WAR 和 JAR 文件的部署使用下列格式来命名：

```
deployment.DEPLOYMENT_NAME
```

例如，**inventory.war** 和 **store.jar** 的模块名分别是 **deployment.inventory.war** 和 **deployment.store.jar**。

- EAR 里的子部署用下列格式来命名：

```
deployment.EAR_NAME.SUBDEPLOYMENT_NAME
```

例如，EAR **accounts.ear** 里的 **reports.war** 的模块名是 **deployment.accounts.ear.reports.war**。

第 7 章 部署应用程序

JBoss EAP 为管理员和开发人员提供了一套应用程序部署和配置选项。对于管理员，[管理控制台](#)和[管理 CLI](#) 提供了在一个生产环境中管理应用程序部署的图形和命令行接口。对于开发人员，一组应用程序部署测试选项包括了一个配置文件系统 [deployment scanner](#)、[HTTP API](#)、一个 IDE（如 Red Hat JBoss Developer Studio）和 [Maven](#)。

在部署应用程序时，您可能希望通过设置 `org.jboss.metadata.parser.validate` 系统属性为 `true` 来启用部署检验。这是通过下列方法实现的：

- 当启动服务器时。

```
$ EAP_HOME/bin/standalone.sh -
Dorg.jboss.metadata.parser.validate=true
```

- 用下列管理 CLI 命令添加它至服务器配置。

```
/system-property=org.jboss.metadata.parser.validate:add(value=true)
```

7.1. 用管理 CLI 部署应用程序

使用管理 CLI 部署应用程序的好处在于，可以使用一个命令行接口来创建并运行部署脚本。您可以使用脚本功能来配置特定应用程序的部署以及管理。您可以管理运行于一个独立服务器上的部署，也可以管理运行于管理域中的整个服务器网络。

7.1.1. 在一个独立服务器上部署应用程序

部署应用程序

在管理 CLI 里，使用 **deploy** 命令指定应用程序部署的路径。

```
deploy /path/to/test-application.war
```

成功的部署不会在管理 CLI 里产生任何输出，但服务器日志会显示部署消息。

```
WFLYSRV0027: Starting deployment of "test-application.war" (runtime-name:
"test-application.war")
WFLYUT0021: Registered web context: /test-application
WFLYSRV0010: Deployed "test-application.war" (runtime-name : "test-
application.war")
```

应用程序已成功部署。

卸载应用程序

在管理 CLI 里使用 **undeploy** 命令并指定部署名称。

- 卸载应用程序并删除部署内容。

```
undeploy test-application.war
```

- 卸载应用程序而不从仓库里删除部署内容。

```
undeploy test-application.war --keep-content
```

这与从管理控制台禁用部署相同。

成功的卸载不会在管理 CLI 里产生任何输出，但服务器日志会显示部署消息。

```
WFLYUT0022: Unregistered web context: /test-application
WFLYSRV0028: Stopped deployment test-application.war (runtime-name: test-
application.war) in 62ms
WFLYSRV0009: Undeployed "test-application.war" (runtime-name: "test-
application.war")
```

应用程序已成功卸载。

列出部署

在管理 CLI 里使用 **deploy -l** 命令来列出所有部署。

```
deploy -l
```

输出即将显示每个部署的细节，如运行时名称和是否已启用。

NAME	RUNTIME-NAME	ENABLED	STATUS
test-application.war	test-application.war	true	OK
jboss-helloworld.war	jboss-helloworld.war	true	OK

7.1.2. 在受管域里部署应用程序

部署应用程序

在管理 CLI 里，使用 **deploy** 命令并指定应用程序部署的路径。您也必须指定应用程序部署的服务器组。

- 要部署应用程序至所有服务器组。

```
deploy /path/to/test-application.war --all-server-groups
```

- 要部署应用程序至特定的服务器组。

```
deploy /path/to/test-application.war --server-groups=main-server-
group,other-server-group
```

成功的部署不会在管理 CLI 里产生任何输出，但服务器日志会显示每个受影响的服务器的部署消息。

```
[Server:server-one] WFLYSRV0027: Starting deployment of "test-
application.war" (runtime-name: "test-application.war")
[Server:server-one] WFLYUT0021: Registered web context: /test-application
[Server:server-one] WFLYSRV0010: Deployed "test-application.war" (runtime-
name : "test-application.war")
```

应用程序已成功部署至受管域里合适的服务器组。

卸载应用程序

在管理 CLI 里，使用 **undeploy** 命令并指定部署名称。您也必须指定应用程序从哪个服务器组中卸载。

- 从所有带有这个部署的服务器组中卸载这个应用程序。

```
undeploy test-application.war --all-relevant-server-groups
```

- 要从特定的服务器组卸载应用程序。请注意 **--keep-content** 参数是必需的，因为内容必须保留在具有该部署的其他服务器组的资料库里。

```
undeploy test-application.war --server-groups=other-server-group --keep-content
```

这与从管理控制台禁用部署相同。

成功的卸载不会在管理 CLI 里产生任何输出，但服务器日志会显示每个受影响的服务器的卸载消息。

```
[Server:server-one] WFLYUT0022: Unregistered web context: /test-application
[Server:server-one] WFLYSRV0028: Stopped deployment test-application.war
(runtime-name: test-application.war) in 74ms
[Server:server-one] WFLYSRV0009: Undeployed "test-application.war"
(runtime-name: "test-application.war")
```

应用程序已成功卸载。

列出部署

在管理 CLI 里使用 **deploy -l** 命令来列出所有部署。

```
deploy -l
```

输出将列出每个部署及其运行时名称。

NAME	RUNTIME-NAME
test-application.war	test-application.war
jboss-helloworld.war	jboss-helloworld.war

7.2. 使用管理控制台部署应用程序

通过管理控制台部署应用程序的好处是，可以使用易用的图形界面。您可以方便地查看部署在服务器或服务组中的应用程序，并根据需要从内容仓库中禁用或删除应用程序。

7.2.1. 在一个独立服务器上部署应用程序

您可以在管理控制台的 **Deployments** 标签页里查看和管理部署。

部署应用程序

点 **Add** 按钮，使用 **New Deployment** 引导程序部署一个应用程序。您可以通过 **上传一个部署** 或 **创建一个未被管理的部署** 来对应用程序进行部署。部署在默认情况下被启用。

- 上传部署
上传要复制到服务器的内容仓库的、并由 JBoss EAP 管理的应用程序。
- 创建不托管的部署
指定部署的位置。这个部署不会复制到服务器的内容仓库且不会被 JBoss EAP 管理。请注意，系统只支持作为非托管部署的展开式部署。

卸载应用程序

选择部署并选择 **Remove** 选项来卸载应用程序。这会卸载部署，并把它从内容仓库中删除。

禁用应用程序

选择部署并选择 **Disable** 选项来禁用应用程序。这会卸载部署，但不会从内容仓库中删除它。

替换一个应用程序

选择部署并选择 **Replace** 选项。选一个新版本的部署，这个新部署需要和原来的部署有相同的名字，点 **Finish**。这会卸载并删除原来的部署版本，然后部署新的版本。

7.2.2. 在受管域里部署应用程序

在 JBoss EAP 管理控制台的 **Deployments** 标签页中可以查看和管理部署：

- **内容仓库**
所有托管和非托管的部署都在内容仓库部分列出。在这里您可以添加和分配部署至服务器组。
- **未分配的内容**
还没分配给任何服务器组的部署在未分配的内容部分列出。在这里您可以分配部署至服务器组或删除部署。
- **服务器组**
已分配给一个或多个服务器组的部署会在服务器组部分里列出。在这里您可以直接启用部署和添加部署至服务器组里。

部署应用程序

1. 在 **Content Repository** 里点击 **Add** 按钮。
2. 通过 *uploading a deployment* 或 *creating an unmanaged deployment* 部署应用程序。
3. 按照这些提示来部署应用程序。
请注意在启用之前部署必须先分配给服务器组。

通过从 **Server Groups** 里添加部署，您可以在一个步骤里添加、分配和启用部署。

为服务器组分配一个应用程序

1. 在 **Unassigned Content** 里选择部署并点击 **Assign** 按钮。
2. 选择这个部署应该分配的一个或多个服务器组。
3. 选择是否启用在所选服务器组上的部署。

从服务器组中取消分配的应用程序

1. 从 **Server Groups** 里选择合适的服务器组。
2. 选择所需的部署并点击 **Unassign** 按钮。

通过选择 **Content Repository** 里的 **Unassign** 按钮，您也可以立即从多个服务器里取消分配。

卸载应用程序

1. 如果部署仍被分配给某个服务器组，请确保取消它的分配。
2. 在 **Content Repository** 里选择部署并点击 **Remove** 按钮。

这会卸载部署，并把它从内容仓库中删除。

禁用应用程序

1. 从 **Server Groups** 里选择合适的服务器组

1. 从 **Server Groups** 里选择合适的服务器组。

2. 选择所需的部署并点击 **Disable** 按钮。

这会卸载部署，但不会把它从内容仓库中删除。

替换一个应用程序

1. 在 **Content Repository** 里选择部署并点击 **Replace** 按钮。

2. 选择部署的新版本（名称必须和原来的部署相同），点 **Finish**。

这会卸载并删除原来的部署，然后部署新的版本。

7.3. 用部署扫描器部署应用程序

部署扫描器（deployment scanner）会监测目录来为应用程序进行部署。在默认情况下，部署扫描器会每 5 秒对 **EAP_HOME/standalone/deployments/** 目录进行扫描以检查是否有变化。marker 文件被用来指示部署的状态，并触发针对部署的操作，如卸载、重新部署。

我们推荐使用管理控制台或管理 CLI 在生产环境中部署应用程序。而部署扫描器为开发人员提供了一定的方便，用户可以在一个具有快速开发周期的环境中使用它创建并测试应用程序。另外，部署扫描器不应该和其它部署方法一起使用。

部署扫描器只有在 JBoss EAP 作为独立服务器运行时才可用。

7.3.1. 在一个独立服务器上部署应用程序

部署扫描器可以被配置为启用或禁用自动部署 XML、zipped 和 exploded 的内容。如果自动部署被禁用，您需要手动创建 marker 文件来触发部署操作。如需了解更多与可用的 marker 文件类型以及它们的作用相关的内容，请参阅 [Deployment Scanner Marker Files](#) 一节。

在默认情况下，自动部署 XML 和 zipped 内容是被启用的。如需了解为每类内容配置自动部署的内容，请参阅 [Configure the Deployment Scanner](#)。



警告

用部署扫描器进行部署的目的是便于开发人员使用，但我们不推荐在产品环境里使用这种方法。它也不应该和其他部署方法一起使用。

部署应用程序

复制内容至 deployment 文件夹。

```
$ cp /path/to/test-application.war EAP_HOME/standalone/deployments/
```

如果启用了自动部署，这个文件将被自动提取和部署，且会创建一个 **.deployed** marker 文件。如果没有启用自动部署，您需要手动添加 **.dodeploy** marker 文件来触发部署。

```
$ touch EAP_HOME/standalone/deployments/test-application.war.dodeploy
```

卸载应用程序

删除 **.deployed** marker 文件将触发卸载。

```
$ rm EAP_HOME/standalone/deployments/test-application.war.deployed
```

如果启用了自动部署，您也可以删除 **test-application.war** 文件，这将触发卸载。请注意这不适用于展开式部署。

重部署应用程序

创建 **.dodeploy** marker 文件来初始化重部署。

```
$ touch EAP_HOME/standalone/deployments/test-application.war.dodeploy
```

7.3.2. 配置部署扫描器

部署扫描器可以通过使用管理控制台或管理 CLI 进行配置。您可以配置部署扫描器的行为，如扫描的间隔时间、部署目录的位置以及自动部署的特定应用程序文件类型。您也可以完全禁用部署扫描器。

关于所有可用的部署扫描器属性的细节，请参考[部署扫描器属性](#)章节。

使用下列管理 CLI 命令来配置默认的部署扫描器。

禁用部署扫描器

```
/subsystem=deployment-scanner/scanner=default:write-attribute(name=scan-enabled,value=false)
```

这禁用了默认的部署扫描器。

修改扫描间隔

```
/subsystem=deployment-scanner/scanner=default:write-attribute(name=scan-interval,value=10000)
```

这将扫描间隔从 **5000** 毫秒（5 秒）更新为 **10000** 毫秒（10 秒）。

修改部署文件夹

```
/subsystem=deployment-scanner/scanner=default:write-attribute(name=path,value=/path/to/deployments)
```

这将部署文件夹的位置从默认的 **EAP_HOME/standalone/deployments** 修改为 **/path/to/deployments**。

path 值将作为绝对路径对待，除非指定了 **relative-to** 属性。

启用展开内容的自动部署

```
/subsystem=deployment-scanner/scanner=default:write-attribute(name=auto-deploy-exploded,value=true)
```

这启用了展开内容的自动部署（默认是禁用的）。

禁用压缩内容的自动部署

```
/subsystem=deployment-scanner/scanner=default:write-attribute(name=auto-deploy-zipped,value=false)
```

这禁用了压缩内容的自动部署（默认是启用的）。

禁用 **XML** 内容的自动部署

```
/subsystem=deployment-scanner/scanner=default:write-attribute(name=auto-deploy-xml,value=false)
```

这禁用了 XML 内容的自动部署（默认是启用的）。

7.3.3. 定义自定义部署扫描器

您可以通过使用管理 CLI，或使用管理控制台的 **Configuration** 标签页中的 **Deployment Scanners** 子系统来添加一个新的部署扫描器。这会定义一个新的目录用于部署扫描。默认的部署扫描器会监测 **EAP_HOME/standalone/deployments**。如需了解如何配置一个存在的部署扫描器的信息，请参阅 [Configure the Deployment Scanner](#)。

下面的管理 CLI 命令会添加一个新的部署扫描器，它每隔 5 秒将检查 **EAP_HOME/standalone/new_deployment_dir** 里的部署。

```
/subsystem=deployment-scanner/scanner=new-scanner:add(path=new_deployment_dir,relative-to=jboss.server.base.dir,scan-interval=5000)
```



注意

指定的目录必须已经存在，否则这个命令将运行失败。

已定义一个新的部署扫描器，它将监控指定的目录。

7.4. 用 MAVEN 部署应用程序

用 Apache Maven 部署应用程序允许您轻易地将 JBoss EAP 部署合并至现有的开发流程里。

您可以通过 [WildFly Maven Plugin](#) 来使用 Maven 在 JBoss EAP 中部署应用程序，它提供了简便的操作来为应用服务器部署或卸载应用程序。

7.4.1. 在一个独立服务器上部署应用程序

下面的说明展示了如何在一个独立的服务器上使用 Maven 部署和卸载 JBoss EAP **helloworld** quickstart。

如需了解更多与 JBoss EAP quickstarts 相关的内容，请参阅 JBoss EAP *Getting Started Guide* 中的 [Using the Quickstart Examples](#)。

部署应用程序

在 Maven **pom.xml** 文件中初始 the WildFly Maven Plugin。这应该已在 JBoss EAP quickstart **pom.xml** 文件中进行了配置。

```
<plugin>
  <groupId>org.wildfly.plugins</groupId>
```



```
<artifactId>wildfly-maven-plugin</artifactId>
<version>${version.wildfly.maven.plugin}</version>
</plugin>
```

从 **helloworld** quickstart 目录里执行下列 Maven 命令。

```
$ mvn clean install wildfly:deploy
```

在执行 Maven 命令进行部署时，终端窗口会显示下列输出，表示部署成功。

```
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 2.981 s
[INFO] Finished at: 2015-12-23T15:06:13-05:00
[INFO] Final Memory: 21M/231M
[INFO] -----
```

您也可以查看服务器日志来确认部署成功。

```
WFLYSRV0027: Starting deployment of "jboss-helloworld.war" (runtime-name:
"jboss-helloworld.war")
WFLYUT0021: Registered web context: /jboss-helloworld
WFLYSRV0010: Deployed "jboss-helloworld.war" (runtime-name : "jboss-
helloworld.war")
```

卸载应用程序

从 **helloworld** quickstart 目录里执行下列 Maven 命令。

```
$ mvn wildfly:undeploy
```

在执行 Maven 命令进行卸载时，终端窗口会显示下列输出，表示部署成功。

```
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.237 s
[INFO] Finished at: 2015-12-23T15:09:10-05:00
[INFO] Final Memory: 10M/183M
[INFO] -----
```

您也可以查看服务器日志来确认卸载成功。

```
WFLYUT0022: Unregistered web context: /jboss-helloworld
WFLYSRV0028: Stopped deployment jboss-helloworld.war (runtime-name: jboss-
helloworld.war) in 27ms
WFLYSRV0009: Undeployed "jboss-helloworld.war" (runtime-name: "jboss-
helloworld.war")
```

7.4.2. 在受管域里部署应用程序

下面的说明展示了如何在一个管理的域中使用 Maven 部署和卸载 JBoss EAP **helloworld** quickstart。

如需了解更多与 JBoss EAP quickstarts 相关的内容，请参阅 JBoss EAP *Getting Started Guide* 中的 [Using the Quickstart Examples](#)。

部署应用程序

当在受管域中部署一个应用程序时，您需要指定这个应用程序要部署的服务器组。这会在 Maven **pom.xml** 文件中配置。

以下是 **pom.xml** 中的配置，它初始 WildFly Maven Plugin 并指定 **main-server-group** 作为应用程序要部署到的服务器组。

```
<plugin>
  <groupId>org.wildfly.plugins</groupId>
  <artifactId>wildfly-maven-plugin</artifactId>
  <version>${version.wildfly.maven.plugin}</version>
  <configuration>
    <domain>
      <server-groups>
        <server-group>main-server-group</server-group>
      </server-groups>
    </domain>
  </configuration>
</plugin>
```

从 **helloworld** quickstart 目录里执行下列 Maven 命令。

```
$ mvn clean install wildfly:deploy
```

在执行 Maven 命令进行部署时，终端窗口会显示下列输出，表示部署成功。

```
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 4.005 s
[INFO] Finished at: 2016-09-02T14:36:17-04:00
[INFO] Final Memory: 21M/226M
[INFO] -----
```

您也可以查看服务器日志来确认部署成功。

```
WFLYSRV0027: Starting deployment of "jboss-helloworld.war" (runtime-name:
"jboss-helloworld.war")
WFLYUT0021: Registered web context: /jboss-helloworld
WFLYSRV0010: Deployed "jboss-helloworld.war" (runtime-name : "jboss-
helloworld.war")
```

卸载应用程序

从 **helloworld** quickstart 目录里执行下列 Maven 命令。

```
$ mvn wildfly:undeploy
```

在执行 Maven 命令进行卸载时，终端窗口会显示下列输出，表示部署成功。

```
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.750 s
[INFO] Finished at: 2016-09-02T14:45:10-04:00
[INFO] Final Memory: 10M/184M
[INFO] -----
-----
```

您也可以查看服务器日志来确认卸载成功。

```
WFLYUT0022: Unregistered web context: /jboss-helloworld
WFLYSRV0028: Stopped deployment jboss-helloworld.war (runtime-name: jboss-helloworld.war) in 106ms
WFLYSRV0009: Undeployed "jboss-helloworld.war" (runtime-name: "jboss-helloworld.war")
```

7.5. 用 HTTP API 部署应用程序

您可以通过 **curl** 命令用 HTTP API 来部署应用程序至 JBoss EAP。

部署应用程序

```
$ curl --digest -L -D - http://HOST:PORT/management --header "Content-Type: application/json" -d '{"operation": "composite", "address": [], "steps": [{"operation": "add", "address": {"deployment": "test-application.war"}, "content": [{"url": "file:/path/to/test-application.war"}]}, {"operation": "deploy", "address": {"deployment": "test-application.war"}}], "json.pretty": 1}' -u USER:PASSWORD
```

卸载应用程序

```
$ curl --digest -L -D - http://HOST:PORT/management --header "Content-Type: application/json" -d '{"operation": "composite", "address": [], "steps": [{"operation": "undeploy", "address": {"deployment": "test-application.war"}}, {"operation": "remove", "address": {"deployment": "test-application.war"}}], "json.pretty": 1}' -u USER:PASSWORD
```

关于在程序里生成 JSON 请求，请参考[红帽知识库文章](#)。

7.6. 自定义部署行为

7.6.1. 为部署内容定义自定义目录

您可以为 JBoss EAP 定义自定义位置来存储部署的内容。

为独立服务器定义自定义目录

在默认的情况下，独立服务器里部署的内容保存在 **EAP_HOME/standalone/data/content** 目录里。这个位置可以通过启动服务器时传入 **-Djboss.server.deploy.dir** 参数来修改。

```
$ EAP_HOME/bin/standalone.sh -
Djboss.server.deploy.dir=/path/to/new_deployed_content
```

选定的位置应该在 JBoss EAP 实例里是唯一的。



注意

jboss.server.deploy.dir 属性用来指定一个目录，这个目录用于存储通过管理控制台或管理 CLI 部署的内容。如果需要指定一个部署扫描器监控的部署目录，请参阅 [Configure the Deployment Scanner](#)。

为受管与定义自定义目录

在默认的情况下，受管域里部署的内容保存在 **EAP_HOME/domain/data/content** 目录里。这个位置可以通过启动域时传入 **-Djboss.domain.deployment.dir** 参数来修改。

```
$ EAP_HOME/bin/domain.sh -
Djboss.domain.deployment.dir=/path/to/new_deployed_content
```

选定的位置应该在 JBoss EAP 实例里是唯一的。

7.6.2. 控制部署的顺序

在服务器启动时，JBoss EAP 对部署顺序提供了细颗粒度的控制。您可以指定多个 EAR 文件里的应用程序的严格的部署顺序以及重启后持久化的顺序。

您可以用 **jboss-all.xml** 部署描述符文件来声明顶层部署间的依赖关系。

例如，如果 **app.ear** 依赖于 **framework.ear** 的部署，那您可以创建一个下面这样的 **app.ear/META-INF/jboss-all.xml** 文件。

```
<jboss xmlns="urn:jboss:1.0">
  <jboss-deployment-dependencies xmlns="urn:jboss:deployment-
dependencies:1.0">
    <dependency name="framework.ear" />
  </jboss-deployment-dependencies>
</jboss>
```

这确保了 **framework.ear** 在 **app.ear** 之前被部署。

7.6.3. 覆盖部署内容

使用 *部署覆盖 (deployment overlay)* 可以在不物理修改部署归档内容的情况下覆盖一个存在的部署内容。您可以使用它在运行时覆盖部署描述符、JAR 文件、类、JSP 页，以及其它文件，而不需要重新构建归档。

通过使用部署覆盖，可以在为一个有不同配置或设置的环境进行部署时进行相应的改变。例如，在应用程序生命周期的不同阶段（开发、测试、预生产、生产）进行部署时，您可以需要交换部署描述符、修改静

态 web 资源来改变应用程序的品牌设置、或根据不同的目标环境使用不同版本的 JAR 库。另外，这个功能能在一个安装需要修改配置，但因为策略或安全限制的原因而无法改变归档的情况下非常有用。

在定义一个部署覆盖时，您可以指定文件系统中的文件来替换部署归档中的文件。您还需要指定哪个部署会受到这个部署覆盖的影响。任何受影响的部署都需要被重新部署来使改变生效。

使用 **deployment-overlay add** 管理 CLI 命令来添加部署覆盖。

```
deployment-overlay add --name=new-deployment-overlay --content=WEB-INF/web.xml=/path/to/other/web.xml --deployments=test-application.war --redeploy-affected
```



注意

在受管域里，用 **--server-groups** 指定适用的服务器组或用 **--all-server-groups** 指定全部的服务器组。

创建之后，您可以在现有的覆盖里添加内容、链接覆盖至部署或删除覆盖。关于完整的用法，请执行 **deployment-overlay --help**。

参数

name

部署覆盖的名称。

content

用逗号隔开的列表，映射文件系统上的文件至它将替换的归档里的文件。条目格式是 **ARCHIVE_PATH=FILESYSTEM_PATH**。

deployments

这个覆盖将链接的用逗号隔开的部署列表。

redeploy-affected

重新部署所有受影响的部署。

7.6.4. 指定操作头

当使用管理 CLI 部署应用程序时，需要指定操作的头数据（header）来控制部署操作要如何执行。使用 **-headers** 参数可以把以下操作的头数据传递给 **deploy** 命令。

- **allow-resource-service-restart** - 是否重启要求重启以使操作修改生效的运行时服务。它默认是 **false**。
- **blocking-timeout** - 在操作被回滚前，操作被堵塞在完成操作所经历的任何阶段的最长时间（以秒为单位）。它的默认值是 **300** 秒。
- **rollback-on-runtime-failure** - 当对运行时应用改变失败时，具有持久性的配置是否需要被还原。它的默认设置是 **true**。
- **rollout** - 受管域部署的 Rollout 计划。详情请参考[使用 Rollout 计划](#)章节。

例如：

```
deploy /path/to/deployment.war --headers={allow-resource-service-
restart=true}
```

使用分号 (;) 来隔离多个操作头部。

7.6.5. 使用 Rollout 计划

关于 **Rollout** 计划

在一个受管域中，针对于域或主机一级资源的操作可能会对多个服务器产生影响。这些操作可以包括一个执行计划，它指定了操作应用到服务器的顺序，以及当操作在一些服务器上执行失败时操作是否需要被复原。如果没有指定执行计划，[默认的运行计划](#)会被使用。

以下是一个涉及到 5 个服务器组的 rollout 计划示例。操作可以按顺序在服务器组中应用 (**in-series**) 或同时应用 (**concurrent-groups**)。 [Rollout Plan Syntax](#) 详细介绍了它的语法。

```
{ "my-rollout-plan" => { "rollout-plan" => {
  "in-series" => [
    { "concurrent-groups" => {
      "group-A" => {
        "max-failure-percentage" => "20",
        "rolling-to-servers" => "true"
      },
      "group-B" => undefined
    } },
    { "server-group" => { "group-C" => {
      "rolling-to-servers" => "false",
      "max-failed-servers" => "1"
    } } },
    { "concurrent-groups" => {
      "group-D" => {
        "max-failure-percentage" => "20",
        "rolling-to-servers" => "true"
      },
      "group-E" => undefined
    } }
  ],
  "rollback-across-groups" => "true"
} } }
```

在上面的示例中，对域中的服务器应用操作分为 3 个阶段。如果任何一个服务器组的策略触发了一个回滚操作，则所有其它服务器组都会回滚。

1. 服务器组 *group-A* 和 *group-B* 会同时应用操作。这个操作会在 *group-A* 中的服务器上顺序应用，而会在 *group-B* 中的服务器上同时应用。如果 *group-A* 中有超过 20% 服务器应用操作失败，则会触发全组范围的回滚操作；如果 *group-B* 中有任何服务器应用操作失败，则会触发全组范围的回滚操作。
2. 当 *group-A* 和 *group-B* 中的所有服务器都已完成时，操作会在 *group-C* 中的服务器上应用。这些服务器会同时处理操作，如果 *group-C* 中的有一个以上的服务器应用操作失败，它将会在这个组的范围内进行回滚。
3. 当 *group-C* 中的所有服务器都完成后，服务器组 *group-D* 和 *group-E* 会同时应用操作。这个操作会在 *group-D* 中的服务器上顺序应用，而会在 *group-E* 中的服务器上同时应用。如果 *group-D* 中有超过 20% 服务器应用操作失败，则会触发全组范围的回滚操作；如果 *group-D* 中有任何服务器应用操作失败，则会触发全组范围的回滚操作。

Rollout 计划语法

您可以以下列方式之一指定 Rollout 计划。

- 在 **deploy** 命令操作头部定义 Rollout 计划。详情请参考[用 Rollout 计划进行部署](#)。
- 使用 **rollout-plan** 命令保存 rollout 计划，然后在 **deploy** 命令的操作头中参考这个计划名。如需更详细的相关信息，请参阅[Deploy Using a Stored Rollout Plan](#)。

虽然每个方法都具有不同的初始命令，两者都使用 **rollout** 操作头部来定义 Rollout 计划。它使用了下列语法。

```
rollout (id=PLAN_NAME | SERVER_GROUP_LIST) [rollback-across-groups]
```

- **PLAN_NAME** 是用 **rollout-plan** 命令保存的 Rollout 计划的名称。
- **SERVER_GROUP_LIST** 是服务器组列表。使用一个逗号 (,) 分隔多个服务器组意味着，操作应该按顺序在每个服务器组上运行。使用 ^ 分隔意味着，操作需要在每个服务器组上同时执行。
 - 对于每个服务器组，请在参数里设置下列策略。多个策略间请用逗号隔开。
 - **rolling-to-servers** : 一个布尔值。如果设置为 **true**，操作会按顺序在组中的每个服务器上执行；如果设置为 **false** 或没有指定，操作会在组中的每个服务器上同时执行。
 - **max-failed-servers** : 一个整数值。当在组中进行操作失败的服务器数量达到这个数值时，组中的所有服务器都需要被复原。它的默认值是 **0**，这代表当操作在任何一个服务器上失败时，在组中都会触发一个回滚操作。
 - **max-failure-percentage** : **0** 和 **100** 间的一个整数。当在组中进行操作失败的服务器数量的百分比达到这个数值时，组中的所有服务器都需要被复原。它的默认值是 **0**，这代表当操作在任何一个服务器上失败时，在组中都会触发一个回滚操作。



注意

如果 **max-failed-servers** 和 **max-failure-percentage** 都被设置为非零值，那么将优先使用 **max-failure-percentage**。

- **rollback-across-groups** : 一个布尔值。它指定了，当需要在一个服务器组中的所有服务器上执行回滚操作时，是否会触发在所有服务器组中的回滚操作。它的默认值是 **false**。

用 Rollout 计划进行部署

您可以通过在 **deploy** 命令的 **headers** 参数中使用 **rollout** 设置来为它直接提供一个 rollout 计划的详细信息。如需了解更多相关信息，请参阅[Rollout Plan Syntax](#)。

下列管理 CLI 命令部署了应用程序至 **main-server-group** 服务器组，它使用了为序列部署指定 **rolling-to-servers=true** 的部署计划。

```
deploy /path/to/test-application.war --server-groups=main-server-group --headers={rollout main-server-group(rolling-to-servers=true)}
```

用 Rollout 计划进行卸载

rollout 计划可能会非常负载，您可以选择保存一个 rollout 计划的详细信息。当您需要使用它时，可以通过计划的名称指定它，而不需要提供完整的 rollout 计划详情。

1. 使用 **rollout-plan** 管理 CLI 命令来保存 Rollout 计划。关于格式的详情，请参考 [Rollout 计划语法](#)。

```
rollout-plan add --name=my-rollout-plan --content={rollout main-
server-group(rolling-to-servers=false,max-failed-servers=1),other-
server-group(rolling-to-servers=true,max-failure-percentage=20)
rollback-across-groups=true}
```

这创建了下列部署计划。

```
"rollout-plan" => {
  "in-series" => [
    {"server-group" => {"main-server-group" => {
      "rolling-to-servers" => false,
      "max-failed-servers" => 1
    }}},
    {"server-group" => {"other-server-group" => {
      "rolling-to-servers" => true,
      "max-failure-percentage" => 20
    }}}
  ],
  "rollback-across-groups" => true
}
```

2. 在部署应用程序时指定存储的 Rollout 计划名称。

下列管理 CLI 命令用存储的 **my-rollout-plan** Rollout 计划部署了一个应用程序至所有的服务组。

```
deploy /path/to/test-application.war --all-server-groups --headers=
{rollout id=my-rollout-plan}
```

删除一个保存的 **rollout** 计划

您可以在 **rollout-plan** 管理 CLI 命令中使用 rollout 计划的名称来删除它。

```
rollout-plan remove --name=my-rollout-plan
```

默认的 **Rollout** 计划

所有影响多个服务器的操作都将用 Rollout 计划来执行。如果在操作请求里没有指定 Rollout 计划，将生成一个默认的 Rollout 计划。这个计划将具有下列特点。

- 只有单个高级别的阶段。这个操作影响的所有服务器组都将并发地应用操作。
- 在每个服务器组里，这个操作将并行地应用于所有服务器。
- 在服务器组里任何一个服务器上失败都将导致整个组的回滚。
- 任何服务器组的失败将导致所有其他服务器组的回滚。

第 8 章 域管理

本节讨论受管域操作模式专有的概念和操作。

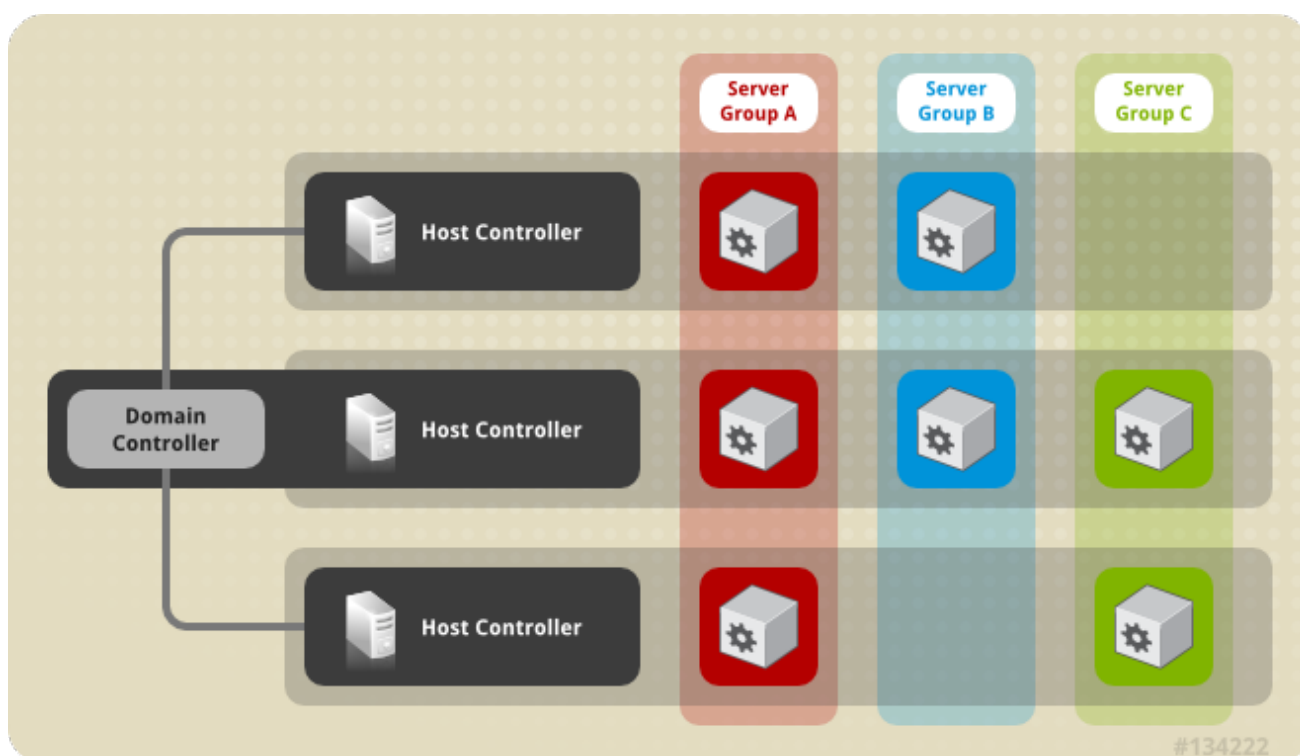
关于保护受管域的详情，请参考 *如何配置服务器安全性* 里的 [保护受管域](#) 章节。

8.1. 关于受管域

受管域（managed domain）操作模式允许从单个控制点管理多个 JBoss EAP 实例。

集中管理的 JBoss EAP 服务器集合被称为域成员。域里所有的 JBoss EAP 实例都共享一个公用的管理策略。

域由一个域控制器、一个或多个主机控制器，以及每个主机的零或多个服务器组组成。



域控制器是控制域的一个中心点。它确保了每个服务器都根据域的管理策略进行配置。域控制器同时也是一个主机控制器。

主机控制器是一个物理主机或一个虚拟主机，它用来和域控制器进行交流，从而控制在主机上运行的应用程序服务器实例的生命周期，并协助域控制器对它们进行管理。每个主机可以包括多个服务器组。

服务器组就是一组安装了JBoss EAP、并被它管理和配置的**服务器**实例。域控制器管理在服务器组中部署的应用程序的配置，因此，服务器组中的每个服务器都会共享相同的配置和部署。

主机控制器和特定的物理（或虚拟）主机相绑定。您可以在一个硬件上运行多个主机控制器，但需要使用不同的配置以确保它们的端口和其它资源没有冲突。域控制器、主机控制器和多个服务器可以在相同物理系统中的同一个 JBoss EAP 实例中运行。

8.1.1. 关于域控制器

域控制器是充当域的中央管理点的 JBoss EAP 服务器实例。主机控制器实例可以配置为域控制器。

域控制器的主要职责是：

- 维护域的集中管理策略。
- 确保所有的主机控制器意识到它当前的内容。
- 协助主机控制器确保所有运行的 JBoss EAP 服务器实例按照这个策略进行配置。

在默认情况下，集中管理策略保存在 `EAP_HOME/domain/configuration/domain.xml` 文件里。设置为域控制器的主机控制器的这个目录里要求存在这个文件。

`domain.xml` 文件包括了可以被域中的服务器使用的配置集配置。一个配置集包括了这个配置集中的不同子系统的详细设置。域配置还包括了套接字组的定义和服务器组的定义。



注意

JBoss EAP 7 域控制器可以管理 JBoss EAP 6 主机和服务器，以及运行 JBoss EAP 6.2 或更高版本的主机和服务器。

关于更多的信息，请参考[启动受管域](#)和[配置域控制器](#)章节。

8.1.2. 关于主机控制器

主机控制器的主要职责是管理服务器。它负责委派域管理任务，并负责启动和停止运行在主机上的应用服务器进程。

它和域控制器进行交流，来帮助管理服务器和域控制器间的通讯。一个域的多个主机控制器可以只和一个域控制器进行交流。在一个独立域模式中运行的所有主机控制器和服务器实例会有一个独立的域控制器，它们需要属于相同的域。

在默认情况下，每个主机控制器都会从主机文件系统中的解包的 JBoss EAP 安装文件中的 `EAP_HOME/domain/configuration/host.xml` 文件中读取它的配置信息。`host.xml` 文件包括了以下与特定主机相关的配置信息：

- 从这个安装中运行的服务器实例名。
- 针对于本地物理安装的配置。例如，在 `domain.xml` 中声明的命名接口定义可以被映射到 `host.xml` 中的一个实际机器的 IP 地址。`domain.xml` 中的抽象路径名会被映射到 `host.xml` 中的实际文件系统路径。
- 任何下列的配置：
 - 主机控制器如何联系域控制器来注册自己并访问域配置。
 - 如何找到并联系远程域控制器。
 - 主机控制器是否充当域控制器。

关于更多的信息，请参考[启动受管域](#)和[配置主机控制器](#)章节。

8.1.3. 关于进程控制器

进程控制器就是一个小型轻量级进程，它的作用是产生主机控制器进程并监控它的生命周期。如果主机控制器崩溃，进程控制器会对它进行重启。进程控制器还会根据主机控制器的指示启动服务器进程，但在崩溃时不会自动重启服务器进程。

进程控制器的日志被记录在 `EAP_HOME/domain/log/process-controller.log` 文件中。您可以在 `EAP_HOME/bin/domain.conf` 文件中使用 `PROCESS_CONTROLLER_JAVA_OPTS` 变量为进程控制器设置 JVM 选项。

8.1.4. 关于服务器组

服务器组就是一个作为一个整体被管理和配置的一组服务器实例集合。在一个受管域中，每个应用服务器实例都属于一个服务器组（即使它是唯一成员）。一个组中的服务器共享相同的配置集配置和部署内容。

域控制器和主机控制器在其域的每个服务器组的所有服务器实例上，强制标准配置的实施。

域由多个服务器组组成。不同的服务器组可以用不同的配置集和部署来进行配置。例如，域可以被配置为，使用不同的服务器层来提供不同服务。

不同的服务器组也可以有相同的配置集和部署。这样，就可以在一个服务器组中对应用程序进行升级，然后在第二个服务器组中进行更新，而不需要完全停止服务。

关于更多的信息，请参考[配置服务器组](#)章节。

8.1.5. 关于服务器

服务器代表一个应用服务器实例。在受管域里，所有的服务器实例都是服务器组的成员。主机控制器在自己的 JVM 进程里启动每个服务器实例。

关于更多的信息，请参考[配置服务器](#)章节。

8.2. 管理域配置

JBoss EAP 提供可扩充的管理接口来支持小型和大型受管域。

管理控制台

JBoss EAP 管理控制台提供了域的图形视图，使您可以容易地管理主机、服务器、部署和配置集。

配置

在 **Configuration** 标签页里，您可以为域里使用的每个配置集配置子系统。域里不同的服务器组可能根据所需的功能使用不同的配置集。

一旦您选择了所要的配置集，所有可用的子系统将被列出。关于配置集的更多信息，请参考[管理 JBoss EAP 配置集](#)。

运行时

在 **Runtime** 标签页里，您可以管理服务器和服务器组以及主机配置。您可以按主机或服务器组来浏览域。

在 **Hosts** 标签页里，您可以配置主机属性和 JVM 设置，添加并配置服务器。

在 **Server Groups** 标签页里，您可以添加新的服务器组、配置属性和 JVM 设置以及为该服务器组添加和配置服务器。您可以对所选服务器组里的所有服务器执行启动、停止、挂起和重载等操作。

通过 **Hosts** 或 **Server Groups**，可以添加新的服务器，配置服务器属性和 JVM 设置。您可以执行启动、停止、挂起、重新加载所选服务器等操作。您还可以查看运行时信息，如 JVM 的使用情况、服务器日志信息、与子系统相关的信息。

部署

在 **Deployments** 标签页里，您可以为服务器组添加和分配部署。您可以查看内容仓库里的所有部署，查看未分配的部署，或者查看分配给特定服务器组的部署。

关于使用管理控制台来部署应用程序的更多信息，请参考[在受管域里部署应用程序](#)。

管理 CLI

JBoss EAP 管理 CLI 提供了命令行接口来管理主机、服务器、部署和配置集。

选择了合适的配置集后，您可以访问子系统配置。

```
/profile=PROFILE_NAME/subsystem=SUBSYSTEM_NAME:read-
resource(recursive=true)
```

注意

本指南里的说明和示例都包含作为独立服务器运行时适用于子系统配置的管理 CLI 命令，例如：

```
/subsystem=datasources/data-source=ExampleDS:read-resource
```

要调整这些管理 CLI 命令以用于受管域，您必须指定合适的配置集进行配置，例如：

```
/profile=default/subsystem=datasources/data-
source=ExampleDS:read-resource
```

在指定了合适的主机后，您可以配置主机设置并在该主机的服务器上执行操作。

```
/host=HOST_NAME/server=SERVER_NAME:read-resource
```

在指定合适的主机后，您可以配置该主机的服务器。

```
/host=HOST_NAME/server-config=SERVER_NAME:write-
attribute(name=ATTRIBUTE_NAME,value=VALUE)
```

在指定合适的服务器组后，您可以配置服务器组设置并在所选的服务器组上的所有服务器上执行操作。

```
/server-group=SERVER_GROUP_NAME:read-resource
```

您可以通过使用 **deploy** 管理 CLI 命令并指定服务器组属性来在一个受管域中部署应用程序。如需了解更多信息，请参阅[在受管域里部署应用程序](#)

8.3. 启动受管域

8.3.1. 启动受管域

使用 JBoss EAP 提供的 **domain.sh** 或 **domain.bat** 脚本可以启动域和主机控制器。如需了解这些启动脚本使用的完整参数列表，请使用 **--help** 参数，或参阅[服务器运行时参数](#)一节。

域控制器必须在域里任何服务器组的从服务器之前启动。首先启动域控制器，然后启动每个关联的主机控制器。

启动域控制器

使用为特定域控制器预配置的 **host-master.xml** 配置文件启动域控制器。

```
$ EAP_HOME/bin/domain.sh --host-config=host-master.xml
```

根据域的具体设置，您将需要进行额外的配置以允许主机控制器进行连接。请参考下面的域设置示例：

- [在单台主机上设立受管域](#)
- [在两台主机上设立受管域](#)

启动主机控制器

使用为从主机控制器预配置的 **host-slave.xml** 配置文件启动主机控制器。

```
$ EAP_HOME/bin/domain.sh --host-config=host-slave.xml
```

根据您的域设置，您将需要进行额外的配置来连接域控制器并保证不冲突。请参考下面的域设置示例：

- [在单台主机上设立受管域](#)
- [在两台主机上设立受管域](#)

8.3.2. 配置域控制器

将主机配置为域控制器

如果主机在其 **<domain-controller>** 声明里包含 **<local/>** 元素，那它就被指定为域控制器。

```
<domain-controller>
  <local/>
</domain-controller>
```

作为域控制器的主机需要提供一个管理接口，在这个域中的其它主机需要可以访问到这个接口。虽然这个接口不需要是一个 HTTP(S) 管理接口，但我们推荐使用它来访问管理控制台。

```
<management-interfaces>
  <native-interface security-realm="ManagementRealm">
    <socket interface="management"
port="${jboss.management.native.port:9999}"/>
  </native-interface>
  <http-interface security-realm="ManagementRealm" http-upgrade-
enabled="true">
    <socket interface="management"
port="${jboss.management.http.port:9990}"/>
  </http-interface>
</management-interfaces>
```

EAP_HOME/domain/configuration/host-master.xml 文件已用这些设置进行预配置以充当域控制器。

8.3.3. 配置主机控制器

连接域控制器

主机控制器需要可以连接到域控制台，从而在域中注册自己。这是通过配置中的 **<domain-controller>** 项进行配置的。

```
<domain-controller>
  <remote security-realm="ManagementRealm">
```

```

    <discovery-options>
      <static-discovery name="primary"
protocol="${jboss.domain.master.protocol:remote}"
host="${jboss.domain.master.address}"
port="${jboss.domain.master.port:9999}"/>
    </discovery-options>
  </remote>
</domain-controller>

```

EAP_HOME/domain/configuration/host-slave.xml 文件已被预先进行了这些配置，从而使它可以连接到域控制器。在启动主机控制器时，需要提供 **jboss.domain.master.address** 属性。

```

$ EAP_HOME/bin/domain.sh --host-config=host-slave.xml -
Djboss.domain.master.address=IP_ADDRESS

```

关于域控制器发现的详情，请参考[域控制器的发现和故障转移](#)章节。

根据域设置，您可能需要提供一个身份验证机制来使域控制器可以验证主机控制器。如需了解使用一个密码的值产生一个管理用户并使用这个值更新主机控制器配置的信息，请参阅[在两台机器上设立受管域](#)。

8.3.3.1. 配置主机的名称

运行在一个受管域中的每个主机都需要有一个唯一的名称。为了简化管理并在多个主机上使用相同的配置文件，服务器会使用以下方法决定主机名。

1. 如果设置，则是 **host.xml** 配置文件里的主机元素名称属性。
2. **jboss.host.name** 系统属性的值。
3. **jboss.qualified.host.name** 系统属性中最后一个 . 后面的值。如果没有 .，则使用这个值。
4. 对于基于 POSIX 的操作系统，**HOSTNAME** 环境变量中 . 后面的值；对于 Microsoft Windows 系统，**COMPUTERNAME** 环境变量中 . 后面的值。如果没有 .，使用整个值。

主机控制器的名称在相关的 **host.xml** 配置文件顶部的 **host** 元素里进行配置，例如：

```

<host xmlns="urn:jboss:domain:4.0" name="host1">

```

用管理 CLI 通过下列过程来更新主机名。

1. 启动 JBoss EAP 主机控制器。

```

$ EAP_HOME/bin/domain.sh --host-config=host-slave.xml

```

2. 启动管理 CLI，连接到域控制器。

```

$ EAP_HOME/bin/jboss-cli.sh --connect --
controller=DOMAIN_CONTROLLER_IP_ADDRESS

```

3. 使用下列命令来设置新的主机名。

```

/host=EXISTING_HOST_NAME:write-
attribute(name=name,value=NEW_HOST_NAME)

```


这修改了 `host-slave.xml` 文件里的主机名属性：

```
<host name="NEW_HOST_NAME" xmlns="urn:jboss:domain:4.0">
```

4. 重载主机控制器使修改生效。

```
reload --host=EXISTING_HOST_NAME
```

如果主机控制器没有在配置文件里设置名称，您也可以在运行时传入主机名。

```
$ EAP_HOME/bin/domain.sh --host-config=host-slave.xml -
Djboss.host.name=HOST_NAME
```

8.3.4. 域控制器的发现和故障转移

当设置一个受管域时，每个主机控制器都需要被配置为需要和域控制器进行联系。在 JBoss EAP 中，每个主机控制器都可以配置多个选项来查找域控制器。主机控制器会尝试使用每个选项，直到可以成功联系到域控制器。

因此，主机控制器可以预配置一个备份域控制器的联系信息。当主域控制器出现问题时，一个备份域控制器就可以被升级为主域控制器，从而使主机控制器可以自动切换到新的域控制器。

下面是如何用多个选项配置主机控制器来寻找域控制器的例子。

示例：有多个域控制器选项的主机控制器

```
<domain-controller>
  <remote security-realm="ManagementRealm">
    <discovery-options>
      <static-discovery name="primary"
protocol="${jboss.domain.master.protocol:remote}" host="172.16.81.100"
port="${jboss.domain.master.port:9999}"/>
      <static-discovery name="backup"
protocol="${jboss.domain.master.protocol:remote}" host="172.16.81.101"
port="${jboss.domain.master.port:9999}"/>
    </discovery-options>
  </remote>
</domain-controller>
```

静态发现选项包含下列必需的属性：

name

这个域控制器发现选项的名称。

host

远程域控制器的主机名。

port

远程域控制器的端口。

在上面的例子里，第一个发现选项是我们期望可以成功的选项。第二个选项可以用在失效切换的场景里。

如果主域控制器出现问题，以 `--backup` 选项启动的主机控制器可以提升为域控制器。



注意

使用 **--backup** 选项启动一个主机控制器会在本地保存一个域配置信息。当主机控制器被预配置为作为一个域控制器使用时，这个配置会被使用。

提升主机控制器为域控制器

1. 确保已停止原来的域控制器。
2. 使用管理 CLI 命令来连接成为新的域控制器的主机控制器。
3. 执行下列命令配置主机控制器成为新的域控制器。

```
/host=HOST_NAME:write-local-domain-controller
```

4. 执行下列命令来重载主机控制器。

```
reload --host=HOST_NAME
```

主机控制器现在成为了域控制器。

8.4. 管理服务器

8.4.1. 配置服务器组

下面是一个服务器组定义的示例：

```
<server-group name="main-server-group" profile="full">
  <jvm name="default">
    <heap size="64m" max-size="512m"/>
  </jvm>
  <socket-binding-group ref="full-sockets"/>
  <deployments>
    <deployment name="test-application.war" runtime-name="test-
application.war"/>
    <deployment name="jboss-helloworld.war" runtime-name="jboss-
helloworld.war" enabled="false"/>
  </deployments>
</server-group>
```

服务器组可以用管理 CLI 或管理控制台的 **Runtime** 标签页进行配置。

添加服务器组

您可以使用下列管理 CLI 命令来添加服务器组。

```
/server-group=SERVER_GROUP_NAME:add(profile=PROFILE_NAME,socket-binding-
group=SOCKET_BINDING_GROUP_NAME)
```

更新服务器组

您可以使用下列管理 CLI 命令来更新服务器组属性。

```
/server-group=SERVER_GROUP_NAME:write-
attribute(name=ATTRIBUTE_NAME,value=VALUE)
```


删除服务器组

您可以使用下列管理 CLI 命令来删除服务器组。

```
/server-group=SERVER_GROUP_NAME:remove
```

服务器组属性

服务器组要求下列属性：

- **name**：服务器组的名称。
- **profile**：服务器配置集的名称。
- **socket-binding-group**：用于组里的服务器的默认套接字绑定组。您可以对每个服务器进行覆盖。

服务器组包含下列可选属性：

- **management-subsystem-endpoint**：当把它设置为 **true** 时，属于这个服务器组的服务器会使用它们的 **remoting** 子系统的端点来连接回主机控制器（这需要有 **remoting** 子系统）。
- **socket-binding-default-interface**：这个服务器的套接字组的默认接口。
- **socket-binding-port-offset**：添加至套接字绑定组指定的端口的偏移量。
- **deployments**：部署在组里的服务器上的部署内容。
- **jvm**：组中所有服务器的默认 JVM 设置。主机控制器会把这些设置和 **host.xml** 所提供的其它配置进行合并，组成启动服务器的 JVM 时使用的配置。
- **deployment-overlays**：定义的部署覆盖和服务器组里的部署间的连接。
- **system-properties**：对组里服务器设置的系统属性。

8.4.2. 配置服务器

定义三个服务器的默认 **host.xml** 配置：

```
<servers>
  <server name="server-one" group="main-server-group">
  </server>
  <server name="server-two" group="main-server-group" auto-start="true">
    <socket-bindings port-offset="150"/>
  </server>
  <server name="server-three" group="other-server-group" auto-
start="false">
    <socket-bindings port-offset="250"/>
  </server>
</servers>
```

一个名为 **server-one** 的服务器实例和 **main-server-group** 相关联，并继承由那个服务器组所提供的子系统配置和套接字绑定。一个名为 **server-two** 的服务器实例也会和 **main-server-group** 相关联，但同时会定义一个套接字绑定 **port-offset** 的值，从而不会和 **server-one** 使用的端口值相冲突。一个名为 **server-three** 的服务器实例会和 **other-server-group** 相关联，并使用那个组的配置。它同时会定义一个 **port-offset** 值，并把 **auto-start** 设置为 **false**，从而在主机控制器启动时，这个服务器不会启动。

服务器可以用管理 CLI 或管理控制台的 **Runtime** 标签页进行配置。

添加服务器

您可以使用下列管理 CLI 命令来添加服务器。

```
/host=HOST_NAME/server-config=SERVER_NAME:add(group=SERVER_GROUP_NAME)
```

更新服务器

您可以使用下列管理 CLI 命令来更新服务器属性。

```
/host=HOST_NAME/server-config=SERVER_NAME:write-attribute(name=ATTRIBUTE_NAME,value=VALUE)
```

删除服务器

您可以使用下列管理 CLI 命令来删除服务器。

```
/host=HOST_NAME/server-config=SERVER_NAME:remove
```

服务器属性

服务器要求下列属性：

- **name**：服务器的名称。
- **group**：域模型里的服务器组的名称。

服务器包含下列可选属性：

- **auto-start**：在主机服务器启动时这个服务器是否应该启动。
- **socket-binding-group**：这个服务器所属的套接字绑定组。
- **socket-binding-port-offset**：添加至这个服务器的套接字绑定组指定的端口值的偏移量。
- **update-auto-start-with-server-status**：用服务器的状态更新 **auto-start** 属性。
- **interface**：这个服务器上可用的网络接口的全限定名称列表。
- **jvm**：这个服务器的 JVM 设置。如果未声明，这个设置将从父服务器组或主机继承。
- **path**：文件系统路径的列表。
- **system-property**：在这个服务器上设置的系统属性列表。

8.4.3. 启动和停止服务器

您可以进入管理控制台的 **Runtime** 标签页并选择合适的主机和服务器组来执行服务器操作，如启动、停止和重载。

请参考下列用管理 CLI 命令执行的操作。

启动服务器

您可以在特定的主机上启动单个服务器。

```
/host=HOST_NAME/server-config=SERVER_NAME:start
```

您也可以启动指定的服务器组里的所有服务器。

```
/server-group=SERVER_GROUP_NAME:start-servers
```

停止服务器

您可以停止特定主机上的单个服务器。

```
/host=HOST_NAME/server-config=SERVER_NAME:stop
```

您也可以停止指定的服务器组里的所有服务器。

```
/server-group=SERVER_GROUP_NAME:stop-servers
```

重载服务器

您可以重载特定主机上的单个服务器。

```
/host=HOST_NAME/server-config=SERVER_NAME:reload
```

您也可以重载指定的服务器组里的所有服务器。

```
/server-group=SERVER_GROUP_NAME:reload-servers
```

8.5. 管理域设置

8.5.1. 在单台机器上设立受管域

您可以用 `jboss.domain.base.dir` 属性在单台机器上运行多个主机控制器。

1. 复制域控制器的 `EAP_HOME/domain` 目录。

```
$ cp -r EAP_HOME/domain /path/to/domain1
```

2. 复制主机控制器的 `EAP_HOME/domain` 目录。

```
$ cp -r EAP_HOME/domain /path/to/host1
```

3. 用 `/path/to/domain1` 启动域控制器。

```
$ EAP_HOME/bin/domain.sh --host-config=host-master.xml -  
Djboss.domain.base.dir=/path/to/domain1
```

4. 用 `/path/to/host1` 启动主机控制器。

```
$ EAP_HOME/bin/domain.sh --host-config=host-slave.xml -  
Djboss.domain.base.dir=/path/to/host1 -  
Djboss.domain.master.address=IP_ADDRESS -  
Djboss.management.native.port=PORT
```



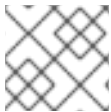
注意

在启动主机控制器时，您必须用 `jboss.domain.master.address` 属性指定域控制器的地址。

另外，因此这个主机控制器和域控制器运行在同一个机器上，所以需要修改管理接口使它不会和域控制器的管理接口相冲突。这个命令会设置 `jboss.management.native.port` 属性。

使用这种方法启动的所有实例都会共享基础安装路径（例如，`EAP_HOME/modules/`）中的其它资源，但域配置是由 `jboss.domain.base.dir` 所指定的目录提供的。

8.5.2. 在两台机器上设立受管域



注意

要运行这个示例，您可能需要配置您的防火墙。

您可以在两台机器上创建受管域，其中一台是域控制器而另外一台是主机。详情请参考[关于域控制器](#)。

- **IP1** = 域控制器（机器 1）的 IP 地址
- **IP2** = 主机（机器 2）的 IP 地址

在两台机器上创建受管域

1. 在机器 1 上

- 添加一个管理域用户以让主机可以被域控制器验证。
使用 `add-user.sh` 脚本为主机控制器（`HOST_NAME`）添加管理用户。为最后一个提示回答 **yes**，并记录下系统提供的保密值（`<secret value="SECRET_VALUE" />`）。这个保密值会用于主机控制器的配置。
- 启动域控制器。
指定为域控制器专门预配置的 `host-master.xml` 配置文件。而且设置 `jboss.bind.address.management` 属性使域控制器为其他机器可见。

```
$ EAP_HOME/bin/domain.sh --host-config=host-master.xml -
Djboss.bind.address.management=IP1
```

2. 在机器 2 上

- 使用用户凭证更新主机的配置。
编辑 `EAP_HOME/domain/configuration/host-slave.xml` 并设置主机名（`HOST_NAME`）和秘密值（`SECRET_VALUE`）。

```
<host xmlns="urn:jboss:domain:1.6" name="HOST_NAME">
  <management>
    <security-realms>
      <security-realm name="ManagementRealm">
        <server-identities>
          <secret value="SECRET_VALUE" />
        </server-identities>
      </security-realm>
    </security-realms>
  </management>
</host>
```

■

b. 启动主机控制器。

指定为从主机控制器专门预配置的 **host-slave.xml** 配置文件。而且设置 **jboss.domain.master.address** 属性以连接域控制器并设置 **jboss.bind.address** 属性来指定主机控制器绑定地址。

```
$ EAP_HOME/bin/domain.sh --host-config=host-slave.xml -
Djboss.domain.master.address=IP1 -Djboss.bind.address=IP2
```

现在，您可以在启动时用 **--controller** 参数指定域控制器地址。这样，就可以通过管理 CLI 对域进行管理。

```
$ EAP_HOME/bin/jboss-cli.sh --connect --controller=IP1
```

或者您可以通过管理控制台管理域：<http://IP1:9990>。

8.6. 管理 JBOSS EAP 配置集

8.6.1. 关于配置集

JBoss EAP 使用 *配置集 (Profile)* 来设置服务器可用的子系统。配置集由可用子系统的集合以及每个子系统的专有配置组成。带有大量子系统的配置集可以让服务器具有强大的功能集合；而带有少量专有子系统的配置集则使得服务器功能较少，但消耗的资源也较少。

JBoss EAP comes with four predefined profiles that should satisfy most use cases:

default

包括常用的子系统，如

logging、**security**、**datasources**、**infinispan**、**webservices**、**ee**、**ejb3**、**transactions** 等。

ha

包含 *default* 配置集里提供的子系统，以及 **jgroups** 和 **modcluster** 子系统来提供高可用性。

full

包含 *default* 配置集里提供的子系统，以及 **messaging-activemq** 和 **iiop-openjdk** 子系统。

full-ha

包含 *full* 配置集里提供的子系统，以及 **jgroups** 和 **modcluster** 子系统来提供高可用性。



注意

JBoss EAP 提供了通过从现有配置集里删除子系统来禁用扩展或手动卸载驱动和其他服务的能力。然而，多数情况下这是没有必要的。既然 JBoss EAP 是在有需要时才动态地加载子系统，如果服务器或应用程序从未使用某个子系统，它就不会被加载。

当现有的配置集不能提供必要的功能时，JBoss EAP 也提供了定义自定义配置集的能力。

8.6.2. 克隆配置集

JBoss EAP 允许您通过克隆现有的配置集在受管域里创建新的配置集。这将创建原始配置集的配置和子系统的备份。

配置集可以用管理 CLI 或 **clone** 操作进行克隆。

```
/profile=full-ha:clone(to-profile=cloned-profile)
```

您也可以在管理控制台里选择所需的配置集并点击 **Clone** 来克隆配置集。

8.6.3. 创建分层的配置集

在受管域里，您可以创建分层的配置集。这允许您创建具有其他配置可以继承的公用扩展的基础配置集。

受管域在 **domain.xml** 里定义了几个配置集。如果多个配置集使用了特定子系统的相同配置，您可以只在一个地方而不是在不同的配置集里进行配置。这可用于不能被覆盖的父配置集里。

通过管理 CLI 的 **list-add** 操作，配置集可以包含多层的配置集。

```
/profile=new-profile:list-add(name=includes, value=PROFILE_NAME)
```

第 9 章 配置 JVM 设置

Java Virtual Machine (JVM) 设置对于独立服务器和受管域里的 JBoss EAP 服务器是不同的。

对于 JBoss EAP 独立服务器实例，服务器启动进程在启动时将 JVM 设置传入 JBoss EAP 服务器。在启动 JBoss EAP 前，您可以从命令行声明或者使用管理控制台里的 **System Properties** 屏幕。

在受管域里，JVM 设置可以在 `host.xml` 和 `domain.xml` 配置文件里进行声明，而且在主机、服务器组和服务器级别进行配置。



注意

系统属性必须在 **JAVA_OPTS** 里进行配置以在启动时为 JBoss EAP 模块（如 logging manager）所用。

9.1. 配置独立服务器的 JVM 设置

独立的 JBoss EAP 服务器实例的 JVM 设置可以在启动服务器前设置 **JAVA_OPTS** 环境变量进行声明。

下面是在 Linux 上设置 **JAVA_OPTS** 环境变量的例子。

```
$ export JAVA_OPTS="-Xmx1024M"
```

在 Microsoft Windows 环境里可以使用相同的设置：

```
set JAVA_OPTS="Xmx1024M"
```

或者，JVM 设置可以添加到 **EAP_HOME/bin** 里的 `standalone.conf` 文件，它包含了传递给 JVM 的选项示例。



警告

设置 **JAVA_OPTS** 环境变量将覆盖 `standalone.conf` 的默认值，这可能导致 JBoss EAP 的启动问题。

9.2. 为受管域配置 JVM 设置

在 JBoss EAP 受管域里，您可以在多个级别定义 JVM 设置。您可以在特定主机上定义自定义 JVM 设置，然后应用这些设置到服务器组，或者单独的服务器实例。

在默认情况下，服务器组和单独的服务器将从父服务器继承 JVM 设置，但您可以选择在每个级别都覆盖 JVM 设置。



注意

`domain.conf` 里的 JVM 设置应用到 JBoss EAP 主机控制器而不是该主机控制器控制的单独 JBoss EAP 服务器实例的 Java 进程。

9.2.1. 在主机控制器上定义 JVM 设置

您可以在主机控制器上定义 JVM 设置，然后应用这些设置到服务器组或单独的服务器。JBoss EAP 附带一个 **default** JVM 设置，但下面的管理 CLI 命令演示了创建名为 **production_jvm** 的带有一些自定义 JVM 设置和选项的新的 JVM 设置。

```
/host=HOST_NAME/jvm=production_jvm:add(
    heap-size=2048m,
    max-heap-size=2048m,
    max-permgen-size=512m,
    stack-size=1024k,
    jvm-options=["-XX:-UseParallelGC"]
)
```

您也可以在管理控制台里创建和编辑 JVM 设置，选择 **Runtime** 标签页，然后选择 **Hosts** 并点击要编辑的主机上的 **JVM**。

这些设置存储在 **host.xml** 里的 **<jvm>** 标签下。

9.2.2. 应用 JVM 设置到服务器组

当创建服务器组时，您可以指定组里所有服务器将使用的 JVM 配置。下面的管理 CLI 命令演示了创建名为 **groupA** 的服务器组，它使用[之前例子](#)里展示的 **production_jvm** JVM 设置。

```
/server-group=groupA:add(profile=default, socket-binding-group=standard-sockets)
/server-group=groupA/jvm=production_jvm:add()
```

服务器组里的所有服务器将从 **production_jvm** 继承 JVM 设置。

您也可以覆盖服务器组级别的专有 JVM 设置。例如，要设置不同的堆大小，您可以使用下列命令：

```
/server-group=groupA/jvm=production_jvm:write-attribute(name=heap-size,value="1024m")
```

应用上述命令后，服务器组 **groupA** 将从 **production_jvm** 继承 JVM 设置，除了堆大小被覆盖为 **1024m**。

您也可以在管理控制台里编辑服务器组的 JVM 设置，选择 **Runtime** 标签页，然后选择 **Server Groups** 并点击 **View** 来编辑服务器组。

这些服务器组的设置保存在 **domain.xml** 里。

9.2.3. 应用 JVM 设置到单独的服务器

在默认情况下，单独的服务器实例将从所属的服务器组继承 JVM 设置。然而，您可以选择用主机控制器的其他完整 JVM 设置覆盖继承的设置，或者选择覆盖具体的 JVM 设置。

例如，下面的命令覆盖了[之前例子](#)里的服务器组的 JVM 定义，并把 **server-one** 的 JVM 设置设为 **default** JVM 定义：

```
/host=HOST_NAME/server-config=server-one/jvm=default:add()
```


和服务器组相似，您也可以在服务器级别覆盖具体的 JVM 设置。例如，要设置不同的堆大小，您可以使用下列命令：

```
/host=HOST_NAME/server-config=server-one/jvm=default:write-attribute(name=heap-size,value="1024m")
```

您也可以在管理控制台里编辑服务器 JVM 设置，选择 **Runtime** 标签页，然后选择 **Hosts** 及相关的主机，并点击 **View** 来编辑服务器。

这些单独服务器的设置保存在 `host.xml` 里。

9.3. 显示 JVM 状态

您可以通过管理控制台查看独立服务器或受管域服务器的 JVM 资源的状态，如堆和线程的使用情况。虽然这些统计数据不是实时显示，但您可以点 **Refresh Results** 来查看 JVM 资源的最新数据。

要显示独立 JBoss EAP 服务器的 JVM 状态：

- 选择 **Runtime** 标签页，然后选择 **Standalone Server**。在 **Monitor** 列里选择 **JVM**，然后点击 **View**。

要显示受管域里 JBoss EAP 服务器的 JVM 状态：

- 选择 **Runtime** 标签页，然后选择您要查看的服务器组和服务器。在 **Monitor** 列里选择 **JVM**，然后点击 **View**。

它显示了堆的以下使用信息：

Max

可以用于内存管理的最大内存数量。

Used

已使用的内存数量。

Committed

已提交 JVM 使用的内存数量。

其它信息（如 JVM 在线时间和线程使用情况）也可以获得。

9.4. 指定 32 位或 64 位 JVM 架构

在某些环境里，如 Hewlett-Packard HP-UX 和 Solaris，您可以使用 **-d32** 或 **-d64** 参数来指定运行在 32 位还是 64 位 JVM 里。如果没有指定则默认为 32 位。

为**独立服务器指定 64 位架构**

1. 打开 `EAP_HOME/bin/standalone.conf`。
2. 添加下列内容，将 **-d64** 选项附到 **JAVA_OPTS** 后面。

```
JAVA_OPTS="$JAVA_OPTS -d64"
```

为**受管域指定 64 位架构**

当运行在受管域时，除了服务器实例之外，您可以为主机和进程控制器指定 64 位环境。

1. 设置运行在 64 位 JVM 里的主机和进程控制器。

- a. 打开 **EAP_HOME/bin/domain.conf**。
- b. 添加下列行以将 **-d64** 选项附到 **JAVA_OPTS** 后面。请确保是在设置 **PROCESS_CONTROLLER_JAVA_OPTS** 和 **HOST_CONTROLLER_JAVA_OPTS** 之前插入这些内容的。

```
JAVA_OPTS="$JAVA_OPTS -d64"
```

domain.conf 里的 JVM 选项示例

```
#
# Specify options to pass to the Java VM.
#
if [ "x$JAVA_OPTS" = "x" ]; then
    JAVA_OPTS="-Xms64m -Xmx512m -XX:MaxPermSize=256m -
Djava.net.preferIPv4Stack=true"
    JAVA_OPTS="$JAVA_OPTS -
Djboss.modules.system.pkgs=$JBOSS_MODULES_SYSTEM_PKGS -
Djava.awt.headless=true"
    JAVA_OPTS="$JAVA_OPTS -Djboss.modules.policy-permissions=true"
    JAVA_OPTS="$JAVA_OPTS -d64"
else
    echo "JAVA_OPTS already set in environment; overriding default
settings with values: $JAVA_OPTS"
fi
```

2. 设置运行在 64 位 JVM 里的服务器实例。

将 **-d64** 作为 JVM 选项添加至 JVM 配置。下面的命令展示了将其添加到 **default** JVM 配置。

```
/host=HOST_NAME/jvm=default:add-jvm-option(jvm-option="-d64")
```

第 10 章 MAIL 子系统

10.1. 配置 MAIL 子系统

mail 子系统允许您在 JBoss EAP 里配置邮件会话，并使用 JNDI 把这些会话注入到应用程序。它也支持使用 Java EE 7 `@MailSessionDefinition` 和 `@MailSessionDefinitions` 注解的配置。

配置应用程序里使用的 SMTP 服务器

1. 用下列 CLI 命令配置 SMTP 服务器和转出套接字绑定，例如：

```
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=my-smtp:add(host=localhost, port=25)
```

```
/subsystem=mail/mail-session=mySession:add(jndi-name=java:jboss/mail/MySession)
```

```
/subsystem=mail/mail-session=mySession/server=smtp:add(outbound-socket-binding-ref=my-smtp, username=user, password=pass, tls=true)
```

2. 在应用程序里调用配置的邮件会话

```
@Resource(lookup="java:jboss/mail/MySession")
private Session session;
```

10.2. 配置自定义传输

当使用标准的邮件服务器（如 POP3 或 IMAP）时，邮件服务器可以定义一系列属性，而一些属性是强制性的。其中最重要的是 **outbound-socket-binding-ref**，它是对转出邮件套接字绑定的引用，通过主机地址和端口号来定义。

对于使用多个主机来进行负载平衡的用户来说，定义 **outbound-socket-binding-ref** 可能不是最有效的解决方案。标准的 JavaMail 不支持使用多个主机进行负载平衡的主机配置。因此，使用这样的配置的用户必须实现自定义邮件传输。这些自定义邮件传输不要求 **outbound-socket-binding-ref** 并允许自定义的主机属性格式。

您可以用管理 CLI 配置自定义邮件传输。

1. 添加新的邮件会话并指定 JNDI 名称。

```
/subsystem=mail/mail-session=mySession:add(jndi-name=java:jboss/mail/MySession)
```

2. 添加转出套接字绑定并指定主机和端口。

```
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=my-smtp-binding:add(host=localhost, port=25)
```

3. 添加 SMTP 服务器并指定转出套接字绑定、用户名和密码。

```
/subsystem=mail/mail-session=mySession/server=smtp:add(outbound-socket-binding-ref=my-smtp-binding, username=user, password=pass, tls=true)
```

注意

您可以用类似的步骤配置 POP3 或 IMAP 服务器。

POP3 服务器

```
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=my-pop3-binding:add(host=localhost, port=110)
/subsystem=mail/mail-session=mySession/server=pop3:add(outbound-socket-binding-ref=my-pop3-binding, username=user, password=pass)
```

IMAP 服务器

```
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=my-imap-binding:add(host=localhost, port=143)
/subsystem=mail/mail-session=mySession/server=imap:add(outbound-socket-binding-ref=my-imap-binding, username=user, password=pass)
```

要使用自定义服务器，您可以创建一个没有转出套接字绑定的自定义邮件服务器。您可以在自定义邮件服务器的属性定义里指定主机信息，例如：

```
/subsystem=mail/mail-session=mySession/custom=myCustomServer:add(username=user, password=pass, properties={"host" => "myhost", "my-property" => "value"})
```

如果您定义了一个自定义协议，包含句点 (.) 的任何属性名都被当作全限定名且被直接处理。任何其他格式，如 **my-property**，将用下列格式进行转换：**mail.server.name.my-property**。

下面是一个包含自定义服务器的邮件 XML 配置示例。

```
<subsystem xmlns="urn:jboss:domain:mail:2.0">
  <mail-session name="default" jndi-name="java:jboss/mail/Default">
    <smtp-server outbound-socket-binding-ref="mail-smtp"/>
  </mail-session>
  <mail-session name="myMail" from="user.name@domain.org" jndi-name="java:/Mail">
    <smtp-server password="password" username="user" tls="true" outbound-socket-binding-ref="mail-smtp"/>
    <pop3-server outbound-socket-binding-ref="mail-pop3"/>
    <imap-server password="password" username="nobody" outbound-socket-binding-ref="mail-imap"/>
  </mail-session>
  <mail-session name="custom" jndi-name="java:jboss/mail/Custom" debug="true">
    <custom-server name="smtp" password="password">
```

```
username="username">
    <property name="host" value="mail.example.com"/>
  </custom-server>
</mail-session>
<mail-session name="custom2" jndi-name="java:jboss/mail/Custom2"
debug="true">
  <custom-server name="pop3" outbound-socket-binding-ref="mail-
pop3">
    <property name="custom-prop" value="some-custom-prop-value"/>
  </custom-server>
</mail-session>
</subsystem>
```

第 11 章 配置 WEB SERVICES

JBoss EAP 可以使用管理控制台或管理 CLI 通过 **webservices** 子系统来配置 Web 服务的行为。您可以配置已发布的端点地址和处理程序链。您也可以启用 Web 服务的运行时统计信息收集。

如需了解更多相关信息，请参阅为 *JBoss EAP* 开发 Web 服务应用程序中的[配置 Web 服务子系统](#)。

第 12 章 JBOSS EAP 的日志

JBoss EAP 提供了高度可配置的日志功能，它既可用于内部使用，也可用于部署的应用程序。**logging** 子系统基于 JBoss LogManager，并支持 JBoss Logging 之外的几个第三方的应用程序日志框架。

12.1. 关于服务器日志

12.1.1. 服务器日志

在默认情况下，所有的 JBoss EAP 日志条目都会写入 **server.log** 文件。这个文件的位置取决于操作模式。

- 独立服务器：**EAP_HOME/standalone/log/server.log**
- 受管域：**EAP_HOME/domain/servers/SERVER_NAME/log/server.log**

这个文件通常被称为服务器日志。详情请参考 [Root Logger](#) 一节。

12.1.2. 引导日志

在引导期间，JBoss EAP 记录关于 Java 环境和每个服务启动的日志信息。这个日志可用于故障解除。在默认情况下，所有的日志条目都被写入 [server log](#)。

引导日志配置是在 **logging.properties** 文件里进行配置的，它在 JBoss EAP **logging** 子系统启动并接管后才会处于活动状态。这个文件的位置取决于您的操作模式。

- 独立服务器：**EAP_HOME/standalone/configuration/logging.properties**
- 受管域：
域控制器和每台服务器都有一个 **logging.properties** 文件。
 - 域控制器：**EAP_HOME/domain/configuration/logging.properties**
 - 服务器：**EAP_HOME/domain/servers/SERVER_NAME/data/logging.properties**



警告

我们推荐您不要直接编辑 **logging.properties** 文件，除非是用于特殊的情况。在进行编辑之前，我们推荐您通过[红帽客户门户网站](#)开一个问题单。

对 **logging.properties** 文件的手动修改在启动时会被覆盖。

12.1.2.1. 查看引导错误

在排除 JBoss EAP 的故障时，第一个步骤应该是检查引导过程中出现的错误。您可以使用这些信息来诊断并找到原因。另外，您可以打一个支持问题单来寻求解决引导故障的帮助。

查看引导错误有两种方式，各自都有自己的优势。您可以直接查看 **server.log** 文件或通过 **read-boot-errors** 管理 CLI 命令来查看引导错误。

查看服务器日志文件

您可以打开 **server.log** 文件来查看引导期间发生的任何错误。

这个方法允许您查看每个错误信息以及相关的信息，从而可以帮助您了解这些错误为什么会发生。您可以查看所有普通文本格式的错误信息。

1. 在文件参看器里打开 **server.log**。
2. 进入文件的结尾处。
3. 向后搜索 **WFLYSRV0049**，这标记着最新的引导序列的起始处。
4. 搜索日志里出现的 **ERROR**。每条信息都包括错误的描述以及相关的模块。

下面是 **server.log** 日志文件里的错误描述示例。

```
2016-03-16 14:32:01,627 ERROR [org.jboss.msc.service.fail] (MSC service
thread 1-7) MSC000001: Failed to start service
jboss.undertow.listener.default: org.jboss.msc.service.StartException in
service jboss.undertow.listener.default: Could not start http listener
    at
org.wildfly.extension.undertow.ListenerService.start(ListenerService.java:
142)
    at
org.jboss.msc.service.ServiceControllerImpl$StartTask.startService(Service
ControllerImpl.java:1948)
    at
org.jboss.msc.service.ServiceControllerImpl$StartTask.run(ServiceControlle
rImpl.java:1881)
    at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:
1142)
    at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java
:617)
    at java.lang.Thread.run(Thread.java:745)
Caused by: java.net.BindException: Address already in use
...
```

通过管理 CLI 命令读取引导错误

如果服务器在引导过程中报告错误，您可以使用 **read-boot-errors** 命令查看引导错误。

这个方法不要求访问服务器的文件系统，这对于没有访问权限却负责监控错误的用户来说是很有用的。因为这是一个管理 CLI 命令，它可以用在脚本里。例如，您可以编写一个脚本来启动多个 JBoss EAP 实例，然后检查引导过程中的错误。

运行下列管理 CLI 命令。

```
/core-service=management:read-boot-errors
```

在引导过程中出现的任何错误都会被列出。

```
{
  "outcome" => "success",
  "result" => [
```



```

    {
        "failed-operation" => {
            "operation" => "add",
            "address" => [
                ("subsystem" => "undertow"),
                ("server" => "default-server"),
                ("http-listener" => "default")
            ]
        },
        "failure-description" => "{\\"WFLYCTL0080: Failed services\\" =>
        {\\"jboss.undertow.listener.default\\" =>
        \\"org.jboss.msc.service.StartException in service
        jboss.undertow.listener.default: Could not start http listener
        Caused by: java.net.BindException: Address already in use\\"}}",
        "failed-services" => {"jboss.undertow.listener.default" =>
        "org.jboss.msc.service.StartException in service
        jboss.undertow.listener.default: Could not start http listener
        Caused by: java.net.BindException: Address already in use"}
    }
    ...
]
}
```

12.1.3. 垃圾收集日志

垃圾收集日志把所有垃圾收集活动记录到普通文本日志文件里。这些日志文件可用于诊断目的。除了 IBM JDK 以外，垃圾收集日志会在所有支持的环境中的 JBoss EAP 独立服务器上默认启用。

垃圾收集日志的默认位置是 **EAP_HOME/standalone/log/gc.log.DIGIT.current**。在默认情况下，垃圾收集日志的每个文件的大小被限制为 3MB，最多 5 个文件进行轮换。

12.1.4. 默认的日志文件位置

下面的日志文件是为默认的日志配置创建的。默认的配置使用 periodic 日志处理程序写入服务器日志文件。

表 12.1. 独立服务器的默认日志文件

日志文件	描述
EAP_HOME/standalone/log/server.log	包含服务器日志信息，其中包含服务器启动信息。
EAP_HOME/standalone/log/gc.log.DIGIT.current	包括垃圾收集的详细信息。

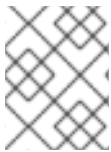
表 12.2. 受管域的默认日志文件

日志文件	描述
EAP_HOME/domain/log/host-controller.log	包含和主机控制器启动相关的日志信息。
EAP_HOME/domain/log/process-controller.log	包含和进程控制器启动相关的日志信息。

日志文件	描述
EAP_HOME/domain/servers/SERVER_NAME/log/server.log	包含命名服务器的日志信息，其中包含服务器启动信息。

12.1.5. Set the Default Locale of the Server

You can configure the default locale for JBoss EAP by setting JVM properties in the appropriate startup configuration file. The startup configuration file is **EAP_HOME/bin/standalone.conf** for a standalone server or **EAP_HOME/bin/domain.conf** for a managed domain.



注意

For Windows Server, the JBoss EAP startup configuration files are **standalone.conf.bat** and **domain.conf.bat**.

Log messages that have been internationalized and localized will use this default locale. See the JBoss EAP *Development Guide* for information on [creating internationalized log messages](#).

Set the Language

Specify the language by setting the **user.language** property using the **JAVA_OPTS** variable. For example, add the following line to the startup configuration file to set a French locale.

```
JAVA_OPTS="$JAVA_OPTS -Duser.language=fr"
```

Log messages that have been internationalized and localized will now output in French.

Set the Language and Country

In addition to the language, it may also be necessary to specify the country by setting the **user.country** property. For example, add the following line to the startup configuration file to set the Portuguese locale for Brazil.

```
JAVA_OPTS="$JAVA_OPTS -Duser.language=pt -Duser.country=BR"
```

Log messages that have been internationalized and localized will now output in Brazilian Portuguese.

12.2. 查看日志文件

查看服务器和应用程序日志对于诊断错误、性能及其他问题是很重要的。某些用户可能宁愿直接在服务器文件系统里查看日志。对于没有直接访问文件系统权限的用户、或者希望使用图形界面的用户来说，JBoss EAP 允许您在管理控制台里查看日志。您也可以通过管理 CLI 查看日志。

要使日志可以从其中一个管理界面进行访问，它必须位于服务器的 **jboss.server.log.dir** 属性指定的目录且定义为 **file**、**periodic rotating**、**size rotating** 或 **periodic size rotating** 日志处理程序。RBAC 角色分配也会被考虑，所以登录至管理控制台或 CLI 的用户只能查看被授权访问的日志。

在管理控制台里查看日志

您可以直接在管理控制台里查看日志。

- 选择 **Runtime** 标签页。

- 选择 **Standalone Server**。如果您运行在受管域里，请选择适当的服务器。
- 选择 **Log Files** 并点击 **View**。

在从列表里选择了日志文件后，就可以直接在管理控制台里查看和搜索日志内容。您也可以将日志文件下载到本地文件系统里。



警告

如果需要查看非常大的日志文件（如大于 100MB 的文件），则不适合使用管理控制台日志查看器。如果打开大于 15MB 的文件，您会被提示进行确认。在管理控制台里打开大型文件可能导致浏览器崩溃，所以您应该下载大型日志文件到本地，然后在文件编辑器里打开它。

通过管理 CLI 查看日志

您可以通过管理 CLI 的 **read-log-file** 命令阅读日志文件的内容。在默认情况下，它会显示指定的日志文件的最近的 **10** 行内容。

```
/subsystem=logging/log-file=LOG_FILE_NAME:read-log-file
```



注意

在受管域里，请在这个命令之前使用 **/host=HOST_NAME/server=SERVER_NAME**。

您可以使用下列参数来定制日志输出。

encoding

用于读取文件的字符编码。

lines

从文件里读取的行数。-1 将读取日志的全部内容。默认值为 **10**。

skip

读取文件前忽略的行数。默认值为 **0**。

tail

是否从文件的结尾开始读取。默认为 **true**。

例如，下列管理 CLI 命令从 **server.log** 日志文件的顶部读取开始的 **5** 行内容。

```
/subsystem=logging/log-file=server.log:read-log-file(lines=5,tail=false)
```

这会产生下列输出。

```
{
  "outcome" => "success",
  "result" => [
    "2016-03-24 08:49:26,612 INFO [org.jboss.modules] (main) JBoss
    Modules version 1.5.1.Final-redhat-1",
```

```

        "2016-03-24 08:49:26,788 INFO [org.jboss.msc] (main) JBoss MSC
version 1.2.6.Final-redhat-1",
        "2016-03-24 08:49:26,863 INFO [org.jboss.as] (MSC service thread
1-7) WFLYSRV0049: JBoss EAP 7.0.0.GA (WildFly Core 2.0.13.Final-redhat-1)
starting",
        "2016-03-24 08:49:27,973 INFO [org.jboss.as.server] (Controller
Boot Thread) WFLYSRV0039: Creating http management service using socket-
binding (management-http)",
        "2016-03-24 08:49:27,994 INFO [org.xnio] (MSC service thread 1-1)
XNIO version 3.3.4.Final-redhat-1"
    ]
}

```

12.3. 关于日志子系统

JBoss EAP **logging** 子系统使用 [日志类别](#) 和 [日志处理程序](#)。日志类别定义要抓取哪些消息，而日志处理程序则定义如何处理这些消息，如写入到磁盘还是发送到控制台。

[日志配置集](#) 允许创建唯一命名的配置，并分配给独立于任何其他日志配置的应用程序。日志配置集的配置和主 **logging** 子系统几乎相同。

12.3.1. Root Logger

JBoss EAP 的 Root logger 抓取所有发送到服务器上的、没有被任何日志类别抓取的日志消息（等于或高于特定级别）。

在默认情况下，Root logger 会被配置成使用控制台或定期日志处理程序。定期日志处理程序写入到 **server.log** 文件，这个文件通常被称为服务器日志。

关于更多的信息，请参考 [配置 Root Logger](#)。

12.3.2. 日志类别

日志类别定义要捕获的一系列日志信息以及处理这些信息的一个或多个日志处理程序。

要捕获的日志信息是通过原始 Java 软件包和日志级别来指定的。来自该软件包里的类和日志级别的信息将被日志类别捕获并发送至指定的日志处理程序。



注意

虽然日志类别通常是 Java 软件包和类名，它也可以是 **Logger.getLogger(LOGGER_NAME)** 方法指定的任何名称。

日志类别也可以选择使用 Root Logger 而不是自己的日志处理程序。

详情请参考 [配置日志类别](#)。

12.3.3. 日志处理程序

日志处理程序定义如何记录捕获的日志信息。可用的日志处理程序类型有 *console*、*file*、*periodic*、*size*、*periodic size*、*syslog*、*custom* 和 *async*。



注意

日志处理程序必须添加到至少一个 logger 才能生效。

日志处理程序类型

Console

Console 日志处理程序将日志信息写入到主机操作系统的标准输出 (**stdout**) 或标准错误 (**stderr**)。当从命令行提示运行 JBoss EAP 时会显示这些信息。来自 Console 日志处理程序的信息通常不会被保存，除非配置了操作系统捕获标准输出或标准错误流。

File

File 日志处理程序将日志信息写入到指定的文件。

Periodic

一个定期的日志处理程序将日志信息写入到某个命名文件，直至超过指定的时间。超时后文件将按照指定的时间戳重命名，而日志处理程序则继续写入到新创建的具有原来名称的日志文件。

Size

Size 日志处理程序将日志信息写入到某个命名文件，直至文件达到指定的大小。此时，文件将用一个数字后缀重命名，而日志处理程序则继续写入到新创建的具有原来名称的日志文件。每个 Size 日志处理程序都必须指定保持这种形式的最大文件数量。

Periodic Size

Periodic Size 日志处理程序将日志信息写入到某个命名文件，直至文件达到指定的大小或超过指定的时间。此时，文件将重命名，而日志处理程序则继续写入到新创建的具有原来名称的日志文件。

这是一个 Periodic 和 Size 日志处理程序的组合且支持联合的属性。

Syslog

Syslog 日志处理程序可以用来发送信息给远程的日志服务器。这允许多个应用程序发送日志信息到相同的服务器，在一起进行解析。

Custom

一个自定义日志处理程序让您配置新的日志处理程序类型。自定义处理程序必须实现为继承 **java.util.logging.Handler** 的 Java 类且包含在模块里。您也可以将 Log4J appender 用作自定义日志处理程序。

Async

Async 日志处理程序是一个日志处理程序 Wrapper，它为一个或多个日志处理程序提供异步行为。它可用于具有高延迟或其他性能问题（如写入日志信息到网络文件系统）的日志处理程序。

关于配置这些日志处理程序的细节，请参考[配置日志处理程序](#)章节。

12.3.4. 日志级别

日志级别是一个枚举值，它表示日志信息的性质和严重性。作为开发人员，您可以使用所选日志框架的相关方法指定日志级别来发送信息。

JBoss EAP 支持受支持的应用程序日志框架使用的所有日志级别。最常用的日志级别，从低到高分别是：**TRACE**、**DEBUG**、**INFO**、**WARN**、**ERROR** 和 **FATAL**。

日志类别和处理程序使用日志级别来限制它们负责的日志信息。每个日志级别都有一个分配的数字值，指示其相对其他日志级别的顺序。日志类别和处理程序都分配一个日志级别，它们只处理该级别及高于该级别的日志信息。例如，具有 **WARN** 级别的日志处理程序只记录级别为 **WARN**、**ERROR** 和 **FATAL** 的信息。

支持的日志级别

日志级别	值	描述
ALL	Integer.MIN_VALUE	提供所有的日志信息。
FINEST	300	-
FINER	400	-
TRACE	400	TRACE 级别的日志提供了应用程序的运行状态的详细信息，它通常用于调试过程。
DEBUG	500	DEBUG 级别的日志指示单独请求或应用程序活动的进展，它通常在调试过程中使用。
FINE	500	-
CONFIG	700	-
INFO	800	INFO 级别的日志指示应用程序的总体进展。它通常用于应用程序启动、关闭和其他主要的生命周期事件。
WARN	900	WARN 级别的日志信息代表了一个不理相但也不是错误的状况。 WARN 日志信息可以指示将来可能导致错误的情况。
WARNING	900	-
ERROR	1000	ERROR 级别的日志指示可能阻止当前活动完成但不会阻止应用程序运行的错误已发生。
SEVERE	1000	-
FATAL	1100	FATAL 级别的日志指示可能导致严重服务故障和应用程序关闭以及可能导致 JBoss EAP 关闭的事件。
OFF	Integer.MAX_VALUE	不显示任何日志信息。



注意

ALL 是最低的日志级别，它表示所有级别的日志信息。它提供了最多的日志信息。

FATAL 是最高的日志级别，它只包含该级别的日志信息。它提供了最少的日志信息。

12.3.5. 日志格式器

日志格式器定义了日志信息的外观。它是一个使用基于 `java.util.logging.Formatter` 类的语法的字符串。

例如，对于服务器日志信息，默认的配置使用下列日志格式器字符串：**%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c] (%t) %s%n**。这将创建类似于下面的日志信息。

```
2016-03-18 15:49:32,075 INFO [org.jboss.as] (Controller Boot Thread)
WFLYSRV0051: Admin console listening on http://127.0.0.1:9990
```

关于日志格式器的更多信息，请参考[配置命名模式格式器](#)或[配置自定义日志格式器](#)。

下表是日志格式器字符串使用的语法。

日志格式器语法

语法	描述
%c	日志事件的类别。
%p	日志条目的级别（INFO、DEBUG 等）。
%P	日志条目的本地化级别。
%d	当前的日期/时间（ yyyy-MM-dd HH:mm:ss,SSS 格式）。
%r	相对时间（自日志初始化后的毫秒数）。
%z	时区，必须在日期（%d）之前指定。例如， %z{GMT}%d{HH:mm:ss,SSS} 。
%k	日志资源关键字（用于日志信息的本地化）。
%m	日志信息（包含异常跟踪信息）。
%s	简单日志信息（不包含异常跟踪信息）。
%e	异常栈跟踪信息（不包含扩展的模块信息）。
%E	异常栈跟踪信息（包含扩展的模块信息）。
%t	当前线程的名称。
%n	换行符。
%C	调用日志方法（慢）的代码的类。
%F	调用日志方法（慢）的类的文件名。
%l	调用日志方法（慢）的代码的源码位置。
%L	调用日志方法（慢）的代码的行号。
%M	调用日志方法（慢）的代码的方法。

语法	描述
%x	嵌套的诊断性上下文。
%X	消息诊断性上下文。
%%	百分比 (%) 字符（脱字符）。

12.3.6. 过滤器表达式

用 **filter-spec** 属性配置的过滤器表达式用于根据不同的标准记录日志信息。过滤器检查总是在原始的未格式化的信息上完成的。您可以在 logger 或处理程序里包含过滤器，但 logger 过滤器优先于处理程序上的过滤器。



注意

其他 logger 不会继承为 Root logger 指定的 **filter-spec**。相反您必须对每个处理程序指定 **filter-spec**。

表 12.3. 用于日志的过滤器表达式

过滤器表达式	描述
accept	接受所有的日志信息。
deny	拒绝所有的日志信息。
not[filter expression]	返回单个过滤器表达式的反转值。例如： not(match("WFLY"))
all[filter expression]	返回连接用逗号隔开的过滤器表达式的值。例如： all(match("WFLY"),match("WELD"))
any[filter expression]	从用逗号隔开的过滤器表达式列表返回一个值。例如： any(match("WFLY"),match("WELD"))
levelChange[level]	用指定的级别更新日志记录。例如： levelChange(WARN)
levels[levels]	过滤用逗号隔开的级别列表里列出的级别的日志信息。例如： levels(DEBUG,INFO,WARN,ERROR)

过滤器表达式	描述
<code>levelRange[minLevel,maxLevel]</code>	<p>过滤指定日志级别范围内的日志信息。[和] 字符用来表示包含的级别。(和) 字符则用来表示排斥的级别。例如：</p> <ul style="list-style-type: none"> • <code>levelRange[INFO,ERROR]</code> <ul style="list-style-type: none"> ◦ 最小的级别必须大于或等于 INFO，而最大的级别必须小于或等于 ERROR。 • <code>levelRange[DEBUG,ERROR]</code> <ul style="list-style-type: none"> ◦ 最小的级别必须大于或等于 DEBUG，而最大的级别必须小于 ERROR。
<code>match["pattern"]</code>	<p>过滤使用提供的常规表达式的日志信息。例如：</p> <p><code>match("WFLY\d+")</code></p>
<code>substitute["pattern","replacement value"]</code>	<p>用替代文本（第二个参数）替换模式的第一次匹配（第一个参数）的过滤器。例如：</p> <p><code>substitute("WFLY","EAP")</code></p>
<code>substituteAll["pattern","replacement value"]</code>	<p>用替代文本（第二个参数）替换模式的所有匹配（第一个参数）的过滤器。例如：</p> <p><code>substituteAll("WFLY","EAP")</code></p>

注意

当用管理 CLI 配置过滤器表达式时，请确保转义过滤文本里的逗号和引号以正确处理字符串。您必须在逗号和引号之前使用反斜杠（\）并用引号包括整个表达式。下面是一个正确转义 `substituteAll("WFLY","YLFW")` 的示例。

```
/subsystem=logging/console-handler=CONSOLE:write-attribute(name=filter-spec,value="substituteAll(\"WFLY\\\", \"YLFW\\\")")
```

12.3.7. 隐性的日志依赖关系

在默认情况下，JBoss EAP **logging** 子系统会添加隐性的日志 API 依赖关系至部署里。您可以通过 **add-logging-api-dependencies** 属性控制是否添加这些隐性的依赖关系至部署，它的默认值是 **true**。

通过管理 CLI，您可以设置 **add-logging-api-dependencies** 属性为 **false**，这样隐性的日志 API 依赖关系就不会添加至部署里。

```
/subsystem=logging:write-attribute(name=add-logging-api-dependencies,value=false)
```

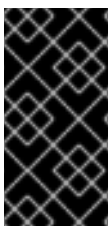
关于 **logging** 子系统的隐性依赖关系的更多信息，请参考《JBoss EAP 开发指南》里的『[隐性的模块依赖关系](#)』章节。#1635；

12.4. 配置日志类别

本节展示了如何用管理 CLI 配置日志类别。您也可以用管理控制台配置日志类别，您可以先从 **Configuration** 标签页进入 **Logging** 子系统，然后选择 **Log Categories** 标签页。

配置日志类别将执行的主要任务是：

- [添加新的日志类别](#)。
- [配置日志类别设置](#)。
- [分配日志处理程序给日志类别](#)。



重要

如果您为日志配置集配置日志类别，命令必须以 `/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/` 而不是 `/subsystem=logging/` 起始。

此外，如果您运行在受管域里，请在命令里先使用 `/profile=PROFILE_NAME`。

添加日志类别

日志类别名称是用原始 Java 软件包定义的。只要该软件包里的消息遵守其他设置（如日志级别），它们将被捕获。

```
/subsystem=logging/logger=LOG_CATEGORY:add
```

配置日志类别设置

根据具体要求，您可能需要设置一个或多个下列的日志类别属性。关于日志类别属性的完整列表及其描述，请参考[日志类别属性](#)。

- 设置日志级别。
设置日志类别的日志级别。默认是 **ALL**。关于所有可用的选项，请参考[日志级别](#)。

```
/subsystem=logging/logger=LOG_CATEGORY:write-attribute(name=level,value=LEVEL)
```

- 设置这个类别是否应该使用 Root Logger 的日志处理程序。
在默认情况下，处理自己的日志处理程序之外，日志类别将使用 Root logger。如果日志类别应该只使用分配的处理程序，请将 **use-parent-handlers** 属性为 **false**。

```
/subsystem=logging/logger=LOG_CATEGORY:write-attribute(name=use-parent-handlers,value=USE_PARENT_HANDLERS)
```

- 设置过滤器表达式
设置日志类别过滤日志信息的表达式。请确保转义任何逗号和引号并用引号括起表达式。例如，对于下面的可替换变量 **FILTER_EXPRESSION**，需要用 `"not(match(\"WFLY\"))"` 替换过滤表达式 `not(match("WFLY"))`。

```
/subsystem=logging/logger=LOG_CATEGORY:write-attribute(name=filter-spec,value=FILTER_EXPRESSION)
```

■

关于可用的过滤器表达式的更多信息，请参考[过滤器表达式](#)章节。

分配处理程序

为日志类别分配日志处理程序。

```
/subsystem=logging/logger=LOG_CATEGORY:add-handler(name=LOG_HANDLER_NAME)
```

删除日志类别

您可以用 **remove** 操作删除日志类别。

```
/subsystem=logging/logger=LOG_CATEGORY:remove
```

12.5. 配置日志处理程序

日志处理程序定义如何记录捕获的日志信息。关于配置您需要的日志处理程序类型，请参考相关的章节。

- [控制台日志处理程序](#)
- [文件日志处理程序](#)
- [定期轮换日志处理程序](#)
- [大小轮换日志处理程序](#)
- [定期大小轮换日志处理程序](#)
- [Syslog 处理程序](#)
- [自定义日志处理程序](#)
- [异步日志处理程序](#)

12.5.1. 配置控制台日志处理程序

本节展示了如何用管理 CLI 配置 Console 日志处理程序。您也可以使用管理控制台配置 Console 日志处理程序，您可以先从 **Configuration** 标签页进入 **Logging** 子系统，然后选择 **Handler** 标签页，并从左侧的菜单选择 **Console**。

配置 Console 日志处理程序时要执行的主要任务是：

- [添加新的控制台日志处理程序。](#)
- [配置控制台日志处理程序。](#)
- [分配控制台日志处理程序至 logger。](#)



重要

如果您为日志配置集配置这个日志处理程序，命令将以 **/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/** 而不是 **/subsystem=logging/** 开始。

此外，如果您运行在受管域里，请在命令里先使用 **/profile=PROFILE_NAME**。

添加控制台日志处理程序

```
/subsystem=logging/console-handler=CONSOLE_HANDLER_NAME:add
```

配置控制台日志处理程序设置

根据具体要求，您可能需要设置一个或多个 Console 日志处理程序属性。关于 Console 日志处理程序属性及其描述的完整列表，请参考 [Console 日志处理程序属性](#)。

- 设置日志级别。
设置处理程序的日志级别。默认值是 **ALL**。关于所有可用的选项，请参考 [日志级别](#)。

```
/subsystem=logging/console-handler=CONSOLE_HANDLER_NAME:write-attribute(name=level,value=LEVEL)
```

- 设置目标。
设置处理程序的目标，它可以是 **System.out**、**System.err** 或 **console** 中的一个。默认值是 **System.out**。

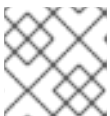
```
/subsystem=logging/console-handler=CONSOLE_HANDLER_NAME:write-attribute(name=target,value=TARGET)
```

- 设置编码。
设置处理程序的编码，例如 **utf-8**。

```
/subsystem=logging/console-handler=CONSOLE_HANDLER_NAME:write-attribute(name=encoding,value=ENCODING)
```

- 设置日志格式器。
设置处理程序的格式器字符串。例如，默认的格式字符串是 **%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%n**。请确保用引号包括 **FORMAT** 值。

```
/subsystem=logging/console-handler=CONSOLE_HANDLER_NAME:write-attribute(name=formatter,value=FORMAT)
```



注意

如果您想引用 [saved formatter](#)，您可以使用 **named-formatter** 属性。

- 设置自动冲刷。
设置在每次写入后是否自动冲刷。默认值是 **true**。

```
/subsystem=logging/console-handler=CONSOLE_HANDLER_NAME:write-attribute(name=autoflush,value=AUTO_FLUSH)
```

- 设置过滤器表达式
设置处理程序过滤日志信息的表达式。请确保转义任何逗号和引号并用引号括起表达式。例如，对于下面的可替换变量 **FILTER_EXPRESSION**，需要用 **"not(match(\"WFLY\"))"** 替换过滤表达式 **not(match("WFLY"))**。

```
/subsystem=logging/console-handler=CONSOLE_HANDLER_NAME:write-attribute(name=filter-spec,value=FILTER_EXPRESSION)
```

关于可用的过滤器表达式的更多信息，请参考[过滤器表达式](#)章节。

分配 **Console** 日志处理程序给 **Logger**

要激活日志处理程序，您必须将其分配给一个 Logger。

下面的管理 CLI 命令将 Console 日志处理程序分配给 Root Logger。

```
/subsystem=logging/root-logger=ROOT:add-handler(name=CONSOLE_HANDLER_NAME)
```

下面的管理 CLI 命令将 Console 日志处理程序分配给名称由 **CATEGORY** 指定的 Logger。

```
/subsystem=logging/logger=CATEGORY:add-handler(name=CONSOLE_HANDLER_NAME)
```

删除 **Console** 日志处理程序

日志处理程序可以用 **remove** 操作删除。如果日志处理程序目前分配给了 logger 或异步日志处理程序，它就无法被删除。

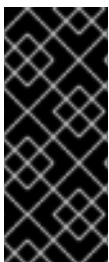
```
/subsystem=logging/console-handler=CONSOLE_HANDLER_NAME:remove
```

12.5.2. 配置 File 日志处理程序

本节展示了如何用管理 CLI 配置 File 日志处理程序。您也可以使用管理控制台配置 File 日志处理程序，您可以先从 **Configuration** 标签页进入 **Logging** 子系统，然后选择 **Handler** 标签页，并从左侧的菜单选择 **File**。

配置 File 日志处理程序时要执行的主要任务是：

- [添加新的 File 日志处理程序](#)。
- [配置 File 日志处理程序](#)。
- [分配 File 日志处理程序给 logger](#)。



重要

如果您为日志配置集配置这个日志处理程序，命令将以 `/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/` 而不是 `/subsystem=logging/` 开始。

此外，如果您运行在受管域里，请在命令里先使用 `/profile=PROFILE_NAME`。

添加 **File** 日志处理程序

在添加 File 日志处理程序时，您必须用 **file** 属性指定文件路径，它由 **path** 和 **relative-to** 属性组成。请使用 **path** 属性来设置日志的文件路径，其中包括名称，例如 `my-log.log`。您也可以选择 **relative-to** 属性来设置 **path** 为相对路径，例如 `jboss.server.log.dir`。

```
/subsystem=logging/file-handler=FILE_HANDLER_NAME:add(file={path=FILE_PATH,relative-to=RELATIVE_TO_PATH})
```

配置 **File** 日志处理程序设置

根据具体要求，您可能需要设置一个或多个 File 日志处理程序属性。关于 File 日志处理程序属性及其描述的完整列表，请参考 [File 日志处理程序属性](#)。

- 设置日志级别。
设置处理程序的日志级别。默认值是 **ALL**。关于所有可用的选项，请参考[日志级别](#)。

```
/subsystem=logging/file-handler=FILE_HANDLER_NAME:write-attribute(name=level,value=LEVEL)
```

- 设置附加行为。
在默认情况下，服务器重启时，JBoss EAP 将日志信息附加到相同的文件里。您可以设置 **append** 属性为 **false**，在服务器重启时覆盖这个文件。

```
/subsystem=logging/file-handler=FILE_HANDLER_NAME:write-attribute(name=append,value=APPEND)
```

- 设置编码。
设置处理程序的编码，例如 **utf-8**。

```
/subsystem=logging/file-handler=FILE_HANDLER_NAME:write-attribute(name=encoding,value=ENCODING)
```

- 设置日志格式器。
设置处理程序的格式器字符串。例如，默认的格式字符串是 **%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n**。请确保用引号包括 **FORMAT** 值。

```
/subsystem=logging/file-handler=FILE_HANDLER_NAME:write-attribute(name=formatter,value=FORMAT)
```



注意

如果您想引用 [saved formatter](#)，您可以使用 **named-formatter** 属性。

- 设置自动冲刷。
设置在每次写入后是否自动冲刷。默认值是 **true**。

```
/subsystem=logging/file-handler=FILE_HANDLER_NAME:write-attribute(name=autoflush,value=AUTO_FLUSH)
```

- 设置过滤器表达式
设置处理程序过滤日志信息的表达式。请确保转义任何逗号和引号并用引号括起表达式。例如，对于下面的可替换变量 **FILTER_EXPRESSION**，需要用 **"not(match(\\"WFLY\\"))"** 替换过滤表达式 **not(match("WFLY"))**。

```
/subsystem=logging/file-handler=FILE_HANDLER_NAME:write-attribute(name=filter-spec, value=FILTER_EXPRESSION)
```

关于可用的过滤器表达式的更多信息，请参考[过滤器表达式](#)章节。

分配 File 日志处理程序给 Logger

要激活日志处理程序，您必须将其分配给一个 Logger。

下面的管理 CLI 命令将 File 日志处理程序分配给 Root Logger。

```
/subsystem=logging/root-logger=ROOT:add-handler(name=FILE_HANDLER_NAME)
```

下面的管理 CLI 命令将 File 日志处理程序分配给名称由 **CATEGORY** 指定的 Logger。

```
/subsystem=logging/logger=CATEGORY:add-handler(name=FILE_HANDLER_NAME)
```

删除 **File** 日志处理程序

日志处理程序可以用 **remove** 操作删除。如果日志处理程序目前分配给了 logger 或异步日志处理程序，它就无法被删除。

```
/subsystem=logging/file-handler=FILE_HANDLER_NAME:remove
```

12.5.3. 配置 Periodic Rotating 日志处理程序

本节展示了如何用管理 CLI 配置 Periodic 日志处理程序。您也可以使用管理控制台配置 Periodic 日志处理程序，您可以先从 **Configuration** 标签页进入 **Logging** 子系统，然后选择 **Handler** 标签页，并从左侧的菜单选择 **Periodic**。

配置 Periodic 日志处理程序时要执行的主要任务是：

- 添加新的 Periodic 日志处理程序。
- 配置 Periodic 日志处理程序设置。
- 配置 Periodic 日志处理程序至 logger。



重要

如果您为日志配置集配置这个日志处理程序，命令将以 **/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/** 而不是 **/subsystem=logging/** 开始。

此外，如果您运行在受管域里，请在命令里先使用 **/profile=PROFILE_NAME**。

添加 **Periodic** 日志处理程序

在添加 Periodic 日志处理程序时，您必须用 **file** 属性指定文件路径，它由 **path** 和 **relative-to** 属性组成。请使用 **path** 属性来设置日志的文件路径，其中包括名称，例如 **my-log.log**。您也可以选择 **relative-to** 属性来设置 **path** 为相对路径，例如 **jboss.server.log.dir**。

您也必须用 **suffix** 属性为轮换日志文件设置后缀。它必须是 **java.text.SimpleDateFormat** 可以理解的格式，例如 **.yyyy-MM-dd-HH**。轮换的周期会根据这个后缀自动计算。

```
/subsystem=logging/periodic-rotating-file-  
handler=PERIODIC_HANDLER_NAME:add(file={path=FILE_PATH,relative-  
to=RELATIVE_TO_PATH},suffix=SUFFIX)
```

配置 **Periodic** 日志处理程序设置

根据具体要求，您可能需要设置一个或多个 Periodic 日志处理程序属性。关于 Periodic 日志处理程序属性及其描述的完整列表，请参考 [Periodic 日志处理程序属性](#)。

- 设置日志级别。
设置处理程序的日志级别。默认值是 **ALL**。关于所有可用的选项，请参考 [日志级别](#)。

■


```
/subsystem=logging/periodic-rotating-file-
handler=PERIODIC_HANDLER_NAME:write-
attribute(name=level,value=LEVEL)
```

- 设置附加行为。

在默认情况下，服务器重启时，JBoss EAP 将日志信息附加到相同的文件里。您可以设置 **append** 属性为 **false**，在服务器重启时覆盖这个文件。

```
/subsystem=logging/periodic-rotating-file-
handler=PERIODIC_HANDLER_NAME:write-
attribute(name=append,value=APPEND)
```

- 设置编码。

设置处理程序的编码，例如 **utf-8**。

```
/subsystem=logging/periodic-rotating-file-
handler=PERIODIC_HANDLER_NAME:write-
attribute(name=encoding,value=ENCODING)
```

- 设置日志格式器。

设置处理程序的格式器字符串。例如，默认的格式字符串是 `%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%n`。请确保用引号包括 **FORMAT** 值。

```
/subsystem=logging/periodic-rotating-file-
handler=PERIODIC_HANDLER_NAME:write-
attribute(name=formatter,value=FORMAT)
```



注意

如果您想引用 [saved formatter](#)，您可以使用 **named-formatter** 属性。

- 设置自动冲刷。

设置在每次写入后是否自动冲刷。默认值是 **true**。

```
/subsystem=logging/periodic-rotating-file-
handler=PERIODIC_HANDLER_NAME:write-
attribute(name=autoflush,value=AUTO_FLUSH)
```

- 设置过滤器表达式

设置处理程序过滤日志信息的表达式。请确保转义任何逗号和引号并用引号括起表达式。例如，对于下面的可替换变量 **FILTER_EXPRESSION**，需要用 `"not(match(\"WFLY\"))"` 替换过滤表达式 `not(match("WFLY"))`。

```
/subsystem=logging/periodic-rotating-file-
handler=PERIODIC_HANDLER_NAME:write-attribute(name=filter-spec,
value=FILTER_EXPRESSION)
```

关于可用的过滤器表达式的更多信息，请参考[过滤器表达式](#)章节。

分配 **Periodic** 日志处理程序给 **Logger**

要激活日志处理程序，您必须将其分配给一个 **Logger**。

下面的管理 CLI 命令将 Periodic 日志处理程序分配给 Root Logger。

```
/subsystem=logging/root-logger=ROOT:add-  
handler(name=PERIODIC_HANDLER_NAME)
```

下面的管理 CLI 命令将 Periodic 日志处理程序分配给名称由 **CATEGORY** 指定的 Logger。

```
/subsystem=logging/logger=CATEGORY:add-handler(name=PERIODIC_HANDLER_NAME)
```

删除 **Periodic** 日志处理程序

日志处理程序可以用 **remove** 操作删除。如果日志处理程序目前分配给了 logger 或异步日志处理程序，它就无法被删除。

```
/subsystem=logging/periodic-rotating-file-  
handler=PERIODIC_HANDLER_NAME:remove
```

12.5.4. 配置 Size Rotating 日志处理程序

本节展示了如何用管理 CLI 配置 Size 日志处理程序。您也可以用管理控制台配置 Size 日志处理程序，您可以先从 **Configuration** 标签页进入 **Logging** 子系统，然后选择 **Handler** 标签页，并从左侧的菜单选择 **Size**。

配置 Size 日志处理程序时要执行的主要任务是：

- 添加新的 [Size 日志处理程序](#)。
- 配置 [Size 日志处理程序设置](#)。
- 配置 [Size 日志处理程序至 logger](#)。



重要

如果您为日志配置集配置这个日志处理程序，命令将以 **/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/** 而不是 **/subsystem=logging/** 开始。

此外，如果您运行在受管域里，请在命令里先使用 **/profile=PROFILE_NAME**。

添加 **Size** 日志处理程序

在添加 Size 日志处理程序时，您必须用 **file** 属性指定文件路径，它由 **path** 和 **relative-to** 属性组成。请使用 **path** 属性来设置日志的文件路径，其中包括名称，例如 **my-log.log**。您也可以选择 **relative-to** 属性来设置 **path** 为相对路径，例如 **jboss.server.log.dir**。

```
/subsystem=logging/size-rotating-file-handler=SIZE_HANDLER_NAME:add(file=  
{path=FILE_PATH,relative-to=RELATIVE_TO_PATH})
```

配置 **Size** 日志处理程序设置

根据具体要求，您可能需要设置一个或多个 Size 日志处理程序属性。关于 Size 日志处理程序属性及其描述的完整列表，请参考 [Size 日志处理程序属性](#)。

- 设置日志级别。
设置处理程序的日志级别。默认值是 **ALL**。关于所有可用的选项，请参考 [日志级别](#)。

■

```
/subsystem=logging/size-rotating-file-
handler=SIZE_HANDLER_NAME:write-attribute(name=level,value=LEVEL)
```

- 设置轮换日志文件的后缀。
设置后缀字符串，它必须是 `java.text.SimpleDateFormat` 可理解的格式，例如 `.yyyy-MM-dd-HH`。轮换的周期将根据这个后缀自动计算。

```
/subsystem=logging/size-rotating-file-
handler=SIZE_HANDLER_NAME:write-attribute(name=suffix, value=SUFFIX)
```

- 设置轮换大小。
设置文件的最大值。当文件大小达到这个值时就需要进行轮换。默认值是 **2m**，亦即 2MB。

```
/subsystem=logging/size-rotating-file-
handler=SIZE_HANDLER_NAME:write-attribute(name=rotate-size,
value=ROTATE_SIZE)
```

- 设置保留的备份日志的最大数目。
设置保留的备份数量。默认值是 **1**。

```
/subsystem=logging/size-rotating-file-
handler=SIZE_HANDLER_NAME:write-attribute(name=max-backup-index,
value=MAX_BACKUPS)
```

- 设置是否在引导时轮换日志文件。
在默认情况下，服务器重启时不会创建新的日志文件。您可以将它设置为 **true** 在服务器重启时轮换日志文件。

```
/subsystem=logging/size-rotating-file-
handler=SIZE_HANDLER_NAME:write-attribute(name=rotate-on-boot,
value=ROTATE_ON_BOOT)
```

- 设置附加行为。
在默认情况下，服务器重启时，JBoss EAP 将日志信息附加到相同的文件里。您可以设置 **append** 属性为 **false**，在服务器重启时覆盖这个文件。

```
/subsystem=logging/size-rotating-file-
handler=SIZE_HANDLER_NAME:write-attribute(name=append,value=APPEND)
```

- 设置编码。
设置处理程序的编码，例如 **utf-8**。

```
/subsystem=logging/size-rotating-file-
handler=SIZE_HANDLER_NAME:write-
attribute(name=encoding,value=ENCODING)
```

- 设置日志格式器。
设置处理程序的格式器字符串。例如，默认的格式字符串是 `%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%n`。请确保用引号包括 **FORMAT** 值。

```
/subsystem=logging/size-rotating-file-
handler=SIZE_HANDLER_NAME:write-
attribute(name=formatter,value=FORMAT)
```



注意

如果您想引用 [saved formatter](#)，您可以使用 **named-formatter** 属性。

- 设置自动冲刷。
设置在每次写入后是否自动冲刷。默认值是 **true**。

```
/subsystem=logging/size-rotating-file-
handler=SIZE_HANDLER_NAME:write-
attribute(name=autoflush,value=AUTO_FLUSH)
```

- 设置过滤器表达式
设置处理程序过滤日志信息的表达式。请确保转义任何逗号和引号并用引号括起表达式。例如，对于下面的可替换变量 **FILTER_EXPRESSION**，需要用 **"not(match(\"WFLY\"))"** 替换过滤表达式 **not(match("WFLY"))**。

```
/subsystem=logging/size-rotating-file-
handler=SIZE_HANDLER_NAME:write-attribute(name=filter-spec,
value=FILTER_EXPRESSION)
```

关于可用的过滤器表达式的更多信息，请参考[过滤器表达式](#)章节。

分配 **Size** 日志处理程序给 **Logger**

要激活日志处理程序，您必须将其分配给一个 Logger。

下面的管理 CLI 命令将 Size 日志处理程序分配给 Root Logger。

```
/subsystem=logging/root-logger=ROOT:add-handler(name=SIZE_HANDLER_NAME)
```

下面的管理 CLI 命令将 Size 日志处理程序分配给名称由 **CATEGORY** 指定的 Logger。

```
/subsystem=logging/logger=CATEGORY:add-handler(name=SIZE_HANDLER_NAME)
```

删除 **Size** 日志处理程序

日志处理程序可以用 **remove** 操作删除。如果日志处理程序目前分配给了 logger 或异步日志处理程序，它就无法被删除。

```
/subsystem=logging/size-rotating-file-handler=SIZE_HANDLER_NAME:remove
```

12.5.5. 配置 **Periodic Size Rotating** 日志处理程序

本节展示了如何用管理 CLI 配置 Periodic Size 日志处理程序。您也可以使用管理控制台配置 Periodic Size 日志处理程序，您可以进入 **Logging** 子系统，然后选择 **Handler** 标签页，并从左侧的菜单选择 **Periodic Size**。

配置 Periodic Size 日志处理程序时要执行的主要任务是：

- 添加新的 [Periodic size](#) 日志处理程序。

- [配置 Periodic size 日志处理程序设置。](#)
- [分配 Periodic size 日志程序给 Logger。](#)



重要

如果您为日志配置集配置这个日志处理程序，命令将以 `/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/` 而不是 `/subsystem=logging/` 开始。

此外，如果您运行在受管域里，请在命令里先使用 `/profile=PROFILE_NAME`。

添加 Periodic Size 日志处理程序

在添加 Periodic Size 日志处理程序时，您必须用 **file** 属性指定文件路径，它由 **path** 和 **relative-to** 属性组成。请使用 **path** 属性来设置日志的文件路径，其中包括名称，例如 `my-log.log`。您也可以选择 **relative-to** 属性来设置 **path** 为相对路径，例如 `jboss.server.log.dir`。

您也必须用 **suffix** 属性为轮换日志文件设置后缀。它必须是 `java.text.SimpleDateFormat` 可以理解的格式，例如 `.yyyy-MM-dd-HH`。轮换的周期会根据这个后缀自动计算。

```
/subsystem=logging/periodic-size-rotating-file-
handler=PERIODIC_SIZE_HANDLER_NAME:add(file={path=FILE_PATH,relative-
to=RELATIVE_TO_PATH},suffix=SUFFIX)
```

配置 Periodic Size 日志处理程序设置

根据具体要求，您可能需要设置一个或多个 Periodic Size 日志处理程序属性。关于 Periodic Size 日志处理程序属性及其描述的完整列表，请参考 [Periodic Size 日志处理程序属性](#)。

- 设置日志级别。
设置处理程序的日志级别。默认值是 **ALL**。关于所有可用的选项，请参考 [日志级别](#)。

```
/subsystem=logging/periodic-size-rotating-file-
handler=PERIODIC_SIZE_HANDLER_NAME:write-
attribute(name=level,value=LEVEL)
```

- 设置轮换大小。
设置文件的最大值。当文件大小达到这个值时就需要进行轮换。默认值是 **2m**，亦即 2MB。

```
/subsystem=logging/periodic-size-rotating-file-
handler=PERIODIC_SIZE_HANDLER_NAME:write-attribute(name=rotate-size,
value=ROTATE_SIZE)
```

- 设置保留的备份日志的最大数目。
设置保留的备份数量。默认值是 **1**。

```
/subsystem=logging/periodic-size-rotating-file-
handler=PERIODIC_SIZE_HANDLER_NAME:write-attribute(name=max-backup-
index, value=MAX_BACKUPS)
```

- 设置是否在引导时轮换日志文件。
在默认情况下，服务器重启时不会创建新的日志文件。您可以将它设置为 **true** 在服务器重启时轮换日志文件。

```
/subsystem=logging/periodic-size-rotating-file-
handler=PERIODIC_SIZE_HANDLER_NAME:write-attribute(name=rotate-on-
boot, value=ROTATE_ON_BOOT)
```

- 设置附加行为。
在默认情况下，服务器重启时，JBoss EAP 将日志信息附加到相同的文件里。您可以设置 **append** 属性为 **false**，在服务器重启时覆盖这个文件。

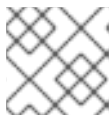
```
/subsystem=logging/periodic-size-rotating-file-
handler=PERIODIC_SIZE_HANDLER_NAME:write-
attribute(name=append, value=APPEND)
```

- 设置编码。
设置处理程序的编码，例如 **utf-8**。

```
/subsystem=logging/periodic-size-rotating-file-
handler=PERIODIC_SIZE_HANDLER_NAME:write-
attribute(name=encoding, value=ENCODING)
```

- 设置日志格式器。
设置处理程序的格式器字符串。例如，默认的格式字符串是 **%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%n**。请确保用引号包括 **FORMAT** 值。

```
/subsystem=logging/periodic-size-rotating-file-
handler=PERIODIC_SIZE_HANDLER_NAME:write-
attribute(name=formatter, value=FORMAT)
```



注意

如果您想引用 [saved formatter](#)，您可以使用 **named-formatter** 属性。

- 设置自动冲刷。
设置在每次写入后是否自动冲刷。默认值是 **true**。

```
/subsystem=logging/periodic-size-rotating-file-
handler=PERIODIC_SIZE_HANDLER_NAME:write-
attribute(name=autoflush, value=AUTO_FLUSH)
```

- 设置过滤器表达式
设置处理程序过滤日志信息的表达式。请确保转义任何逗号和引号并用引号括起表达式。例如，对于下面的可替换变量 **FILTER_EXPRESSION**，需要用 **"not(match(\"WFLY\"))"** 替换过滤表达式 **not(match("WFLY"))**。

```
/subsystem=logging/periodic-size-rotating-file-
handler=PERIODIC_SIZE_HANDLER_NAME:write-attribute(name=filter-spec,
value=FILTER_EXPRESSION)
```

关于可用的过滤器表达式的更多信息，请参考[过滤器表达式](#)章节。

分配 **Periodic Size** 日志处理程序给 **Logger**

要激活日志处理程序，您必须将其分配给一个 **Logger**。

下面的管理 CLI 命令将 Periodic Size 日志处理程序分配给 Root Logger。

```
/subsystem=logging/root-logger=ROOT:add-  
handler(name=PERIODIC_SIZE_HANDLER_NAME)
```

下面的管理 CLI 命令将 Periodic Size 日志处理程序分配给名称由 **CATEGORY** 指定的 Logger。

```
/subsystem=logging/logger=CATEGORY:add-  
handler(name=PERIODIC_SIZE_HANDLER_NAME)
```

删除 **Periodic Size** 日志处理程序

日志处理程序可以用 **remove** 操作删除。如果日志处理程序目前分配给了 logger 或异步日志处理程序，它就无法被删除。

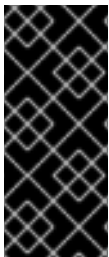
```
/subsystem=logging/periodic-size-rotating-file-  
handler=PERIODIC_SIZE_HANDLER_NAME:remove
```

12.5.6. 配置 Syslog 处理程序

本节展示了如何用管理 CLI 配置 Syslog 处理程序，它可以用来发送消息至支持 Syslog 协议（RFC-3164 或 RFC-5424）的远程日志服务器。您也可以用管理控制台配置 Syslog 处理程序，您可以先从 **Configuration** 标签页进入 **Logging** 子系统，然后选择 **Handler** 标签页，并从左侧的菜单选择 **Syslog**。

配置 Syslog 处理程序时要执行的主要任务是：

- 添加新的 Syslog 处理程序。
- 配置 Syslog 处理程序设置。
- 分配 Syslog 处理程序到 logger。



重要

如果您为日志配置集配置这个日志处理程序，命令将以 **/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/** 而不是 **/subsystem=logging/** 开始。

此外，如果您运行在受管域里，请在命令里先使用 **/profile=PROFILE_NAME**。

添加 **Syslog** 处理程序

```
/subsystem=logging/syslog-handler=SYSLOG_HANDLER_NAME:add
```

配置 **Syslog** 处理程序设置

根据具体要求，您可能需要设置一个或多个 Syslog 日志处理程序属性。关于 Syslog 日志处理程序属性及其描述的完整列表，请参考 [Syslog 处理程序属性](#)。

- 设置处理程序的日志级别。默认值是 **ALL**。关于所有可用的选项，请参考 [日志级别](#)。

```
/subsystem=logging/syslog-handler=SYSLOG_HANDLER_NAME:write-  
attribute(name=level,value=LEVEL)
```

- 设置要登记日志的应用程序的名称。其默认值是 **java**。

```
/subsystem=logging/syslog-handler=SYSLOG_HANDLER_NAME:write-attribute(name=app-name,value=APP_NAME)
```

- 设置 syslog 服务器的地址。默认地址是 **localhost**。

```
/subsystem=logging/syslog-handler=SYSLOG_HANDLER_NAME:write-attribute(name=server-address,value=SERVER_ADDRESS)
```

- 设置 syslog 服务器的端口。默认端口是 **514**。

```
/subsystem=logging/syslog-handler=SYSLOG_HANDLER_NAME:write-attribute(name=port,value=PORT)
```

- 设置 syslog 格式（由 RFC 规格定义）。默认格式是 **RFC5424**。

```
/subsystem=logging/syslog-handler=SYSLOG_HANDLER_NAME:write-attribute(name=syslog-format,value=SYSLOG_FORMAT)
```

分配 Syslog 处理程序给 Logger

要激活日志处理程序，您必须将其分配给一个 Logger。

下面的管理 CLI 命令将 Syslog 日志处理程序分配给 Root Logger。

```
/subsystem=logging/root-logger=ROOT:add-handler(name=SYSLOG_HANDLER_NAME)
```

下面的管理 CLI 命令将 Syslog 日志处理程序分配给名称由 **CATEGORY** 指定的 Logger。

```
/subsystem=logging/logger=CATEGORY:add-handler(name=SYSLOG_HANDLER_NAME)
```

删除 Syslog 处理程序

日志处理程序可以用 **remove** 操作删除。如果日志处理程序目前分配给了 logger 或异步日志处理程序，它就无法被删除。

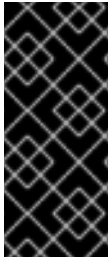
```
/subsystem=logging/syslog-handler=SYSLOG_HANDLER_NAME:remove
```

12.5.7. 配置自定义日志处理程序

本节展示了如何用管理 CLI 配置 custom 日志处理程序。您也可以用管理控制台配置 custom 日志处理程序，您可以先从 **Configuration** 标签页进入 **Logging** 子系统，然后选择 **Handler** 标签页，并从左侧的菜单选择 **File**。

配置 Custom 日志处理程序时要执行的主要任务是：

- 添加新的 Custom 日志处理程序。
- 配置 Custom 日志处理程序设置。
- 分配 Custom 日志处理程序到 logger。



重要

如果您为日志配置集配置这个日志处理程序，命令将以 `/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/` 而不是 `/subsystem=logging/` 开始。

此外，如果您运行在受管域里，请在命令里先使用 `/profile=PROFILE_NAME`。

添加 Custom 日志处理程序

在添加 Custom 日志处理程序时，您必须指定处理程序的 Java 类及它所在的 JBoss EAP 模块。这个类必须继承 `java.util.logging.Handler`。



注意

您必须已经创建了包含自定义 logger 的模块，否则这个命令将无法运行。

```
/subsystem=logging/custom-  
handler=CUSTOM_HANDLER_NAME:add(class=CLASS_NAME,module=MODULE_NAME)
```

配置 Custom 日志处理程序设置

根据具体要求，您可能需要设置一个或多个 Custom 日志处理程序属性。关于 Custom 日志处理程序属性及其描述的完整列表，请参考 [Custom 日志处理程序属性](#)。

- 设置日志级别。
设置处理程序的日志级别。默认值是 **ALL**。关于所有可用的选项，请参考 [日志级别](#)。

```
/subsystem=logging/custom-handler=CUSTOM_HANDLER_NAME:write-  
attribute(name=level,value=LEVEL)
```

- 设置属性。
设置日志处理程序所必需的属性。这些属性必须可以通过 setter 方法访问。

```
/subsystem=logging/custom-handler=CUSTOM_HANDLER_NAME:write-  
attribute(name=properties.PROPERTY_NAME,value=PROPERTY_VALUE)
```

- 设置编码。
设置处理程序的编码，例如 **utf-8**。

```
/subsystem=logging/custom-handler=CUSTOM_HANDLER_NAME:write-  
attribute(name=encoding,value=ENCODING)
```

- 设置日志格式器。
设置处理程序的格式器字符串。例如，默认的格式字符串是 `%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%n`。请确保用引号包括 **FORMAT** 值。

```
/subsystem=logging/custom-handler=CUSTOM_HANDLER_NAME:write-  
attribute(name=formatter,value=FORMAT)
```



注意

如果您想引用 [saved formatter](#)，您可以使用 **named-formatter** 属性。

- 设置过滤器表达式
设置处理程序过滤日志信息的表达式。请确保转义任何逗号和引号并用引号括起表达式。例如，对于下面的可替换变量 **FILTER_EXPRESSION**，需要用 `"not(match(\"WFLY\"))"` 替换过滤表达式 `not(match("WFLY"))`。

```
/subsystem=logging/custom-handler=CUSTOM_HANDLER_NAME:write-attribute(name=filter-spec, value=FILTER_EXPRESSION)
```

关于可用的过滤器表达式的更多信息，请参考[过滤器表达式](#)章节。

分配 **Custom** 日志处理程序给 **Logger**

要激活日志处理程序，您必须将其分配给一个 Logger。

下面的管理 CLI 命令将 Custom 日志处理程序分配给 Root Logger。

```
/subsystem=logging/root-logger=ROOT:add-handler(name=CUSTOM_HANDLER_NAME)
```

下面的管理 CLI 命令将 Custom 日志处理程序分配给名称由 **CATEGORY** 指定的 Logger。

```
/subsystem=logging/logger=CATEGORY:add-handler(name=CUSTOM_HANDLER_NAME)
```

删除 **Custom** 日志处理程序

日志处理程序可以用 **remove** 操作删除。如果日志处理程序目前分配给了 logger 或异步日志处理程序，它就无法被删除。

```
/subsystem=logging/custom-handler=CUSTOM_HANDLER_NAME:remove
```

12.5.8. 配置 **Async** 日志处理程序

本节展示了如何用管理 CLI 配置 Async 日志处理程序。您也可以用管理控制台配置 Async 日志处理程序，您可以先从 **Configuration** 标签页进入 **Logging** 子系统，然后选择 **Handler** 标签页，并从左侧的菜单选择 **Async**。

配置 Async 日志处理程序时要执行的主要任务是：

- 添加新的 [Async 日志处理程序](#)。
- 添加子处理程序到 [Async 日志处理程序](#)。
- 配置 [Async 日志处理程序设置](#)。
- 分配 [Async 日志处理程序到 logger](#)。



重要

如果您为日志配置集配置这个日志处理程序，命令将以 `/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/` 而不是 `/subsystem=logging/` 开始。

此外，如果您运行在受管域里，请在命令里先使用 `/profile=PROFILE_NAME`。

添加 **Async** 日志处理程序

当添加 Async 日志处理程序时，您必须指定队列长度。这是可以保持在队列里的日志请求的最大数目。

```
/subsystem=logging/async-handler=ASYNC_HANDLER_NAME:add(queue-length=QUEUE_LENGTH)
```

添加子处理程序

您可以添加一个或多个处理程序作为这个 Async 日志处理程序的子处理程序。请注意，这些处理程序必须已存在于配置里，否则这个命令会无法运行。

```
/subsystem=logging/async-handler=ASYNC_HANDLER_NAME:add-handler(name=HANDLER_NAME)
```

配置 Async 日志处理程序设置

根据具体要求，您可能需要设置一个或多个 Async 日志处理程序属性。关于 Async 日志处理程序属性及其描述的完整列表，请参考 [Async 日志处理程序属性](#)。

- 设置日志级别。
设置处理程序的日志级别。默认值是 **ALL**。关于所有可用的选项，请参考 [日志级别](#)。

```
/subsystem=logging/async-handler=ASYNC_HANDLER_NAME:write-attribute(name=level,value=LEVEL)
```

- 设置溢出操作。
设置溢出时进行的操作。默认值是 **BLOCK**，表示线程在队列已满时会阻塞。您可以将这个值改为 **DISCARD**，也就是在队列已满时丢弃日志信息。

```
/subsystem=logging/async-handler=ASYNC_HANDLER_NAME:write-attribute(name=overflow-action,value=OVERFLOW_ACTION)
```

- 设置过滤器表达式
设置处理程序过滤日志信息的表达式。请确保转义任何逗号和引号并用引号括起表达式。例如，对于下面的可替换变量 **FILTER_EXPRESSION**，需要用 `"not(match(\"WFLY\"))"` 替换过滤表达式 `not(match("WFLY"))`。

```
/subsystem=logging/async-handler=ASYNC_HANDLER_NAME:write-attribute(name=filter-spec,value=FILTER_EXPRESSION)
```

关于可用的过滤器表达式的更多信息，请参考 [过滤器表达式](#) 章节。

分配 Async 日志处理程序给 Logger

要激活日志处理程序，您必须将其分配给一个 Logger。

下面的管理 CLI 命令将 Async 日志处理程序分配给 Root Logger。

```
/subsystem=logging/root-logger=ROOT:add-handler(name=ASYNC_HANDLER_NAME)
```

下面的管理 CLI 命令将 Async 日志处理程序分配给名称由 **CATEGORY** 指定的 Logger。

```
/subsystem=logging/logger=CATEGORY:add-handler(name=ASYNC_HANDLER_NAME)
```

删除 Async 日志处理程序

日志处理程序可以用 **remove** 操作删除。但如果日志处理程序目前分配给了 logger，它就无法被删除。

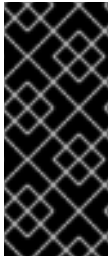
```
/subsystem=logging/async-handler=ASYNC_HANDLER_NAME:remove
```

12.6. 配置 ROOT LOGGER

root logger 抓取所有发送到服务器上的、没有被任何日志类别抓取的日志消息（等于或高于特定级别）。

本节展示了如何用管理 CLI 配置 Root Logger。您也可以用管理控制台配置 Root Logger，您可以先从 **Configuration** 标签页进入 **Logging** 子系统，然后选择 **Root Logger** 标签页。

配置 Root Logger



重要

如果您为日志配置集配置这个日志处理程序，命令将以 `/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/` 而不是 `/subsystem=logging/` 开始。

此外，如果您运行在受管域里，请在命令里先使用 `/profile=PROFILE_NAME`。

1. 将日志处理程序添加至 Root Logger。
添加日志处理程序。

```
/subsystem=logging/root-logger=ROOT:add-  
handler(name=LOG_HANDLER_NAME)
```

删除日志处理程序。

```
/subsystem=logging/root-logger=ROOT:remove-  
handler(name=LOG_HANDLER_NAME)
```

2. 设置日志级别。

```
/subsystem=logging/root-logger=ROOT:write-  
attribute(name=level,value=LEVEL)
```

关于可用的 Root Logger 属性及其描述的完整列表，请参考 [Root Logger 属性](#)。

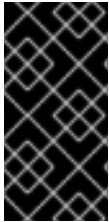
12.7. 配置日志格式器

日志格式器定义处理程序的日志信息的外观。您可以配置[命名模式格式器](#)或[自定义日志格式器](#)。

12.7.1. 配置命名模式格式器（Named Pattern Formatter）

您可以创建日志处理程序用来格式化日志信息的命名模式格式器。

本节展示了如何用管理 CLI 配置日志格式器。您也可以用管理控制台配置日志格式器，您可以先从 **Configuration** 标签页进入 **Logging** 子系统，然后选择 **Formatter** 标签页，并从左侧的菜单选择 **Pattern**。

**重要**

如果您为日志配置集配置这个日志格式器，命令将以 `/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/` 而不是 `/subsystem=logging/` 开始。

此外，如果您运行在受管域里，请在命令里先使用 `/profile=PROFILE_NAME`。

创建命名格式器

在定义格式器时，您要提供模式字符串来格式化日志消息。关于模式语法的详情，请参考[日志格式器](#)。

```
/subsystem=logging/pattern-formatter=PATTERN_FORMATTER_NAME:add(pattern=PATTERN_STRING)
```

您也可以定义一个颜色表来为不同的日志级别分配不同颜色。其格式是用逗号分隔的 **LEVEL:COLOR** 列表。

- 有效级别：**finest, finer, fine, config, trace, debug, info, warning, warn, error, fatal, severe**
- 有效颜色：**black, green, red, yellow, blue, magenta, cyan, white, brightblack, brightred, brightgreen, brightblue, brightyellow, brightmagenta, brightcyan, brightwhite**

```
/subsystem=logging/pattern-formatter=PATTERN_FORMATTER_NAME:write-attribute(name=color-map,value="LEVEL:COLOR,LEVEL:COLOR")
```

为日志处理程序分配命名格式器

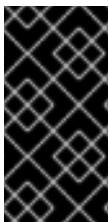
下面的管理 CLI 命令分配了一个由定期轮换文件处理程序使用的模式格式器。

```
/subsystem=logging/periodic-rotating-file-handler=FILE_HANDLER_NAME:write-attribute(name=named-formatter,value=PATTERN_FORMATTER_NAME)
```

12.7.2. 配置自定义日志格式器

您也可以创建日志处理程序和格式化消息都可以使用的自定义日志格式器。

本节展示了如何用管理 CLI 配置自定义日志格式器。您也可以使用管理控制台配置日志格式器，您可以先从 **Configuration** 标签页进入 **Logging** 子系统，然后选择 **Formatter** 标签页，并从左侧的菜单选择 **Custom**。

配置自定义日志格式器**重要**

如果您为日志配置集配置这个日志格式器，命令将以 `/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/` 而不是 `/subsystem=logging/` 开始。

此外，如果您运行在受管域里，请在命令里先使用 `/profile=PROFILE_NAME`。

1. 添加自定义日志格式器。

在添加 Custom 日志格式器时，您必须指定格式器的 Java 类及它所在的 JBoss EAP 模块。这个类必须继承 `java.util.logging.Formatter`。



注意

您必须也创建一个包含自定义格式器的模块，否则这个命令会执行失败。

```
/subsystem=logging/custom-  
formatter=CUSTOM_FORMATTER_NAME:add(class=CLASS_NAME,  
module=MODULE_NAME)
```

2. 为日志格式器设置必要的属性。
这些属性必须用 `setter` 方法来访问。

```
/subsystem=logging/custom-formatter=CUSTOM_FORMATTER_NAME:write-  
attribute(name=properties.PROPERTY_NAME,value=PROPERTY_VALUE)
```

3. 为日志处理程序分配自定义格式器
下面的管理 CLI 命令分配了一个由定期轮换文件处理程序使用的自定义格式器。

```
/subsystem=logging/periodic-rotating-file-  
handler=FILE_HANDLER_NAME:write-attribute(name=named-formatter,  
value=CUSTOM_FORMATTER_NAME)
```

自定义 XML 格式器示例

下面的例子配置了一个自定义 XML 格式器。它使用 `org.jboss.logmanager` 模块里提供的 `java.util.logging.XMLFormatter` 类并将其分配至控制台日志处理程序。

```
/subsystem=logging/custom-formatter=custom-xml-  
formatter:add(class=java.util.logging.XMLFormatter,  
module=org.jboss.logmanager)  
/subsystem=logging/console-handler=CONSOLE:write-attribute(name=named-  
formatter, value=custom-xml-formatter)
```

使用这个格式器的日志消息将有如下格式。

```
<record>  
  <date>2016-03-23T12:58:13</date>  
  <millis>1458752293091</millis>  
  <sequence>93963</sequence>  
  <logger>org.jboss.as</logger>  
  <level>INFO</level>  
  <class>org.jboss.as.server.BootstrapListener</class>  
  <method>logAdminConsole</method>  
  <thread>22</thread>  
  <message>WFLYSRV0051: Admin console listening on http://%s:%d</message>  
  <param>127.0.0.1</param>  
  <param>9990</param>  
</record>
```

12.8. 关于应用程序日志

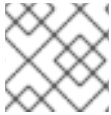
应用程序的日志可以用 JBoss EAP 的 `logging` 子系统配置或对每个部署进行配置。

关于使用 JBoss EAP 日志类别和处理程序来采集日志信息，请参考[关于日志子系统](#)。

关于应用程序日志的更多信息，如支持的应用程序日志框架和配置 per-deployment 日志，请参考《JBoss EAP 部署指南》里的 [Logging for Developers](#) 章节。

12.8.1. Per-deployment 日志

Per-deployment 日志允许开发人员提前为应用程序配置日志。当部署应用程序时，日志按照预先定义的配置进行记录。通过这个配置创建的日志文件只包含和应用程序行为相关的信息。



注意

如果没有进行部署前的日志设置，则应用程序和服务器会使用 **logging** 子系统的设置。

和使用系统范围的日志功能相比，这个方法有利有弊。它的优点是，JBoss EAP 实例的管理员不需要配置除服务器日志以外的任何日志功能。而缺点是，部署前的日志配置只在启动时可读，因此无法在运行时进行修改。

关于在您的应用程序里使用 per-deployment 日志的说明，请参考《JBoss EAP 部署指南》里的『[添加 Per-deployment 日志到应用程序](#)』章节。

12.8.1.1. 禁用 Per-deployment 日志

您可以用下列方法之一禁用 Per-deployment 日志

- 设置 **use-deployment-logging-config** 属性为 **false**。
use-deployment-logging-config 属性控制是否您的部署启用 per-deployment 日志。它的默认值为 **true**。您可以设置这个属性为 **false** 来禁用 per-deployment 日志。

```
/subsystem=logging:write-attribute(name=use-deployment-logging-config,value=false)
```

- 用 **jboss-deployment-structure.xml** 文件排除 **logging** 子系统。
相关的说明，请参考《JBoss EAP 部署指南》里的[从部署里排除子系统](#)章节。

12.8.2. 日志配置集

日志配置集是独立的日志配置组，它可以分配给部署的应用程序。作为一般的 **logging** 子系统，日志配置集可以定义 handler、类型以及一个根日志程序，但它无法参考其它配置集或主 **logging** 子系统的配置。日志配置集的结构和 **logging** 子系统一样，这样可以减轻配置负担。

日志配置集允许管理员，在不影响其它日志配置的情况下，针对一个或多个应用程序创建日志配置。因为每个配置集都是在服务器配置中定义的，所以可以在不重新部署受影响的应用程序的情况下，修改日志配置。

每个日志配置集可以有：

- 一个唯一的名称（必需）
- 任意数量的日志处理程序
- 任意数量的日志类别
- 最多一个 Root Logger

应用程序可以在它的 **MANIFEST.MF** 文件里用 **Logging-Profile** 属性指定一个日志配置集。

12.8.2.1. 配置日志配置集

日志配置集可用日志处理程序、类别和 Root Logger 来配置。它使用与主 **logging** 子系统几乎完全相同的语法，除了两点不同：

- 根配置路径是 **/subsystem=logging/logging-profile=NAME**。
- 日志配置集不能包含其他日志配置集。

创建和配置日志配置集

下面的过程使用管理 CLI 来创建日志配置集并设置了一个文件处理程序和 Logger 类别。

1. 创建日志配置集。

```
/subsystem=logging/logging-profile=PROFILE_NAME:add
```

2. 创建文件处理程序。

```
/subsystem=logging/logging-profile=PROFILE_NAME/file-  
handler=FILE_HANDLER_NAME:add(file={path=>"LOG_NAME.log", "relative-  
to"=>"jboss.server.log.dir"})
```

```
/subsystem=logging/logging-profile=PROFILE_NAME/file-  
handler=FILE_HANDLER_NAME:write-attribute(name="level",  
value="DEBUG")
```

3. 创建 Logger 类别。

```
/subsystem=logging/logging-  
profile=PROFILE_NAME/logger=CATEGORY_NAME:add(level=TRACE)
```

4. 分配文件处理程序给类别。

```
/subsystem=logging/logging-  
profile=PROFILE_NAME/logger=CATEGORY_NAME:add-  
handler(name="FILE_HANDLER_NAME")
```

您可以在应用程序的 **MANIFEST.MF** 文件中设置日志配置集。如需了解更详细的信息，请参阅 *开发指南* 中的 [Specify a Logging Profile in an Application](#)。

12.8.2.2. 日志配置集示例

这个例子展示了日志配置集和使用它的应用程序的配置。它也展示了管理 CLI 命令、结果 XML 和应用程序的 **MANIFEST.MF** 文件。

这个日志配置集示例具有下列特点：

- 它的名称是 **accounts-app-profile**。
- 日志类别是 **com.company.accounts.ejbs**。
- 日志级别是 **TRACE**。
- 日志处理程序是使用 **ejb-trace.log** 的文件处理程序。

管理 CLI 会话

```

/subsystem=logging/logging-profile=accounts-app-profile:add

/subsystem=logging/logging-profile=accounts-app-profile/file-handler=ejb-
trace-file:add(file={path=>"ejb-trace.log", "relative-
to"=>"jboss.server.log.dir"})

/subsystem=logging/logging-profile=accounts-app-profile/file-handler=ejb-
trace-file:write-attribute(name="level", value="DEBUG")

/subsystem=logging/logging-profile=accounts-app-
profile/logger=com.company.accounts.ejbs:add(level=TRACE)

/subsystem=logging/logging-profile=accounts-app-
profile/logger=com.company.accounts.ejbs:add-handler(name="ejb-trace-
file")

```

XML 配置

```

<logging-profiles>
  <logging-profile name="accounts-app-profile">
    <file-handler name="ejb-trace-file">
      <level name="DEBUG"/>
      <file relative-to="jboss.server.log.dir" path="ejb-trace.log"/>
    </file-handler>
    <logger category="com.company.accounts.ejbs">
      <level name="TRACE"/>
      <handlers>
        <handler name="ejb-trace-file"/>
      </handlers>
    </logger>
  </logging-profile>
</logging-profiles>

```

应用程序的 MANIFEST.MF 文件

```

Manifest-Version: 1.0
Logging-Profile: accounts-app-profile

```

12.8.3. 查看部署日志配置

您可以用下列管理 CLI 命令获得特定部署的日志配置信息。

```

/deployment=DEPLOYMENT_NAME/subsystem=logging/configuration=CONFIG:read-
resource

```

部署日志的 **configuration** 值可以是：

- 如果部署使用 **logging 子系统**，则是 **default**。这将输出 **logging** 子系统配置。
- 如果部署使用 **logging** 子系统里定义的 **日志配置集**，则是 **profile-PROFILE_NAME**。这将输出日志配置集配置。

- 被使用的配置文件的路径，例如：`myear.ear/META-INF/logging.properties`。

例如，下面的管理 CLI 命令显示指定的部署使用的 **MYPROFILE** 日志配置集的配置。

```
/deployment=mydeployment.war/subsystem=logging/configuration=profile-
MYPROFILE:read-resource(recursive=true,include-runtime=true)
```

这将输出下列信息。

```
{
  "outcome" => "success",
  "result" => {
    "error-manager" => undefined,
    "filter" => undefined,
    "formatter" => {
      "MYFORMATTER" => {
        "class-name" =>
"org.jboss.logmanager.formatters.PatternFormatter",
        "module" => undefined,
        "properties" => {"pattern" => "%d{HH:mm:ss,SSS} %-5p [%c]
(%t) %s%e%n"}
      }
    },
    "handler" => {
      "MYPERIODIC" => {
        "class-name" =>
"org.jboss.logmanager.handlers.PeriodicRotatingFileHandler",
        "encoding" => undefined,
        "error-manager" => undefined,
        "filter" => undefined,
        "formatter" => "MYFORMATTER",
        "handlers" => [],
        "level" => "ALL",
        "module" => undefined,
        "properties" => {
          "append" => "true",
          "autoFlush" => "true",
          "enabled" => "true",
          "suffix" => ".yyyy-MM-dd",
          "fileName" =>
"EAP_HOME/standalone/log/deployment.log"
        }
      }
    },
    "logger" => {"MYCATEGORY" => {
      "filter" => undefined,
      "handlers" => [],
      "level" => "DEBUG",
      "use-parent-handlers" => true
    }},
    "pojo" => undefined
  }
}
```

您也可以使用递归的 **read-resource** 操作来读取部署的日志配置和其他信息。

```
/deployment=DEPLOYMENT_NAME/subsystem=logging:read-resource(include-  
runtime=true, recursive=true)
```

第 13 章 数据源管理

13.1. 关于 JBOSS EAP 数据源

关于 JDBC

JDBC API 是一个标准，它定义了 Java 应用程序如何访问数据库。一个应用程序需要配置一个数据源（datasource）来指代一个 JDBC 驱动，然后，应用程序代码就可以对驱动进行写操作，而不需要直接对数据库进行写操作。驱动会把相应的代码转换为数据库的语言。这意味着，通过安装正确的驱动，应用程序可以使用任何支持的数据库。

更多信息请参考 [JDBC 4.0 规格](#)。

支持的数据库

关于 JBoss EAP 7 支持的兼容 JDBC 的数据库的列表，请参考 [JBoss EAP 支持的配置](#)。

数据源类型

数据源一般被分为两类：*非 XA 数据源*和*XA 数据源*。

非 XA 数据源

用于不使用事务的应用程序，或者使用的事务只有一个数据库的应用程序。

XA 数据源

将多个数据库或其他 XA 资源作为 XA 事务的一部分使用的应用程序会使用它。XA 数据源会导致额外的负荷。

当使用 JBoss EAP 管理接口创建数据源时，您需要指定所用的数据源类型。

ExampleDS 数据源

JBoss EAP 附带了数据源配置示例（*ExampleDS*），它演示了如何定义数据源。这个数据源使用一个 H2 数据库，它是一个轻量级关系数据库管理系统，为开发人员提供了快速构建应用程序的功能。



警告

ExampleDS 数据源和 H2 数据库不应该用在生产环境里。这是一个非常小的、自包含的数据源，它支持所有测试和构建应用程序所需的标准，但对于生产环境，它不够稳定且扩展性不强。

13.2. JDBC 驱动

在 JBoss EAP 里定义数据源之前，您必须首先安装合适的 JDBC 驱动。

13.2.1. 将 JDBC 驱动安装为核心模块

按照以下步骤，使用管理 CLI 把 JDBC 驱动安装为核心模块。

1. 下载 JDBC 驱动。

从您的数据库厂商下载合适的 JDBC 驱动。关于常用数据库的 JDBC 驱动的标准下载位置，请参考 [JDBC 驱动下载位置](#)。

如果 JDBC 驱动 JAR 文件包含在 ZIP 或 TAR 归档里，请确保解压这个归档文件。

2. 启动 JBoss EAP 服务器。
3. 启动管理 CLI，但请不要使用 **--connect** 或 **-c** 参数来连接运行的实例。

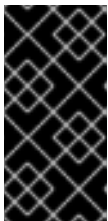
```
$ EAP_HOME/bin/jboss-cli.sh
```

4. 使用 **module add** 管理 CLI 命令来添加新的核心模块。

```
module add --name=MODULE_NAME --resources=PATH_TO_JDBC_JAR --
dependencies=DEPENDENCIES
```

例如，下列命令添加了一个 MySQL JDBC 驱动模块。

```
module add --name=com.mysql --resources=/path/to/mysql-connector-
java-5.1.36-bin.jar --dependencies=javax.api,javax.transaction.api
```



重要

Using the **module** management CLI command to add and remove modules is provided as [technology preview](#) only. This command is not appropriate for use in a managed domain or when connecting to the management CLI remotely. Modules should be [added](#) and [removed](#) manually in a production environment.

关于使用这个命令添加或删除模块的详情，请执行 **module --help**。

5. 使用 **connect** 管理 CLI 命令来连接运行的实例。

```
connect
```

6. 注册 JDBC 驱动。当运行在受管域里时，请确保在命令之前使用 **/profile=PROFILE_NAME**。

```
/subsystem=datasources/jdbc-driver=DRIVER_NAME:add(driver-
name=DRIVER_NAME,driver-module-name=MODULE_NAME,driver-xa-
datasource-class-name=XA_DATASOURCE_CLASS_NAME,driver-class-
name=DRIVER_CLASS_NAME)
```



注意

driver-class-name 参数只有在 JDBC 驱动 JAR 在其 **/META-INF/services/java.sql.Driver** 文件里定义了多个类时才需要。

例如，MySQL 5.1.36 JDBC 驱动 JAR 里的 **/META-INF/services/java.sql.Driver** 文件定义了两个类：

- `com.mysql.jdbc.Driver`
- `com.mysql.fabric.jdbc.FabricMySQLDriver`

在这例子里，您会传入 **driver-class-name=com.mysql.jdbc.Driver**。

例如，下列命令注册了 MySQL JDBC 驱动。

```
/subsystem=datasources/jdbc-driver=mysql:add(driver-
name=mysql,driver-module-name=com.mysql,driver-xa-datasource-class-
name=com.mysql.jdbc.jdbc2.optional.MysqlXADataSource, driver-class-
name=com.mysql.jdbc.Driver)
```

JDBC 驱动现在可被应用程序数据源所引用了。

13.2.2. 将 JDBC 驱动安装为 JAR 部署

JDBC 驱动可以用管理 CLI 或管理控制台安装为 JAR 部署。只要这个驱动兼容 JDBC 4，它就可以自动被识别并在部署时被安装为 JDBC 驱动。

下列步骤描述了如何用管理 CLI 安装 JDBC 驱动。



注意

我们推荐的安装 JDBC 驱动的方法是将其安装为[核心模块](#)。

1. 下载 JDBC 驱动。

从您的数据库厂商下载合适的 JDBC 驱动。关于常用数据库的 JDBC 驱动的标准下载位置，请参考[JDBC 驱动的下載位置](#)。

如果 JDBC 驱动 JAR 文件包含在 ZIP 或 TAR 归档里，请确保解压这个归档文件。

2. 如果 JDBC 驱动不兼容 JDBC 4，请遵照[更新 JDBC 驱动 JAR 使其兼容 JDBC 4](#) 里的步骤。

3. 部署 JAR 至 JBoss EAP。

```
部署 PATH_TO_JDBC_JAR
```



注意

在受管域里，指定合适的服务器组。

例如，下列命令部署了 MySQL JDBC 驱动。

```
deploy /path/to/mysql-connector-java-5.1.36-bin.jar
```

JBoss EAP 服务器日志里将显示一条含有定义数据源时使用的驱动名的消息。

```
WFLYJCA0018: Started Driver service with driver-name = mysql-
connector-java-5.1.36-bin.jar_com.mysql.jdbc.Driver_5_1
```

JDBC 驱动现在可被应用程序数据源所引用了。

将 JDBC 驱动 JAR 更新为兼容 JDBC 4

如果 JDBC JAR 不兼容 JDBC 4，请执行以下操作。

1. 创建一个空的临时目录。
2. 创建一个 **META-INF** 子目录。

- 3. 创建一个 **META-INF/services** 子目录。
- 4. 创建一个 **META-INF/services/java.sql.Driver** 文件并添加一行指定 JDBC 驱动的全限定类名。
例如，对于 MySQL JDBC 驱动将添加下列命令行。

```
com.mysql.jdbc.Driver
```

- 5. 使用 JAR 命令行工具在 JAR 文件里添加这个新的文件。

```
jar \-uf jdbc-driver.jar META-INF/services/java.sql.Driver
```

13.2.3. JDBC 驱动的下载位置

下表提供了 JBoss EAP 使用的常用数据库的 JDBC 驱动的标准下载位置。



注意

这些链接指向不受红帽控制的第三方网站。如需获得数据库的最新驱动，请访问相关厂商的文档和网站。

表 13.1. JDBC 驱动的下载位置

提供商	下载位置
MySQL	http://www.mysql.com/products/connector/
PostgreSQL	http://jdbc.postgresql.org/
Oracle	http://www.oracle.com/technetwork/database/features/jdbc/index-091264.html
IBM	http://www-306.ibm.com/software/data/db2/java/
Sybase	jConnect JDBC 驱动是 SAP ASE SDK 的一部分。目前这个驱动还没有单独的下载站点。
Microsoft	http://msdn.microsoft.com/data/jdbc/

13.2.4. 访问厂商专有的类

在某些情况下，应用程序需要使用厂商专用的、不属于 JDBC API 的功能。此时，您可以通过在应用程序里声明依赖关系来访问厂商专有的 API。



警告

这是高级的用法。只有应用程序需要使用 JDBC API 里没有的功能时才执行这个操作。



重要

当使用重验证机制，并访问厂商的专有类时，需要执行这个操作。

您可以用 **MANIFEST.MF** 文件或 **jboss-deployment-structure.xml** 文件来定义依赖关系。

如果您还没有这样做，请将 [JDBC 驱动安装为核心模块](#)。

使用 **MANIFEST.MF** 文件

1. 编辑应用程序的 **META-INF/MANIFEST.MF** 文件。
2. 添加 **Dependencies** 行并指定模块名称。
例如，下列行将 **com.mysql** 模块声明为依赖关系。

```
Dependencies: com.mysql
```

使用 **jboss-deployment-structure.xml** 文件

1. 在应用程序的 **META-INF/** 或 **WEB-INF/** 文件夹里创建一个名为 **jboss-deployment-structure.xml** 的文件。
2. 用 **dependencies** 元素指定模块。
例如，下面的 **jboss-deployment-structure.xml** 文件将 **com.mysql** 模块声明为依赖关系。

```
<jboss-deployment-structure>
  <deployment>
    <dependencies>
      <module name="com.mysql"/>
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

下面的示例代码访问了 MySQL API。

```
import java.sql.Connection;
import org.jboss.jca.adapters.jdbc.WrappedConnection;

...

Connection c = ds.getConnection();
WrappedConnection wc = (WrappedConnection)c;
com.mysql.jdbc.Connection mc = wc.getUnderlyingConnection();
```

**重要**

请严格按照厂商专有的 API 准则，因为连接是由 IronJacamar 容器所控制的。

13.3. 创建数据源

数据源可以用管理控制台或管理 CLI 来创建。

JBoss EAP 7 允许您在数据源属性值里使用表达式，例如 **enabled** 属性。关于在配置里使用表达式的细节，请参考[属性替换](#)章节。

13.3.1. 创建非 XA 数据源

非 XA 数据源可以用 **data-source add** 管理 CLI 命令来定义。您也可以使用管理控制台定义非 XA 数据源，请选择 **Configuration** → **Subsystems** → **Datasources** → **Non-XA** 并点击 **Add** 打开 **Create Datasource** 向导。

下面的步骤描述了如何用管理 CLI 定义非 XA 数据源。

1. 如果您还没有这样做，请将 [JDBC 驱动安装和注册为核心模块](#)。
2. 请用 **data-source add** 命令定义数据源，并指定合适的参数值。

```
data-source add --name=DATASOURCE_NAME --jndi-name=JNDI_NAME --
driver-name=DRIVER_NAME --connection-url=CONNECTION_URL
```

**注意**

在受管域里，您必须指定 **--profile=PROFILE_NAME** 参数。

关于这些参数值的详情，请参考下面的[数据源参数](#)章节。

关于详细的例程，请参考[数据源配置示例](#)。

数据源参数

jndi-name

数据源的 JNDI 名称必须以 **java:/** 或 **java:jboss/** 开始。例如，**java:jboss/datasources/ExampleDS**。

driver-name

驱动名称的值取决于 JDBC 驱动是否被安装为核心模块或 JAR 部署。

1. 对于核心模块，驱动名的值将是注册时赋予 JDBC 驱动的名称。
2. 对于 JAR 部署，如果 **/META-INF/services/java.sql.Driver** 文件里只列出了一个类，驱动名就是 JAR 的名称。如果列出了多个类，那它的值是 **JAR_NAME + "_" + DRIVER_CLASS_NAME + "_" + MAJOR_VERSION + "_" + MINOR_VERSION**（例如 **mysql-connector-java-5.1.36-bin.jar_com.mysql.jdbc.Driver_5_1**）。当部署 JDBC JAR 时，您也可以在 JBoss EAP 服务器日志里查看驱动名称。

```
WFLYJCA0018: Started Driver service with driver-name = mysql-
connector-java-5.1.36-bin.jar_com.mysql.jdbc.Driver_5_1
```


connection-url

关于受支持的数据库的连接 URL 格式的 details, 请参考[数据源连接 URL](#) 列表。

关于所有可用的数据源参数的完整列表, 请参考[数据源参数](#)章节。

13.3.2. 创建 XA 数据源

XA 数据源可以用 **xa-data-source add** 管理 CLI 命令来定义。您也可以用管理控制台定义 XA 数据源, 请选择 **Configuration** → **Subsystems** → **Datasources** → **XA** 并点击 **Add** 打开 **Create XA Datasource** 向导。

下面的步骤描述了如何用管理 CLI 定义 XA 数据源。



注意

在受管域里, 您需要指定使用的配置集。根据管理 CLI 命令的格式, 您要么在命令前使用 **/profile=PROFILE_NAME**, 要么传入 **--profile=PROFILE_NAME** 参数。

1. 如果您还没有这样做, 请将 [JDBC 驱动安装和注册为核心模块](#)。
2. 请用 **xa-data-source add** 命令定义数据源, 并指定合适的参数值。

```
xa-data-source add --name=XA_DATASOURCE_NAME --jndi-name=JNDI_NAME -
-driver-name=DRIVER_NAME --xa-datasource-class=XA_DATASOURCE_CLASS -
-xa-datasource-properties=
{"ServerName"=>"HOSTNAME", "DatabaseName"=>"DATABASE_NAME"}
```

关于这些参数值的详情, 请参考下面的[数据源参数](#)一节。

3. 设置 XA 数据源属性。

当定义 XA 数据源时要求至少有一个 XA 数据源属性, 否则您在添加它时会遇到错误。任何在定义 XA 数据源时没有设置的属性都可以在之后单独进行设置。

- a. 设置服务器名称。

```
/subsystem=datasources/xa-data-source=XA_DATASOURCE_NAME/xa-
datasource-properties=ServerName:add(value=HOSTNAME)
```

- b. 设置数据库名称。

```
/subsystem=datasources/xa-data-source=XA_DATASOURCE_NAME/xa-
datasource-properties=DatabaseName:add(value=DATABASE_NAME)
```

关于详细的例程, 请参考[数据源配置示例](#)。

数据源参数

jndi-name

数据源的 JNDI 名称必须以 **java:/** 或 **java:jboss/** 开始。例如, **java:jboss/datasources/ExampleDS**。

driver-name

驱动名称的值取决于 JDBC 驱动是否被安装为核心模块或 JAR 部署。

1. 对于核心模块，驱动名的值将是注册时赋予 JDBC 驱动的名称。
2. 对于 JAR 部署，如果 `/META-INF/services/java.sql.Driver` 文件里只列出了一个类，驱动名就是 JAR 的名称。如果列出了多个类，那它的值是 `JAR_NAME + "_" + DRIVER_CLASS_NAME + "_" + MAJOR_VERSION + "_" + MINOR_VERSION`（例如 `mysql-connector-java-5.1.36-bin.jar_com.mysql.jdbc.Driver_5_1`）。当部署 JDBC JAR 时，您也可以在 JBoss EAP 服务器日志里查看驱动名称。

```
WFLYJCA0018: Started Driver service with driver-name = mysql-connector-java-5.1.36-bin.jar_com.mysql.jdbc.Driver_5_1
```

xa-datasource-class

为 `javax.sql.XADataSource` 类的 JDBC 驱动实现指定 XA 数据源类。

-properties

当定义 XA 数据源时要求至少有一个 XA 数据源属性，否则您在添加它时会遇到错误。在定义之后可以在 XA 数据源里添加其他的属性。

关于所有可用的数据源参数的完整列表，请参考[数据源参数](#)章节。

13.4. 修改数据源

使用管理控制台或管理 CLI 可以修改数据源。

JBoss EAP 7 允许您在数据源属性值里使用表达式，例如 `enabled` 属性。关于在配置里使用表达式的细节，请参考[属性替换](#)章节。

13.4.1. 修改非 XA 数据源

非 XA 数据源设置可以用 `data-source` 管理 CLI 命令更新。您也可以从管理控制台的 `datasources` 子系统里更新数据源属性。



注意

非 XA 数据源可以与 JTA 事务集成。要继承数据源和 JTA，请确保 `jta` 参数被设置为 `true`。

数据源的设置可以用下列管理 CLI 命令来更新。

```
data-source --name=DATASOURCE_NAME --ATTRIBUTE_NAME=ATTRIBUTE_VALUE
```



注意

在受管域里，您必须指定 `--profile=PROFILE_NAME` 参数。

要使更新生效，必须重新启动服务器。

13.4.2. 修改 XA 数据源

XA 数据源设置可以用 `xa-data-source` 管理 CLI 命令更新。您也可以从管理控制台的 `datasources` 子系统里更新数据源属性。

- XA 数据源的设置可以用下列管理 CLI 命令来更新。

```
xa-data-source --name=XA_DATASOURCE_NAME --
ATTRIBUTE_NAME=ATTRIBUTE_VALUE
```



注意

在受管域里，您必须指定 **--profile=PROFILE_NAME** 参数。

- XA 数据源属性可以用下列管理 CLI 命令来更新。

```
/subsystem=datasources/xa-data-source=XA_DATASOURCE_NAME/xa-
datasource-properties=PROPERTY:add(value=VALUE)
```



注意

在受管域里，您必须在这个命令之前使用 **/profile=PROFILE_NAME**。

要使更新生效，必须重新启动服务器。

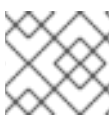
13.5. 删除数据源

数据源可以用管理控制台或管理 CLI 来删除。

13.5.1. 删除非 XA 数据源

您可以用 **data-source remove** 管理 CLI 命令删除非 XA 数据源。您可以在管理控制台的 **datasources** 子系统里删除非 XA 数据源。

```
data-source remove --name=DATASOURCE_NAME
```



注意

在受管域里，您必须指定 **--profile=PROFILE_NAME** 参数。

删除数据源后要求重启服务器。

13.5.2. 删除 XA 数据源

您可以用 **xa-data-source remove** 管理 CLI 命令删除 XA 数据源。您可以在管理控制台的 **datasources** 子系统里删除 XA 数据源。

```
xa-data-source remove --name=XA_DATASOURCE_NAME
```



注意

在受管域里，您必须指定 **--profile=PROFILE_NAME** 参数。

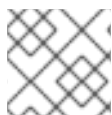
删除 XA 数据源后要求重启服务器。

13.6. 测试数据源连接

一旦在 JBoss EAP 里添加了数据源，您可以测试连接以检验设置是否正确。数据源连接可以用管理 CLI 命令或管理控制台的 **datasources** 子系统里的 **Test Connection** 按钮进行测试。

您可以使用下列管理 CLI 命令来测试数据源连接。

```
/subsystem=datasources/data-source=DATASOURCE_NAME:test-connection-in-pool
```



注意

在受管域里，您必须在这个命令之前使用 `/host=HOSTNAME/server=SERVER_NAME`。

13.7. XA 数据源的恢复

XA 数据源就是一个可以参与到一个 XA 全局事务中的数据源，它由事务管理器（transaction manager）进行协调，并可以在一个事务中分散到多个资源。如果其它的一个参与者无法提供它的改变，其它参与者会终止这个事务，并把它们的状态恢复到事务发生以前的状态。这么做的目的是，维护数据的一致性，并避免潜在的数据丢失或破坏。

XA 恢复（XA recovery）是一个过程，它确保了受一个事务影响的所有资源都会被更新或回滚，即使任何资源或事务参与者崩溃或无法访问。XA 恢复不需要用户的干涉。

每个 XA 资源都需要一个和它的配置相关联的恢复模块。这个模块包括了在恢复发生时需要执行的代码。JBoss EAP 会自动为 JDBC XA 资源注册恢复模块。如果需要执行自定义的恢复代码，则需要在 XA 数据源中注册一个自定义的模块。恢复模块需要扩展类

`com.arjuna.ats.jta.recovery.XAResourceRecovery`。

13.7.1. 配置 XA 恢复

对于多数的 JDBC 资源，恢复模块会自动和资源关联。此时，您只需要配置允许恢复模块连接资源以执行恢复的选项。

下表描述了 XA 恢复专有的 XA 数据源参数。这些配置属性都可以在数据源创建期间或之后进行设置。您可以用管理控制台或管理 CLI 命令来设置它们。关于配置 XA 数据源的信息，请参考[修改 XA 数据源](#)。

表 13.2. XA 恢复的数据源参数

属性	描述
recovery-username	连接恢复资源所使用的用户名称。请注意如果没有指定，将使用数据源安全设置。
recovery-password	连接恢复资源所使用的密码。请注意如果没有指定，将使用数据源安全设置。
recovery-security-domain	连接用于恢复的资源时使用的安全域。
recovery-plugin-class-name	如果您需要使用自定义的恢复模块，请将这个属性设置为模块的全限定类名。这个模块必须继承 <code>com.arjuna.ats.jta.recovery.XAResourceRecovery</code> 。

属性	描述
recovery-plugin-properties	如果您使用了要求设置属性的自定义模块，请将这个属性设置为用逗号隔开的 KEY=VALUE 列表。

禁用 XA 恢复

如果多个 XA 数据源连接至相同的物理数据库，那 XA 恢复通常都需要对其中一个进行配置。

使用下列管理 CLI 命令来禁用 XA 数据源的恢复。

```
/subsystem=datasources/xa-data-source=XA_DATASOURCE_NAME:write-attribute(name=no-recovery,value=true)
```

13.7.2. 厂商专有的 XA 恢复

厂商专有的配置

某些数据库要求专门的配置与 JBoss EAP 事务管理者管理的 XA 事务进行协作。如需了解最新的相关信息，请参考数据库厂商的文档。

MySQL

这里不要求特殊的配置。更多的信息请参考 MySQL 文档。

PostgreSQL 和 Postgres Plus Advanced Server

要使 PostgreSQL 可以处理 XA 事务，把配置参数 **max_prepared_transactions** 设置为一个大于 0 并大于或等于 **max_connections** 的值。

Oracle

请确保 Oracle 用户 (**USER**) 具有对恢复所需的表的访问权限。

```
GRANT SELECT ON sys.dba_pending_transactions TO USER;
GRANT SELECT ON sys.pending_trans$ TO USER;
GRANT SELECT ON sys.dba_2pc_pending TO USER;
GRANT EXECUTE ON sys.dbms_xa TO USER;
```

如果 Oracle 用户没有正确的权限，您可能会遇到这样的错误信息：

```
WARN [com.arjuna.ats.jta.logging.loggerI18N]
[com.arjuna.ats.internal.jta.recovery.xarecovery1] Local
XARecoveryModule.xaRecovery got XA exception
javax.transaction.xa.XAException, XAException.XAER_RMERR
```

Microsoft SQL 服务器

如需了解更多信息，请参考 Microsoft SQL 服务器文档，包括 <http://msdn.microsoft.com/en-us/library/aa342335.aspx>。

IBM DB2

这里不要求特殊的配置。更多的信息请参考 IBM DB2 文档。

Sybase

Sybase 需要 XA 事务在数据库中启用。如果没有正确的数据库设置，XA 事务将无法正常工作。使用 **enable xact coordination** 参数可以启用或禁用 Adaptive Server 事务协调服务。当启用这个参

数时，Adaptive Server 会确保，在远程 Adaptive Server 数据提交或回滚时，使用原始事务进行更新。

要启用事务协调，请使用：

```
sp_configure 'enable xact coordination', 1
```

MariaDB

这里不要求特殊的配置。更多的信息请参考 MariaDB 文档。

已知问题

这些与处理 XA 事务相关的已知问题是针对于 JBoss EAP 7 支持的特定数据库和 JDBC 驱动版本的。如需了解支持数据库的最新信息，请参考[JBoss EAP 支持的配置](#)。

MySQL

MySQL 并不能完全处理 XA 事务。如果客户端断开与 MySQL 的连接，那么事务的所有信息都会丢失。详情请参考 [MySQL 程序错误](#)。请注意 MySQL 5.7 已修复了这个问题。

PostgreSQL 和 Postgres Plus Advanced Server

在二阶段提交（2PC）的提交阶段发生网络故障时，JDBC 驱动会返回 **XAER_RMERR** XAException 错误代码。这个错误代表了事务管理器的一个不可恢复的严重事件，但在数据库端，这个事务会处于 **in-doubt** 状态，并在网络连接重新建立后可以容易地进行修复。正确的返回错误代码应该是 **XAER_RMFAIL** 或 **XAER_RETRY**。不正确的错误代码会导致事务在 JBoss EAP 中处于 **Heuristic** 状态，并会在数据库中被锁定，需要手工干预才可以解决这个问题。如需了解更多相关信息，请参阅 [PostgreSQL bug](#)。

如果在使用一阶段提交时出现连接错误，JDBC 驱动会返回 **XAER_RMERR**，但正确的错误返回代码应该是 **XAER_RMFAIL**。因此，如果数据库在一阶段提交的提交阶段出现连接问题时，会导致数据不统一的问题，客户端会接收到事务被回滚的信息。

Postgres Plus JDBC 驱动会为 Postgres Plus Server 中存在的所有准备的事务返回 XID，因此无法决定 XID 属于哪个数据库。如果为 JBoss EAP 中的同一个数据库定义了多个数据源，一个 in-doubt 事务恢复尝试会在错误的账户中返回，这将导致恢复失败。

Oracle

当恢复管理器使用由一些用户的凭证信息配置的数据源调用恢复时，JDBC 驱动返回属于数据库实例中所有用户的 XID。JDBC 驱动会产生异常 **ORA-24774: cannot switch to specified transaction**，这是因为它尝试恢复属于其它用户的 XID。

解决这个问题一个临时方法是，为用来配置恢复数据源的用户分配 **FORCE ANY TRANSACTION** 权限。如需了解更多相关信息，请参阅

http://docs.oracle.com/database/121/ADMIN/ds_txnman.htm#ADMIN12259

Microsoft SQL 服务器

在二阶段提交（2PC）的提交阶段发生网络故障时，JDBC 驱动会返回 **XAER_RMERR** XAException 错误代码。这个错误代表了事务管理器的一个不可恢复的严重事件，但在数据库一端，这个事务会处于 **in-doubt** 状态，并在网络连接重新建立后可以容易地进行修复。正确的返回错误代码应该是 **XAER_RMFAIL** 或 **XAER_RETRY**。不正确的错误代码会导致事务在 JBoss EAP 中处于 **Heuristic** 状态，并会在数据库中被锁定，需要手工干预才可以解决这个问题。如需了解更多相关信息，请参阅 [Microsoft SQL Server issue report](#)。

如果在使用一阶段提交时出现连接错误，JDBC 驱动会返回 **XAER_RMERR**，但正确的错误返回代码应该是 **XAER_RMFAIL**。因此，如果数据库在一阶段提交的提交阶段出现连接问题时，会导致数据不统一的问题，客户端会接收到事务被回滚的信息。

IBM DB2

如果在使用一阶段提交时出现连接错误，JDBC 驱动会返回 **XAER_RETRY**，但正确的错误返回代码应该是 **XAER_RMFAIL**。因此，如果数据库在一阶段提交的提交阶段出现连接问题时，会导致数据不统一的问题，客户端会接收到事务被回滚的信息。

Sybase

在二阶段提交（2PC）的提交阶段发生网络故障时，JDBC 驱动会返回 **XAER_RMERR** `XAException` 错误代码。这个错误代表了事务管理器的一个不可恢复的严重事件，但在数据库一端，这个事务会处于 **in-doubt** 状态，并在网络连接重新建立后可以容易地进行修复。正确的返回错误代码应该是 **XAER_RMFAIL** 或 **XAER_RETRY**。不正确的错误代码会导致事务在 JBoss EAP 中处于 **Heuristic** 状态，并会在数据库中被锁定，需要手工干预才可以解决这个问题。

如果在使用一阶段提交时出现连接错误，JDBC 驱动会返回 **XAER_RMERR**，但正确的错误返回代码应该是 **XAER_RMFAIL**。因此，如果数据库在一阶段提交的提交阶段出现连接问题时，会导致数据不统一的问题，客户端会接收到事务被回滚的信息。

MariaDB

MySQL 并不能完全处理 XA 事务。如果客户断开 MySQL，那么事务的所有信息都会丢失。

13.8. 数据库连接检验

数据库维护、网络问题或其他断电事件都可能导致 JBoss EAP 丢失数据库连接。为了从这些状况中恢复，您可以为数据源启用数据库连接检验。

在配置数据库连接验证时，需要指定验证时间（验证在什么时间发生）、验证机制（验证如何执行），以及异常处理（验证出现异常时如何处理）。

1. 选择 *其中一个* 检验时间。

validate-on-match

如果 **validate-on-match** 选项被设置为 **true**，每次从连接池中调出数据库连接时，都会使用下一步中指定的验证机制进行验证。

如果连接没有通过验证，则会在日志中写一个警告信息，池中的下一个连接会被调出。这个过程会一直继续，直到获得了一个有效的连接。如果您不希望检查池中的每一个连接，则可以使用 **use-fast-fail** 选项。如果无法在池中找到有效连接，则会向请求的应用程序返回一个异常。

这种设置可以最快获得结果，但对数据库所产生的负载也是最高的。不过，如果性能的小损失不是非常重要，则这种设置会是一个最安全的选择。

background-validation

当 **background-validation** 选项被设置为 **true** 时，连接会在使用前，在后台线程中被定期验证。验证的频率由 **background-validation-millis** 属性指定。**background-validation-millis** 的默认值是 **0**，这意味着它被禁用。

在确定 **background-validation-millis** 属性的值时，请考虑下列情况：

- 这个值不应该和 **idle-timeout-minutes** 设置相同。
- 这个值越低，池中的连接会被越频繁地验证，无效连接会被更快地从池中删除。
- 如果这个值比较低，则会消耗更多数据库资源。较高值意味着验证检查发生的频率会降低，并使用较少的数据库资源，但需要更长的时间来发现无效的连接。



注意

这些选项是互斥的。如果 **validate-on-match** 被设置为 **true**，则 **background-validation** 必须被设置为 **false**。如果 **background-validation** 被设置为 **true**，则 **validate-on-match** 必须被设置为 **false**。

2. 选择其中一个检验机制。

valid-connection-checker-class-name

valid-connection-checker-class-name 是首选的验证机制。它指定了一个连接检查程序（connection checker）类用来对使用的特定数据库的连接进行验证。JBoss EAP 提供了以下的连接检查程序：

- `org.jboss.jca.adapters.jdbc.extensions.db2.DB2ValidConnectionChecker`
- `org.jboss.jca.adapters.jdbc.extensions.mssql.MSSQLValidConnectionChecker`
- `org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLReplicationValidConnectionChecker`
- `org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChecker`
- `org.jboss.jca.adapters.jdbc.extensions.novendor.JDBC4ValidConnectionChecker`
- `org.jboss.jca.adapters.jdbc.extensions.novendor.NullValidConnectionChecker`
- `org.jboss.jca.adapters.jdbc.extensions.oracle.OracleValidConnectionChecker`
- `org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidConnectionChecker`
- `org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseValidConnectionChecker`

check-valid-connection-sql

通过 **check-valid-connection-sql** 提供用于检验连接的 SQL 语句。

下面是您可以用来检验 Oracle 连接的 SQL 语句示例。

```
select 1 from dual
```

下面是您可以用来检验 MySQL 或 PostgreSQL 连接的 SQL 语句示例。

```
select 1
```

3. 设置 exception sorter 的类名。

当一个异常被标记为致命 (fatal) 时, 连接会被马上关闭, 即使这个连接参与了一个事务。在致命连接异常发生后, 使用 `exception sorter` 选项来正确发现并清理。这需要为您的数据源类型选择恰当的 JBoss EAP exception sorter。

- `org.jboss.jca.adapters.jdbc.extensions.db2.DB2ExceptionSorter`
- `org.jboss.jca.adapters.jdbc.extensions.informix.InformixExceptionSorter`
- `org.jboss.jca.adapters.jdbc.extensions.mssql.MSQLExceptionSorter`
- `org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter`
- `org.jboss.jca.adapters.jdbc.extensions.novendor.NullExceptionSorter`
- `org.jboss.jca.adapters.jdbc.extensions.oracle.OracleExceptionSorter`
- `org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLExceptionSorter`
- `org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseExceptionSorter`

13.9. 数据源的安全性

数据源安全是指对数据源连接的密码进行加密。这些密码可以以明文的形式保存在配置文件中, 但这将会导致潜在的安全风险。

数据源安全的首选解决方案是使用安全域或密码库。下面是两者的例子。

用安全域保护数据源

下面定义了数据源的安全域。

```
<security-domain name="DsRealm" cache-type="default">
  <authentication>
    <login-module code="ConfiguredIdentity" flag="required">
      <module-option name="userName" value="sa"/>
      <module-option name="principal" value="sa"/>
      <module-option name="password" value="sa"/>
    </login-module>
  </authentication>
</security-domain>
```



注意

如果一个安全域将和多个数据源一起使用, 则应该在安全域中禁用数据缓存。把 `cache-type` 参数的值设为 `none`, 或完全删除这个参数。但是, 如果需要使用缓存, 则每个数据源都需要一个独立的安全域。

然后 `DsRealm` 被数据源配置所引用。

```
<datasources>
  <datasource jndi-name="java:jboss/datasources/securityDs"
    pool-name="securityDs">
    <connection-url>jdbc:h2:mem:test;DB_CLOSE_DELAY=-1</connection-url>
```

```

        <driver>h2</driver>
        <new-connection-sql>select current_user()</new-connection-sql>
        <security>
            <security-domain>DsRealm</security-domain>
        </security>
    </datasource>
</datasources>

```

关于安全域的更多信息，请参考 [如何配置身份管理](#) 章节。

用密码库保护数据源

```

<security>
    <user-name>admin</user-name>

    <password>${VAULT::ds_ExampleDS::password::N2NhZDYzOTMtNWE0OS00ZGQ0LWE4MmE
tMWNI1MDMyNDdmNmI2TEl0RV9CUkVBS3ZhdWx0}</password>
</security>

```

关于密码库的更多信息，请参考 [如何配置服务器安全性](#) 章节。

13.10. 数据源统计

您可以查看定义的数据源的核心池和 *JDBC* 运行时的统计数据。如需了解统计数据的完整列表，请参阅 [数据源统计](#) 一节。

启用数据源统计

在默认情况下，数据源统计是不启用的。下面的管理 CLI 命令启用了 **ExampleDS** 数据源的统计收集。



注意

在受管域里，命令前加入 **/profile=PROFILE_NAME**。

```

/subsystem=datasources/data-source=ExampleDS/statistics=pool:write-
attribute(name=statistics-enabled,value=true)

```

```

/subsystem=datasources/data-source=ExampleDS/statistics=jdbc:write-
attribute(name=statistics-enabled,value=true)

```

查看数据源统计

所有的数据源统计信息都可以通过 CLI 命令来获取。这些统计的子集可以从管理控制台的 **Runtime** 标签页里查看。

下面的管理 CLI 命令获取 **ExampleDS** 数据源的核心池统计。



注意

在受管域里，请在这些命令之前使用 **/host=HOST_NAME/server=SERVER_NAME**。

```

/subsystem=datasources/data-source=ExampleDS/statistics=pool:read-
resource(include-runtime=true)
{

```

```

"outcome" => "success",
"result" => {
    "ActiveCount" => 1,
    "AvailableCount" => 20,
    "AverageBlockingTime" => 0L,
    "AverageCreationTime" => 122L,
    "AverageGetTime" => 128L,
    "AveragePoolTime" => 0L,
    "AverageUsageTime" => 0L,
    "BlockingFailureCount" => 0,
    "CreatedCount" => 1,
    "DestroyedCount" => 0,
    "IdleCount" => 1,
    ...
}

```

下面的管理 CLI 命令获取了 **ExampleDS** 数据源的 *JDBC* 统计信息。

```

/subsystem=datasources/data-source=ExampleDS/statistics=jdbc:read-
resource(include-runtime=true)
{
    "outcome" => "success",
    "result" => {
        "PreparedStatementCacheAccessCount" => 0L,
        "PreparedStatementCacheAddCount" => 0L,
        "PreparedStatementCacheCurrentSize" => 0,
        "PreparedStatementCacheDeleteCount" => 0L,
        "PreparedStatementCacheHitCount" => 0L,
        "PreparedStatementCacheMissCount" => 0L,
        "statistics-enabled" => true
    }
}

```



注意

因为统计都是运行时信息，请确保指定了 **include-runtime=true** 参数。

13.11. 容量策略

JBoss EAP 支持为 JCA 部署定义包括数据源在内的容量策略。容量策略定义了一个池的物理连接如何创建（增加容量）和销毁（减少容量）。默认的策略是，在增加容量时，为每个请求创建一次连接；当调度空闲超时来减少容量时，销毁所有超时的连接。

要配置容量策略，您需要指定一个容量 **incrementer** 或 **decrementer** 类：

命令示例

```

/subsystem=datasources/data-source=ExampleDS:write-
attribute(name=capacity-incrementer-class,
value="org.jboss.jca.core.connectionmanager.pool.capacity.SizeIncrementer"
)

/subsystem=datasources/data-source=ExampleDS:write-

```

```
attribute(name=capacity-decrementer-class,
value="org.jboss.jca.core.connectionmanager.pool.capacity.SizeDecrementer"
)
```

您也可以在指定的容量 incrementer 或 decrementer 类上配置属性：

命令示例

```
/subsystem=datasources/data-source=ExampleDS:write-
attribute(name=capacity-incrementer-properties.size, value=2)

/subsystem=datasources/data-source=ExampleDS:write-
attribute(name=capacity-decrementer-properties.size, value=2)
```

MaxPoolSize Incrementer 策略

类名：`org.jboss.jca.core.connectionmanager.pool.capacity.MaxPoolSizeIncrementer`

MaxPoolSize incrementer 策略会为每个请求把池添充到最大值。如果您需要一直保持有最多数量的可用连接时，可以使用这个策略。

Size Incrementer 策略

类名：`org.jboss.jca.core.connectionmanager.pool.capacity.SizeIncrementer`

Size incrementer 策略会为每个请求在池中添充指定数量的连接。如果您预测每个请求的下一个请求也需要连接，所以需要为当前的请求增加一定数量的连接时，可以使用这个策略。

表 13.3. Size 策略属性

名称	描述
Size	应该创建的连接数量



注意
这是默认策略，它的 *size* 值为 1。

Watermark Incrementer 策略

类名：`org.jboss.jca.core.connectionmanager.pool.capacity.WatermarkIncrementer`

Watermark incrementer 策略会为每个请求把池添充到特定连接数量。如果您需要池中总保留特定数量的连接时，可以使用这个策略。

表 13.4. Watermark 策略属性

名称	描述
Watermark	连接数量的 watermark 值

MinPoolSize Decrementer 策略

类名：`org.jboss.jca.core.connectionmanager.pool.capacity.MinPoolSizeDecrementer`

MinPoolSize decrementer 策略会为每个请求把池减到它的最小值。如果您希望在每个空闲超时请求时限制连接数量，可以使用这个策略。这个池会以先进先出（FIFO）的模式操作。

Size Decrementer 策略

类名: `org.jboss.jca.core.connectionmanager.pool.capacity.SizeDecrementer`

Size decrementer 策略会在每个空闲超时请求时，为池减少特定数量的连接。

表 13.5. Size 策略属性

名称	描述
Size	需要被销毁的连接数量

如果您预测随着时间的推移，池的使用情况会下降，因此希望在每个空闲请求时减少特定数量的连接时，可以使用这个策略。

池会以先进先出（FIFO）模式操作。

TimedOut Decrementer 策略

类名: `org.jboss.jca.core.connectionmanager.pool.capacity.TimedOutDecrementer`

TimedOut decrementer 策略会为每个空闲超时请求删除池中所有的超时连接。池以先进后出（FILO）的形式操作。



注意

这个策略是默认的 decrement 策略。

TimedOut/FIFO Decrementer 策略

类名:

`org.jboss.jca.core.connectionmanager.pool.capacity.TimedOutFIFODecrementer`

TimedOutFIFO decrementer 策略会为每个空闲超时请求删除池中所有的超时连接。池以先进先出（FIFO）的形式操作。

Watermark Decrementer 策略

类名: `org.jboss.jca.core.connectionmanager.pool.capacity.WatermarkDecrementer`

Watermark incrementer 策略会为每个空闲超时请求把池减少到特定连接数量。如果您需要池中总保留特定数量的连接时，可以使用这个策略。池会以先进先出（FIFO）的形式操作。

表 13.6. Watermark 策略属性

名称	描述
Watermark	连接数量的 watermark 值

13.12. 登记跟踪

登记跟踪（enlistment trace）会被记录，它会被用来帮助在登记 **XAResource** 实例时定位错误的发生。这个功能会对性能有一定影响，您可能在特定情况下需要禁用它。

您可以使用管理 CLI 禁用记录登记跟踪的功能。您需要把 **enlistment-trace** 属性的值设置为 **false**。

禁用非 XA 数据源的 enlistment 跟踪

```
data-source --name=DATASOURCE_NAME --enlistment-trace=false
```

禁用 XA 数据源的 enlistment 跟踪

```
xa-data-source --name=XA_DATASOURCE_NAME --enlistment-trace=false
```



警告

禁用登记跟踪会使在事务登记的过程中查找错误变得比较困难。

13.13. 数据源配置示例

13.13.1. MySQL 数据源示例

这是一个带有连接信息、基本安全性和检验选项的 MySQL 数据源配置示例。

MySQL 数据源配置示例

```
<datasources>
  <datasource jndi-name="java:jboss/MySqlDS" pool-name="MySqlDS">
    <connection-url>jdbc:mysql://localhost:3306/jbossdb</connection-url>
    <driver>mysql</driver>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChe
cker"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>false</background-validation>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter"/>
    </validation>
  </datasource>
  <drivers>
    <driver name="mysql" module="com.mysql">
      <driver-class>com.mysql.jdbc.Driver</driver-class>
      <xa-datasource-
class>com.mysql.jdbc.jdbc2.optional.MysqlXADataSource</xa-datasource-
class>
```

```

    </driver>
  </drivers>
</datasources>

```

MySQL JDBC Driver module.xml 文件示例

```

<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.mysql">
  <resources>
    <resource-root path="mysql-connector-java-5.1.36-bin.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>

```

管理 CLI 命令示例

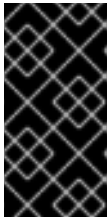
这个配置示例可以用下列管理 CLI 命令来实现。

1. 将 MySQL JDBC 驱动添加为核心模块。

```

module add --name=com.mysql --resources=/path/to/mysql-connector-
java-5.1.36-bin.jar --dependencies=javax.api,javax.transaction.api

```



重要

Using the **module** management CLI command to add and remove modules is provided as [technology preview](#) only. This command is not appropriate for use in a managed domain or when connecting to the management CLI remotely. Modules should be [added](#) and [removed](#) manually in a production environment.

2. 注册 MySQL JDBC 驱动。

```

/subsystem=datasources/jdbc-driver=mysql:add(driver-
name=mysql,driver-module-name=com.mysql,driver-xa-datasource-class-
name=com.mysql.jdbc.jdbc2.optional.MysqlXADataSource, driver-class-
name=com.mysql.jdbc.Driver)

```

3. 添加 MySQL 数据源。

```

data-source add --name=MySqlDS --jndi-name=java:jboss/MySqlDS --
driver-name=mysql --connection-
url=jdbc:mysql://localhost:3306/jbossdb --user-name=admin --
password=admin --validate-on-match=true --background-
validation=false --valid-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnecti
onChecker --exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSort
er

```

13.13.2. MySQL XA 数据源示例

这是一个带有 XA 数据源属性、基本安全性和检验选项的 MySQL XA 数据源配置示例。

MySQL XA 数据源配置示例

```
<datasources>
  <xa-datasource jndi-name="java:jboss/MySQLXADS" pool-name="MySQLXADS">
    <xa-datasource-property name="ServerName">
      localhost
    </xa-datasource-property>
    <xa-datasource-property name="DatabaseName">
      mysqlldb
    </xa-datasource-property>
    <driver>mysql</driver>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChe
cker"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>false</background-validation>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter"/>
    </validation>
  </xa-datasource>
  <drivers>
    <driver name="mysql" module="com.mysql">
      <driver-class>com.mysql.jdbc.Driver</driver-class>
      <xa-datasource-
class>com.mysql.jdbc.jdbc2.optional.MysqlXADataSource</xa-datasource-
class>
    </driver>
  </drivers>
</datasources>
```

MySQL JDBC Driver module.xml 文件示例

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.mysql">
  <resources>
    <resource-root path="mysql-connector-java-5.1.36-bin.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

管理 CLI 命令示例

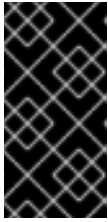
这个配置示例可以用下列管理 CLI 命令来实现。

1. 将 MySQL JDBC 驱动添加为核心模块。

```
module add --name=com.mysql --resources=/path/to/mysql-connector-
```



```
java-5.1.36-bin.jar --dependencies=javax.api,javax.transaction.api
```



重要

Using the **module** management CLI command to add and remove modules is provided as [technology preview](#) only. This command is not appropriate for use in a managed domain or when connecting to the management CLI remotely. Modules should be [added](#) and [removed](#) manually in a production environment.

2. 注册 MySQL JDBC 驱动。

```
/subsystem=datasources/jdbc-driver=mysql:add(driver-
name=mysql,driver-module-name=com.mysql,driver-xa-datasource-class-
name=com.mysql.jdbc.jdbc2.optional.MysqlXADataSource, driver-class-
name=com.mysql.jdbc.Driver)
```

3. 添加 MySQL XA 数据源。

```
xa-data-source add --name=MySqlXADS --jndi-name=java:jboss/MySqlXADS
--driver-name=mysql --user-name=admin --password=admin --validate-
on-match=true --background-validation=false --valid-connection-
checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnecti
onChecker --exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSort
er --xa-datasource-properties=
{"ServerName"=>"localhost", "DatabaseName"=>"mysqldb"}
```

13.13.3. PostgreSQL 数据源示例

这是一个带有连接信息、基本安全性和检验选项的 PostgreSQL 数据源配置示例。

PostgreSQL 数据源配置示例

```
<datasources>
  <datasource jndi-name="java:jboss/PostgresDS" pool-name="PostgresDS">
    <connection-
url>jdbc:postgresql://localhost:5432/postgresdb</connection-url>
    <driver>postgresql</driver>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidConne
ctionChecker"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>false</background-validation>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLExceptionS
orter"/>
    </validation>
  </datasource>
```

```

<drivers>
  <driver name="postgresql" module="com.postgresql">
    <xa-datasource-class>org.postgresql.xa.PGXADatasource</xa-
datasource-class>
  </driver>
</drivers>
</datasources>

```

PostgreSQL JDBC 驱动的 module.xml 文件示例

```

<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.postgresql">
  <resources>
    <resource-root path="postgresql-9.3-1102.jdbc4.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>

```

管理 CLI 命令示例

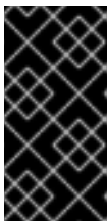
这个配置示例可以用下列管理 CLI 命令来实现。

1. 将 PostgreSQL JDBC 驱动添加为核心模块。

```

module add --name=com.postgresql --resources=/path/to/postgresql-
9.3-1102.jdbc4.jar --dependencies=javax.api,javax.transaction.api

```



重要

Using the **module** management CLI command to add and remove modules is provided as [technology preview](#) only. This command is not appropriate for use in a managed domain or when connecting to the management CLI remotely. Modules should be [added](#) and [removed](#) manually in a production environment.

2. 注册 PostgreSQL JDBC 驱动。

```

/subsystem=datasources/jdbc-driver=postgresql:add(driver-
name=postgresql,driver-module-name=com.postgresql,driver-xa-
datasource-class-name=org.postgresql.xa.PGXADatasource)

```

3. 添加 PostgreSQL 数据源。

```

data-source add --name=PostgresDS --jndi-name=java:jboss/PostgresDS
--driver-name=postgresql --connection-
url=jdbc:postgresql://localhost:5432/postgresdb --user-name=admin --
password=admin --validate-on-match=true --background-
validation=false --valid-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValid
ConnectionChecker --exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLExcep
tionSorter

```

13.13.4. PostgreSQL XA 数据源示例

这是一个带有 XA 数据源属性、基本安全性和检验选项的 PostgreSQL XA 数据源配置示例。

PostgreSQL XA 数据源配置示例

```
<datasources>
  <xa-datasource jndi-name="java:jboss/PostgresXADS" pool-
name="PostgresXADS">
    <xa-datasource-property name="ServerName">
      localhost
    </xa-datasource-property>
    <xa-datasource-property name="PortNumber">
      5432
    </xa-datasource-property>
    <xa-datasource-property name="DatabaseName">
      postgresdb
    </xa-datasource-property>
    <driver>postgresql</driver>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidConne
ctionChecker"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>false</background-validation>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLExceptions
orter"/>
    </validation>
  </xa-datasource>
  <drivers>
    <driver name="postgresql" module="com.postgresql">
      <xa-datasource-class>org.postgresql.xa.PGXADatasource</xa-
datasource-class>
    </driver>
  </drivers>
</datasources>
```

PostgreSQL JDBC 驱动的 module.xml 文件示例

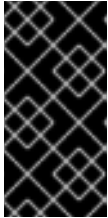
```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.postgresql">
  <resources>
    <resource-root path="postgresql-9.3-1102.jdbc4.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

管理 CLI 命令示例

这个配置示例可以用下列管理 CLI 命令来实现。

1. 将 PostgreSQL JDBC 驱动添加为核心模块。

```
module add --name=com.postgresql --resources=/path/to/postgresql-9.3-1102.jdbc4.jar --dependencies=javax.api,javax.transaction.api
```



重要

Using the **module** management CLI command to add and remove modules is provided as [technology preview](#) only. This command is not appropriate for use in a managed domain or when connecting to the management CLI remotely. Modules should be [added](#) and [removed](#) manually in a production environment.

2. 注册 PostgreSQL JDBC 驱动。

```
/subsystem=datasources/jdbc-driver=postgresql:add(driver-name=postgresql,driver-module-name=com.postgresql,driver-xa-datasource-class-name=org.postgresql.xa.PGXADatasource)
```

3. 添加 PostgreSQL XA 数据源。

```
xa-data-source add --name=PostgresXADS --jndi-name=java:jboss/PostgresXADS --driver-name=postgresql --user-name=admin --password=admin --validate-on-match=true --background-validation=false --valid-connection-checker-class-name=org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidConnectionChecker --exception-sorter-class-name=org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLExceptionSorter --xa-datasource-properties={"ServerName"=>"localhost", "PortNumber"=>"5432", "DatabaseName"=>"postgresdb"}
```

13.13.5. Oracle 数据源示例

这是一个带有连接信息、基本安全性和检验选项的 Oracle 数据源配置示例。

Oracle 数据源配置示例

```
<datasources>
  <datasource jndi-name="java:jboss/OracleDS" pool-name="OracleDS">
    <connection-url>jdbc:oracle:thin:@localhost:1521:XE</connection-url>
    <driver>oracle</driver>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <valid-connection-checker class-name="org.jboss.jca.adapters.jdbc.extensions.oracle.OracleValidConnectionChecker"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>false</background-validation>
      <stale-connection-checker class-
```

```

name="org.jboss.jca.adapters.jdbc.extensions.oracle.OracleStaleConnectionC
hecker"/>
    <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.oracle.OracleExceptionSorter"
/>
    </validation>
</datasource>
<drivers>
    <driver name="oracle" module="com.oracle">
        <xa-datasource-class>oracle.jdbc.xa.client.OracleXADataSource</xa-
datasource-class>
    </driver>
</drivers>
</datasources>

```

Oracle JDBC 驱动的 module.xml 文件示例

```

<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.oracle">
    <resources>
        <resource-root path="ojdbc7.jar"/>
    </resources>
    <dependencies>
        <module name="javax.api"/>
        <module name="javax.transaction.api"/>
    </dependencies>
</module>

```

管理 CLI 命令示例

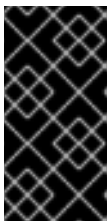
这个配置示例可以用下列管理 CLI 命令来实现。

1. 将 Oracle JDBC 驱动添加为核心模块。

```

module add --name=com.oracle --
resources=/path/to/misc/jdbc_drivers/oracle/ojdbc7.jar --
dependencies=javax.api,javax.transaction.api

```



重要

Using the **module** management CLI command to add and remove modules is provided as [technology preview](#) only. This command is not appropriate for use in a managed domain or when connecting to the management CLI remotely. Modules should be [added](#) and [removed](#) manually in a production environment.

2. 注册 Oracle JDBC 驱动。

```

/subsystem=datasources/jdbc-driver=oracle:add(driver-
name=oracle,driver-module-name=com.oracle,driver-xa-datasource-
class-name=oracle.jdbc.xa.client.OracleXADataSource)

```

3. 添加 Oracle 数据源。

```

data-source add --name=OracleDS --jndi-name=java:jboss/OracleDS --
driver-name=oracle --connection-

```

```
url=jdbc:oracle:thin:@localhost:1521:XE --user-name=admin --
password=admin --validate-on-match=true --background-
validation=false --valid-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.oracle.OracleValidConnec
tionChecker --exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.oracle.OracleExceptionSo
rter --stale-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.oracle.OracleStaleConnec
tionChecker
```

13.13.6. Oracle XA 数据源示例

重要

下面的设置必须应用于访问 Oracle XA 数据源的用户以让 XA 恢复正常操作。**user** 的值是连接 JBoss EAP 至 Oracle 的用户。

- **GRANT SELECT ON sys.dba_pending_transactions TO user;**
- **GRANT SELECT ON sys.pending_trans\$ TO user;**
- **GRANT SELECT ON sys.dba_2pc_pending TO user;**
- **GRANT EXECUTE ON sys.dbms_xa TO user;**

这是一个带有 XA 数据源属性、基本安全性和检验选项的 Oracle XA 数据源配置示例。

Oracle XA 数据源配置示例

```
<datasources>
  <xa-datasource jndi-name="java:jboss/OracleXADS" pool-name="OracleXADS">
    <xa-datasource-property name="URL">
      jdbc:oracle:thin:@oracleHostName:1521:orcl
    </xa-datasource-property>
    <driver>oracle</driver>
    <xa-pool>
      <is-same-rm-override>>false</is-same-rm-override>
    </xa-pool>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.oracle.OracleValidConnectionC
hecker"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>false</background-validation>
      <stale-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.oracle.OracleStaleConnectionC
hecker"/>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.oracle.OracleExceptionSorter"
/>
    </validation>
```

```

</xa-datasource>
<drivers>
  <driver name="oracle" module="com.oracle">
    <xa-datasource-class>oracle.jdbc.xa.client.OracleXADataSource</xa-
datasource-class>
  </driver>
</drivers>
</datasources>

```

Oracle JDBC 驱动的 module.xml 文件示例

```

<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.oracle">
  <resources>
    <resource-root path="ojdbc7.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>

```

管理 CLI 命令示例

这个配置示例可以用下列管理 CLI 命令来实现。

1. 将 Oracle JDBC 驱动添加为核心模块。

```

module add --name=com.oracle --
resources=/path/to/misc/jdbc_drivers/oracle/ojdbc7.jar --
dependencies=javax.api,javax.transaction.api

```



重要

Using the **module** management CLI command to add and remove modules is provided as [technology preview](#) only. This command is not appropriate for use in a managed domain or when connecting to the management CLI remotely. Modules should be [added](#) and [removed](#) manually in a production environment.

2. 注册 Oracle JDBC 驱动。

```

/subsystem=datasources/jdbc-driver=oracle:add(driver-
name=oracle,driver-module-name=com.oracle,driver-xa-datasource-
class-name=oracle.jdbc.xa.client.OracleXADataSource)

```

3. 添加 Oracle XA 数据源。

```

xa-data-source add --name=OracleXADS --jndi-
name=java:jboss/OracleXADS --driver-name=oracle --user-name=admin --
password=admin --validate-on-match=true --background-
validation=false --valid-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.oracle.OracleValidConnec-
tionChecker --exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.oracle.OracleExceptionSo-
rter --stale-connection-checker-class-

```



```
name=org.jboss.jca.adapters.jdbc.extensions.oracle.OracleStaleConnectionChecker --same-rm-override=false --xa-datasource-properties={\"URL\"=>\"jdbc:oracle:thin:@oracleHostName:1521:orcl\"}
```

13.13.7. Microsoft SQL Server 数据源示例

这是一个带有连接信息、基本安全性和检验选项的 Microsoft SQL 服务器数据源配置示例。

Microsoft SQL Server 数据源配置示例

```
<datasources>
  <datasource jndi-name=\"java:jboss/MSSQLDS\" pool-name=\"MSSQLDS\">
    <connection-
url>jdbc:sqlserver://localhost:1433;DatabaseName=MyDatabase</connection-
url>
    <driver>sqlserver</driver>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <valid-connection-checker class-
name=\"org.jboss.jca.adapters.jdbc.extensions.mssql.MSQLValidConnectionChe
cker\"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>>false</background-validation>
      <exception-sorter class-
name=\"org.jboss.jca.adapters.jdbc.extensions.mssql.MSQLExceptionSorter\"/>
    </validation>
  </datasource>
</drivers>
  <driver name=\"sqlserver\" module=\"com.microsoft\">
    <xa-datasource-
class>com.microsoft.sqlserver.jdbc.SQLServerXADataSource</xa-datasource-
class>
  </driver>
</drivers>
</datasources>
```

Microsoft SQL Server JDBC 驱动的 module.xml 文件示例

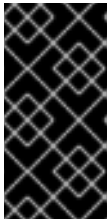
```
<?xml version=\"1.0\" ?>
<module xmlns=\"urn:jboss:module:1.1\" name=\"com.microsoft\">
  <resources>
    <resource-root path=\"sqljdbc41.jar\"/>
  </resources>
  <dependencies>
    <module name=\"javax.api\"/>
    <module name=\"javax.transaction.api\"/>
  </dependencies>
</module>
```

管理 CLI 命令示例

这个配置示例可以用下列管理 CLI 命令来实现。

1. 将 Microsoft SQL Server JDBC 驱动添加为核心模块。

```
module add --name=com.microsoft --resources=/path/to/sqljdbc41.jar -
-dependencies=javax.api,javax.transaction.api
```



重要

Using the **module** management CLI command to add and remove modules is provided as [technology preview](#) only. This command is not appropriate for use in a managed domain or when connecting to the management CLI remotely. Modules should be [added](#) and [removed](#) manually in a production environment.

2. 注册 Microsoft SQL Server JDBC 驱动。

```
/subsystem=datasources/jdbc-driver=sqlserver:add(driver-
name=sqlserver,driver-module-name=com.microsoft,driver-xa-
datasource-class-
name=com.microsoft.sqlserver.jdbc.SQLServerXADataSource)
```

3. 添加 Microsoft SQL Server 数据源

```
data-source add --name=MSSQLDS --jndi-name=java:jboss/MSSQLDS --
driver-name=sqlserver --connection-
url=jdbc:sqlserver://localhost:1433;DatabaseName=MyDatabase --user-
name=admin --password=admin --validate-on-match=true --background-
validation=false --valid-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.mssql.MSSQLValidConnecti
onChecker --exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.mssql.MSQLExceptionSort
er
```

13.13.8. Microsoft SQL Server XA 数据源示例

这是一个带有 XA 数据源属性、基本安全性和检验选项的 Microsoft SQL Server XA 数据源配置示例。

Microsoft SQL Server XA 数据源配置示例

```
<datasources>
  <xa-datasource jndi-name="java:jboss/MSSQLXADS" pool-name="MSSQLXADS">
    <xa-datasource-property name="ServerName">
      localhost
    </xa-datasource-property>
    <xa-datasource-property name="DatabaseName">
      mssqlldb
    </xa-datasource-property>
    <xa-datasource-property name="SelectMethod">
      cursor
    </xa-datasource-property>
    <driver>sqlserver</driver>
    <xa-pool>
      <is-same-rm-override>false</is-same-rm-override>
    </xa-pool>
    <security>
      <user-name>admin</user-name>
```

```

        <password>admin</password>
    </security>
    <validation>
        <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.mssql.MSSQLValidConnectionChe
cker"/>
        <validate-on-match>true</validate-on-match>
        <background-validation>false</background-validation>
        <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.mssql.MSQLExceptionSorter"/>
    </validation>
</xa-datasource>
<drivers>
    <driver name="sqlserver" module="com.microsoft">
        <xa-datasource-
class>com.microsoft.sqlserver.jdbc.SQLServerXADataSource</xa-datasource-
class>
    </driver>
</drivers>
</datasources>

```

Microsoft SQL Server JDBC 驱动的 module.xml 文件示例

```

<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.microsoft">
    <resources>
        <resource-root path="sqljdbc41.jar"/>
    </resources>
    <dependencies>
        <module name="javax.api"/>
        <module name="javax.transaction.api"/>
    </dependencies>
</module>

```

管理 CLI 命令示例

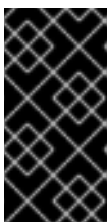
这个配置示例可以用下列管理 CLI 命令来实现。

1. 将 Microsoft SQL Server JDBC 驱动添加为核心模块。

```

module add --name=com.microsoft --resources=/path/to/sqljdbc41.jar -
-dependencies=javax.api,javax.transaction.api

```



重要

Using the **module** management CLI command to add and remove modules is provided as [technology preview](#) only. This command is not appropriate for use in a managed domain or when connecting to the management CLI remotely. Modules should be [added](#) and [removed](#) manually in a production environment.

2. 注册 Microsoft SQL Server JDBC 驱动。

```

/subsystem=datasources/jdbc-driver=sqlserver:add(driver-
name=sqlserver,driver-module-name=com.microsoft,driver-xa-
datasource-class-

```

```
name=com.microsoft.sqlserver.jdbc.SQLServerXADataSource)
```

3. 添加 Microsoft SQL Server XA 数据源

```
xa-data-source add --name=MSSQLXADS --jndi-name=java:jboss/MSSQLXADS
--driver-name=sqlserver --user-name=admin --password=admin --
validate-on-match=true --background-validation=false --background-
validation=false --valid-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.mssql.MSQLValidConnecti
onChecker --exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.mssql.MSQLExceptionSort
er --same-rm-override=false --xa-datasource-properties=
{"ServerName"=>"localhost", "DatabaseName"=>"mssqldb", "SelectMethod"=
>"cursor"}
```

13.13.9. IBM DB2 数据源示例

这是一个带有连接信息、基本安全性和检验选项的 IBM DB2 数据源配置示例。

IBM DB2 数据源配置示例

```
<datasources>
  <datasource jndi-name="java:jboss/DB2DS" pool-name="DB2DS">
    <connection-url>jdbc:db2://localhost:50000/ibmdb2db</connection-url>
    <driver>ibmdb2</driver>
    <pool>
      <min-pool-size>0</min-pool-size>
      <max-pool-size>50</max-pool-size>
    </pool>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.db2.DB2ValidConnectionChecker
"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>false</background-validation>
      <stale-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.db2.DB2StaleConnectionChecker
"/>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.db2.DB2ExceptionSorter"/>
    </validation>
  </datasource>
</drivers>
  <driver name="ibmdb2" module="com.ibm">
    <xa-datasource-class>com.ibm.db2.jcc.DB2XADataSource</xa-
datasource-class>
  </driver>
</drivers>
</datasources>
```

IBM DB2 JDBC 驱动的 module.xml 文件示例

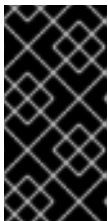
```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.ibm">
  <resources>
    <resource-root path="db2jcc4.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

管理 CLI 命令示例

这个配置示例可以用下列管理 CLI 命令来实现。

1. 将 IBM DB2 JDBC 驱动添加为核心模块。

```
module add --name=com.ibm --resources=/path/to/db2jcc4.jar --
dependencies=javax.api,javax.transaction.api
```



重要

Using the **module** management CLI command to add and remove modules is provided as [technology preview](#) only. This command is not appropriate for use in a managed domain or when connecting to the management CLI remotely. Modules should be [added](#) and [removed](#) manually in a production environment.

2. 注册 IBM DB2 JDBC 驱动。

```
/subsystem=datasources/jdbc-driver=ibmdb2:add(driver-
name=ibmdb2,driver-module-name=com.ibm,driver-xa-datasource-class-
name=com.ibm.db2.jcc.DB2XADataSource)
```

3. 添加 IBM DB2 数据源。

```
data-source add --name=DB2DS --jndi-name=java:jboss/DB2DS --driver-
name=ibmdb2 --connection-url=jdbc:db2://localhost:50000/ibmdb2db --
user-name=admin --password=admin --validate-on-match=true --
background-validation=false --valid-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.db2.DB2ValidConnectionCh
ecker --exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.db2.DB2ExceptionSorter -
-stale-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.db2.DB2StaleConnectionCh
ecker --min-pool-size=0 --max-pool-size=50
```

13.13.10. IBM DB2 XA 数据源示例

这是一个带有 XA 数据源属性、基本安全性和检验选项的 IBM DB2 XA 数据源配置示例。

IBM DB2 XA 数据源配置示例

```
<datasources>
  <xa-datasource jndi-name="java:jboss/DB2XADS" pool-name="DB2XADS">
```

```

    <xa-datasource-property name="ServerName">
        localhost
    </xa-datasource-property>
    <xa-datasource-property name="DatabaseName">
        ibmdb2db
    </xa-datasource-property>
    <xa-datasource-property name="PortNumber">
        50000
    </xa-datasource-property>
    <xa-datasource-property name="DriverType">
        4
    </xa-datasource-property>
    <driver>ibmdb2</driver>
    <xa-pool>
        <is-same-rm-override>false</is-same-rm-override>
    </xa-pool>
    <security>
        <user-name>admin</user-name>
        <password>admin</password>
    </security>
    <recovery>
        <recover-plugin class-
name="org.jboss.jca.core.recovery.ConfigurableRecoveryPlugin">
            <config-property name="EnableIsValid">
                false
            </config-property>
            <config-property name="IsValidOverride">
                false
            </config-property>
            <config-property name="EnableClose">
                false
            </config-property>
        </recover-plugin>
    </recovery>
    <validation>
        <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.db2.DB2ValidConnectionChecker
"/>
            <validate-on-match>true</validate-on-match>
            <background-validation>false</background-validation>
            <stale-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.db2.DB2StaleConnectionChecker
"/>
                <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.db2.DB2ExceptionSorter"/>
            </validation>
        </xa-datasource>
    <drivers>
        <driver name="ibmdb2" module="com.ibm">
            <xa-datasource-class>com.ibm.db2.jcc.DB2XADataSource</xa-
datasource-class>
        </driver>
    </drivers>
</datasources>

```

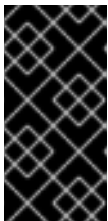
```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.ibm">
  <resources>
    <resource-root path="db2jcc4.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

管理 CLI 命令示例

这个配置示例可以用下列管理 CLI 命令来实现。

1. 将 IBM DB2 JDBC 驱动添加为核心模块。

```
module add --name=com.ibm --resources=/path/to/db2jcc4.jar --
dependencies=javax.api,javax.transaction.api
```



重要

Using the **module** management CLI command to add and remove modules is provided as [technology preview](#) only. This command is not appropriate for use in a managed domain or when connecting to the management CLI remotely. Modules should be [added](#) and [removed](#) manually in a production environment.

2. 注册 IBM DB2 JDBC 驱动。

```
/subsystem=datasources/jdbc-driver=ibmdb2:add(driver-
name=ibmdb2,driver-module-name=com.ibm,driver-xa-datasource-class-
name=com.ibm.db2.jcc.DB2XADataSource)
```

3. 添加 IBM DB2 XA 数据源。

```
xa-data-source add --name=DB2XADS --jndi-name=java:jboss/DB2XADS --
driver-name=ibmdb2 --user-name=admin --password=admin --validate-on-
match=true --background-validation=false --background-
validation=false --valid-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.db2.DB2ValidConnectionCh
ecker --exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.db2.DB2ExceptionSorter -
-stale-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.db2.DB2StaleConnectionCh
ecker --same-rm-override=false --recovery-plugin-class-
name=org.jboss.jca.core.recovery.ConfigurableRecoveryPlugin --
recovery-plugin-properties=
{"EnableIsValid"=>"false","IsValidOverride"=>"false","EnableClose"=>
"false"} --xa-datasource-properties=
{"ServerName"=>"localhost","DatabaseName"=>"ibmdb2db","PortNumber"=>
"50000","DriverType"=>"4"}
```

13.13.11. Sybase 数据源示例

这是一个带有连接信息、基本安全性和检验选项的 Sybase 数据源配置示例。

Sybase 数据源配置示例

```
<datasources>
  <datasource jndi-name="java:jboss/SybaseDB" pool-name="SybaseDB">
    <connection-url>jdbc:sybase:Tds:localhost:5000/DATABASE?
JCONNECT_VERSION=6</connection-url>
    <driver>sybase</driver>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseValidConnectionC
hecker"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>>false</background-validation>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseExceptionSorter"
/>
    </validation>
  </datasource>
</drivers>
  <driver name="sybase" module="com.sybase">
    <xa-datasource-class>com.sybase.jdbc4.jdbc.SybXADataSource</xa-
datasource-class>
  </driver>
</drivers>
</datasources>
```

Sybase JDBC 驱动的 module.xml 文件示例

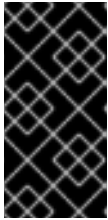
```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.sybase">
  <resources>
    <resource-root path="jconn4.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

管理 CLI 命令示例

这个配置示例可以用下列管理 CLI 命令来实现。

1. 将 Sybase JDBC 驱动添加为核心模块。

```
module add --name=com.sybase --resources=/path/to/jconn4.jar --
dependencies=javax.api,javax.transaction.api
```

重要

Using the **module** management CLI command to add and remove modules is provided as [technology preview](#) only. This command is not appropriate for use in a managed domain or when connecting to the management CLI remotely. Modules should be [added](#) and [removed](#) manually in a production environment.

2. 注册 Sybase JDBC 驱动。

```
/subsystem=datasources/jdbc-driver=sybase:add(driver-
name=sybase,driver-module-name=com.sybase,driver-xa-datasource-
class-name=com.sybase.jdbc4.jdbc.SybXADataSource)
```

3. 添加 Sybase 数据源。

```
data-source add --name=SybaseDB --jndi-name=java:jboss/SybaseDB --
driver-name=sybase --connection-
url=jdbc:sybase:Tds:localhost:5000/DATABASE?JCONNECT_VERSION=6 --
user-name=admin --password=admin --validate-on-match=true --
background-validation=false --valid-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseValidConnec-
tionChecker --exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseExceptionSo-
rter
```

13.13.12. Sybase XA 数据源示例

这是一个带有 XA 数据源属性、基本安全性和检验选项的 Sybase XA 数据源配置示例。

Sybase XA 数据源配置示例

```
<datasources>
  <xa-datasource jndi-name="java:jboss/SybaseXADS" pool-name="SybaseXADS">
    <xa-datasource-property name="ServerName">
      localhost
    </xa-datasource-property>
    <xa-datasource-property name="DatabaseName">
      mydatabase
    </xa-datasource-property>
    <xa-datasource-property name="PortNumber">
      4100
    </xa-datasource-property>
    <xa-datasource-property name="NetworkProtocol">
      Tds
    </xa-datasource-property>
    <driver>sybase</driver>
    <xa-pool>
      <is-same-rm-override>false</is-same-rm-override>
    </xa-pool>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <valid-connection-checker class-
```



```

name="org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseValidConnectionC
hecker"/>
    <validate-on-match>true</validate-on-match>
    <background-validation>false</background-validation>
    <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseExceptionSorter"
/>
    </validation>
</xa-datasource>
<drivers>
    <driver name="sybase" module="com.sybase">
        <xa-datasource-class>com.sybase.jdbc4.jdbc.SybXADataSource</xa-
datasource-class>
    </driver>
</drivers>
</datasources>

```

Sybase JDBC 驱动的 module.xml 文件示例

```

<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.sybase">
    <resources>
        <resource-root path="jconn4.jar"/>
    </resources>
    <dependencies>
        <module name="javax.api"/>
        <module name="javax.transaction.api"/>
    </dependencies>
</module>

```

管理 CLI 命令示例

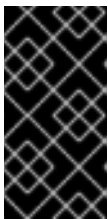
这个配置示例可以用下列管理 CLI 命令来实现。

1. 将 Sybase JDBC 驱动添加为核心模块。

```

module add --name=com.sybase --resources=/path/to/jconn4.jar --
dependencies=javax.api,javax.transaction.api

```



重要

Using the **module** management CLI command to add and remove modules is provided as [technology preview](#) only. This command is not appropriate for use in a managed domain or when connecting to the management CLI remotely. Modules should be [added](#) and [removed](#) manually in a production environment.

2. 注册 Sybase JDBC 驱动。

```

/subsystem=datasources/jdbc-driver=sybase:add(driver-
name=sybase,driver-module-name=com.sybase,driver-xa-datasource-
class-name=com.sybase.jdbc4.jdbc.SybXADataSource)

```

3. 添加 Sybase XA 数据源。

```

xa-data-source add --name=SybaseXADS --jndi-

```

```

name=java:jboss/SybaseXADS --driver-name=sybase --user-name=admin --
password=admin --validate-on-match=true --background-
validation=false --background-validation=false --valid-connection-
checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseValidConnec
tionChecker --exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseExceptionSo
rter --same-rm-override=false --xa-datasource-properties=
{"ServerName"=>"localhost", "DatabaseName"=>"mydatabase", "PortNumber"
=>"4100", "NetworkProtocol"=>"Tds"}

```

13.13.13. MariaDB 数据源示例

这是一个带有连接信息、基本安全性和检验选项的 MariaDB 数据源配置示例。

MariaDB 数据源配置示例

```

<datasources>
  <datasource jndi-name="java:jboss/MariaDBDS" pool-name="MariaDBDS">
    <connection-url>jdbc:mariadb://localhost:3306/jbossdb</connection-url>
    <driver>mariadb</driver>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChe
cker"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>false</background-validation>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter"/>
    </validation>
  </datasource>
  <drivers>
    <driver name="mariadb" module="org.mariadb">
      <driver-class>org.mariadb.jdbc.Driver</driver-class>
      <xa-datasource-class>org.mariadb.jdbc.MySQLDataSource</xa-
datasource-class>
    </driver>
  </drivers>
</datasources>

```

MariaDB JDBC 驱动的 module.xml 文件示例

```

<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="org.mariadb">
  <resources>
    <resource-root path="mariadb-java-client-1.2.3.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
  </dependencies>
</module>

```

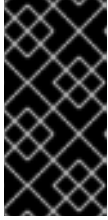
```
<module name="javax.transaction.api"/>
</dependencies>
</module>
```

管理 CLI 命令示例

这个配置示例可以用下列管理 CLI 命令来实现。

1. 将 MariaDB JDBC 驱动添加为核心模块。

```
module add --name=org.mariadb --resources=/path/to/mariadb-java-
client-1.2.3.jar --dependencies=javax.api,javax.transaction.api
```



重要

Using the **module** management CLI command to add and remove modules is provided as [technology preview](#) only. This command is not appropriate for use in a managed domain or when connecting to the management CLI remotely. Modules should be [added](#) and [removed](#) manually in a production environment.

2. 注册 MariaDB JDBC 驱动。

```
/subsystem=datasources/jdbc-driver=mariadb:add(driver-
name=mariadb,driver-module-name=org.mariadb,driver-xa-datasource-
class-name=org.mariadb.jdbc.MySQLDataSource, driver-class-
name=org.mariadb.jdbc.Driver)
```

3. 添加 MariaDB 数据源。

```
data-source add --name=MariaDBDS --jndi-name=java:jboss/MariaDBDS --
driver-name=mariadb --connection-
url=jdbc:mariadb://localhost:3306/jbossdb --user-name=admin --
password=admin --validate-on-match=true --background-
validation=false --valid-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnecti
onChecker --exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSort
er
```

13.13.14. MariaDB XA 数据源示例

这是一个带有 XA 数据源属性、基本安全性和检验选项的 MariaDB XA 数据源配置示例。

MariaDB XA 数据源配置示例

```
<datasources>
  <xa-datasource jndi-name="java:jboss/MariaDBXADS" pool-
name="MariaDBXADS">
    <xa-datasource-property name="ServerName">
      localhost
    </xa-datasource-property>
    <xa-datasource-property name="DatabaseName">
      mariadbdb
    </xa-datasource-property>
    <driver>mariadb</driver>
```

```

<security>
  <user-name>admin</user-name>
  <password>admin</password>
</security>
<validation>
  <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChe
cker"/>
  <validate-on-match>true</validate-on-match>
  <background-validation>>false</background-validation>
  <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter"/>
</validation>
</xa-datasource>
<drivers>
  <driver name="mariadb" module="org.mariadb">
    <driver-class>org.mariadb.jdbc.Driver</driver-class>
    <xa-datasource-class>org.mariadb.jdbc.MySQLDataSource</xa-
datasource-class>
  </driver>
</drivers>
</datasources>

```

MariaDB JDBC 驱动的 `module.xml` 文件示例

```

<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="org.mariadb">
  <resources>
    <resource-root path="mariadb-java-client-1.2.3.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>

```

管理 CLI 命令示例

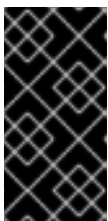
这个配置示例可以用下列管理 CLI 命令来实现。

1. 将 MariaDB JDBC 驱动添加为核心模块。

```

module add --name=org.mariadb --resources=/path/to/mariadb-java-
client-1.2.3.jar --dependencies=javax.api,javax.transaction.api

```



重要

Using the **module** management CLI command to add and remove modules is provided as [technology preview](#) only. This command is not appropriate for use in a managed domain or when connecting to the management CLI remotely. Modules should be [added](#) and [removed](#) manually in a production environment.

2. 注册 MariaDB JDBC 驱动。

```

/subsystem=datasources/jdbc-driver=mariadb:add(driver-

```

```
name=mariadb,driver-module-name=org.mariadb,driver-xa-datasource-  
class-name=org.mariadb.jdbc.MySQLDataSource, driver-class-  
name=org.mariadb.jdbc.Driver)
```

3. 添加 MariaDB XA 数据源。

```
xa-data-source add --name=MariaDBXADS --jndi-  
name=java:jboss/MariaDBXADS --driver-name=mariadb --user-name=admin  
--password=admin --validate-on-match=true --background-  
validation=false --valid-connection-checker-class-  
name=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnecti  
onChecker --exception-sorter-class-  
name=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSort  
er --xa-datasource-properties=  
{"ServerName"=>"localhost", "DatabaseName"=>"mariadbdb"}
```

第 14 章 配置事务

14.1. 事务子系统配置

14.1.1. 配置事务管理者

您可以用基于 Web 的管理控制台，或命令行管理 CLI 配置事务管理者（transaction manager）。

用管理控制台配置事务管理者

下面的步骤解释了如何用基于 Web 的管理控制台配置事务管理者：

1. 从屏幕的顶部选择 **Configuration** 标签页。
2. 如果您在受管域里运行 JBoss EAP，请选择要修改的配置集。
3. 从 **Subsystem** 列表里，选择 **Transactions** 并点击 **View**。
4. 在相关的标签页里点 **Edit** 来对相关设置进行编辑。例如，在 **Recovery** 标签页中对恢复选项进行编辑。
5. 进行修改后点击 **Save** 保存。
6. 点击 **Need Help?** 显示帮助文本。

用管理 CLI 配置事务管理者

通过管理 CLI，您可以用一系列命令来配置事务管理者。对于独立服务器，这些命令以 `/subsystem=transactions` 开始，而对于受管域里的 `default` 配置集，则是 `/profile=default/subsystem=transactions/`。

关于事务管理者配置选项的详细列表，请参考 JBoss EAP [事务管理者配置选项](#)。

14.1.2. 配置数据源来使用 JTA

这个任务向您展示如何在数据源上启用 Java Transaction API（JTA）。

先决条件

- 您的数据库必须支持 JTA。如果不确定，请参考数据库的文档。
- 创建一个 [非 XA 数据源](#)。



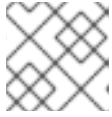
注意

[XA 数据源](#) 在默认情况下已支持 JTA。

配置数据源使用 JTA

1. 请用下列管理 CLI 命令将 `jta` 属性设置为 `true`。

```
/subsystem=datasources/data-source=DATASOURCE_NAME:write-attribute(name=jta,value=true)
```



注意

在受管域里，您必须在这个命令之前使用 `/host=PROFILE_NAME`。

2. 重新载入服务器来使所做的修改生效。

重新载入

您的数据源现在可以使用 JTA 了。

14.1.3. 关于事务日志消息

您可以为事务 Logger 设置 **DEBUG** 日志级别，使日志文件保持可读来跟踪事务状态。如果需要进行更具体的故障排除，则使用 **TRACE** 日志级别。关于配置事务 Logger 的信息，请参考[配置事务子系统的日志](#)。

当配置为 **TRACE** 日志级别时，事务管理者（TM）可以生成大量的日志信息。下面是一些常见的信息。这个列表并不完整，所以您也可能会遇到其他的信息。

表 14.1. 事务状态的改变

Transaction Begin（事务开始）	当事务开始时， <code>com.arjuna.ats.arjuna.coordinator.BasicAction</code> 类的 <code>Begin</code> 方法将被执行，在日志里会出现 <code>BasicAction::Begin() for action-id <transaction uid></code> 信息。
Transaction Commit（事务提交）	当事务被提交时， <code>com.arjuna.ats.arjuna.coordinator.BasicAction</code> 类的 <code>Commit</code> 方法将被执行，在日志里会出现 <code>BasicAction::Commit() for action-id <transaction uid></code> 信息。
Transaction Rollback（事务回滚）	当事务回滚时， <code>com.arjuna.ats.arjuna.coordinator.BasicAction</code> 类的 <code>Rollback</code> 方法将被执行，在日志里会出现 <code>BasicAction::Rollback() for action-id <transaction uid></code> 信息。
Transaction Timeout（事务超时）	当事务超时时， <code>com.arjuna.ats.arjuna.coordinator.TransactionReaper</code> 类的 <code>doCancellations</code> 方法将被执行，在日志里会出现 <code>Reaper Worker <thread id> attempting to cancel <transaction uid></code> 信息。然后您会看到相同线程的事务回滚信息（如前所述）。

14.1.4. 配置事务子系统的日志

您可以控制关于事务的日志信息量，这取决于 JBoss EAP 里的其他日志设置。您可以用管理控制台或 CLI 来配置日志设置。

用管理控制台配置事务 Logger

1. 进入 **Logging** 子系统配置。

- a. 在管理控制台里，点击 **Configuration** 标签页。如果您使用了受管域，则需要首先选择适当的服务器配置集。
 - b. 选择 **Logging** 子系统并点击 **View**。
2. 编辑 **com.arjuna** 属性。
选择 **Log Categories** 标签页。**com.arjuna** 条目已经存在。选择 **com.arjuna** 并点击 **Attributes** 部分的 **Edit**。您可以修改日志级别并选择是否使用父处理程序。
 - 日志级别：
一个事务可能会产生大量日志输出，默认的日志级别被设置为 **WARN**，从而使服务器的日志不会被事务日志大量占用。如果需要查看事务处理的详情，请使用 **TRACE** 日志级别，这会显示事务 ID。
 - 使用父处理程序：
父处理程序指示 **Logger** 是否应该发送输出到其父 **Logger**。其默认设置是 **true**。
 3. 点击 **Save** 保存修改。

用管理 CLI 配置事务 **Logger**

在管理 CLI 里使用下列命令来设置日志级别。对于独立服务器，请从命令里删除 **/profile=default**。

```
/profile=default/subsystem=logging/logger=com.arjuna:write-attribute(name=level,value=VALUE)
```

14.2. 事务管理

14.2.1. 浏览和管理事务

管理 CLI 具有浏览和操纵事务记录的功能。这个功能是通过 TM 和 JBoss EAP 的管理 API 进行交互来实现的。

TM 会把每个待定事务及事务的参与者信息存储到一个持久性存储里，这个持久性存储被称为“对象库（object store）”。管理 API 将对象库作为 **log-store** 资源开放。名为 **probe** 的 API 操作读取事务日志并为每条记录创建一个节点路径。每当需要刷新 **log-store** 时，可以手动调用 **probe** 命令。事务日志快速地显示和消失是正常的。

刷新日志库

这个命令刷新受管域里使用 **default** 配置集的服务器组的日志库。对于独立服务器，请从命令里删除 **profile=default**。

```
/profile=default/subsystem=transactions/log-store=log-store/:probe
```

查看所有预备的事务

要查看所有预备的事务，首先刷新日志库（请参考[刷新日志库](#)），然后运行下列命令（和文件系统的 **ls** 命令类似）。

```
ls /profile=default/subsystem=transactions/log-store=log-store/transactions
```

或者


```
/host=master/server=server-one/subsystem=transactions/log-store=log-
store:read-children-names(child-type=transactions)
```

显示每个事务，包括其唯一标识符。可以为单独的事务运行单独的操作（请参考[管理事务](#)）。

14.2.1.1. 管理事务

查看事务的属性

要查看事务的信息，如 JNDI 名称、EIS 产品名和版本或者状态，请使用 **:read-resource** 管理 CLI 命令。

```
/profile=default/subsystem=transactions/log-store=log-
store/transactions=0\:\ffff7f000001\:-b66efc2\:\4f9e6f8f\:\9:read-resource
```

查看事务的参与者

每条事务日志都包含一个名为 **participants** 的子元素。在此元素上使用 **read-resource** 管理 CLI 命令来查看事务的参与者。参与者是通过它们的 JNDI 名称标识的。

```
/profile=default/subsystem=transactions/log-store=log-
store/transactions=0\:\ffff7f000001\:-
b66efc2\:\4f9e6f8f\:\9/participants=java\:\JmsXA:read-resource
```

结果类似于：

```
{
  "outcome" => "success",
  "result" => {
    "eis-product-name" => "ActiveMQ",
    "eis-product-version" => "2.0",
    "jndi-name" => "java:/JmsXA",
    "status" => "HEURISTIC",
    "type" => "/StateManager/AbstractRecord/XAResourceRecord"
  }
}
```

这里显示的结果状态是 **HEURISTIC**，可以进行恢复。详情请参考[恢复事务](#)。

特殊情况下可以在对象库里创建孤立记录，这是在日志里没有任何对应事务的 XAResourceRecords。例如，XA 资源已预备但在 TM 记录前崩溃，域管理 API 无法进行访问。要访问这样的记录，您需要将管理选项 **expose-all-logs** 设置为 **true**。这个选项没有保存在管理模型里，当服务器重启时会恢复至 **false**。

```
/profile=default/subsystem=transactions/log-store=log-store:write-
attribute(name=expose-all-logs, value=true)
```

您也可以使用以下命令，它会以组合的形式显示事务参与者 ID。

```
/host=master/server=server-one/subsystem=transactions/log-store=log-
store/transactions=0\:\ffff7f000001\:-b66efc2\:\4f9e6f8f\:\9:read-children-
names(child-type=participants)
```

删除事务

每个事务日志都支持 **:delete** 操作来删除事务日志。

```
/profile=default/subsystem=transactions/log-store=log-
store/transactions=0\:ffff7f000001\:-b66efc2\:4f9e6f8f\:9:delete
```

恢复事务

每个事务参与者都支持通过 **:recover** 管理 CLI 命令进行恢复。

```
/profile=default/subsystem=transactions/log-store=log-
store/transactions=0\:ffff7f000001\:-
b66efc2\:4f9e6f8f\:9/participants=2:recover
```

- 如果事务的状态是 **HEURISTIC**，恢复操作修改状态为 **PREPARE** 并触发恢复操作。
- 如果其中一个事务参与者的状态为 **heuristic**，恢复操作会试图重新执行 **commit** 操作。如果成功，参与者将从事务日志里删除。您可以通过，在 **log-store** 上重新运行 **:probe** 操作并检查参与者已不再被列出，来验证这一点。如果这是最后一个参与者，事务也将被删除。

刷新需要恢复事务的状态

如果事务需要恢复，在进行实际恢复前，可以使用 **:refresh** 管理 CLI 命令来确定它当前仍需要恢复。

```
/profile=default/subsystem=transactions/log-store=log-
store/transactions=0\:ffff7f000001\:-
b66efc2\:4f9e6f8f\:9/participants=2:refresh
```

14.2.2. 查看事务统计信息

如果启用了事务管理者统计，您可以通过事务管理者查看已处理事务的统计信息。关于如何启用事务管理者统计的信息，请参考[配置事务管理者](#)。

您可以用管理控制台或管理 CLI 来查看统计信息。在管理控制台里，从 **Runtime** 标签页的 **Transactions** 子系统里可以看到事务统计信息。但管理控制台里并没有所有的统计信息。

下表展示了每个可用的统计及其描述。

表 14.2. 事务子系统统计信息

统计信息	描述
number-of-transactions	这个服务器上事务管理者处理的事务总数。
number-of-committed-transactions	这个服务器上事务管理者提交的事务数量。
number-of-aborted-transactions	这个服务器上事务管理者中止的事务数量。
number-of-timed-out-transactions	这个服务器上事务管理者处理的超时事务数量。这个数量也计入被中止的事务数量。
number-of-heuristics	处于 heuristic 状态的事务的数量。
number-of-inflight-transactions	已开始但还未终止的事务的数量。

统计信息	描述
number-of-application-rollback	失败源自应用程序的失败事务的数量。
number-of-resource-rollback	失败源自资源的失败事务的数量。

14.2.3. 事务对象库

事务需要一个地方来存储对象。其中一个选择是 JDBC 数据源。与文件库相比，JDBC 接口可以从相同网络上的任何系统上访问数据库。如果您使用文件库，在访问数据库之前磁盘必须共享或迁移。如果考虑到性能，JDBC 对象库可能比文件系统或 ActiveMQ 日志对象库要慢。



重要

如果您配置 **transactions** 子系统将 Apache ActiveMQ Artemis 日志用作事务日志的存储类型，那么就不允许两个 JBoss EAP 实例使用相同的目录来存储日志。应用服务器实例不能共享相同的位置，每个实例都需要配置唯一的位置。

将 **JDBC** 数据源用作事务对象库

按照下列步骤将 JDBC 数据源用作事务对象库。

1. 创建数据源（例如，**TransDS**）。相关信息请参阅[创建非 XA 数据源](#)。请注意，数据源的 JDBC 驱动必须[作为一个核心模块被安装](#)，而不是作为一个 JAR 部署，只有这样才可以确保对象存储可以正常工作。
2. 将数据源的 **jta** 属性设置为 **false**。

```
/subsystem=datasources/data-source=TransDS:write-attribute(name=jta,
value=false)
```

3. 把 **jdbc-store-datasource** 属性设置为数据源要使用的 JNDI 名称，如 **java:jboss/datasources/TransDS**。

```
/subsystem=transactions:write-attribute(name=jdbc-store-datasource,
value=java:jboss/datasources/TransDS)
```

4. 把 **use-jdbc-store** 属性设置为 **true**。

```
/subsystem=transactions:write-attribute(name=use-jdbc-store,
value=true)
```

5. 重新载入 JBoss EAP 服务器来使所做的修改生效。

重新载入

下表列出所有和 JDBC 对象存储相关的可用属性。

表 14.3. 事务 JDBC 库属性

属性	描述
use-jdbc-store	把它设置为 true 在事务里启用 JDBC 库。
jdbc-store-datasource	用于存储的 JDBC 数据源的 JNDI 名称。
jdbc-action-store-drop-table	是否在启动时删除或重新创建操作库。默认值是 false 。
jdbc-action-store-table-prefix	Aaction store 表名称的前缀。
jdbc-communication-store-drop-table	启动时是否删除并重新创建 Communication store 表。默认为 false 。
jdbc-communication-store-table-prefix	Communication store 表名称的前缀。
jdbc-state-store-drop-table	启动时是否删除并重新创建 State store 表。默认为 false 。
jdbc-state-store-table-prefix	State store 表名称的前缀。

第 15 章 ORB 配置

15.1. 关于公共对象请求代理架构（CORBA）

公共对象请求代理架构（Common Object Request Broker Architecture，简称 CORBA）是一个标准，使用这个标准，可以使应用程序和服务一起工作，即使它们是使用相互不兼容的不同语言编写的，或部署在不同的平台上。CORBA 请求由一个名为对象请求代理（Object Request Broker，简称 ORB）的服务器端组件进行代理。JBoss EAP 通过 Open JDK ORB 组件提供了一个 ORB 实例。

ORB 用于内部的 JTS 事务，另外也可以被应用程序使用。

15.2. 配置 JTS 事务的 ORB

在默认的 JBoss EAP 安装里，事务的 ORB 支持是禁用的。您可以用管理 CLI 或管理控制台在 **iiop-openjdk** 子系统里配置 ORB。



注意

在受管域里使用 *full* 或 *full-ha* 配置集，或对独立服务器使用 **standalone-ha.xml** 或 **standalone-full-ha.xml** 配置集时，**iiop-openjdk** 子系统都是可用的。

用管理 CLI 配置 ORB

您可以用管理 CLI 配置 ORB 的各个方面。这是用于 JTS 的 ORB 的最小配置。

下面的管理 CLI 命令是为使用 **full** 配置集的受管域配置的。如果有必要，请修改这个配置集来满足您的需要。如果您使用的是独立服务器，请忽略这个命令的 **/profile=full** 部分。

启用安全拦截器

设置这个值为 **identity** 来启用 **security** 属性。

```
/profile=full/subsystem=iiop-openjdk:write-attribute(name=security,value=identity)
```

启用 IIOP 子系统里的事务

要为 JTS 启用 ORB，将属性 **transactions** 的值设置为 **full** 而不是默认的 **spec**。

```
/profile=full/subsystem=iiop-openjdk:write-attribute(name=transactions,value=full)
```

在 Transactions 子系统中启用 JTS

```
/profile=full/subsystem=transactions:write-attribute(name=jts,value=true)
```



注意

要激活 JTS，服务器必须重启，仅重新加载是不够的。

用管理控制台来配置 ORB。

1. 从管理控制台顶部选择 **Configuration** 标签页。

2. 选择 **Subsystems**。在受管域里，需要首先选择适当的配置集。
3. 选择 **IIOP** 子系统并点击 **View**。
4. 点击 **Edit** 按钮并按需要修改属性。关于每个字段的详细说明，请点击 **Need Help?** 链接。
5. 点击 **Save** 保存修改。

第 16 章 JAVA 连接器架构 (JCA) 管理

16.1. 关于 JAVA 连接器架构 (JCA)

Java EE 连接器架构 (Java EE Connector Architecture, 简称 JCA) 为 Java EE 系统和外部的异构企业级信息系统 (EIS) 间定义一个标准的架构。EIS 的例子包括 ERP 系统、大型机事务处理 (TP)、数据库和消息系统。[资源适配器](#)是一个实现 Java EE Connector API 架构的组件。它和数据源对象类似。

JCA 1.7 提供这些管理功能：

- 连接
- 事务
- 安全
- 生命周期
- 工作实例
- 事务流
- 消息流

JCA 1.7 是根据 Java Community Process [JSR-322](#) 开发的。

16.2. 关于资源适配器

资源适配器是提供 Java EE 应用程序和使用 JCA 规格的 EIS 系统间通讯的可部署 Java EE 组件。EIS 厂商经常提供资源适配器，允许他们的产品与 Java EE 应用程序轻易集成。

企业信息系统 (Enterprise Information System, EIS) 可以是机构里的其他软件包系统。如 ERP 系统、数据库系统、电子邮件服务器和专有消息系统。

资源适配器打包在资源适配器归档 (Resource Adapter Archive, RAR) 文件里，它可以部署至 JBoss EAP 服务器。RAR 文件也可以包含在 EAR 部署里。

16.3. 配置 JCA 子系统

`jca` 子系统用来控制 JCA 容器和资源适配器部署的常用设置。您可以使用管理控制台或管理 CLI 来配置 `jca` 子系统。

要配置的主要的 JCA 元素是：

- [归档校验](#)
- [Bean 校验](#)
- [工作管理者](#)
- [引导上下文](#)
- [缓存的连接管理者](#)

用管理控制台配置 JCA 设置

jca 子系统可以在管理控制台里配置，请选择 **Configuration** → **Subsystems** → **JCA**。然后，选择适当的标签页：

- **Common Config** - 包含缓存连接管理者、归档校验、Bean 校验的设置。每个设置都有自己的标签页。打开相应的标签页并点击 **Edit** 按钮来修改这些设置。
- **Bootstrap Contexts** - 包含配置的引导上下文列表。您可以添加、删除和配置新的引导上下文对象。每个引导上下文都必须分配一个工作管理者（Work Manager）。
- **Work Manager** - 包含配置的工作管理者的列表。您可以添加、删除新的工作管理者并配置其线程池。每个工作管理者都可以有一个短期线程池和可选的长期线程池。线程池属性可以点击所选工作管理者的 **View** 来进行配置。

用管理 CLI 配置 JCA 设置

jca 子系统可以通过 `/subsystem=jca` 地址使用管理 CLI 进行配置。在受管域里，您必须在这个命令之前使用 `/host=PROFILE_NAME`。

归档校验

确定是否在部署单元上进行归档校验。下表描述了您可以为归档校验设置的属性。

表 16.1. 归档校验属性

属性	默认值	描述
enabled	true	指定是否启用归档校验。
fail-on-error	true	指定归档校验错误报告是否使部署失败。
fail-on-warn	false	指定归档校验警告报告是否使部署失败。

如果归档没有正确实现 JCA 规格且启用了归档校验，部署期间将出现一个描述该问题的错误信息。例如：

```
Severity: ERROR
Section: 19.4.2
Description: A ResourceAdapter must implement a "public int hashCode()"
method.
Code: com.mycompany.myproject.ResourceAdapterImpl

Severity: ERROR
Section: 19.4.2
Description: A ResourceAdapter must implement a "public boolean
equals(Object)" method.
Code: com.mycompany.myproject.ResourceAdapterImpl
```

如果没有指定归档校验，它会认为是启用的，且 **enabled** 属性默认是 **true**。

Bean 校验

这个设置确定是否在部署单元上执行 Bean 校验（JSR-303）。下表描述了您可以设置的 Bean 校验属性。

表 16.2. Bean 校验属性

属性	默认值	描述
enabled	true	指定是否启用 Bean 校验。

如果没有指定 Bean 校验，它会认为是启用的，且 **enabled** 属性默认是 **true**。

工作管理者

有两种类型的工作管理者：

默认的工作管理者

默认的工作管理者及其线程池。

自定义的工作管理者

自定义的工作管理者及其线程池。

下表描述了您可以为工作管理者设置的属性。

表 16.3. 工作管理者属性

属性	描述
name	指定工作管理者的名称。
short-running-threads	标准 Work 实例的线程池。每个工作管理者都有一个短期线程池。
long-running-threads	设置 LONG_RUNNING 提示的 JCA 1.7 Work 实例的线程池。每个工作管理者可以有一个可选的长期线程池。

下表描述了您可以为工作管理者线程池设置的属性。

表 16.4. 线程池属性

属性	默认值	描述
name	default	指定线程池的名称。
keepalive-time	10 seconds	指定池线程在完成工作后应该保持的时间。
allow-core-timeout	false	布尔值，确定核心线程是否会超时。默认值是 false。
thread-factory		对线程工厂的引用。
max-thread	50	线程池的最大尺寸。
core-threads	50	核心线程池的大小。这必须等于或小于最大的线程池尺寸。
queue-length	50	最大的队列长度。

引导上下文

用于定义自定义引导上下文。下表描述了您可以为引导上下文设置的属性。

表 16.5. 引导上下文属性

属性	描述
name	指定引导上下文的名称。
workmanager	指定用于这个上下文的工作管理者的名称。

缓存的连接管理者

这用于调试连接和支持事务里的连接的惰性征募（lazy enlistment），跟踪应用程序是否正确地使用和释放它们。下表描述了您可以为缓存连接管理者设置的属性。

表 16.6. 缓存连接管理者属性

属性	默认值	描述
debug	false	在失败时输出警告来显性地关闭连接。
error	false	在失败时抛出异常来显性地关闭连接。
ignore-unknown-connections	false	指定不会缓存未知的连接。

16.4. 配置资源适配器

16.4.1. 部署资源适配器

资源适配器可以像其他部署一样用管理 CLI 或管理控制台进行部署。当运行为独立服务器时，您也可以将归档复制到部署目录，让部署扫描器检测到。

用管理 CLI 部署资源适配器

要部署资源适配器至独立服务器，请输入下列管理 CLI 命令。

```
deploy /path/to/resource-adapter.rar
```

要部署资源适配器至受管域里的所有服务器组，请输入下列管理 CLI 命令。

```
deploy /path/to/resource-adapter.rar --all-server-groups
```

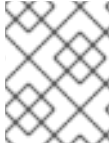
用管理控制台部署资源适配器

1. 登录至管理控制台并点击 **Deployments** 标签页。
2. 点击 **Add**。在受管域里，您首先需要选择 **Content Repository**。
3. 选择 **Upload a new deployment** 选项并点击 **Next**。
4. 浏览资源适配器归档并点击 **Next**。

5. 检验要上传的文件并点击 **Finish**。
6. 在受管域里，分配部署至合适的服务器组并启用部署。

用部署扫描器部署资源适配器。

要手动部署资源适配器至独立服务器，请将资源适配器归档复制到服务器的 `deployments` 目录，例如 `EAP_HOME/standalone/deployments/`。部署扫描器将检测并部署它。



注意

这个选项对于受管域不可用。您必须使用管理控制台或管理 CLI 来部署资源适配器至服务器组。

16.4.2. 配置资源适配器

您可以用管理界面来配置资源适配器。下面的例子展示了如何用管理 CLI 配置资源适配器。关于受支持的属性和其他重要信息，请参考资源适配器厂商的相关文档。

添加资源适配器配置
添加资源适配器配置。

```
/subsystem=resource-adapters/resource-adapter=eis.rar:add(archive=eis.rar,
transaction-support=XATransaction)
```

配置资源适配器设置
按需要配置下列设置。

- 配置 **config-properties**。
添加 **server** 配置属性。

```
/subsystem=resource-adapters/resource-adapter=eis.rar/config-
properties=server:add(value=localhost)
```

添加 **port** 配置属性。

```
/subsystem=resource-adapters/resource-adapter=eis.rar/config-
properties=port:add(value=9000)
```

- 配置 **admin-objects**。
添加一个 **admin** 对象。

```
/subsystem=resource-adapters/resource-adapter=eis.rar/admin-
objects=aoName:add(class-name=com.acme.eis.ra.EISAdminObjectImpl,
jndi-name=java:/eis/AcmeAdminObject)
```

配置 **admin** 对象的属性。

```
/subsystem=resource-adapters/resource-adapter=eis.rar/admin-
objects=aoName/config-properties=threshold:add(value=10)
```

- 配置 **connection-definitions**。
为受管连接工厂添加连接定义。

■

```
/subsystem=resource-adapters/resource-adapter=eis.rar/connection-
definitions=cfName:add(class-
name=com.acme.eis.ra.EISManagedConnectionFactory, jndi-
name=java:/eis/AcmeConnectionFactory)
```

配置受管连接工厂的配置属性。

```
/subsystem=resource-adapters/resource-adapter=eis.rar/connection-
definitions=cfName/config-properties=name:add(value=Acme Inc)
```

配置是否记录征募跟踪信息。您可以将 **enlistment-trace** 属性设置为 **false** 来禁用征募跟踪信息的记录。

```
/subsystem=resource-adapters/resource-adapter=eis.rar/connection-
definitions=cfName:write-attribute(name=enlistment-
trace,value=false)
```



警告

禁用登记跟踪会使在事务登记的过程中查找错误变得比较困难。

关于资源适配器的所有可用的配置属性，请参考[资源适配器属性](#)。

激活资源适配器
激活资源适配器。

```
/subsystem=resource-adapters/resource-adapter=eis.rar:activate
```



注意

您也可以定义资源适配器的功能策略。更多的信息请参考[功能策略](#)章节。

16.4.3. 部署和配置 Websphere MQ 资源适配器

您可以在《为 JBoss EAP 配置消息系统》的 [部署 Websphere MQ 资源适配器](#)里找到相关的说明。

16.4.4. 部署和配置通用的 JMS 资源适配器

您可以在《为 JBoss EAP 配置消息系统》的 [配置通用的 JMS 资源适配器](#)里找到相关的说明。

16.5. 配置受管连接池

JBoss EAP 提供了三个 **ManagedConnectionPool** 接口的实现。

```
org.jboss.jca.core.connectionmanager.pool.mcp.SemaphoreConcurrentLinkedQueueM
anagedConnectionPool
```

这是 JBoss EAP 7 的默认连接池且提供了最佳开箱即用性能。

`org.jboss.jca.core.connectionmanager.pool.mcp.SemaphoreArrayListManagedConnectionPool`

这是之前的 JBoss EAP 版本里的默认连接池。

`org.jboss.jca.core.connectionmanager.pool.mcp.LeakDumperManagedConnectionPool`

这个连接池仅用于调试目的，它将在关闭或冲刷时报告任何泄漏。

您可以用下列管理 CLI 命令为数据源设置受管连接池实现。

```
/subsystem=datasources/data-source=DATA_SOURCE:write-attribute(name=mcp,value=MCP_CLASS)
```

您可以用下列管理 CLI 命令为资源适配器设置受管连接池实现。

```
/subsystem=resource-adapters/resource-adapter=RESOURCE_ADAPTER/connection-definitions=CONNECTION_DEFINITION:write-attribute(name=mcp,value=MCP_CLASS)
```

您可以用下列管理 CLI 命令为消息服务器设置受管连接池实现。

```
/subsystem=messaging-activemq/server=SERVER/pooled-connection-factory=CONNECTION_FACTORY:write-attribute(name=managed-connection-pool,value=MCP_CLASS)
```

16.6. 查看连接统计信息

您可以从 `/deployment=NAME.rar` 子树读取连接的统计信息。这使您可以访问任何 RAR 的统计信息，即使它没有在 `standalone.xml` 或 `domain.xml` 配置里定义。

```
/deployment=NAME.rar/subsystem=resource-adapters/statistics=statistics/connection-definitions=java\:\testMe:read-resource(include-runtime=true)
```



注意

请确保指定了 `include-runtime=true` 参数，因为所有统计都只是运行时信息。

关于可用的统计信息的详情，请参考[资源适配器统计](#)。

第 17 章 配置 WEB 服务器（UNDERTOW）

17.1. UNDERTOW 子系统概述



重要

在 JBoss EAP 7 里，**undertow** 取代了之前 JBoss EAP 版本里的 **web** 子系统。

undertow 子系统允许您配置 Web 服务器和 Servlet 容器设置。它实现了 [Java Servlet 3.1 规格](#) 以及 Websocket 并支持 HTTP Upgrade，而且在 Servlet 部署里使用了高性能的非阻塞式处理程序。**undertow** 子系统也能够充当支持 `mod_cluster` 的高性能反向代理。

在 **undertow** 子系统里要配置 5 个主要的组件：

- [缓冲缓存（buffer caches）](#)
- [服务器（server）](#)
- [servlet 容器](#)
- [处理程序（handlers）](#)
- [过滤器（filters）](#)



注意

虽然 JBoss EAP 提供了升级这些组件的配置的能力，默认的配置还是适用于多数情况且提供了合理的性能设置。

默认的 Undertow 子系统配置

```
<subsystem xmlns="urn:jboss:domain:undertow:3.1">
  <buffer-cache name="default"/>
  <server name="default-server">
    <http-listener name="default" socket-binding="http" redirect-
socket="https"/>
    <host name="default-host" alias="localhost">
      <location name="/" handler="welcome-content"/>
      <filter-ref name="server-header"/>
      <filter-ref name="x-powered-by-header"/>
    </host>
  </server>
  <servlet-container name="default">
    <jsp-config/>
    <websockets/>
  </servlet-container>
  <handlers>
    <file name="welcome-content" path="${jboss.home.dir}/welcome-
content"/>
  </handlers>
  <filters>
    <response-header name="server-header" header-name="Server" header-
value="JBoss-EAP/7"/>
  </filters>
</subsystem>
```

```
<response-header name="x-powered-by-header" header-name="X-Powered-By" header-value="Undertow/1"/>
</filters>
</subsystem>
```



重要

undertow 子系统也依赖 **io** 子系统来提供 XNIO 工作节点和缓冲池。**io** 子系统是单独配置的，它提供了在多数情况下应该展现优化性能的默认配置。



注意

和之前的 JBoss EAP 版本里的 **web** 子系统相比，JBoss EAP 7 里的 **undertow** 子系统有不同的 [HTTP 方法](#) 的默认行为。

17.2. 配置缓冲缓存

缓冲缓存 (Buffer cache) 被用来缓存静态资源。JBoss EAP 启用了部署可引用和配置的多组缓存，允许不同的部署使用不同的缓存大小。缓冲在区 (region) 中进行分配且大小固定。使用的总共空间可以通过缓冲大小乘以每个区的缓冲数量以及最大的区数量来计算。默认的缓冲缓存大小是 10MB。

JBoss EAP 默认提供单个缓存：

默认的 Undertow 子系统配置

```
<subsystem xmlns="urn:jboss:domain:undertow:3.1">
  <buffer-cache name="default"/>
  ....
</subsystem>
```

更新现有的缓冲缓存

要更新现有的缓冲缓存：

```
/subsystem=undertow/buffer-cache=default/:write-attribute(name=buffer-size,value=2048)
```

重新载入

创建新的缓冲缓存

要创建新的缓冲缓存：

```
/subsystem=undertow/buffer-cache=new-buffer:add
```

删除缓冲缓存

要删除缓冲缓存：

```
/subsystem=undertow/buffer-cache=new-buffer:remove
```

重新载入

关于缓冲缓存的可配置属性的完整列表，请参考 [Undertow 子系统属性](#) 章节。

17.3. 配置服务器

服务器代表一个 Undertow 实例并由几个元素组成：

- host
- http-listener
- https-listener
- ajp-listener

host 元素提供虚拟主机配置，而三个 listener 元素则提供到 Undertow 实例的不同类型的连接。



注意

您可以配置多个服务器，从而允许部署和服务器完全隔离。在某些情况下，如多租户环境里，这很有用。

JBoss EAP 默认提供单个服务：

默认的 Undertow 子系统配置

```
<subsystem xmlns="urn:jboss:domain:undertow:3.1">
  <buffer-cache name="default"/>
  <server name="default-server">
    <http-listener name="default" socket-binding="http" redirect-
socket="https"/>
    <host name="default-host" alias="localhost">
      <location name="/" handler="welcome-content"/>
      <filter-ref name="server-header"/>
      <filter-ref name="x-powered-by-header"/>
    </host>
  </server>
  ...
</subsystem>
```

更新现有的服务器

要更新现有的服务器：

```
/subsystem=undertow/server=default-server:write-attribute(name=default-
host,value=default-host)
```

重新载入

创建新的服务器

要创建新的服务器

```
/subsystem=undertow/server=new-server:add
```

重新载入

删除服务器

要删除服务器：

```
/subsystem=undertow/server=new-server:remove
```

重新载入

关于服务器的可配置属性的完整列表，请参考 [Undertow 子系统属性](#) 章节。

17.4. 配置 SERVLET 容器

Servlet 容器提供所有 servlet、JSP 和 Websocket 相关的配置，包括和会话相关的配置。虽然多数服务器只需要单个的 Servlet 容器，您也可以添加额外的 *servlet-container* 元素来配置多个 Servlet 容器。使用多个 Servlet 容器可以把多个部署部署至不同虚拟主机的相同上下文路径上。



注意

Servlet 容器提供的多数配置都可以被部署应用程序的 **web.xml** 文件单独覆盖。

JBoss EAP 默认提供了一个 Servlet 容器：

默认的 Undertow 子系统配置

```
<subsystem xmlns="urn:jboss:domain:undertow:3.1">
  <buffer-cache name="default"/>
  <server name="default-server">
    ...
  </server>
  <servlet-container name="default">
    <jsp-config/>
    <websockets/>
  </servlet-container>
  ...
</subsystem>
```

更新现有的 Servlet 容器

更新现有的 Servlet 容器：

```
/subsystem=undertow/servlet-container=default:write-attribute(name=ignore-flush,value=true)
```

重新载入

创建新的 Servlet 容器

创建新的 Servlet 容器：

```
/subsystem=undertow/servlet-container=new-servlet-container:add
```

重新载入

删除 Servlet 容器

删除 Servlet 容器：

-

```
/subsystem=undertow/servlet-container=new-servlet-container:remove
```

重新载入

关于 Servlet 容器的可配置属性的完整列表，请参考 [Undertow 子系统属性](#) 章节。

17.5. 配置处理程序

JBoss EAP 允许配置两种类型的处理程序：

- 文件处理程序
- reverse-proxy 处理程序

文件处理程序服务静态文件。每个文件处理程序必须附加到虚拟主机里的某个位置。Reverse-proxy 处理程序允许 JBoss EAP 充当高性能的反向代理。它可以处理 AJP、HTTP 和 HTTP.2 后台并支持 mod_cluster。

JBoss EAP 默认提供了一个文件处理程序：

默认的 Undertow 子系统配置

```
<subsystem xmlns="urn:jboss:domain:undertow:3.1">
  <buffer-cache name="default"/>
  <server name="default-server">
    ...
  </server>
  <servlet-container name="default">
    ...
  </servlet-container>
  <handlers>
    <file name="welcome-content" path="${jboss.home.dir}/welcome-
content"/>
  </handlers>
  ...
</subsystem>
```

对静态资源使用 WebDAV

之前的 JBoss EAP 版本允许将 WebDAV 和 **web** 子系统一起使用（**WebdavServlet**）来容纳静态资源，并启用额外的 HTTP 方法以访问和操作这些文件。在 JBoss EAP 7 里，**undertow** 子系统提供了通过文件处理程序服务静态文件的机制。但 **undertow** 子系统不支持 WebDAV。如果您想在 JBoss EAP 7 里使用 WebDAV，可以编写自定义的 WebDAV servlet。

更新现有的文件处理程序

更新现有的文件处理程序：

```
/subsystem=undertow/configuration=handler/file=welcome-content:write-
attribute(name=case-sensitive,value=true)
```

重新载入

创建新的文件处理程序

创建新的文件处理程序：

```
/subsystem=undertow/configuration=handler/file=new-file-handler:add
```

删除文件处理程序

删除文件处理程序

```
/subsystem=undertow/configuration=handler/file=new-file-handler:remove
```

重新载入

关于处理程序的可配置属性的完整列表，请参考 [Undertow 子系统属性](#) 章节。

17.6. 配置过滤器

过滤器可以修改请求的某些方面，且可以使用 predicate 来控制过滤器何时执行。过滤器的常见用例包括：设置头数据、进行 GZIP 压缩。



注意

过滤器从功能上来说等同于之前的 JBoss EAP 版本里使用的全局阀（Valve）。

您可以定义下列过滤器类型：

- custom-filter
- error-page
- expression-filter
- gzip
- mod-cluster
- request-limit
- response-header
- rewrite

JBoss EAP 默认提供两个过滤器：

默认的 Undertow 子系统配置

```
<subsystem xmlns="urn:jboss:domain:undertow:3.1">
  <buffer-cache name="default"/>
  <server name="default-server">
    ...
  </server>
  <servlet-container name="default">
    ...
  </servlet-container>
  <handlers>
    ...
  </handlers>
  <filters>
```

```

        <response-header name="server-header" header-name="Server" header-
value="JBoss-EAP/7"/>
        <response-header name="x-powered-by-header" header-name="X-Powered-
By" header-value="Undertow/1"/>
    </filters>
</subsystem>

```

更新现有的过滤器

要更新现有的过滤器：

```

/subsystem=undertow/configuration=filter/response-header=server-
header:write-attribute(name=header-value,value="JBoss-EAP")

```

重新载入

创建新的过滤器

要创建新的过滤器：

```

/subsystem=undertow/configuration=filter/response-header=new-response-
header:add(header-name=new-response-header,header-value="My Value")

```

删除过滤器

要删除过滤器：

```

/subsystem=undertow/configuration=filter/response-header=new-response-
header:remove

```

重新载入

关于过滤器的可配置属性的完整列表，请参考 [Undertow 子系统属性](#) 章节。

17.7. 配置默认的 WELCOME WEB 应用程序

JBoss EAP 包含一个默认的 **welcome** 应用程序，它默认显示在端口 8080 的根上下文里。

在 Undertow 里预配置了一个默认的服务器来提供欢迎内容。

默认的 Undertow 子系统配置

```

<subsystem xmlns="urn:jboss:domain:undertow:3.1">
    ...
    <server name="default-server">
        <http-listener name="default" socket-binding="http" redirect-
socket="https"/>
        <host name="default-host" alias="localhost">
            <location name="/" handler="welcome-content"/>
            <filter-ref name="server-header"/>
            <filter-ref name="x-powered-by-header"/>
        </host>
    </server>
    ...
    <handlers>
        <file name="welcome-content" path="${jboss.home.dir}/welcome-

```

```
content"/>
</handlers>
...
</subsystem>
```

默认的服务器 (**default-server**) 配置了一个默认的主机 (**default-host**)。这个默认的主机用 **<location>** 元素和 **welcome-content** 文件处理程序处理到服务器根目录的请求。**welcome-content** 处理程序在 **path** 属性指定的位置上提供内容。

您可以用自己的 Web 应用程序替换默认的 **Welcome** 应用程序。这可以用下列两种方法之一来配置：

- 修改 **welcome-content** 文件处理程序
- 您也可以修改 **default-web-module**

您也可以禁用欢迎内容。

修改 welcome-content 文件处理程序

修改现有的 **welcome-content** 文件处理程序的路径来指向新的部署。

```
/subsystem=undertow/configuration=handler/file=welcome-content:write-
attribute(name=path,value="/path/to/content")
```

注意

或者，您可以创建一个服务器根目录使用的不同文件处理程序。

```
/subsystem=undertow/configuration=handler/file=NEW_FILE_HANDLER
:add(path="/path/to/content")
/subsystem=undertow/server=default-server/host=default-
host/location=/:write-
attribute(name=handler,value=NEW_FILE_HANDLER)
```

重新载入服务器来使所做的修改生效。

重新载入

修改 default-web-module

映射部署的 Web 应用程序至服务器的根目录。

```
/subsystem=undertow/server=default-server/host=default-host:write-
attribute(name=default-web-module,value=hello.war)
```

重新载入服务器来使所做的修改生效。

重新载入

禁用默认的 Welcome Web 应用程序

通过删除 **default-host** 的 **location** 条目 (/) 来禁用欢迎应用程序。

```
/subsystem=undertow/server=default-server/host=default-
host/location=/:remove
```

重新载入服务器来使所做的修改生效。

重新载入

17.8. 配置 HTTPS

关于配置 HTTPS 用于 Web 应用程序以及管理接口的更多信息，请参考[如何配置服务器安全性](#)。

17.9. 配置 HTTP 会话超时

HTTP 会话超时定义声明 HTTP 会话无效所需的失效时间。例如，用户访问部署在 JBoss EAP 里并创建 HTTP 会话的应用程序。如果这个用户在 HTTP 会话超时后试图再次访问该应用程序，原来的 HTTP 会话将失效，而用户将被迫创建一个新的 HTTP 会话。这可能导致非持久性数据的丢失或必须进行重新验证。

HTTP 超时会话是在应用程序的 `web.xml` 文件里配置的，但默认的 HTTP 会话超时可以在 JBoss EAP 里指定。服务器的超时值将适用于所有部署的应用程序，但应用程序的 `web.xml` 将覆盖服务器的值。

服务器的值是在 `default-session-timeout` 属性里指定的，它位于 `undertow` 子系统的 `servlet-container` 部分。`default-session-timeout` 的单位是分钟，默认值是 30。

配置默认的会话超时

要配置 `default-session-timeout`

```
/subsystem=undertow/servlet-container=default:write-attribute(name=default-session-timeout, value=60)
```

重新载入



重要

修改 HTTP 会话超时要求所有受影响的 JBoss EAP 实例重启。在重启完全之前，仍将应用原始的 HTTP 会话超时。

17.10. 配置 HTTP-ONLY 会话管理 COOKIE

会话管理 Cookie 可以通过 HTTP API 和非 HTTP API（如 JavaScript）访问。JBoss EAP 提供将 `HttpOnly` 头部作为 `Set-Cookie` 响应头部的一部分发送给客户（通常是浏览器）的能力。在被支持的浏览器里，启用这个头部告诉浏览器它应该防止通过非 HTTP API 来访问会话管理 Cookie。限制会话管理 Cookie 至 HTTP API 有助于减轻通过跨站点脚本攻击窃取会话 Cookie 的威胁。要启用这个行为，`http-only` 属性应该被设置为 `true`。



重要

使用 `HttpOnly` 头部自身无法实际防止跨站点脚本攻击，它仅仅是通知浏览器而已。浏览器也必须支持 `HttpOnly` 以使这个行为生效。



重要

使用 `http-only` 属性仅将这个限制应用到会话管理 Cookie 而不是其他的浏览器 Cookie。

`http-only` 属性在 `undertow` 子系统的两个地方进行设置：

- 在 Servlet 容器里作为会话 Cookie 设置
- 在服务器的主机设置部分作为单点登录属性

为 Servlet 容器会话 Cookie 配置 *host-only* 属性

要为 Servlet 容器会话 Cookie 配置 *host-only* 属性：

```
/subsystem=undertow/servlet-container=default/setting=session-cookie:add
```

```
/subsystem=undertow/servlet-container=default/setting=session-cookie:write-attribute(name=http-only,value=true)
```

重新载入

为主机单点登录配置 *host-only*

要为主机单点登录配置 *host-only*

```
/subsystem=undertow/server=default-server/host=default-host/setting=single-sign-on:add
```

```
/subsystem=undertow/server=default-server/host=default-host/setting=single-sign-on:write-attribute(name=http-only,value=true)
```

重新载入

17.11. 配置 HTTP/2

Undertow 允许使用 [HTTP/2](#) 标准，这通过压缩头部和在相同的 TCP 连接上多路传输多个数据流来减少延迟。它也为服务器提供了，在客户请求资源前将资源主动推送至客户端的能力，从而可以提高页面加载的速度。Undertow 也与 HTTP/2 以前的 [SPDY](#) 相兼容，因此可以支持没有更新至新规格的客户端。



重要

在 JBoss EAP 7.0 里 HTTP/2 仅作为技术预览支持，它只能用于支持 HTTP/2 标准的浏览器里。



重要

HTTP/2 不但需要使用 Java 8，而且还需要在 Casspath 上设立 [ALPN](#)。这是因为 HTTP/2 需要支持 ALPN 的 TLS 栈，而 Java 8 的默认安装是不提供的。

17.11.1. 配置 Undertow 来使用 HTTP/2

要配置 Undertow 使用 HTTP/2，请完成下列步骤：

配置 Undertow 使用 HTTPS

关于配置 Undertow 将 HTTPS 用于 Web 应用程序，请参考[如何配置服务器的安全性](#)。



注意

换句话说，使用 HTTP/2 时不需要 HTTPS，只要使用 HTTP Upgrade 的普通 HTTP 就可以了。此时，您不需要安装 ALPN，只要在 Undertow 里简单地启用 HTTP/2：

```
/subsystem=undertow/server=default-server/http-  
listener=default:write-attribute(name=enable-http2,value=true)
```

下载 ALPN JAR

首先确定 Java 的具体版本。从终端运行下列命令来查看 Java 的版本：

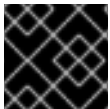
```
java -version
```

根据您的版本，请访问[这个页面](#)来确定从[这个页面](#)下载正确的 ALPN JAR 版本。例如，如果您运行的是 Java 1.8.0_51，您需要 ALPN 8.1.4.v20150727 并下载 `alpn-boot-8.1.4.v20150727.jar`。

把 ALPN JAR 添加到 boot Classpath

在下载了正确的 ALPN JAR 版本后，请将其复制到 `EAP_HOME/bin`。您也必须添加下列内容至 `bin/standalone.conf`（如果运行在受管域里则是 `bin/domain.conf`），并用合适的值替换 `$JBOSS_HOME` 和 `$ALPN_VERSION`。

```
JAVA_OPTS="$JAVA_OPTS -Xbootclasspath/p:$JBOSS_HOME/bin/alpn-boot-  
$ALPN_VERSION.jar"
```



重要

您必须重启 JBoss EAP 以使 classpath 的修改生效。

在 HTTPS Listener 启用 HTTP/2

要在 Undertow 里启用 HTTPS Listener 来使用 HTTP/2，需要把 `enable-http2` 属性设置为 `true`：

```
/subsystem=undertow/server=default-server/https-listener=https:write-  
attribute(name=enable-http2,value=true)
```

检验 HTTP/2 是否被使用

要检验 Undertow 是否在使用 HTTP/2，您需要检查来自 Undertow 的头部信息。用 HTTPS 访问您的 JBoss EAP 实例，例如 <https://localhost:8443>，并使用您的浏览器的开发人员工具来检查头部信息。某些浏览器，如 Google Chrome，在使用 HTTP/2 时将显示 HTTP/2 伪头部信息（`:path`、`:authority`、`:method` 和 `:scheme`），而其他浏览器（如 Firefox 和 Safari）将会报告头部的状态或版本为 `HTTP/2.0`。

17.12. 配置 REQUESTDUMPING 处理程序

RequestDumping 处理程序（`io.undertow.server.handlers.RequestDumpingHandler`）登记 JBoss EAP 里 Undertow 处理的请求，以及相关的响应对象的细节。



重要

虽然这个处理程序可用于调试，但它可能也会记录下敏感信息。因此，在启用这个处理程序时请记住这一点。

**注意**

RequestDumping 处理程序替代了之前的 JBoss EAP 版本里的 *RequestDumperValve*。

您可以直接在服务器级别或单独的应用程序里配置 *RequestDumping* 处理程序。

17.12.1. 配置服务器上的 RequestDumping 处理程序

RequestDumping 处理程序应该配置为表达式过滤器。要将 *RequestDumping* 处理程序配置为表达式过滤器，您需要：

用 *RequestDumping* 处理程序创建一个新的表达式过滤器

```
/subsystem=undertow/configuration=filter/expression-  
filter=requestDumperExpression:add(expression="dump-request")
```

在 **Undertow Web Server** 里启用表达式过滤器

```
/subsystem=undertow/server=default-server/host=default-host/filter-  
ref=requestDumperExpression:add
```

**重要**

以这种方式将 *RequestDumping* 启用为表达式过滤器时，Undertow Web Server 处理的所有请求和对应的响应将被记录。

为特定的 **URL** 配置 *RequestDumping* 处理程序

除了记录所有请求以外，您也可以使用表达式过滤器来记录特定 URL 的请求和对应的响应。这可以通过在表达式使用 predicate 来实现，如 *path*、*path-prefix* 或 *path-suffix*。例如，如果您想记录所有的请求和对应的响应至 */myApplication/test*，在创建表达式过滤器时您可以使用表达式 **"path(/myApplication/test) -> dump-request"** 而不是 **"dump-request"**。这将带有完全匹配 */myApplication/test* 路径的请求指引至 *RequestDumping* 处理程序。

17.12.2. 在应用程序里配置 RequestDumping 处理程序

除了在服务器里配置 *RequestDumping* 处理程序，您还可以在单独的应用程序里配置它。这将限制处理程序的作用域至特定的应用程序。*RequestDumping* 处理程序应该在 **WEB-INF/undertow-handlers.conf** 里进行配置。

要在 **WEB-INF/undertow-handlers.conf** 里配置 *RequestDumping* 处理程序以记录这个应用程序的所有请求和对应的响应，请添加下列表达式至 **WEB-INF/undertow-handlers.conf**：

WEB-INF/undertow-handlers.conf 示例

```
dump-request
```

要在 **WEB-INF/undertow-handlers.conf** 里配置 *RequestDumping* 应用程序来记录这个应用程序里特定 URL 的请求和响应，您可以在表达式里使用 predicate，如 *path*、*path-prefix* 或 *path-suffix*。例如，要记录所有的请求和对应的响应至您的应用程序里的 **test**，您可以使用下列带有 *path* predicate 的表达式：

WEB-INF/undertow-handlers.conf 示例

```
path(/test) -> dump-request
```



注意

当在应用程序的**WEB-INF/undertow-handlers.conf** 里定义的表达式里使用 *path*、*path-prefix* 或 *path-suffix* 等 predicate 时，所有的值将和应用程序的根上下文相关。例如，如果应用程序的根上下文是 *myApplication* 并在 **WEB-INF/undertow-handlers.conf** 里配置了表达式 **path(/test) -> dump-request**，它只会记录请求和对应的响应至 */myApplication/test*。

第 18 章 配置 REMOTING

18.1. 关于 REMOTING 子系统

remoting 子系统允许您为本地和远程服务配置入站（inbound）和出站（outbound）连接，以及这些连接的设置。

JBoss Remoting 包括以下可配置的元素：端点（endpoint）、连接器（connector），以及一系列的本地和远程 URI。除非用户的应用程序需要使用自定义的连接器，用户一般不需要配置 **remoting** 子系统。那些需要作为远程客户端的应用程序（如 EJB）则需要独立的配置来连接到一个特定的连接器。

默认的 Remoting 子系统配置

```
<subsystem xmlns="urn:jboss:domain:remoting:3.0">
  <endpoint/>
  <http-connector name="http-remoting-connector" connector-ref="default"
    security-realm="ApplicationRealm"/>
</subsystem>
```

如需获得 **remoting** 子系统可用的完整属性列表，请参阅 [Remoting 子系统属性](#)。

Remoting 端点

Remoting 端点使用了 **io** 子系统声明和配置的 XNIO 工作节点。

如需了解如何配置远程端点的信息，请参阅[配置端点](#)。

连接器

连接器是主要的 Remoting 配置元素。JBoss EAP 允许多个连接器。每个连接器都由带有几个子元素及其他一些属性的 **<connector>** 元素组成。默认的连接被 JBoss EAP 的几个子系统使用。自定义连接器的元素和属性的专有设置取决于您的应用程序。更多信息请联系红帽全球支持服务。

如需了解如何配置连接器的信息，请参阅[配置一个连接器](#)。

出站连接

您可以指定三种不同类型的出站连接：

- 由一个 URI 指定的[出站连接](#)
- 连接到一个本地资源（如一个端口）的[本地出站连接](#)
- 连接到一个远程资源，并使用一个安全域（security realm）进行验证的[远程出站连接](#)

额外配置

Remoting 还依赖于 **remoting** 子系统以外的一些元素，如网络接口和 IO worker。

如需了解更多信息，请参阅[额外的远程配置](#)。

18.2. 配置端点



重要

在 JBoss EAP 6 中，worker 线程池在 **remoting** 子系统中直接配置。在 JBoss EAP 7 中，remoting 端点配置会从 **io** 子系统中参考一个 worker。

JBoss EAP 默认提供下列端点配置。

```
<subsystem xmlns="urn:jboss:domain:remoting:3.0">
  <endpoint/>
  ...
</subsystem>
```

更新现有的端点配置

```
/subsystem=remoting/configuration=endpoint:write-attribute(name=authentication-retries,value=2)
```

重新载入

创建新的端点配置

```
/subsystem=remoting/configuration=endpoint:add
```

删除端点配置

```
/subsystem=remoting/configuration=endpoint:remove
```

重新载入

如需了解端点配置可用的属性，请参阅[端点属性](#)。

18.3. 配置连接器

连接器是和 Remoting 相关的主要配置元素，它包含几个用于其他配置的子元素。

更新现有的连接器配置

```
/subsystem=remoting/connector=new-connector:write-attribute(name=socket-binding,value=my-socket-binding)
```

重新载入

创建新的连接器

```
/subsystem=remoting/connector=new-connector:add(socket-binding=my-socket-binding)
```

删除连接器

```
/subsystem=remoting/connector=new-connector:remove
```

重新载入

关于配置连接器的可用属性的完整列表，请参考 [Remoting 子系统属性](#) 章节。

18.4. 配置 HTTP 连接器

HTTP 连接器提供基于 HTTP Upgrade 的 Remoting 连接器的配置。JBoss EAP 默认提供了下列 **http-connector** 配置。

```
<subsystem xmlns="urn:jboss:domain:remoting:3.0">
    ...
    <http-connector name="http-remoting-connector" connector-ref="default"
security-realm="ApplicationRealm"/>
</subsystem>
```

在默认情况下，这个 HTTP connector 会连接到一个名为 **default** 的 HTTP listener 中（在 **undertow** 子系统中配置）。如需了解更多相关信息，请参阅[配置 Web 服务器（Undertow）](#)。

更新现有的 HTTP 连接器配置

```
/subsystem=remoting/http-connector=new-connector:write-
attribute(name=connector-ref,value=new-connector-ref)
```

重新载入

创建新的 HTTP 连接器

```
/subsystem=remoting/http-connector=new-connector:add(connector-
ref=default)
```

删除 HTTP 连接器

```
/subsystem=remoting/http-connector=new-connector:remove
```

如需了解配置一个 HTTP connector 的完整属性列表，请参阅[连接器属性](#)。

18.5. 配置出站连接

出站连接是一个由 URI 完全指定的通用 remoting 出站连接。

更新现有的出站连接

```
/subsystem=remoting/outbound-connection=new-outbound-connection:write-
attribute(name=uri,value=http://example.com)
```

创建新的出站连接

```
/subsystem=remoting/outbound-connection=new-outbound-
connection:add(uri=http://example.com)
```

删除出站连接

```
/subsystem=remoting/outbound-connection=new-outbound-connection:remove
```

如需了解配置一个出站连接的完整属性列表，请参阅[出站连接属性](#)。

18.6. 配置远程出站连接

远程出站连接是由协议、出站连接套接字绑定、用户名和安全区指定的。这个协议可以是 **remote**、**http-remoting** 或 **https-remoting**。

更新现有的远程出站连接

```
/subsystem=remoting/remote-outbound-connection=new-remote-outbound-connection:write-attribute(name=outbound-socket-binding-ref,value=outbound-socket-binding)
```

创建新的远程出站连接

```
/subsystem=remoting/remote-outbound-connection=new-remote-outbound-connection:add(outbound-socket-binding-ref=outbound-socket-binding)
```

删除远程出站连接

```
/subsystem=remoting/remote-outbound-connection=new-remote-outbound-connection:remove
```

如需了解配置一个远程出站连接的完整属性列表，请参阅[远程出站连接属性](#)。

18.7. 配置本地出站连接

本地出站连接是一个由转出套接字绑定指定的使用 **local** 协议的 Remoting 出站连接。

更新现有的本地出站连接

```
/subsystem=remoting/local-outbound-connection=new-local-outbound-connection:write-attribute(name=outbound-socket-binding-ref,value=outbound-socket-binding)
```

创建新的本地出站连接

```
/subsystem=remoting/local-outbound-connection=new-local-outbound-connection:add(outbound-socket-binding-ref=outbound-socket-binding)
```

删除本地出站连接

```
/subsystem=remoting/local-outbound-connection=new-local-outbound-connection:remove
```

如需了解配置一个本地出站连接的完整属性列表，请参阅[本地出站连接属性](#)。

18.8. 额外的远程配置

一些远程项需要在 **remoting** 子系统以外配置。

IO worker

使用下列命令设置 Remoting 的 IO 工作节点：

■

```
/subsystem=remoting/configuration=endpoint:write-attribute(name=worker,
value=WORKER_NAME)
```

如需了解如何配置一个 IO worker 的信息，请参阅[配置工作节点](#)。

网络接口

remoting 子系统使用的网络接口是 **public**，这个接口同时也被其它子系统使用，因此在需要修改它时请小心处理。

```
<interfaces>
  <interface name="management">
    <inet-address
value="${jboss.bind.address.management:127.0.0.1}"/>
  </interface>
  <interface name="public">
    <inet-address value="${jboss.bind.address:127.0.0.1}"/>
  </interface>
  <interface name="unsecure">
    <inet-address value="${jboss.bind.address.unsecure:127.0.0.1}"/>
  </interface>
</interfaces>
```

在一个受管域中，每个主机在它的 **host.xml** 文件中都定义了 **public** 接口。

套接字绑定

remoting 子系统使用的默认套接字绑定是绑定到端口 **8080**。

如需了解更多与套接字绑定和套接字绑定组相关的信息，请参阅[套接字绑定](#)。

EJB 的远程 connector 参考

ejb3 子系统包括了一个到远程 connector 的参考用来进行远程方法的调用。以下是默认的设置：

```
<remote connector-ref="remoting-connector" thread-pool-name="default"/>
```

安全传输配置

如果客户有要求，Remoting 传输使用 STARTTLS 来保护连接安全，如 HTTPS、Secure Servlet。安全和非安全的连接使用相同的套接字绑定（网络端口），所以不需要额外的服务器端的配置。客户根据需要请求安全或非安全的传输方式。使用 Remoting 的 JBoss EAP 组件（如 EJB、ORB 和 JMS 供应商）都默认请求使用安全的接口。



警告

STARTTLS 的工作方式是，当客户端请求一个安全连接时，它会激活一个安全连接，在其它情况下，都会使用一个未加密的连接。这就可能会导致“中间人（man-in-the-middle）”攻击：攻击者会截获客户端请求，然后把请求修改为请求一个未加密的连接。为了避免这个问题，客户端需要对接收到的连接进行检查。如果请求的是一个安全的连接，而返回的是一个未加密的连接，则需要相关的错误处理。

第 19 章 配置 IO 子系统

19.1. IO 子系统概述

io 子系统定义其他子系统（如 Undertow 和 Remoting）使用的 XNIO **工作节点（worker）** 和 **缓冲池**。这些工作节点和缓冲池在 **io** 子系统的下列组件里定义：

默认的 IO 子系统配置

```
<subsystem xmlns="urn:jboss:domain:io:1.1">
  <worker name="default"/>
  <buffer-pool name="default"/>
</subsystem>
```

19.2. 配置工作节点

工作节点（worker）就是 XNIO 工作节点实例，它是 Java NIO API 的抽象层，提供管理 IO 和工作节点线程的功能，及对 SSL 的支持。在默认情况下，JBoss EAP 提供一个名为 **default** 的工作节点，但您也可以定义多个工作节点。

更新现有的工作节点

要更新现有的工作节点：

```
/subsystem=io/worker=default:write-attribute(name=io-threads,value=10)
```

重新载入

创建新的工作节点

要创建新的工作节点：

```
/subsystem=io/worker=newWorker:add
```

删除工作节点

要删除工作节点：

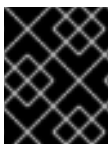
```
/subsystem=io/worker=newWorker:remove
```

重新载入

关于配置工作节点的可用属性的完整列表，请参考 [IO 子系统属性](#) 章节。

19.3. 配置缓冲池

缓冲池就是包括一组 NIO 缓冲实例的池。



重要

修改缓冲大小对应用程序的性能有巨大的影响。对于多数的服务器来说，理想的大小通常是 16K。

更新现有的缓冲池

要更新现有的缓冲池：

```
/subsystem=io/buffer-pool=default:write-attribute(name=direct-  
buffers,value=true)
```

重新载入

创建缓冲池

要创建新的缓冲池：

```
/subsystem=io/buffer-pool=newBuffer:add
```

删除缓冲池

要删除缓冲池：

```
/subsystem=io/buffer-pool=newBuffer:remove
```

重新载入

关于配置缓冲池的可用属性的完整列表，请参考 [IO 子系统属性](#) 章节。

第 20 章 配置批应用程序

JBoss EAP 7 引入了 [JSR-352](#) 里定义的 Java 批（batch）应用程序。您可以为运行批应用程序配置环境并用 **batch-jberet** 子系统管理批任务。

关于开发批应用程序的信息，请参考《JBoss EAP 开发指南》里的『[Java 批应用程序开发](#)』章节。

20.1. 配置批任务

您可以用基于 JBeret 实现的 **batch-jberet** 子系统配置批任务。

默认的 **batch-jberet** 子系统配置定义了一个 in-memory 任务库和默认的线程池设置。

```
<subsystem xmlns="urn:jboss:domain:batch-jberet:1.0">
  <default-job-repository name="in-memory"/>
  <default-thread-pool name="batch"/>
  <job-repository name="in-memory">
    <in-memory/>
  </job-repository>
  <thread-pool name="batch">
    <max-threads count="10"/>
    <keepalive-time time="30" unit="seconds"/>
  </thread-pool>
</subsystem>
```

在默认情况下，任何批任务都会在服务器挂起时停止，并在服务器恢复运行时重新开始。您可以将 **restart-jobs-on-resume** 属性设置为 **false** 以让任务保持在 **STOPPED** 状态。

```
/subsystem=batch-jberet:write-attribute(name=restart-jobs-on-resume,value=false)
```

您也可以为批任务仓库和线程池配置设置。

20.1.1. 配置批任务仓库

本节介绍了如何使用管理 CLI 配置 in-memory 和 JDBC 任务仓库来存储批任务信息。您也可以使用管理控制台来配置任务仓库。通过 **Configuration** 标签页进入 **Batch** 子系统，从左面的菜单中选 **In Memory** 或 **JDBC**。

添加 **In-memory** 任务仓库

您也可以添加在内存里保存批任务信息的任务仓库。

```
/subsystem=batch-jberet/in-memory-job-repository=REPOSITORY_NAME:add
```

添加 **JDBC** 任务仓库

您也可以添加在数据库里保存任务信息的任务仓库。您必须指定 **data-source** 来连接数据库。

```
/subsystem=batch-jberet/jdbc-job-repository=REPOSITORY_NAME:add(data-source=java:jboss/datasources/DATASOURCE)
```

设置默认的任务仓库

您可以设置 in-memory 或 JDBC 任务仓库为批应用程序的默认任务库。

```
/subsystem=batch-jberet:write-attribute(name=default-job-
repository,value=REPOSITORY_NAME)
```

这要求服务器重启。

重新载入

20.1.2. 配置批线程池

本节介绍了如何使用管理 CLI 配置线程池和线程工厂以供批任务使用。您也可用通过管理控制台配置它们。从 **Configuration** 标签页进入 **Batch** 子系统，从左面菜单中选 **Thread Pools** 或 **Thread Factories**。

配置线程池

当添加线程池时，您必须指定 **max-threads**，它的值需要大于 **3**，因为要保留两个线程来确保分区任务可以如期执行。

1. 添加线程池。

```
/subsystem=batch-jberet/thread-pool=THREAD_POOL_NAME:add(max-
threads=10)
```

2. 如果需要，请设置 **keepalive-time** 值。

```
/subsystem=batch-jberet/thread-pool=THREAD_POOL_NAME:write-
attribute(name=keepalive-time,value={time=60,unit=SECONDS})
```

使用线程工厂

1. 添加线程工厂。

```
/subsystem=batch-jberet/thread-factory=THREAD_FACTORY_NAME:add
```

2. 配置线程工厂的属性。

- **group-name** - 为这个线程工厂创建的线程组的名称。
- **priority** - 所创建线程的优先级。
- **thread-name-pattern** - 用于创建线程名称的模板。您可以使用下列形式：
 - **%%** - 百分比符号
 - **%t** - 工厂线程的序列号
 - **%g** - 全局线程序列号
 - **%f** - 工厂序列号
 - **%i** - 线程 ID

3. 分配线程工厂至线程池。

```
/subsystem=batch-jberet/thread-pool=THREAD_POOL_NAME:write-attribute(name=thread-factory,value=THREAD_FACTORY_NAME)
```

这要求服务器重启。

重新载入

设置默认的线程池

您可以指定不同的线程池作为默认线程池。

```
/subsystem=batch-jberet:write-attribute(name=default-thread-pool,value=THREAD_POOL_NAME)
```

这要求服务器重启。

重新载入

查看线程池统计信息

您可用 **read-resource** 管理 CLI 操作查看批线程池的运行时信息。您必须使用 **include-runtime=true** 参数才能查看这些信息。

```
/subsystem=batch-jberet/thread-pool=THREAD_POOL_NAME:read-resource(include-runtime=true)
{
  "outcome" => "success",
  "result" => {
    "active-count" => 0,
    "completed-task-count" => 0L,
    "current-thread-count" => 0,
    "keepalive-time" => undefined,
    "largest-thread-count" => 0,
    "max-threads" => 15,
    "name" => "THREAD_POOL_NAME",
    "queue-size" => 0,
    "rejected-count" => 0,
    "task-count" => 0L,
    "thread-factory" => "THREAD_FACTORY_NAME"
  }
}
```

您也可以通过管理控制台来查看批线程池的运行时信息，如从 **Runtime** 标签页进入 **Batch** 子系统。

20.2. 管理批任务

部署的 **batch-jberet** 子系统资源允许您启动、停止和重启批任务。您也可以查看任务执行的细节。

重启批任务

您可以用 Execution ID 和其他属性重启处于 **STOPPED** 或 **FAILED** 状态的任务。

```
/deployment=DEPLOYMENT_NAME/subsystem=batch-jberet:restart-job(execution-id=EXECUTION_ID,properties={PROPERTY=VALUE})
```

Execution ID 必须是任务实例最近执行的 ID。

启动批任务

您可以通过任务 XML 文件启动批任务，并可以指定在启动批任务时需要使用的属性。

```
/deployment=DEPLOYMENT_NAME/subsystem=batch-jberet:start-job(job-xml-
name=JOB_XML_NAME,properties={PROPERTY=VALUE})
```

停止批任务

您可以通过 execution ID 来停止运行的批任务。

```
/deployment=DEPLOYMENT_NAME/subsystem=batch-jberet:stop-job(execution-
id=EXECUTION_ID)
```

查看批任务的执行细节

您可以查看批任务执行的细节。您必须使用 **include-runtime=true** 参数来查看这些运行时信息。

```
/deployment=DEPLOYMENT_NAME/subsystem=batch-jberet:read-
resource(recursive=true,include-runtime=true)
{
  "outcome" => "success",
  "result" => {"job" => {"import-file" => {
    "instance-count" => 2,
    "running-executions" => 0,
    "execution" => {
      "2" => {
        "batch-status" => "COMPLETED",
        "create-time" => "2016-04-11T22:03:12.708-0400",
        "end-time" => "2016-04-11T22:03:12.718-0400",
        "exit-status" => "COMPLETED",
        "instance-id" => 58L,
        "last-updated-time" => "2016-04-11T22:03:12.719-0400",
        "start-time" => "2016-04-11T22:03:12.708-0400"
      },
      "1" => {
        "batch-status" => "FAILED",
        "create-time" => "2016-04-11T21:57:17.567-0400",
        "end-time" => "2016-04-11T21:57:17.596-0400",
        "exit-status" => "Error :
org.hibernate.exception.ConstraintViolationException: could not execute
statement",
        "instance-id" => 15L,
        "last-updated-time" => "2016-04-11T21:57:17.597-0400",
        "start-time" => "2016-04-11T21:57:17.567-0400"
      }
    }
  }
}
}}}
```

第 21 章 配置 NAMING 子系统

21.1. 关于 NAMING 子系统

naming 子系统为 JBoss EAP 提供了 JNDI 的实现。您可以配置这个子系统来[绑定全局 JNDI 命名空间中的项](#)。您也可以配置它来[激活或取消激活远程 JNDI 接口](#)。

以下是一个包括了所有指定的项和属性的 **naming** 子系统 XML 配置示例。

```
<subsystem xmlns="urn:jboss:domain:naming:2.0">
  <bindings>
    <simple name="java:global/simple-integer-binding" value="100"
type="int" />
    <simple name="java:global/jboss.org/docs/url"
value="https://docs.jboss.org" type="java.net.URL" />
    <object-factory name="java:global/foo/bar/factory"
module="org.foo.bar" class="org.foo.bar.ObjectFactory" />
    <external-context name="java:global/federation/ldap/example"
class="javax.naming.directory.InitialDirContext" cache="true">
      <environment>
        <property name="java.naming.factory.initial"
value="com.sun.jndi.ldap.LdapCtxFactory" />
        <property name="java.naming.provider.url"
value="ldap://ldap.example.com:389" />
        <property name="java.naming.security.authentication"
value="simple" />
        <property name="java.naming.security.principal"
value="uid=admin,ou=system" />
        <property name="java.naming.security.credentials"
value="secret" />
      </environment>
    </external-context>
    <lookup name="java:global/new-alias-name"
lookup="java:global/original-name" />
  </bindings>
  <remote-naming/>
</subsystem>
```

21.2. 配置全局绑定

naming 子系统允许把项绑定到 **java:global**、**java:jboss** 或 **java** 全局 JNDI 命名空间中。但是，推荐使用标准的、可移植的 **java:global** 命名空间。

全局绑定在 **naming** 子系统的 **<bindings>** 项中设置。现在支持 4 类绑定。

- [简单绑定 \(simple binding\)](#)
- [对象工厂绑定 \(object factory binding\)](#)
- [外部上下文绑定 \(external context binding\)](#)
- [绑定查找别名 \(binding lookup alias\)](#)

配置简单绑定

simple XML 配置项用来绑定 primitive 或 `java.net.URL` 项。

- **name** 属性是必需的，它为此条目指定目标 JNDI 名。
- **value** 属性是必需的，它用来定义条目的值。
- 可选的 **type** 属性用来指定这个条目值的类型，它的默认值是 `java.lang.String`。除了 `java.lang.String`，还可用指定 primitive 类型以及它们相应的 object wrapper 类型，如 `int` 或 `java.lang.Integer`，以及 `java.net.URL`。

以下是创建一个简单绑定的管理 CLI 命令的示例。

```
/subsystem=naming/binding=java\:global\simple-integer-binding:add(binding-type=simple, type=int, value=100)
```

XML 配置的结果

```
<subsystem xmlns="urn:jboss:domain:naming:2.0">
  <bindings>
    <simple name="java:global/simple-integer-binding" value="100"
type="int"/>
  </bindings>
  <remote-naming/>
</subsystem>
```

使用以下命令删除绑定。

```
/subsystem=naming/binding=java\:global\simple-integer-binding:remove
```

绑定对象工厂

object-factory XML 配置项绑定 `javax.naming.spi.ObjectFactory` 条目。

- **name** 属性是必需的，它为此条目指定目标 JNDI 名。
- **class** 属性是必需的，它用来定义对象工厂的 Java 类型。
- **module** 属性是必需的，它指定了 JBoss Module ID，对象工厂的 Java 类会从这个 ID 所代表的地方加载。
- **environment** 子项是可选的，它被用来为对象工厂提供一个自定义的环境。

以下是创建一个对象工厂绑定的管理 CLI 命令的示例。

```
/subsystem=naming/binding=java\:global\foo\bar\factory:add(binding-type=object-factory, module=org.foo.bar, class=org.foo.bar.ObjectFactory, environment=[p1=v1, p2=v2])
```

XML 配置的结果

```
<subsystem xmlns="urn:jboss:domain:naming:2.0">
  <bindings>
    <object-factory name="java:global/foo/bar/factory"
module="org.foo.bar" class="org.foo.bar.ObjectFactory">
      <environment>
```

```

        <property name="p1" value="v1" />
        <property name="p2" value="v2" />
    </environment>
</object-factory>
</bindings>
</subsystem>

```

使用以下命令删除绑定。

```
/subsystem=naming/binding=java\:global\foo\bar\factory:remove
```

绑定外部上下文

使用 **external-context** XML 配置项可以联合外部 JNDI 上下文，如 LDAP 上下文。

- **name** 属性是必需的，它为此条目指定目标 JNDI 名。
- **class** 属性是必需的，它指定了用来创建联合上下文（federated context）的 Java 初始命名上下文类型。请注意，这个类型需要有一个带有单一环境映射参数的 constructor。
- **module** 参数是可选的，它指定了 JBoss Module ID，外部 JNDI 上下文所需的类可以从这里加载。
- **cache** 参数是可选的，默认值是 **false**。它指定了外部上下文实例是否可以被缓存。
- **environment** 子项是可选的，它被用来提供查看外部上下文所需的自定义环境。

以下是一个管理 CLI 命令的示例，它创建了一个外部上下文绑定。

```

/subsystem=naming/binding=java\:global\ federation\ldap\example:add(binding-type=external-context, cache=true,
class=javax.naming.directory.InitialDirContext, environment=
[java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory,
java.naming.provider.url=ldap://ldap.example.com:389,
java.naming.security.authentication=simple,
java.naming.security.principal=uid=admin,ou=system,
java.naming.security.credentials= secret])

```

XML 配置的结果

```

<subsystem xmlns="urn:jboss:domain:naming:2.0">
  <bindings>
    <external-context name="java:global/federation/ldap/example"
class="javax.naming.directory.InitialDirContext" cache="true">
      <environment>
        <property name="java.naming.factory.initial"
value="com.sun.jndi.ldap.LdapCtxFactory" />
        <property name="java.naming.provider.url"
value="ldap://ldap.example.com:389" />
        <property name="java.naming.security.authentication"
value="simple" />
        <property name="java.naming.security.principal"
value="uid=admin,ou=system" />
        <property name="java.naming.security.credentials"
value="secret" />
      </environment>
    </external-context>
  </bindings>
</subsystem>

```



```

    </external-context>
  </bindings>
</subsystem>

```

使用以下命令删除绑定。

```
/subsystem=naming/binding=java\:global\:/federation\:/ldap\:/example:remove
```

注意

当查找的资源是一个没有正确实现 **lookup(Name)** 方法的 JNDI 厂商时，会出现一个 "javax.naming.InvalidNameException: Only support CompoundName names" 错误。

这个问题的一个临时解决方法是，指定使用 **lookup(String)** 方法的外部上下文环境，而不添加以下属性。但是，这会导致性能下降。

```

<property name="org.jboss.as.naming.lookup.by.string"
value="true"/>

```

绑定查找别名

lookup 项允许为存在的项绑定额外的名称或别名。

- **name** 属性是必需的，它为此条目指定目标 JNDI 名。
- **lookup** 属性是必需的，它指定了源 JNDI 的名。

以下是把存在的项与一个别名进行绑定的管理 CLI 命令的示例。

```
/subsystem=naming/binding=java\:global\:/new-alias-name:add(binding-
type=lookup, lookup=java\:global\:/original-name)
```

XML 配置的结果

```

<lookup name="java\:global\:/new-alias-name" lookup="java\:global\:/original-
name" />

```

使用以下命令删除绑定。

```
/subsystem=naming/binding=java\:global\:/c:remove
```

21.3. 配置远程 JNDI 接口

远程 JNDI 接口允许客户端查找远程 JBoss EAP 实例中的项。**naming** 子系统可以被配置为启用或禁用这个接口（默认是启用）。远程 JNDI 接口使用 **<remote-naming>** 项进行配置。

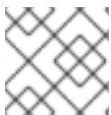
使用以下管理 CLI 命令激活或重新激活远程 JNDI 接口。

```
/subsystem=naming/service=remote-naming:add
```

使用以下管理 CLI 命令取消激活远程 JNDI 接口。

■

```
/subsystem=naming/service=remote-naming:remove
```



注意

只有 **java:jboss/exported** 上下文中的项可以通过远程 JNDI 访问。

第 22 章 配置高可用性

22.1. 高可用性介绍

JBoss EAP 提供下列 *高可用性* 服务来保证部署的 Java EE 应用程序的可用性。

负载均衡

这允许服务将负载分散到多个服务器从而处理大量请求。即使在处理大量请求时，客户都可以及时从服务得到响应。

故障切换

故障切换功能确保了，在硬件或网络出现问题时，客户端对服务的访问也不会被中断。如果服务失败，另外一个集群成员会接替来处理客户端的请求，从而使服务可以继续进行。

集群 (Clustering) 是实现这些功能的一个技术。一个集群中的成员可以被配置为用来共享负载（负载均衡），或在其它集群成员出现故障时接替它们来处理用户的请求（故障切换）。



注意

需要注意的一点是，所选的 JBoss EAP 操作模式（*独立服务器 (standalone server)* 或 *受管域 (managed domain)*）代表了您希望如何管理您的服务器。而无论选择什么操作模式，都可以在 JBoss EAP 中配置高可用性服务。

通过不同的组件，JBoss EAP 可以在不同级别上支持高可用性。这些运行时组件以及应用程序包括：

- 应用服务器实例
- 和内部的 JBoss Web Server、Apache HTTP Server、Microsoft IIS 或 Oracle iPlanet Web Server 一起使用的 Web 应用程序
- 有状态 (Stateful) 和无状态 (stateless) 的 Session Enterprise JavaBeans (EJB)
- 单点登录 (SSO) 机制
- HTTP 会话
- JMS 服务和 message-driven beans (MDB)
- Singleton MSC 服务
- Singleton 部署

通过 **jgroups**、**infinispan** 和 **modcluster** 子系统，可以在 JBoss EAP 中使用高可用性。*ha* 和 *full-ha* 配置集可以启用这些子系统。在 JBoss EAP 中，这些服务会根据需要被启动或关闭，但它们只会当部署在服务器上的应用应用程序被配置为 *distributable* 时才可以启用。

关于如何 [创建分布式应用程序](#)，请参考《JBoss EAP 开发指南》。

22.2. 与 JGROUPS 的集群通讯

22.2.1. 关于 JGroups

JGroups 是创建可靠消息通讯的工具包，它可用来创建集群，其中的节点可以彼此发送消息。

jgroups 子系统为 JBoss EAP 中的高可用性服务提供了组交流的支持。当一个配置提供了高可用性功能时（如在受管域中的 *ha* 或 *full-ha* 配置集；或一个独立服务器的 **standalone-ha.xml** 或 **standalone-full-ha.xml** 配置文件），**jgroups** 子系统才可用。

JBoss EAP 预配置了两个 JGroups 栈：

udp

集群里的节点使用 Datagram Protocol (UDP) 多播进行彼此间的通讯。这是默认设置。

tcp

集群中的节点使用 TCP 协议和其它节点进行通讯。

您可以使用预配置的设置，或根据您的系统的具体情况定义自己的设置。



注意

因为 TCP 需要进行错误检查、数据包排序、阻塞控制等操作，所以会消耗更多额外的资源，并通常被认为比 UDP 的性能要低。JGroups 会为 UDP 处理这些操作，而 TCP 可以保证自己实现这些功能。当在一个不稳定或具有大量数据阻塞的网络中使用 JGroups，或多播不可用时，使用 TCP 是一个好的选择。

22.2.2. 切换默认的 JGroups 频道以使用 TCP

在默认情况下，集群节点使用 UDP 协议来通讯。默认的 **ee** JGroups 频道使用预定义的 **udp** 协议栈。

```
<channels default="ee">
  <channel name="ee" stack="udp"/>
</channels>
<stacks>
  <stack name="udp">
    <transport type="UDP" socket-binding="jgroups-udp"/>
    <protocol type="PING"/>
    ...
  </stack>
  <stack name="tcp">
    <transport type="TCP" socket-binding="jgroups-tcp"/>
    <protocol type="MPING" socket-binding="jgroups-mping"/>
    ...
  </stack>
</stacks>
```

某些网络只允许使用 TCP。使用下列管理 CLI 命令来切换 **ee** 频道以使用预配置的 **tcp** 栈。

```
/subsystem=jgroups/channel=ee:write-attribute(name=stack,value=tcp)
```

这个默认的 **tcp** 栈使用 **MPING** 协议，它使用 IP 多播来发现初始的集群成员资格。关于配置用于其他成员资格发现协议的栈的内容，请参考下列章节：

- 使用 **TCPPING** 协议来定义静态的集群成员资格列表。
- 使用 **TCPGOSSIP** 协议以使用外部的 Gossip 路由器来发现集群的成员。

22.2.3. 配置 TCPPING

以下过程会创建一个新的 JGroups 栈，它使用 **TCPPING** 协议来定义一个静态的集群成员列表。一个基本的脚本会被提供，它会创建一个 **tcpping** 栈，并设置默认的 **ee** 频道来使用这个新栈。这个脚本中的管理 CLI 命令需要根据您的具体环境进行定制，并会以批处理的形式运行。

1. 复制下列脚本至文本编辑器并保存至本地的文件系统。

```
batch
# Add the tcpping stack
/subsystem=jgroups/stack=tcpping:add
/subsystem=jgroups/stack=tcpping/transport=TCP:add(socket-
binding=jgroups-tcp)
/subsystem=jgroups/stack=tcpping/protocol=TCPPING:add
# Set the properties for the TCPPING protocol
/subsystem=jgroups/stack=tcpping/protocol=TCPPING:write-
attribute(name=properties,value=
{initial_hosts="HOST_A[7600],HOST_B[7600]",port_range=0,timeout=3000
})
/subsystem=jgroups/stack=tcpping/protocol=MERGE3:add
/subsystem=jgroups/stack=tcpping/protocol=FD_SOCK:add(socket-
binding=jgroups-tcp-fd)
/subsystem=jgroups/stack=tcpping/protocol=FD:add
/subsystem=jgroups/stack=tcpping/protocol=VERIFY_SUSPECT:add
/subsystem=jgroups/stack=tcpping/protocol=pbcast.NAKACK2:add
/subsystem=jgroups/stack=tcpping/protocol=UNICAST3:add
/subsystem=jgroups/stack=tcpping/protocol=pbcast.STABLE:add
/subsystem=jgroups/stack=tcpping/protocol=pbcast.GMS:add
/subsystem=jgroups/stack=tcpping/protocol=MFC:add
/subsystem=jgroups/stack=tcpping/protocol=FRAG2:add
# Set tcpping as the stack for the ee channel
/subsystem=jgroups/channel=ee:write-
attribute(name=stack,value=tcpping)
run-batch
reload
```

请注意，定义协议的顺序非常重要。

2. 根据您的环境修改这个脚本。

- 如果运行在受管域里，您必须在 **/subsystem=jgroups** 命令中使用 **/profile=PROFILE_NAME** 指定要更新的配置集。
- 根据您的环境调整可选的 TCPPING 属性：
 - **initial_hosts**：一个以逗号分隔的主机列表。这些主机被认为是已知的，并可以用来查找初始的成员。
 - **port_range**：如果需要，可以分配一个端口范围。如果把端口范围设置为 **2**，初始的端口是 **7600**，则 TCPPING 会试图通过端口 **7600-7601** 来联系每个主机。
 - **timeout**：集群成员的超时时间，单位为毫秒。

3. 将脚本文件传入管理 CLI 来运行脚本。

```
$ EAP_HOME/bin/jboss-cli.sh --connect --file=/path/to/SCRIPT_NAME
```

TCPPING 栈现在已可用而 TCP 被用于网络通讯。

22.2.4. 配置 TCPGOSSIP

以下过程会创建一个新的 JGroups 栈，它通过 **TCPGOSSIP** 协议使用一个外部的 gossip 路由器来发现一个集群的成员列表。一个基本的脚本会被提供，它会创建一个 **tcpgossip** 栈，并设置默认的 **ee** 频道来使用这个新栈。这个脚本中的管理 CLI 命令需要根据您的具体环境进行定制，并会以批处理的形式运行。

1. 复制下列脚本至文本编辑器并保存至本地的文件系统。

```
batch
# Add the tcpgossip stack
/subsystem=jgroups/stack=tcpgossip:add
/subsystem=jgroups/stack=tcpgossip/transport=TCP:add(socket-
binding=jgroups-tcp)
/subsystem=jgroups/stack=tcpgossip/protocol=TCPGOSSIP:add
# Set the properties for the TCPGOSSIP protocol
/subsystem=jgroups/stack=tcpgossip/protocol=TCPGOSSIP:write-
attribute(name=properties,value={initial_hosts="HOST_A[13001]"})
/subsystem=jgroups/stack=tcpgossip/protocol=MERGE3:add
/subsystem=jgroups/stack=tcpgossip/protocol=FD_SOCK:add(socket-
binding=jgroups-tcp-fd)
/subsystem=jgroups/stack=tcpgossip/protocol=FD:add
/subsystem=jgroups/stack=tcpgossip/protocol=VERIFY_SUSPECT:add
/subsystem=jgroups/stack=tcpgossip/protocol=pbcst.NAKACK2:add
/subsystem=jgroups/stack=tcpgossip/protocol=UNICAST3:add
/subsystem=jgroups/stack=tcpgossip/protocol=pbcst.STABLE:add
/subsystem=jgroups/stack=tcpgossip/protocol=pbcst.GMS:add
/subsystem=jgroups/stack=tcpgossip/protocol=MFC:add
/subsystem=jgroups/stack=tcpgossip/protocol=FRAG2:add
# Set tcpgossip as the stack for the ee channel
/subsystem=jgroups/channel=ee:write-
attribute(name=stack,value=tcpgossip)
run-batch
reload
```

请注意，定义协议的顺序非常重要。

2. 根据您的环境修改这个脚本。

- 如果运行在受管域里，您必须在 **/subsystem=jgroups** 命令中使用 **/profile=PROFILE_NAME** 指定要更新的配置集。
- 根据您的环境调整可选的 TCPGOSSIP 属性：
 - **initial_hosts**：一个以逗号分隔的主机列表。这些主机被认为是已知的，并可以用来查找初始的成员。
 - **reconnect_interval**：已断开的 stub 试图重连 Gossip 路由器的间隔（毫秒）。
 - **sock_conn_timeout**：套接字创建的最长时间。默认值是 **1000** 毫秒。
 - **sock_read_timeout**：阻塞块读取的最长时间（毫秒）。**0** 表示无限期阻塞。

3. 将脚本文件传入管理 CLI 来运行脚本。

```
$ EAP_HOME/bin/jboss-cli.sh --connect --file=/path/to/SCRIPT_NAME
```

TCPGOSSIP 栈现已可用而 TCP 被用于网络通讯。这个栈被配置和 Gossip 路由器一起使用，所以 JGroups 集群成员可以找到其他的集群成员。

22.2.5. 绑定 JGroups 到网络接口

在默认情况下，JGroups 只绑定 **private** 网络接口，它执行默认设置的 localhost。因为集群的网络数据不应该暴露到公共网络接口，所以为了安全的原因，JGroups 不会绑定在 JBoss EAP 启动时使用 **-b** 参数定义的网络接口。

关于如何配置网络接口的信息，请参考[网络和端口配置](#)章节。



重要

出于安全原因，JGroups 只应该绑定到非公共网络接口。而出于性能考量，我们还推荐 JGroups 的网络接口应该是专属的虚拟局域网（Virtual Local Area Network, VLAN）的一部分。

22.2.6. 保护集群

要安全运行集群，您需要解决几个问题：

- 防止未授权的节点加入集群。这是通过[验证](#)来解决的。
- 防止非成员与集群成员通讯。这是通过[消息加密](#)实现的。

配置验证

JGroups 验证是由 **AUTH** 协议执行的。其目的是确保只有经过验证的节点才能加入集群。

在相关的服务器配置文件里，添加 **AUTH** 协议和合适的属性设置。**AUTH** 协议应该恰好在 **pbcast.GMS** 协议之前配置。

```
<subsystem xmlns="urn:jboss:domain:jgroups:4.0">
  <stacks>
    <stack name="udp">
      <transport type="UDP" socket-binding="jgroups-udp"/>
      <protocol type="PING"/>
      <protocol type="MERGE3"/>
      <protocol type="FD_SOCK" socket-binding="jgroups-udp-fd"/>
      <protocol type="FD_ALL"/>
      <protocol type="VERIFY_SUSPECT"/>
      <protocol type="pbcast.NAKACK2"/>
      <protocol type="UNICAST3"/>
      <protocol type="pbcast.STABLE"/>
      <protocol type="AUTH">
        <property name="auth_class">org.jgroups.auth.MD5Token</property>
        <property name="auth_value">mytoken</property> <!-- Change this
password -->
        <property name="token_hash">MD5</property>
      </protocol>
      <protocol type="pbcast.GMS"/>
      <protocol type="UFC"/>
      <protocol type="MFC"/>
      <protocol type="FRAG2"/>
    </stack>
  </stacks>
</subsystem>
```

```

</stack>
</stacks>
</subsystem>

```

配置加密

为了加密信息，JGroups 使用一个由集群成员共享的密钥。信息的发送者使用这个密钥对信息进行加密，接收者使用相同的密钥进行解密。对于[对称加密](#)（配置为使用 **SYM_ENCRYPT** 协议），节点使用共享的 keystore 来获取这个密钥。对于[非对称加密](#)（配置为使用 **ASYM_ENCRYPT** 协议），节点在使用 **AUTH** 被验证后，从集群的 coordinator 中获得这个密钥。



重要

You must apply [Red Hat JBoss Enterprise Application Platform 7.0 Update 01](#) or a newer cumulative patch to your JBoss EAP installation in order to have access to the **SYM_ENCRYPT** and **ASYM_ENCRYPT** protocols.

See the JBoss EAP [Patching and Upgrading Guide](#) for information on applying cumulative patches.

使用对称加密

要使用 **SYM_ENCRYPT**，您必须设立每个节点在 JGroups 配置里引用的密钥库（keystore）。

1. 创建密钥库

在下面的命令里，请用合适的 JGroups JAR 版本替换 **VERSION**，并用密钥库密码替换 **PASSWORD**。

```

$ java -cp
EAP_HOME/modules/system/layers/base/org/jgroups/main/jgroups-
VERSION.jar org.jgroups.demos.KeyStoreGenerator --alg AES --size 128
--storeName defaultStore.keystore --storepass PASSWORD --alias mykey

```

这将生成一个在 JGroups 配置里引用的 **defaultStore.keystore** 文件。

2. 在 jgroups 子系统中配置 **SYM_ENCRYPT** 协议。

在相关的服务器配置文件里，添加 **SYM_ENCRYPT** 协议和合适的属性设置。 **SYM_ENCRYPT** 协议应该恰好在 **pbcast.NAKACK2** 协议之前配置。

```

<subsystem xmlns="urn:jboss:domain:jgroups:4.0">
  <stacks>
    <stack name="udp">
      <transport type="UDP" socket-binding="jgroups-udp"/>
      <protocol type="PING"/>
      <protocol type="MERGE3"/>
      <protocol type="FD_SOCK" socket-binding="jgroups-udp-fd"/>
      <protocol type="FD_ALL"/>
      <protocol type="VERIFY_SUSPECT"/>
      <protocol type="SYM_ENCRYPT">
        <property name="provider">SunJCE</property>
        <property name="sym_algorithm">AES</property>
        <property name="encrypt_entire_message">true</property>
      </protocol>
      <property name="keystore_name">/path/to/defaultStore.keystore</property>
      <property name="store_password">PASSWORD</property>
      <property name="alias">mykey</property>
    </stack>
  </stacks>
</subsystem>

```



```

    </protocol>
    <protocol type="pbcaster.NAKACK2"/>
    <protocol type="UNICAST3"/>
    <protocol type="pbcaster.STABLE"/>
    <protocol type="pbcaster.GMS"/>
    <protocol type="UFC"/>
    <protocol type="MFC"/>
    <protocol type="FRAG2"/>
  </stack>
</stacks>
</subsystem>

```



注意

在使用 **SYM_ENCRYPT** 时，配置 **AUTH** 是可选的。

使用不对称加密

1. 在 **jgroups** 子系统中配置 **ASYM_ENCRYPT** 协议。

在相关的服务器配置文件里，添加 **ASYM_ENCRYPT** 协议和合适的属性设置。**ASYM_ENCRYPT** 协议应该恰好在 **pbcaster.NAKACK2** 协议之前配置。

```

<subsystem xmlns="urn:jboss:domain:jgroups:4.0">
  <stacks>
    <stack name="udp">
      <transport type="UDP" socket-binding="jgroups-udp"/>
      <protocol type="PING"/>
      <protocol type="MERGE3"/>
      <protocol type="FD_SOCK" socket-binding="jgroups-udp-fd"/>
      <protocol type="FD_ALL"/>
      <protocol type="VERIFY_SUSPECT"/>
      <protocol type="ASYM_ENCRYPT">
        <property name="encrypt_entire_message">true</property>
        <property name="sym_keylength">128</property>
        <property
name="sym_algorithm">AES/ECB/PKCS5Padding</property>
        <property name="asym_keylength">512</property>
        <property name="asym_algorithm">RSA</property>
      </protocol>
      <protocol type="pbcaster.NAKACK2"/>
      <protocol type="UNICAST3"/>
      <protocol type="pbcaster.STABLE"/>
      <!-- Configure AUTH protocol here -->
      <protocol type="pbcaster.GMS"/>
      <protocol type="UFC"/>
      <protocol type="MFC"/>
      <protocol type="FRAG2"/>
    </stack>
  </stacks>
</subsystem>

```

2. 在 **jgroups** 子系统中配置 **AUTH** 协议。

AUTH 是 **ASYM_ENCRYPT** 要求的。相关说明请参考[配置验证](#)章节。

22.2.7. 配置 JGroups 线程池

jgroups 子系统包含 **default**、**internal**、**oob** 和 **timer** 线程池。这些池可以对每个 JGroups 栈进行配置。

下表列出了您可以为每个线程池配置的属性和默认值。

线程池的名称	keepalive-time	max-threads	min-threads	queue-length
default	60000L	300	20	100
internal	60000L	4	2	100
oob	60000L	300	20	0
timer	5000L	4	2	500

使用下列管理 CLI 命令来配置 JGroups 线程池。

```
/subsystem=jgroups/stack=STACK_TYPE/transport=TRANSPORT_TYPE/thread-
pool=THREAD_POOL_NAME:write-attribute(name=ATTRIBUTE_NAME,
value=ATTRIBUTE_VALUE)
```

以下管理 CLI 命令示例会把 **udp** 栈的 **default** 线程池里的 **max-threads** 的值设置为 **500**。

```
/subsystem=jgroups/stack=udp/transport=UDP/thread-pool=default:write-
attribute(name="max-threads", value="500")
```

22.2.8. 配置 JGroups Send 和 Receive 缓冲

解决缓冲区大小警告

在默认情况下，JGroups 被配置为使用特定的接收和发送缓冲区大小。但是，操作系统可能会对缓冲区大小有限制，因此 JBoss EAP 可能无法使用它配置的缓冲区大小的设置。在这种情况下，会在 JBoss EAP 日志中看到和以下类似的警告信息：

```
WARNING [org.jgroups.protocols.UDP] (ServerService Thread Pool -- 68)
JGRP000015: the send buffer of socket DatagramSocket was set to 640KB, but
the OS only allocated 212.99KB.
This might lead to performance problems. Please set your max send buffer
in the OS correctly (e.g. net.core.wmem_max on Linux)
WARNING [org.jgroups.protocols.UDP] (ServerService Thread Pool -- 68)
JGRP000015: the receive buffer of socket DatagramSocket was set to 20MB,
but the OS only allocated 212.99KB.
This might lead to performance problems. Please set your max receive
buffer in the OS correctly (e.g. net.core.rmem_max on Linux)
```

为了解决这个问题，查阅操作系统的文档来获得如何增加缓冲区大小的方法。对于使用 Red Hat Enterprise Linux 的系统，以 root 用户身份编辑 **/etc/sysctl.conf** 来配置缓冲区大小的最大值。这个设置在重启后仍然有效。例如：

```
# Allow a 25MB UDP receive buffer for JGroups
```

```
net.core.rmem_max = 26214400
# Allow a 1MB UDP send buffer for JGroups
net.core.wmem_max = 1048576
```

在修改了 `/etc/sysctl.conf` 之后，运行 `sysctl -p` 以使修改生效。

配置 **JGroups** 缓冲的大小

您可以通过设置 UDP 和 TCP JGroups 栈上的传输属性来配置 JBoss EAP 使用的 JGroups 缓冲的大小。

UDP 栈

- `ucast_recv_buf_size`
- `ucast_send_buf_size`
- `mcast_recv_buf_size`
- `mcast_send_buf_size`

TCP 栈

- `recv_buf_size`
- `send_buf_size`

JGroups 缓冲大小可以用管理控制台或管理 CLI 来配置。

使用下列管理 CLI 命令来设置 JGroups 缓冲大小属性。

```
/subsystem=jgroups/stack=STACK_NAME/transport=TRANSPORT/property=PROPERTY_
NAME:add(value=BUFFER_SIZE)
```

以下是一个管理 CLI 命令示例，它把 **tcp** 栈中的 `recv_buf_size` 属性值设置为 **20000000**。

```
/subsystem=jgroups/stack=tcp/transport=TRANSPORT/property=recv_buf_size:ad
d(value=20000000)
```

JGroups 缓冲区大小也可以通过管理控制台进行配置。从 **Configuration** 标签页进入 **JGroups** 子系统，查看相关的栈，选择 **Transport**，然后选择 **Properties** 标签页。

22.2.9. JGroups 的故障排除

22.2.9.1. 节点没有组成一个集群

确认您的机器已正确配置了 IP 多播。JBoss EAP 带有两个可以用来测试多播的程序：**McastReceiverTest** 和 **McastSenderTest**。

在一个终端中启动 **McastReceiverTest**。

```
$ java -cp EAP_HOME/bin/client/jboss-client.jar
org.jgroups.tests.McastReceiverTest -mcast_addr 230.11.11.11 -port 5555
```

然后，在另外一个终端窗口中启动 **McastSenderTest**。

```
$ java -cp EAP_HOME/bin/client/jboss-client.jar
org.jgroups.tests.McastSenderTest -mcast_addr 230.11.11.11 -port 5555
```

使用 **-bind_addr YOUR_BIND_ADDRESS** 可以绑定一个特定的网络接口卡（NIC），其中的 **YOUR_BIND_ADDRESS** 是要绑定的 NIC 的 IP 地址。在接收方和发送方都要使用这个参数。

当在 **McastSenderTest** 终端窗口中输入时，应该可以在 **McastReceiverTest** 窗口中看到输出。如果没有看到，尝试以下步骤。

- 在 sender 命令中使用 **-ttl VALUE** 来增加多播的 time-to-live 值。这个测试程序使用的默认值是 **32**，**VALUE** 的值不能超过 **255**。
- 如果机器有多个接口，您需要检查是否使用了正确的接口。
- 联系一个系统管理员来确认，多播可以在所选接口中工作。

在您确定多播可以在集群的所有机器上都可以正常工作后，就可以重复上面的测试来对网络进行测试，把 sender 放到一个机器上，把 receiver 放到另外一个机器上。

22.2.9.2. 在故障监测中导致丢失“心跳”的原因

有时，当在一定时间内（**T**，由 **timeout** 和 **max_tries** 指定）故障监测（failure detection - FD）没有获得“心跳”确认时，会认为集群中的一个成员可能出现了问题。

例如，在一个包括节点 A、B、C 和 D 的集群中，A ping B、B ping C、C ping D、D ping A。当出现以下情况时 C 会被认为可能出现了问题：

- B 或 C 的 CPU 使用率为 100% 的时间超过了一定长度（**T** 秒）。当 C 向 B 发送了一个心跳确认时，B 可能因为处于 100% CPU 使用率而无法处理 C 发送的心跳确认。
- B 或 C 正在进行垃圾收集操作，这会和以上情况相同。
- 以上两种情况的组合。
- 网络丢失了数据包。这通常发生在网络中有大量数据时，交换机会开始丢掉数据包，一般是先广播，然后是 IP 多播，最后是 TCP 数据包。
- B 或 C 正在处理回调。例如，C 通过它的频道接收到一个远程方法调用，并用了 **T + 1** 秒的时间进行处理。在这个期间，C 将不会处理包括心跳在内的其它信息。因此，B 将不会收到 C 的心跳确认信息，并认为 C 可能出现了问题。

22.3. INFINISPAN

22.3.1. 关于 Infinispan

Infinispan 是一个 Java 数据网格平台，它提供了一个与 [JSR-107](#) 兼容的缓存接口用来管理缓存的数据。如需了解更多与 Infinispan 相关的信息，请参阅 [Infinispan Documentation](#)。

infinispan 子系统为 JBoss EAP 提供了缓存支持。它允许您配置并查看命名的缓存容器和缓存区的运行时的指标。

在使用一个可以提供高可用性功能的配置中（如受管域中的 *ha* 或 *full-ha* 配置集，或独立服务器的 **standalone-ha.xml** 或 **standalone-full-ha.xml** 配置文件），**infinispan** 子系统提供了缓存、状态复制和状态分布的支持。在一个非高可用性配置中，**infinispan** 子系统提供了本地的缓存支

持。



重要

在 JBoss EAP 中, Infinispan 作为一个私人模块提供, 它为 JBoss EAP 提供了缓存功能。Infinispan 不能被应用程序直接使用。

22.3.2. 缓存容器

缓存容器是子系统使用的缓存的仓库。每个缓存容器都会定义一个要使用的默认缓存。

JBoss EAP 7 定义了下列默认的 Infinispan 缓存容器：

- 用于单点缓存的 **server**
- 用于 Web 会话集群的 **web**
- 用于 stateful session bean 集群的 **ejb**
- 用于实体缓存的 **hibernate**

示例：默认的 Infinispan 配置

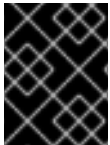
```
<subsystem xmlns="urn:jboss:domain:infinispan:4.0">
  <cache-container name="server" aliases="singleton cluster" default-
cache="default" module="org.wildfly.clustering.server">
    <transport lock-timeout="60000"/>
    <replicated-cache name="default" mode="SYNC">
      <transaction mode="BATCH"/>
    </replicated-cache>
  </cache-container>
  <cache-container name="web" default-cache="dist"
module="org.wildfly.clustering.web.infinispan">
    <transport lock-timeout="60000"/>
    <distributed-cache name="dist" mode="ASYNC" l1-lifespan="0"
owners="2">
      <locking isolation="REPEATABLE_READ"/>
      <transaction mode="BATCH"/>
      <file-store/>
    </distributed-cache>
  </cache-container>
  <cache-container name="ejb" aliases="sfsb" default-cache="dist"
module="org.wildfly.clustering.ejb.infinispan">
    <transport lock-timeout="60000"/>
    <distributed-cache name="dist" mode="ASYNC" l1-lifespan="0"
owners="2">
      <locking isolation="REPEATABLE_READ"/>
      <transaction mode="BATCH"/>
      <file-store/>
    </distributed-cache>
  </cache-container>
  <cache-container name="hibernate" default-cache="local-query"
module="org.hibernate.infinispan">
    <transport lock-timeout="60000"/>
    <local-cache name="local-query">
```

```

        <eviction strategy="LRU" max-entries="10000"/>
        <expiration max-idle="100000"/>
    </local-cache>
    <invalidation-cache name="entity" mode="SYNC">
        <transaction mode="NON_XA"/>
        <eviction strategy="LRU" max-entries="10000"/>
        <expiration max-idle="100000"/>
    </invalidation-cache>
    <replicated-cache name="timestamps" mode="ASync"/>
</cache-container>
</subsystem>

```

记录下每个容器定义的默认缓存。例如，**web** 缓存容器定义了 **dist** 分布的缓存作为默认缓存。因此，当对 **web** 会话进行集群时，**dist** 缓存会被使用。



重要

另外，您还可以添加额外的缓存和缓存容器。如为 HTTP 会话、有状态的会话 bean、singleton 服务或部署增加缓存。用户应用程序不能直接使用这些缓存。

22.3.2.1. 配置缓存容器

缓存容器和缓存属性可以用管理控制台或管理 CLI 来配置。



警告

因为配置中的其它组件可能在使用它们，所以您不应该修改缓存或缓存容器的名称。

用管理控制台配置缓存

从管理控制台的 **Configuration** 标签页进入 **Infinispan** 子系统，您就可以配置缓存和缓存容器。在一个受管域中，确定选择适当的配置集进行配置。

- 添加缓存容器。
点 **Cache Container** 旁边的 **Add** 按钮，输入新缓存容器的设置。
- 更新缓存容器的设置。
选择适当的缓存容器，从下拉菜单中选 **Container Settings**。根据需要配置缓存容器。
- 更新缓存容器的传输设置。
选择适当的缓存容器，从下拉菜单中选 **Transport Settings**。根据需要配置缓存容器的传输设置。
- 配置缓存。
选择适当的缓存容器，选 **View**。在相关的缓存标签页中（如 **Replicated Caches**）可以添加、更新和删除缓存。

用管理 CLI 配置缓存

您可以使用管理 CLI 配置缓存和缓存容器。在一个受管域中，您可以使用带有 **/profile=PROFILE_NAME** 的命令指定要更新的配置集。

- 添加缓存容器。

```
/subsystem=infinispan/cache-container=CACHE_CONTAINER:add
```

- 添加复制缓存。

```
/subsystem=infinispan/cache-container=CACHE_CONTAINER/replicated-  
cache=CACHE:add(mode=MODE)
```

- 设置缓存容器的默认缓存。

```
/subsystem=infinispan/cache-container=CACHE_CONTAINER:write-  
attribute(name=default-cache,value=CACHE)
```

- 配置复制缓存的批处理。

```
/subsystem=infinispan/cache-container=CACHE_CONTAINER/replicated-  
cache=CACHE/component=transaction:write-  
attribute(name=mode,value=BATCH)
```

22.3.3. 集群模式

在使用 Infinispan 的 JBoss EAP 中有两种方式配置集群，而您的应用程序需要使用哪种方式取决于您的具体需要。每种模式都会在可用性、可靠性和可扩展性间进行相应的协调。在选择一个集群模式前，您需要决定在网络中哪些功能是最重要的，并对这些要求进行平衡。

缓存模式

复制模式 (Replicated)

复制模式会自动检测并在集群中添加新实例。对这些实例所做的改变会被复制到集群中的所有节点上。复制模式通常适合于一个小的集群，因为不需要在网络间复制太多信息。Infinispan 可以被配置为使用 UDP 多播，它在一定程度上可以减缓网络堵塞问题。

分布模式 (Distributed)

分布模式允许 Infinispan 对集群进行线性扩展。分布模式使用一个一致哈希算法来决定一个新节点应该放置在集群中的什么地方。需要保存的信息副本（所有者）的数量是可以配置的。需要保存的副本数量、数据保持有效的时间，以及性能间会有相互折衷。保持的副本越多，对性能的影响就越大，但可以减少在服务器出现故障时丢失数据的可能性。这个哈希算法还可以在不进行多播或对元数据进行排序的情况下定位相关的数据，从而达到减少网络流量的效果。

当集群的节点数量超过 6 到 8 个时，可以考虑使用分布模式。分布模式中的数据只会被分布到集群中的一部分节点中，而不是分布到所有节点上。

同步和异步复制

复制可用同步或异步模式，选择的模式取决于您的要求和应用程序。

同步复制

当使用同步复制时，用来处理用户请求的线程会被禁止执行，直到复制已成功完成。当复制成功后，会向客户端发送一个反馈信息，客户端只有在接收到这个响应后才会释放线程。因此同步复制需要从集群中的每个节点上获得响应信息，所以它对网络流量会有影响。但是，它也有自己的优点，它可以保证所有的改变都应用到集群中的所有节点上。

异步复制

复制时，Infinispan 使用一个线程池在后台处理复制操作。发送方不需要等待集群中其它节点的响应。

但是，在前一个复制完成前，同一个会话的读缓存操作会被禁用，这可以保证正在处理中的数据不被读取。复制操作会基于时间被触发，或基于队列的长度触发。如果复制失败，相关信息会写入日志，而不会实时进行通知。

22.3.3.1. 配置缓存模式

您可以用管理 CLI 修改默认的缓存。



注意

本节介绍了配置 web 会话缓存的信息，这个缓存在默认情况下被配置为分配模式。这里所使用的步骤和管理 CLI 命令可以被修改来应用到其它缓存容器中。

修改复制缓存模式

默认 JBoss EAP 7 的 web 会话缓存配置不包括一个 **repl** 复制的缓存。这个缓存需要被首先添加。



注意

以下是一个针对独立服务器的管理 CLI 命令。如果运行在一个受管域中，在 `/subsystem=jgroups` 命令中使用 `/profile=PROFILE_NAME` 指定要更新的配置集。

1. 添加 **repl** 复制缓存。

```
/subsystem=infinispan/cache-container=web/replicated-
cache=repl:add(mode=ASYNC)
/subsystem=infinispan/cache-container=web/replicated-
cache=repl/component=transaction:write-
attribute(name=mode,value=BATCH)
/subsystem=infinispan/cache-container=web/replicated-
cache=repl/component=locking:write-attribute(name=isolation,
value=REPEATABLE_READ)
/subsystem=infinispan/cache-container=web/replicated-
cache=repl/store=file:add
```

2. 修改默认缓存为 **repl** 复制缓存。

```
/subsystem=infinispan/cache-container=web:write-
attribute(name=default-cache,value=repl)
```

3. 重新加载服务器。

重新载入

修改为分布式缓存模式

JBoss EAP 7 对 web 会话缓存的默认配置已经包括了一个 **dist** 分布的缓存。



注意

以下是一个针对独立服务器的管理 CLI 命令。如果运行在一个受管域中，在 `/subsystem=jgroups` 命令中使用 `/profile=PROFILE_NAME` 指定要更新的配置集。

1. 修改默认缓存为 **dist** 分布式缓存。


```
/subsystem=infinispan/cache-container=web:write-attribute(name=default-cache,value=dist)
```

2. 设置分布式缓存的所有者。下列命令设置了 5 个所有者。默认值是 2。

```
/subsystem=infinispan/cache-container=web/distributed-cache=dist/:write-attribute(name=owners,value=5)
```

3. 重新加载服务器。

重新载入

22.3.3.2. 缓存策略性能

当使用 **SYNC** 缓存策略时，复制的代价可以被方便地评估。因为请求只有在复制完成后才可以完成，因此可以直接看到请求响应时间。

虽然从表面上看，**ASYNC** 缓存策略的响应时间会比 **SYNC** 缓存策略的响应时间要快，但这只在特定条件下才是正确的。**ASYNC** 缓存策略不容易被评估，但如果不同请求的间隔足够长（缓存操作可以在这个间隔内完成），这个策略的性能会比 **SYNC** 策略的性能要高。这是因为，复制的成本不会在响应时间中马上显现出来。

如果对同一个会话的两个请求的间隔时间太短，前一个请求所造成的复制成本会显现到后一个请求中，因为后一个请求需要等待前一个请求所造成的复制完成后才可以被处理。当一个请求接收到响应信息后马上就发送下一个请求，**ASYNC** 缓存策略的性能就会比 **SYNC** 的缓存性能差。因此，只有在对同一个会话的不同请求间有一定的间隔时间（间隔阈值）时，**ASYNC** 缓存策略的性能才会比 **SYNC** 缓存策略的性能好。在实际的环境中，对同一会话的不同请求间通常都会有一定的间隔时间，这个间隔时间一般都会有几秒钟或更长。在这种情况下，把 **ASYNC** 缓存策略作为默认设置是合理的，它可以提供更快的响应时间。

22.3.4. 配置 Infinispan 线程池

infinispan 子系统包括 **async-operations**、**expiration**、**listener**、**persistence**、**remote-command**、**state-transfer** 和 **transport** 线程池。可以为任何 Infinispan 缓存容器配置这些池。

下表列出了您可以为 **infinispan** 子系统每个线程池配置的属性和默认值。

线程池的名称	keepalive-time	max-threads	min-threads	queue-length
async-operations	60000L	25	25	1000
expiration	60000L	1	N/A	N/A
listener	60000L	1	1	100000
persistence	60000L	4	1	0
remote-command	60000L	200	1	0
state-transfer	60000L	60	1	0

线程池的名称	keepalive-time	max-threads	min-threads	queue-length
transport	60000L	25	25	100000

使用下列管理 CLI 命令来配置 Infinispan 线程池。

```
/subsystem=infinispan/cache-container=CACHE_CONTAINER_NAME/thread-
pool=THREAD_POOL_NAME:write-attribute(name=ATTRIBUTE_NAME,
value=ATTRIBUTE_VALUE)
```

下面的管理 CLI 命令示例在 **server** 缓存容器的 **persistence** 线程池中把 **max-threads** 的值设为 **10**。

```
/subsystem=infinispan/cache-container=server/thread-
pool=persistence:write-attribute(name="max-threads", value="10")
```

22.3.5. Infinispan 统计

可以启用对 Infinispan 缓存和缓存容器的统计功能来进行监控。因为性能的原因，收集统计数据的功能在默认情况下没有被启用。

统计数据收集功能可以针对每个缓存容器、缓存或缓存容器及缓存启动。缓存容器的统计数据收集功能设置会被容器中的缓存继承，而对具体缓存的设置会覆盖从相应缓存容器中继承的设置。

22.3.5.1. 启用 Infinispan 统计



警告

启用 Infinispan 统计功能可能会对 **infinispan** 子系统的性能产生影响。请只在需要时启用统计功能。

您可以使用管理控制台或管理 CLI 来启用或禁用收集 Infinispan 统计数据的功能。在管理控制台中，从 **Configuration** 标签页中进入 **Infinispan** 子系统，选择相关的缓存或缓存容器，编辑 **Statistics enabled** 属性。使用以下管理 CLI 命令也可以启用统计功能。

为一个缓存容器启用统计数据收集功能。服务器需要被重新加载。

```
/subsystem=infinispan/cache-container=CACHE_CONTAINER:write-
attribute(name=statistics-enabled,value=true)
```

为一个缓存启用统计数据收集功能。服务器需要被重新加载。

```
/subsystem=infinispan/cache-
container=CACHE_CONTAINER/CACHE_TYPE=CACHE:write-
attribute(name=statistics-enabled,value=true)
```



注意

您可以使用以下命令来取消定义一个缓存的 **statistics-enabled** 属性，这个缓存会继承它的缓存容器的 **statistics-enabled** 属性。

```
/subsystem=infinispan/cache-  
container=CACHE_CONTAINER/CACHE_TYPE=CACHE:undefine-  
attribute(name=statistics-enabled)
```

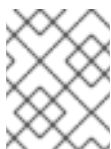
22.3.6. Infinispan 分区处理

*Infinispan 集群*是由一组节点组成的、用来存储数据的环境。为了防止在出现多个节点故障时不会丢失数据，Infinispan 会把相同数据复制到多个节点。使用 **owners** 属性可以配置数据冗余的级别。只要出现问题的节点数量少于配置的数量，Infinispan 就会具有有效的数据。

但是，当集群中的太多节点出现问题时，就可以会造成灾难性的情况发生。

裂脑

把集群分裂为两个或多个分区或子集群，每个分区都独立操作。在这种情况下，在不同分区中读或写的客户端就会使用同一个缓存项的不同版本，这对于许多应用程序来讲都会是问题。



注意

可以使用一些技术来缓解裂脑发生的可能性，如网络冗余，或 [IP 绑定](#)。但是，这只会减少可能发生问题的时间窗口。

多个节点相继崩溃

如果多个节点（特别是所有者）在非常短的时间内相继崩溃，Infinispan 将不会有足够时间在不同的崩溃间适当再平衡它们的状态，从而导致部分的数据丢失。

我们所要实现的目标是，尽量避免当出现裂脑或在短时间内出现多个节点崩溃时，为用户提供不正确数据的情况。

22.3.6.1. 裂脑

对于裂脑的情况，每个网络分区都会安装自己的 JGroups 视图，其中会删除其它分区中的节点。我们没有一个直接的方式来决定集群已被分隔为两个或多个分区，因为分区彼此间不知道其它分区的存在。但是，当一个或多个节点从 JGroups 集群中消失而没有发送明确的离开信息时。我们就认为集群已被分隔。

当分区处理被禁用时，每个这样的分区都会以一个独立集群的形式继续工作。每个分区可能只会看到数据的一部分，并在分区中写有冲突的数据。

当分区处理启动时，在发现分隔情况时每个分区不会马上开始再平衡操作，而是首先检查是否应该进入降级模式：

- 当最少一个段已丢失了它的全部所有者时，就意味着从最后一次再平衡结束后，所指定数量的所有者已离开，分区进入降级模式。
- 如果分区不包括最后稳定拓扑中的大多数节点（节点数量除以 2，结果取整数，再加 1），分区也会进入降级模式。
- 其它所有情况，分区继续工作并开始一个再平衡操作。

当再平衡操作结束，协调程序决定不再需要另外一个再平衡操作时，*稳定拓扑*被更新。这些规则确保了，最起码有一个分区处于可用模式，其它分区变为降级模式。

当一个分区处于降级模式，则只能访问完全拥有的关键字：

- 对在节点上有所有副本的项的请求（读请求和写请求）可以正确处理。
- 对完全或部分由消失节点所拥有的项进行的请求会被拒绝（返回一个 **AvailabilityException**）。

这可以保证，分区不会为相同的项写不同的值（缓存不统一）。另外，一个分区也不会读已在其它分区被更新的关键字（过期数据）。



注意

只要两个分区不进行合并，它们就可以单独启动，并可以读和写不统一的数据。以后，我们可能会允许使用自定义的可用性策略（如检查一个特定节点是集群的一部分，或检查一个外部机器可以被访问）。

22.3.6.2. 配置分区处理

当前，分区处理在默认情况下被禁用。使用以下管理 CLI 命令来启用分区处理功能：

```
/subsystem=infinispan/cache-container=web/distributed-
cache=dist/component=partition-handling:write-attribute(name=enabled,
value=true)
```

22.3.7. 外部化 HTTP 会话到 JBoss Data Grid

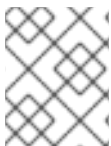


注意

您需要 Red Hat JBoss Data Grid 订阅才可以使用这个功能。

Red Hat JBoss Data Grid 可以被用来作为 JBoss EAP 中特定应用程序（如 HTTP 会话）的外部缓存容器。这可以在独立于应用程序的情况下扩展数据层，并允许处于不同域中的不同 JBoss EAP 集群从相同的 JBoss Data Grid 集群中访问数据。另外，其它应用程序也可以和 Red Hat JBoss Data Grid 提供的缓存有接口。

以下示例显示了如何外部化 HTTP 会话。它适用于独立的 JBoss EAP 实例，也适用于受管域。但是，在一个受管域中，每个服务器组都需要配置一个唯一的远程缓存。当多个服务器组都可以利用同一个 Red Hat JBoss Data Grid 集群时，相应的远程缓存对于 JBoss EAP 服务器组将会是唯一的。



注意

对于每个发布的应用程序，需要创建一个全新的缓存。它可以在一个已存在的缓存容器（如 web）中创建。

外部化 HTTP 会话：

1. 通过在 **socket-binding-group** 中添加网络信息来定义远程 Red Hat JBoss Data Grid 服务器的位置。

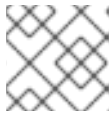
添加远程套接字绑定的示例

```
/socket-binding-group=standard-sockets/remote-destination-outbound-
socket-binding=remote-jdg-server1:add(host=JDGHostName1, port=11222)

/socket-binding-group=standard-sockets/remote-destination-outbound-
socket-binding=remote-jdg-server2:add(host=JDGHostName2, port=11222)
```

XML 结果

```
<socket-binding-group name="standard-sockets" ... >
  ...
  <outbound-socket-binding name="remote-jdg-server1">
    <remote-destination host="JDGHostName1" port="11222"/>
  </outbound-socket-binding>
  <outbound-socket-binding name="remote-jdg-server2">
    <remote-destination host="JDGHostName2" port="11222"/>
  </outbound-socket-binding>
</socket-binding-group>
```



注意

每个 Red Hat JBoss Data Grid 服务器都需要一个远程套接字绑定配置。

2. 确定远程缓存容器已在 JBoss EAP 的 **infinispan** 子系统中进行了定义。在以下的示例中，**remote-store** 项的 **cache** 属性定义了远程 JBoss Data Grid 服务器上的缓存名。如果运行在一个受管域中，在这些命令中使用 **/profile=PROFILE_NAME** 参数。

添加一个远程缓存容器的示例

```
/subsystem=infinispan/cache-container=web/invalidation-
cache=jdg:add(mode=SYNC)

/subsystem=infinispan/cache-container=web/invalidation-
cache=jdg/component=locking:write-
attribute(name=isolation,value=REPEATABLE_READ)

/subsystem=infinispan/cache-container=web/invalidation-
cache=jdg/component=transaction:write-
attribute(name=mode,value=BATCH)

/subsystem=infinispan/cache-container=web/invalidation-
cache=jdg/store=remote:add(remote-servers=["remote-jdg-
server1","remote-jdg-server2"], cache=default, socket-timeout=60000,
passivation=false, purge=false, shared=true)
```

XML 结果

```
<subsystem xmlns="urn:jboss:domain:infinispan:4.0">
  ...
  <cache-container name="web" default-cache="dist"
    module="org.wildfly.clustering.web.infinispan" statistics-
    enabled="true">
    <transport lock-timeout="60000"/>
    <invalidation-cache name="jdg" mode="SYNC">
```

```

        <locking isolation="REPEATABLE_READ"/>
        <transaction mode="BATCH"/>
        <remote-store cache="default" socket-timeout="60000" remote-
servers="remote-jdg-server1 remote-jdg-server2" passivation="false"
purge="false" shared="true"/>
    </invalidation-cache>
    ...
</cache-container>
</subsystem>

```

3. 在应用程序的 `jboss-web.xml` 中添加缓存信息。在以下示例中，`web` 是缓存容器的名称；`jdg` 是在这个容器中的相应缓存的名称。

jboss-web.xml 文件示例

```

<?xml version="1.0" encoding="UTF-8"?>
<jboss-web xmlns="http://www.jboss.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.jboss.com/xml/ns/javaee
http://www.jboss.org/j2ee/schema/jboss-web_10_0.xsd"
    version="10.0">
    <replication-config>
        <replication-granularity>SESSION</replication-granularity>
        <cache-name>web.jdg</cache-name>
    </replication-config>
</jboss-web>

```

22.4. 配置 JBOSS EAP 作为一个前端负载均衡器

您可以把 JBoss EAP 和 `undertow` 子系统配置为一个前端的负载均衡器，使用它来代理到后端 JBoss EAP 服务器的请求。因为 Undertow 使用异步 IO，所以用来负责连接的 IO 线程是唯一参与到请求处理中的线程。这个线程还被用来进行到后端服务器的连接。

您可以使用以下协议：

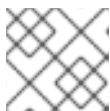
- 使用明文的 HTTP (`http`)，支持 HTTP/1 和 HTTP/2 (`h2c`)



注意

HTTP/2 只作为[技术预览](#)被提供。

- 安全的 HTTP (`https`)，支持 HTTP/1 和 HTTP/2 (`h2`)



注意

HTTP/2 只作为[技术预览](#)被提供。

- AJP (`ajp`)

您可以定义一个[静态负载均衡器](#)并在配置中指定后端主机；或使用 `mod_cluster` 前端来动态更新主机。

22.4.1. 配置 Undertow 作为一个使用 `mod_cluster` 的负载均衡器

您可以使用内建的 `mod_cluster` 前端负载均衡器来进行负载均衡。

这个过程假设，您运行在一个受管域中，并已有以下配置：

- 将充当负载均衡器的 JBoss EAP 服务器。
 - 这个服务器使用了绑定到 **standard-sockets** 套接字绑定组的 **default** 配置集。
- 两个将充当后台服务器的 JBoss EAP 服务器。
 - 这些服务器在集群中运行，并适用绑定了 **ha-sockets** 套接字绑定组的 **ha** 配置集。
- 发布的应用程序在后端服务器中部署了负载均衡功能。

配置 `mod_cluster` 前端负载均衡器

以下步骤在一个受管域中加载负载均衡服务器，但可以修改后应用于独立服务器。请根据您的环境修改相应的管理 CLI 命令。

1. 设置 `mod_cluster` 的 `advertise` 安全密钥。
增加广告密钥可以使负载均衡器和服务器在发现阶段进行身份验证。

使用以下管理 CLI 命令设置 `mod_cluster` 广告密钥。

```
/profile=ha/subsystem=modcluster/mod-cluster-  
config=configuration:write-attribute(name=advertise-security-key,  
value=mypassword)
```

2. 使用多播地址和 `mod_cluster` 的端口创建一个套接字绑定。
您需要为 `mod_cluster` 创建一个套接字配置用于发现将用于进行负载均衡的服务器，并和这些服务器进行通讯。

使用以下管理 CLI 命令添加一个 **modcluster** 套接字绑定来绑定适当的多播地址和配置的端口。

```
/socket-binding-group=standard-sockets/socket-  
binding=modcluster:add(multicast-port=23364, multicast-  
address=224.0.1.105)
```

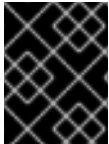
3. 包括 `mod_cluster` 负载均衡器。
在设置了广告安全密钥和套接字绑定后，您就可以为 Undertow 添加 `mod_cluster` 过滤器来作为负载均衡，而不使用 JBoss EAP。

使用以下管理 CLI 命令添加 `mod_cluster` 过滤器。

```
/profile=default/subsystem=undertow/configuration=filter/mod-  
cluster=modcluster:add(management-socket-binding=http, advertise-  
socket-binding=modcluster, security-key=mypassword)
```

使用以下管理 CLI 命令把 `mod_cluster` 过滤器绑定到默认主机。

```
/profile=default/subsystem=undertow/server=default-  
server/host=default-host/filter-ref=modcluster:add
```



重要

我们推荐，`mod_cluster` 使用的管理和广告套接字只在内部网络中可闻，而不是一个公共 IP 地址。

作为负载均衡器的 JBoss EAP 服务器现在可以对两个后端 JBoss EAP 服务器进行负载均衡。

22.4.2. 配置 Undertow 作为一个静态负载均衡器

为了配置 Undertow 作为静态负载均衡器，您需要在 `undertow` 子系统中配置一个代理处理程序（proxy handler）。要在 Undertow 中配置代理处理程序，需要在作为静态负载均衡器的 JBoss EAP 实例上进行以下操作：

1. 添加一个反向代理处理程序
2. 为每个远程主机定义外向的套接字绑定
3. 把每个远程主机添加到反向代理处理程序
4. 添加反向代理位置

以下示例展示了如何配置一个 JBoss EAP 实例作为一个静态负载均衡器。JBoss EAP 实例位于 **lb.example.com**，并会在服务器 **server1.example.com** 和 **server2.example.com** 间进行负载均衡。负载均衡器会把代理反向到 `/app`，并使用 AJP 协议。

1. 添加反向代理处理程序：

```
/subsystem=undertow/configuration=handler/reverse-proxy=my-  
handler:add
```

2. 为每个远程主机定义外向的套接字绑定：

```
/socket-binding-group=standard-sockets/remote-destination-outbound-  
socket-binding=remote-host1/:add(host=server1.example.com,  
port=8009)
```

```
/socket-binding-group=standard-sockets/remote-destination-outbound-  
socket-binding=remote-host2/:add(host=server2.example.com,  
port=8009)
```

3. 把每个远程主机添加到反向代理处理程序：

```
/subsystem=undertow/configuration=handler/reverse-proxy=my-  
handler/host=host1:add(outbound-socket-binding=remote-host1,  
scheme=ajp, instance-id=myroute, path=/test)
```

```
/subsystem=undertow/configuration=handler/reverse-proxy=my-  
handler/host=host2:add(outbound-socket-binding=remote-host2,  
scheme=ajp, instance-id=myroute, path=/test)
```

4. 添加反向代理的位置：

```
/subsystem=undertow/server=default-server/host=default-  
host/location=/app:add(handler=my-handler)
```


当访问 `lb.example.com:8080/app` 时，您将可以看到内容是通过 `server1.example.com` 和 `server2.example.com` 进行代理的。

22.5. 使用一个外部 WEB 服务器作为一个代理服务器

根据外部 web 服务器的配置，JBoss EAP 可以接受来自于使用 HTTP、HTTPS 或 AJP 协议的外部 web 服务器的请求。

如需了解每个 web 服务器所支持的 HTTP 连接器的详细信息，请参阅 [HTTP 连接器概述](#)。在决定使用什么 web 服务器和 HTTP 连接器后，请参阅相关的章节获得如何配置这些连接器的信息：

- 对于 [Apache HTTP Server](#)，请参阅 [mod_cluster](#)、[mod_jk](#) 或 [mod_proxy](#)。
- 对于 Microsoft IIS，请参阅 [ISAPI 连接器](#)。
- 对于 Oracle iPlanet Web Server，请参阅 [NSAPI 连接器](#)。

如需了解支持的 HTTP 连接器配置的最新信息，请参阅 [JBoss EAP supported configurations](#)。

您还需要确定，JBoss EAP 已被配置为[可以接收外部 web 服务器的请求](#)。

22.5.1. HTTP 连接器概述

JBoss EAP 可以使用外部 web 服务器（如 Apache HTTP Server、Microsoft IIS 和 Oracle iPlanet）中内建的、负载均衡和集群机制，或通过 Undertow 实现这些功能。JBoss EAP 使用一个连接器（connector）来和 web 服务器进行通讯。这些连接器在 JBoss EAP 的 `undertow` 子系统中进行配置。

web 服务器包括用来控制把 HTTP 请求路由到 JBoss EAP 节点的软件模块。这些模块中的每个模块的工作方式和配置方法都会不同。它们会被配置来在多个 JBoss EAP 节点间进行负载均衡；或配置来在服务器出现问题时，把工作负载移到其它服务器上；或配置为实现以上两种功能。

JBoss EAP 支持不同的连接器，您需要根据您所使用的 web 服务器，以及所需功能进行选择。下表包括了 JBoss EAP 支持的 HTTP 连接器的配置及功能比较信息。



注意

如果需要使用 JBoss EAP 7 作为一个多平台的负载均衡器，请参阅[使用 mod_cluster 配置 Undertow 作为一个负载均衡器](#)。

如需了解支持的 HTTP 连接器配置的最新信息，请参阅 [JBoss EAP supported configurations](#)。

表 22.1. HTTP 连接器支持的配置

连接器	Web Server	支持的操作系统	支持的协议
mod_cluster	Red Hat JBoss Core Services Apache HTTP Server、Red Hat JBoss Web Server Apache HTTP Server	Red Hat Enterprise Linux、 Microsoft Windows Server、Oracle Solaris	HTTP, HTTPS, AJP

连接器	Web Server	支持的操作系统	支持的协议
mod_jk	Red Hat JBoss Core Services Apache HTTP Server、Red Hat JBoss Web Server Apache HTTP Server	Red Hat Enterprise Linux、 Microsoft Windows Server、Oracle Solaris	AJP
mod_proxy	Red Hat JBoss Core Services Apache HTTP Server、Red Hat JBoss Web Server Apache HTTP Server	Red Hat Enterprise Linux、 Microsoft Windows Server、Oracle Solaris	HTTP, HTTPS, AJP
ISAPI connector	Microsoft IIS	Microsoft Windows Server	AJP
NSAPI connector	Oracle iPlanet Web Server	Oracle Solaris	AJP

表 22.2. HTTP 连接器功能

连接器	支持粘性会话 (Sticky Session)	根据部署状态进行调整
mod_cluster	是	是。检测应用程序的部署或卸载，根据应用程序是否在此服务器上部署来动态决定，是否把客户端请求发送到此服务器。
mod_jk	是	否。无论应用程序是什么状态，只要容器可用，就把客户端请求发送到此容器。
mod_proxy	是	否。无论应用程序是什么状态，只要容器可用，就把客户端请求发送到此容器。
ISAPI connector	是	否。无论应用程序是什么状态，只要容器可用，就把客户端请求发送到此容器。
NSAPI connector	是	否。无论应用程序是什么状态，只要容器可用，就把客户端请求发送到此容器。

22.5.2. Apache HTTP Server

一个独立的 Apache HTTP Server 现在可以使用 Red Hat JBoss Core Services 进行单独下载。这简化了安装和配置的过程，并使更新过程更统一。

22.5.2.1. 安装 Apache HTTP Server

如需了解安装 Apache HTTP Server 的信息，请参阅 JBoss Core Services [Apache HTTP Server Installation Guide](#)。

22.5.3. 接受外部 web 服务器的请求

当配置了正确的协议 handler（如 AJP、HTTP 或 HTTPS）后，JBoss EAP 不需要特殊的配置就可以开始接受一个代理服务器的请求。

如果代理服务器使用 `mod_jk`、`mod_proxy`、ISAPI 或 NSAPI，它会把请求发送到 JBoss EAP，JBoss EAP 将会进行响应。对于 `mod_cluster`，则需要配置网络来允许 JBoss EAP 发送信息（如当前的负载、应用程序生命周期事件、健康状态）到代理服务器，代理服务器会根据这些信息决定把请求发送到什么地方。如需了解更多配置 `mod_cluster` 代理服务器的信息，请参阅 [mod_cluster HTTP 连接器](#)。

更新 JBoss EAP 配置

使用您实际需要的配置替换以下过程中使用的协议和端口。

1. 配置 Undertow 的 **instance-id** 属性。

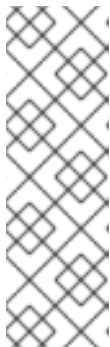
外部 web 服务器在它的连接器配置中使用 **instance-id** 来指代 JBoss EAP 实例。使用以下管理 CLI 命令在 Undertow 中配置 **instance-id** 属性。

```
/subsystem=undertow:write-attribute(name=instance-id,value=node1)
```

在上面的示例中，外部 web 服务器使用 **node1** 代表当前的 JBoss EAP 实例。

2. 为 Undertow 添加所需的 listener。

为了使外部服务器可以连接到 JBoss EAP，Undertow 需要一个 listener。每个协议都需要自己的 listener，它会和一个套接字绑定相关联。



注意

取决于您需要的协议和端口配置，这一步可能并不需要。HTTP listener 在所有默认 JBoss EAP 配置中都进行了配置；如果使用 *ha* 或 *full-ha* 配置集，则 AJP listener 会被配置。

查看默认服务器配置可以知道所需的 listener 是否已被配置：

```
/subsystem=undertow/server=default-server:read-resource
```

为 Undertow 添加一个 listener 需要有一个套接字绑定。套接字绑定被添加到服务器或服务器组所使用的套接字绑定组中。以下管理 CLI 命令把一个 **ajp** 套接字绑定（绑定端口 **8009**）添加到 **standard-sockets** 套接字绑定组中。

```
/socket-binding-group=standard-sockets/socket-binding=ajp:add(port=8009)
```

以下管理 CLI 命令使用 **ajp** 套接字绑定为 Undertow 添加了一个 **ajp** listener。

```
/subsystem=undertow/server=default-server/ajp-listener=ajp:add(socket-binding=ajp)
```

22.6. MOD_CLUSTER HTTP 连接器

`mod_cluster` 连接器是一个基于 Apache HTTP Server 的负载均衡器。它使用一个通讯频道来转发来自 Apache HTTP Server 的请求至一系列应用服务器节点中的一个。

`mod_cluster` 连接器和其它连接器相比有一些优势。

- `mod_cluster` 管理协议（MCMP）是在 JBoss EAP 服务和启用了 `mod_cluster` 模块的 Apache HTTP Server 间的一个额外的连接。JBoss EAP 服务器使用它来通过一组自定义的 HTTP 方法把传输服务器方的负载均衡因子和生命周期事件传输回 Apache HTTP Server。
- 动态配置带有 `mod_cluster` 的 Apache HTTP Server 将可以在不需要手工配置的情况下，使 JBoss EAP 服务器可以加入到负载均衡协议中。
- JBoss EAP 会进行负载均衡因子计算，而不是依赖于带有 `mod_cluster` 的 Apache HTTP Server。因此，它的负载均衡指标数据比其它连接器的数据更准确。
- `mod_cluster` 连接器提供了细颗粒的应用程序生命周期控制。每个 JBoss EAP 服务器都会把 web 应用程序上下文生命周期事件转发到 Apache HTTP 服务器，通知它为特定上下文开始或停止路由请求。这可以防止最终用户因为不可用的资源而接收到错误信息。
- 可以使用 AJP、HTTP 或 HTTPS 传输。

如需了解更多与 `modcluster` 子系统特定配置选项相关的信息，请参阅 [ModCluster 子系统属性](#)。

22.6.1. 在 Apache HTTP Server 中配置 `mod_cluster`

在安装 JBoss Core Services Apache HTTP Server 或使用 JBoss Web Server 时，`mod_proxy` 模块已被包含，并在默认情况下被加载。

根据以下信息配置您的 `mod_cluster` 模块。



注意

红帽客户也可以使用红帽客户门户提供的 [Load Balancer 配置工具](#) 快速生成用于 `mod_cluster` 和其他连接器的优化的配置模板。请注意，您必须先登录才能访问这个工具。

配置 `mod_cluster`

Apache HTTP Server 已经包括了一个 `mod_cluster` 配置文件（`mod_cluster.conf`），它会加载 `mod_cluster` 模块并提供基本的配置。这个文件中的 IP 地址、端口和其它设置都可以根据需要进行修改。

```
# mod_proxy_balancer should be disabled when mod_cluster is used
LoadModule proxy_cluster_module modules/mod_proxy_cluster.so
LoadModule cluster_slotmem_module modules/mod_cluster_slotmem.so
LoadModule manager_module modules/mod_manager.so
LoadModule advertise_module modules/mod_advertise.so
```

```
MemManagerFile cache/mod_cluster
```

```
<IfModule manager_module>
  Listen 6666
  <VirtualHost *:6666>
    <Directory />
      Require ip 127.0.0.1
    </Directory>
    ServerAdvertise on
    EnableMCPMReceive
    <Location /mod_cluster_manager>
      SetHandler mod_cluster-manager
      Require ip 127.0.0.1
    </Location>
  </VirtualHost>
</IfModule>
```

```
</Location>
</VirtualHost>
</IfModule>
```

Apache HTTP Server 服务器可以被配置为一个负载均衡器，并和 JBoss EAP 上运行的 **modcluster** 子系统一起工作。您需要[配置一个 mod_cluster worker 节点](#)才可以使 JBoss EAP 支持 mod_cluster。

如果您需要为 mod_cluster 禁用广告，并配置一个静态代理列表来替代它，请参阅[为 mod_cluster 禁用广告](#)。如需了解在 Apache HTTP Server 中可用的 mod_cluster 配置选项，请参阅[Apache HTTP Server mod_cluster Directives](#)

如需了解更多与配置 mod_cluster 相关的信息，请参阅 JBoss Web Server *HTTP Connectors and Load Balancing Guide* 中的[Configure Load Balancing Using Apache HTTP Server and mod_cluster](#) 一节。

22.6.2. 为 mod_cluster 禁用广告

在默认情况下，**modcluster** 子系统的负载均衡器使用多播 UDP 来向后台的 worker 广告它的可用性。您可用使用以下操作来禁用广告功能，并使用一个代理列表来替代它。



注意

以下操作中使用的管理 CLI 命令假设您在一个受管域中使用 **full-ha** 配置集。如果您没有使用 **full-ha** 配置集，则需要在命令中使用实际的配置集名。如果您运行一个独立的服务器，完全删除 **/profile=full-ha**。

1. 修改 Apache HTTP Server 配置。

编辑 **httpd.conf** Apache HTTP Server 配置文件。使用 **EnableMCPMReceive** 项对监听 MCPM 请求的虚拟主机进行以下更新。

a. 添加一个项来禁用服务器广告。

把 **ServerAdvertise** 项设置为 **Off** 来禁用服务器广告。

```
ServerAdvertise Off
```

b. 禁用广告的频率。

如果配置指定了 **AdvertiseFrequency** 参数，使用 **#** 把它注释掉。

```
# AdvertiseFrequency 5
```

c. 启用接收 MCPM 信息的功能。

确认存在 **EnableMCPMReceive** 项，这将允许 web 服务器从 worker 节点接收 MCPM 信息。

```
EnableMCPMReceive
```

2. 在 JBoss EAP **modcluster** 子系统中禁用广告。

使用以下管理 CLI 命令禁用广告。

```
/profile=full-ha/subsystem=modcluster/mod-cluster-
config=configuration/:write-attribute(name=advertise,value=false)
```



重要

继续以下步骤来提供代理列表。如果代理列表为空，广告将不会被禁用。

3. 在 JBoss EAP **modcluster** 子系统中提供代理列表。

当广告被禁用时，**modcluster** 子系统不会自动发现代理，因此需要提供代理列表。

首先，在适当的套接字绑定组中定义外向的套接字绑定。

```
/socket-binding-group=full-ha-sockets/remote-destination-outbound-
socket-binding=proxy1:add(host=10.33.144.3,port=6666)
/socket-binding-group=full-ha-sockets/remote-destination-outbound-
socket-binding=proxy2:add(host=10.33.144.1,port=6666)
```

接下来，为 **mod_cluster** 配置添加代理。

```
/profile=full-ha/subsystem=modcluster/mod-cluster-
config=configuration:list-add(name=proxies,value=proxy1)
/profile=full-ha/subsystem=modcluster/mod-cluster-
config=configuration:list-add(name=proxies,value=proxy2)
```

Apache HTTP Server 均衡器将不再向 worker 节点广告它的存在，UDP 多播将不再被使用。

22.6.3. 配置一个 **mod_cluster Worker** 节点

mod_cluster worker 节点包括一个 JBoss EAP 服务器。这个服务器可以是一个独立的服务器，也可以是一个受管域中的一个服务器组中的一部分。一个独立的进程会在 JBoss EAP 中运行，它用来管理集群中的所有 worker 节点。它被称为主节点（master）。

在一个受管域中的 worker 节点会在服务器组的范围内共享一个相同的配置。作为独立服务器运行的 worker 节点需要单独配置。配置的步骤是相同的。

- 一个独立的服务器需要在启动时带有 *standalone-ha* 或 *standalone-full-ha* 配置集。
- 受管域中的服务器组需要使用 *ha* 或 *full-ha* 配置集，*ha-sockets* 或 *full-ha-sockets* 套接字绑定组。JBoss EAP 带有一个名为 *other-server-group* 的服务器组，它启用了集群功能，并满足以上要求。

配置一个 **worker** 节点

以下操作中使用的管理 CLI 命令假设您使用了一个带有 **full-ha** 配置集的受管域。如果您运行一个独立的服务器，把命令中的 **/profile=full-ha** 部分删除。

1. 配置网络接口

在默认情况下，网络接口都被默认设置为 **127.0.0.1**。用来运行一个独立服务器或一个服务器组中的一个或多个服务器的主机都需要配置为使用它的公共 IP 地址，从而使其它服务器可以看到它。

根据您的环境，使用以下管理 CLI 命令为 **management**、**public** 和 **unsecure** 接口修改外部 IP 地址。使用主机的实际外部 IP 地址替换命令中的 **EXTERNAL_IP_ADDRESS**。

```
/interface=management:write-attribute(name=inet-
address,value="${jboss.bind.address.management:EXTERNAL_IP_ADDRESS}")
/interface=public:write-attribute(name=inet-
```

```
address,value="${jboss.bind.address.public:EXTERNAL_IP_ADDRESS}")
/interface=unsecure:write-attribute(name=inet-
address,value="${jboss.bind.address.unsecure:EXTERNAL_IP_ADDRESS}")
```

重新加载服务器。

重新载入

2. 配置主机名。

为一个受管域中的每个主机设置一个唯一的主机名。这个名称在所有从节点中需要是唯一的，从主机会使用它在集群中进行指代。因此，把您所使用的名称做一个记录。

- a. 启动 JBoss EAP 从主机，使用适当的 **host.xml** 配置文件。

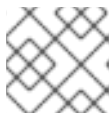
```
$ EAP_HOME/bin/domain.sh --host-config=host-slave.xml
```

- b. 使用以下管理 CLI 命令设置一个唯一的主机名。这个示例中使用 **slave1** 作为新的主机名。

```
/host=EXISTING_HOST_NAME:write-attribute(name=name,value=slave1)
```

如需了解更多与配置一个主机名相关的信息，请参阅[配置主机名](#)。

3. 配置每个主机来连接到域控制器。



注意

独立服务器不需要这个步骤。

对于新创建的、需要加入到一个受管域的主机，需要把本地项删除，并添加指向域控制器的远程项主机属性。

- a. 启动 JBoss EAP 从主机，使用适当的 **host.xml** 配置文件。

```
$ EAP_HOME/bin/domain.sh --host-config=host-slave.xml
```

- b. 使用以下管理 CLI 命令配置域控制器设置。

```
/host=SLAVE_HOST_NAME:write-remote-domain-
controller(host=DOMAIN_CONTROLLER_IP_ADDRESS,port=${jboss.domain.
master.port:9999},security-realm="ManagementRealm")
```

这会修改 host-slave.xml 文件的 XML：

```
<domain-controller>
  <remote host="DOMAIN_CONTROLLER_IP_ADDRESS"
port="${jboss.domain.master.port:9999}" security-
realm="ManagementRealm"/>
</domain-controller>
```

如需了解更多相关信息，请参阅[连接到域控制器](#)。

4. 为每个从主机配置身份验证。

每个从服务器都需要在域控制器或独立主服务器的 ManagementRealm 中创建一个用户名和密码。在域控制器或独立主服务器上为每个主机运行 **EAP_HOME/bin/add-user.sh** 命令。为每个主机添加一个管理用户，它的用户名需要和从服务器中的主机名相匹配。

对最后一个问题 ("Is this new user going to be used for one AS process to connect to another AS process?") 回答 **yes**，您会被提供一个秘密的值。

add-user 脚本输出示例 (已被剪辑)

```
$ EAP_HOME/bin/add-user.sh

What type of user do you wish to add?
  a) Management User (mgmt-users.properties)
  b) Application User (application-users.properties)
(a): a

Username : slave1
Password : changeme
Re-enter Password : changeme
What groups do you want this user to belong to? (Please enter a
comma separated list, or leave blank for none)[ ]:
About to add user 'slave1' for realm 'ManagementRealm'
Is this correct yes/no? yes
Is this new user going to be used for one AS process to connect to
another AS process?
e.g. for a slave host controller connecting to the master or for a
Remoting connection for server to server EJB calls.
yes/no? yes
To represent the user add the following to the server-identities
definition <secret value="SECRET_VALUE" />
```

复制这个输出中的使用 Base64 编码的秘密值 (**SECRET_VALUE**)，它可能需要在下一步中使用。

如需了解更多相关信息，请参阅 [如何配置服务器安全指南](#) 中的 [为主域控制器添加一个用户](#) 一节。

5. 修改从主机的安全域 (security realm) 来使用新的用户验证。

您可以通过在服务器配置中设置秘密的值来指定密码。从 vault 中获得密码，或作为一个系统属性传递密码。

- 使用管理 CLI 在服务器配置文件中指定 Base64 编码的密码值。
使用以下管理 CLI 命令指定秘密值。使用前一步中的 add-user 输出中返回的值替换 **SECRET_VALUE**。

```
/host=SLAVE_HOST_NAME/core-service=management/security-
realm=ManagementRealm/server-
identity=secret:add(value="SECRET_VALUE")
```

您需要重新加载服务器。--host 参数不适用于独立的服务器。

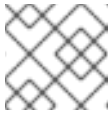
```
reload --host=HOST_NAME
```

如需了解更多相关信息，请参阅 JBoss EAP 的 [如何配置服务器安全指南](#) 中的 [Configuring the Slave Controllers to Use the Credential](#) 一节。

- 配置主机来从 vault 获得密码。

- a. 使用 **EAP_HOME/bin/vault.sh** 脚本产生一个经过掩盖的密码。它会产生一个 **VAULT::secret::password::VAULT_SECRET_VALUE** 格式的字符串。例如：

```
VAULT::secret::password::ODVmYmJjNGMtZDU2ZC00YmNlLWE4ODMtZjQ1N
WNmNDU4ZDc1TEl0RV9CUkVBS3ZhdWx0.
```



注意

在 vault 中创建密码时，需要使用明文，而不是使用 Base64 编码的值。

- b. 使用以下管理 CLI 命令指定秘密值。使用在前一步中创建的已经过掩盖的秘密替换 **VAULT_SECRET_VALUE**。

```
/host=master/core-service=management/security-
realm=ManagementRealm/server-
identity=secret:add(value="${VAULT::secret::password::VAULT_SE
CRET_VALUE}")
```

您需要重新加载服务器。--host 参数不适用于独立的服务器。

```
reload --host=HOST_NAME
```

如需了解更多相关信息，请参阅 JBoss EAP 的 *如何配置服务器安全指南* 中的 [Password Vault](#) 一节。

- 使用一个系统属性指定密码。

以下示例使用 **server.identity.password** 作为密码的系统属性名。

- a. 在服务器的配置文件中设置密码的系统属性名。
使用以下管理 CLI 命令配置安全身份来使用系统属性。

```
/host=SLAVE_HOST_NAME/core-service=management/security-
realm=ManagementRealm/server-
identity=secret:add(value="${server.identity.password}")
```

您需要重新加载服务器。--host 参数不适用于独立的服务器。

```
reload --host=master
```

- b. 在启动服务器时为系统属性设置密码。
您可以使用命令行的一个参数或通过一个属性文件来设置 **server.identity.password** 系统属性。

- i. 使用一个命令行参数。
启动服务器并传递 **server.identity.password** 属性。

```
$ EAP_HOME/bin/domain.sh --host-config=host-slave.xml -
Dserver.identity.password=changeme
```

**警告**

密码需要以明文的方式输入，它对运行 **ps -ef** 命令的任何用户都可见。

- ii. 在属性文件中设置属性。
创建一个属性文件，为它添加相关的“键/值”对。例如：

```
server.identity.password=changeme
```

**警告**

密码以明文的形式存在，并对任何可以访问这个属性文件的用户可见。

使用命令行参数启动服务器。

```
$ EAP_HOME/bin/domain.sh --host-config=host-slave.xml --  
properties=PATH_TO_PROPERTIES_FILE
```

6. 重启服务器

从服务器将使用它的主机名作为用户名，以及加密的字符串作为密码，和主服务器进行验证。

到此，您的独立服务器或一个受管域中的服务器组中的服务器已被配置为 **mod_cluster worker** 节点。如果您需要部署一个集群的应用程序，它的会话会被复制到所有集群节点中以用于故障转移，它可以接收从外部 **web** 器或负载均衡器发来的请求。

22.6.4. 配置 **mod_cluster fail_on_status** 参数

fail_on_status 参数包括了一个 HTTP 状态代码列表，当集群中的一个 **worker** 节点返回了其中的状态代码，则代表这个节点出现故障，负载均衡器将会把请求发送到集群中的其它节点上。这个失败的 **worker** 节点会一直处于一个 **NOTOK** 状态，直到它向负载均衡器发送了一个 **STATUS** 信息。

**注意**

HP 的 HP-UX v11.3 hpws httpd B.2.2.15.15 不支持 **fail_on_status** 参数，所以它不能使用这个参数。

fail_on_status 参数需要在负载均衡器中的 **httpd** 配置文件中进行配置。**fail_on_status** 参数可以指定多个 HTTP 状态代码（以逗号分隔）。下面这个示例为 **fail_on_status** 设置了 HTTP 状态代码 **203** 和 **204**。

fail_on_status 配置示例

```
ProxyPass / balancer://MyBalancer sticky-session=JSESSIONID|jsessionid
nofailover=on failonstatus=203,204
ProxyPassReverse / balancer://MyBalancer
ProxyPreserveHost on
```

22.6.5. 迁移集群间的通讯

在用 JBoss EAP 创建新集群后，作为升级过程的一部分，您可以将之前集群的通讯迁移至新的集群。在这个任务里，您将看到，如何在最小断电或下线时间的情况下迁移这些通讯的策略。

- 新的集群设立（我们将这个集群称为 *ClusterNEW*）。
- 冗余的旧的集群设立（我们称这个集群之为 *ClusterOLD*）。

集群的升级过程 - 负载均衡组

1. 用预备条件里描述的步骤设立新的集群。
2. 在 *ClusterNEW* 和 *ClusterOLD* 里，请确保配置选项 **sticky-session** 被设置为 **true**（它默认是 **true**）。启用这个选项表示对任何集群里的集群节点的所有新的请求都会继续去往各自的集群节点。

```
/profile=full-ha/subsystem=modcluster/mod-cluster-
config=configuration/:write-attribute(name=sticky-
session,value=true)
```

3. 设置 **load-balancing-group** 为 **ClusterOLD**，假设 *ClusterOLD* 里的所有集群节点都是 *ClusterOLD* 负载均衡组里的成员。

```
/profile=full-ha/subsystem=modcluster/mod-cluster-
config=configuration/:write-attribute(name=load-balancing-
group,value=ClusterOLD)
```

```
<subsystem xmlns="urn:jboss:domain:modcluster:2.0">
  <mod-cluster-config load-balancing-group="ClusterOLD" advertise-
socket="modcluster" connector="ajp">
    <dynamic-load-provider>
      <load-metric type="cpu"/>
    </dynamic-load-provider>
  </mod-cluster-config>
</subsystem>
```

4. 使用[配置一个 mod_cluster Worker 节点](#)里描述的步骤单独添加 *ClusterNEW* 里的节点至 *mod_cluster* 配置。此外，按照之前提及的过程并设置负载均衡组为 *ClusterNEW*。此时，您可以在 *mod_cluster-manager* 控制台上看到和下面类似的输出：

```
mod_cluster/<version>

LBGroup ClusterOLD: [Enable Nodes]    [Disable Nodes]    [Stop
Nodes]
    Node node-1-jvmroute (ajp://node1.oldcluster.example:8009):
      [Enable Contexts]    [Disable Contexts]    [Stop
Contexts]
    Balancer: qaccluster, LBGroup: ClusterOLD, Flushpackets:
```

```

Off, ..., Load: 100
    Virtual Host 1:
        Contexts:
            /my-deployed-application-context, Status:
ENABLED Request: 0 [Disable] [Stop]

    Node node-2-jvmroute (ajp://node2.oldcluster.example:8009):
        [Enable Contexts] [Disable Contexts] [Stop
Contexts]
        Balancer: qaclcluster, LBGroup: ClusterOLD, Flushpackets:
Off, ..., Load: 100
        Virtual Host 1:
            Contexts:
                /my-deployed-application-context, Status:
ENABLED Request: 0 [Disable] [Stop]

        LBGroup ClusterNEW: [Enable Nodes] [Disable Nodes] [Stop
Nodes]
        Node node-3-jvmroute (ajp://node3.newcluster.example:8009):
            [Enable Contexts] [Disable Contexts] [Stop
Contexts]
            Balancer: qaclcluster, LBGroup: ClusterNEW, Flushpackets:
Off, ..., Load: 100
            Virtual Host 1:
                Contexts:
                    /my-deployed-application-context, Status:
ENABLED Request: 0 [Disable] [Stop]

            Node node-4-jvmroute (ajp://node4.newcluster.example:8009):
                [Enable Contexts] [Disable Contexts] [Stop
Contexts]
                Balancer: qaclcluster, LBGroup: ClusterNEW, Flushpackets:
Off, ..., Load: 100
                Virtual Host 1:
                    Contexts:
                        /my-deployed-application-context, Status:
ENABLED Request: 0 [Disable] [Stop]

```

5. *ClusterOLD* 组里有旧的活动会话，任何新的会话将在 *ClusterOLD* 或 *ClusterNEW* 组里创建。之后，我们希望禁用整个 *ClusterOLD* 组，这样我们可以关闭它的集群节点而不会在当前活动客户的会话里导致任何错误。

点击 *mod_cluster-manager* web 控制台里 LBGroup *ClusterOLD* 上的 **Disable Nodes** 链接。

从此以后，只有属于已建立会话的请求会被路由至 *ClusterOLD* 负载均衡组的成员。任何新客户的会话将只在 *ClusterNEW* 组里创建。只要 *ClusterOLD* 组里没有活动会话，我们就可以安全地删除它的成员。



注意

使用 **Stop Nodes** 将让负载均衡器停止将任何请求立即路由至这个域。这将强制其他负载均衡组的失效切换，如果 *ClusterNEW* 和 *ClusterOLD* 间没有会话复制，这会导致会话数据丢失。

如果当前的 *ClusterOLD* 设置没有包含任何负载均衡组设置（您可以在 *mod_cluster-manager* 控制台上查看 *LBGroup*），您仍可以禁用 *ClusterOLD* 节点。此时，对每个 *ClusterOLD* 节点点击 **Disable Contexts**。这些节点的上下文将被禁用，一旦不存在活动的会话，它们将可以被删除。新客户的会话将只在启用了上下文的节点上创建，如这个例子的 *ClusterNEW* 成员。

使用管理 CLI

除了使用 *mod_cluster-managerweb* 控制台，您可以调整管理 CLI 来禁用特定的上下文。下面提及的操作被称作 *stop-context*，但它让集群节点发送 *DISABLE-APP* 命令至负载均衡器，这和点击 *mod_cluster-manager* 控制台上特定上下文的 *Disable* 链接有一样的效果（请注意虚拟主机别名，如 *default-host*，已从前提及的 *mod_cluster-manager* 控制台输出示例删除）。

除了使用 *mod_cluster-managerweb* 控制台，您可以调整管理 CLI 来禁用特定的上下文。使用的操作被称作 **stop-context**，但它让集群节点发送 **DISABLE-APP** 命令至负载均衡器，这和点击 *mod_cluster-manager* 控制台上特定上下文的 **Disable** 链接有一样的效果（请注意虚拟主机别名，如 *default-host*，已从前提及的 *mod_cluster-manager* 控制台输出示例删除）。

```
/profile=full-ha/subsystem=modcluster/:stop-context(context=/my-deployed-application-context, virtualhost=default-host, waittime=50)
```

要停止特定的上下文，集群节点或整个负载均衡组会强制平衡器停止立即路由任何请求，从而强制失效切换至其他可用的上下文。要禁用特定的上下文，集群节点或整个负载均衡组需要告诉平衡器不应该在这个特定的上下文/节点/负载均衡组上创建新的会话。

22.7. APACHE MOD_JK HTTP 连接器

Apache mod_jk 是一个 HTTP 连接器，它为特定客户提供了所需的兼容性。

JBoss EAP 可以从 *Apache HTTP* 代理服务器接受工作负载。代理服务器从 *Web* 前端接受客户请求并将工作负荷传递给参与的 JBoss EAP 服务器。如果启用了粘性会话，相同的客户请求总会到达相同的 JBoss EAP 服务器，除非服务器是不可用的。

mod_jk 通过 *AJP 1.3* 协议来通讯。其他协议可以与 *mod_cluster* 或 *mod_proxy* 一起使用。详情请参考 [HTTP 连接器概述](#)。



注意

mod_cluster 是一个比 *mod_jk* 更高级的负载均衡器，它也是我们推荐的 HTTP 连接器。*mod_cluster* 提供了 *mod_jk* 的所有功能以及其他功能。不象 JBoss EAP *mod_cluster* HTTP 连接器，*Apache mod_jk* HTTP 连接器不知道服务器或服务器组上的部署状态，也无法根据工作负载在发送的地点而进行调整。

详情请参考 [Apache mod_jk documentation](#)。

22.7.1. 配置 Apache HTTP 服务器里的 mod_jk

在安装 JBoss Core Services *Apache HTTP Server* 或使用 JBoss *Web Server* 时，*mod_jk* 模块（*mod_jk.so*）是已被包含。然而，默认情况下它不会被加载。使用下列步骤来加载和配置 *Apache HTTP Server* 里的 *mod_jk*。请注意，这些步骤假设您已进入 *Apache HTTP Server* 的 *httpd/* 目录，这个目根根据平台的不同而不同。更多信息请参考《[Apache HTTP Server 安装指南](#)》里的 JBoss Core Services 安装说明。



注意

红帽客户也可以使用客户门户网站上的 [Load Balancer 配置工具](#) 快速生成用于 mod_jk 和其他连接器的优化的配置模板。请注意，您必须先登录才能访问这个工具。

1. 配置 mod_jk 模块。



注意

conf.d/mod_jk.conf.sample 提供了一个 mod_jk 配置文件示例。您可以使用这个示例而不是创建自己的配置文件，您需要删除它的 **.sample** 后缀并按需要修改其内容。

创建一个名为 **conf.d/mod_jk.conf** 的文件。添加下列配置，请确保根据需要修改相关内容。

```
# Load mod_jk module
# Specify the filename of the mod_jk lib
LoadModule jk_module modules/mod_jk.so

# Where to find workers.properties
JkWorkersFile conf.d/workers.properties

# Where to put jk logs
JkLogFile logs/mod_jk.log

# Set the jk log level [debug/error/info]
JkLogLevel info

# Select the log format
JkLogStampFormat "[%a %b %d %H:%M:%S %Y]"

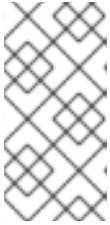
# JkOptions indicates to send SSK KEY SIZE
JkOptions +ForwardKeySize +ForwardURICompat -ForwardDirectories

# JkRequestLogFormat
JkRequestLogFormat "%w %V %T"

# Mount your applications
JkMount /application/* loadbalancer

# Add shared memory.
# This directive is present with 1.2.10 and
# later versions of mod_jk, and is needed for
# for load balancing to work properly
JkShmFile logs/jk.shm

# Add jkstatus for managing runtime data
<Location /jkstatus/>
    JkMount status
    Require ip 127.0.0.1
</Location>
```



注意

JkMount 指令指定 Apache HTTP Server 必须转发哪个 URL 至 mod_jk 模块。根据指令的配置，mod_jk 发送接收的 URL 至正确的工作节点。要直接服务静态的内容且只对 Java 应用程序使用负载均衡器，URL 路径必须为 **/application/***。要将 mod_jk 用作负载均衡器，请使用 **/*** 来转发所有的 URL 至 mod_jk。

除了常用的 mod_jk 配置，这个文件还指定了加载 **mod_jk.so** 模块，并定义了在哪里找到 **workers.properties** 文件。

2. 配置 mod_jk 工作节点。



注意

conf.d/workers.properties.sample 提供了一个工作节点配置文件示例。您可以使用这个示例而不是创建自己的配置文件，您需要删除它的 **.sample** 后缀并按需要修改其内容。

创建一个名为 **conf.d/workers.properties.sample** 的文件。添加下列配置，请确保根据需要修改相关内容。

```
# Define list of workers that will be used
# for mapping requests
worker.list=loadbalancer,status

# Define Node1
# modify the host as your host IP or DNS name.
worker.node1.port=8009
worker.node1.host=node1.mydomain.com
worker.node1.type=ajp13
worker.node1.ping_mode=A
worker.node1.lbfactor=1

# Define Node2
# modify the host as your host IP or DNS name.
worker.node2.port=8009
worker.node2.host=node2.mydomain.com
worker.node2.type=ajp13
worker.node2.ping_mode=A
worker.node2.lbfactor=1

# Load-balancing behavior
worker.loadbalancer.type=lb
worker.loadbalancer.balance_workers=node1,node2
worker.loadbalancer.sticky_session=1

# Status worker for managing load balancer
worker.status.type=status
```

关于 mod_jk **workers.properties** 文件的语法和其他高级配置选项的细节，请参考 [mod_jk 工作节点属性](#)。

3. 您可根据需要指定 JKMountFile 指令。

除了 `mod-jk.conf` 里的 `JKMount` 指令，您还可以指定转发一个包含多个 URL 模式的文件至 `mod_jk`。

- a. 创建 `uriworkermap.properties` 文件。



注意

`conf.d/uriworkermap.properties.sample` 提供了一个 URI 工作节点映射配置文件示例。您可以使用这个示例而不是创建自己的配置文件，您需要删除它的 `.sample` 后缀并按需要修改其内容。

创建一个名为 `conf.d/uriworkermap.properties` 的文件。为每个匹配的 URL 模式添加一行，例如：

```
# Simple worker configuration file
/*=loadbalancer
```

- b. 更新配置使其指向 `uriworkermap.properties` 文件。
在 `conf.d/mod_jk.conf` 里附加下列内容。

```
# Use external file for mount points.
# It will be checked for updates each 60 seconds.
# The format of the file is: /url=worker
# /examples/*=loadbalancer
JkMountFile conf.d/uriworkermap.properties
```

关于配置 `mod_jk` 的更多细节，请参考《JBoss Web Server 的 HTTP 连接器和负载均衡指南》的『[配置 Apache HTTP Server 加载 mod_jk](#)』章节。

22.7.2. 配置 JBoss EAP 与 mod_jk 通讯

JBoss EAP `undertow` 子系统需要指定一个 listener 来接受请求并发送回复至外部的 Web 服务器。既然 `mod_jk` 使用了 AJP 协议，您必须配置 AJP Listener。

如果您在使用其中一个默认的高可用性配置（`ha` 或 `full-ha`），那么已经配置了 AJP Listener。

相关说明请参考[接受外部 web 服务器的请求](#)。

22.8. APACHE MOD_PROXY HTTP 连接器

`Apache mod_proxy` 是一个支持 AJP、HTTP 和 HTTPS 协议连接的 HTTP 连接器。您可以在负载均衡或非负载均衡配置里配置 `mod_proxy`，它支持粘性会话的概念。

`mod_proxy` 模块要求 JBoss EAP 在 `undertow` 子系统里配置 HTTP、HTTPS 或 AJP listener，这取决于您计划使用的协议。



注意

`mod_cluster` 是一个比 `mod_proxy` 更高级的负载均衡器，它也是我们推荐的 HTTP 连接器。`mod_cluster` 提供了 `mod_proxy` 的所有功能以及其他功能。不象 JBoss EAP `mod_cluster` HTTP 连接器，`Apache mod_proxy` HTTP 连接器不知道服务器或服务器组上的部署状态，也无法根据工作负载在发送的地点而进行调整。

详情请参考 [Apache mod_proxy 文档](#)。

22.8.1. 配置 Apache HTTP 里的 mod_proxy

在安装 JBoss Core Services Apache HTTP Server 或使用 JBoss Web Server 时，mod_proxy 模块已被包含，且被默认加载。

请参考下面的内容来配置基本的 [load-balancing](#) 或 [non-load-balancing](#) 代理。这些步骤假定您已经进入了 Apache HTTP Server 的 `httpd/` 目录，这个目录根据平台的不同而可能不同。更多信息请参考 [《Apache HTTP Server 安装指南》](#) 里的 JBoss Core Services 安装说明。这些步骤也假定您已经在 JBoss EAP `undertow` 子系统里配置了必要的 HTTP Listener。



注意

红帽客户也可以使用红帽客户门户网站中的 [Load Balancer 配置工具](#) 快速生成用于 mod_proxy 和其他连接器的优化的配置模板。请注意，您必须先登录才能访问这个工具。

添加 **Non-load-balancing** 代理

在 `conf/httpd.conf` 文件里，把以下配置（根据您的具体设置替换相关的内容）添加到其他 `<VirtualHost>` 指令下面。

```
<VirtualHost *:80>
# Your domain name
ServerName YOUR_DOMAIN_NAME

ProxyPreserveHost On

# The IP and port of JBoss
# These represent the default values, if your httpd is on the same host
# as your JBoss managed domain or server

ProxyPass / http://localhost:8080/
ProxyPassReverse / http://localhost:8080/

# The location of the HTML files, and access control information
DocumentRoot /var/www
<Directory /var/www>
Options -Indexes
Order allow,deny
Allow from all
</Directory>
</VirtualHost>
```

添加 **Load-balancing** 代理



注意

默认的 Apache HTTP Server 配置禁用了 `mod_proxy_balancer.so` 模块，因为它和 `mod_cluster` 不兼容。为了完成这个任务，您将需要加载这个模块并禁用 `mod_cluster` 模块。

要将 mod_proxy 作为负载均衡器，并将工作负载发送到多个 JBoss EAP 实例，把以下配置（根据您的具体情况替换其中的 IP 地址）添加到您的 `conf/httpd.conf` 文件中。

```

<Proxy balancer://mycluster>

Order deny,allow
Allow from all

# Add each JBoss Enterprise Application Server by IP address and port.
# If the route values are unique like this, one node will not fail over to
the other.
BalancerMember http://192.168.1.1:8080 route=node1
BalancerMember http://192.168.1.2:8180 route=node2
</Proxy>

<VirtualHost *:80>
  # Your domain name
  ServerName YOUR_DOMAIN_NAME

  ProxyPreserveHost On
  ProxyPass / balancer://mycluster/

  # The location of the HTML files, and access control information
  DocumentRoot /var/www
  <Directory /var/www>
    Options -Indexes
    Order allow,deny
    Allow from all
  </Directory>

</VirtualHost>

```

上面的例子都使用 HTTP 协议进行通讯。您可以通过加载适当的 `mod_proxy` 模块来使用 AJP 或 HTTPS 协议。详情请参考 [Apache mod_proxy 文档](#)。

启用粘性会话

粘性会话 (*Sticky session*) 表示如果客户请求原本是去往特定的 JBoss EAP 工作节点，那所有将来的请求都送往相同的工作节点，除非它不可用。这一直是我们推荐的行为。

要启用 `mod_proxy` 的粘性会话，请将 **stickysession** 参数添加到 **ProxyPass** 语句里。

```
ProxyPass / balancer://mycluster stickysession=JSESSIONID
```

您可以在 **ProxyPass** 语句里指定其他参数，如 **lbmethod** 和 **nofailover**。关于可用参数的更多信息，请参考 [Apache mod_proxy 文档](#)。

22.8.2. 配置 JBoss EAP 与 `mod_proxy` 通讯

JBoss EAP **undertow** 子系统需要指定一个 listener 来接受来自外部 Web 服务器的请求，并向外部 Web 服务器发送回复。根据您所使用的协议，可能还需要配置 Listener。

JBoss EAP 的默认配置里配置了 HTTP Listener。如果您在使用其中一个默认的高可用性配置 (*ha* 或 *full-ha*)，那么已经配置了 AJP Listener。

相关说明请参考[接受外部 web 服务器的请求](#)。

22.9. MICROSOFT ISAPI CONNECTOR

Internet Server API (ISAPI) 是用于写入 OLE 服务器扩展的一系列 API 及用于 Web 服务器（如 Microsoft 的 IIS）的过滤器。**isapi_redirect.dll** 是针对 IIS 调整的 *mod_jk* 的扩展。**isapi_redirect.dll** 让您可以将 JBoss EAP 配置为工作节点，而将 IIS 作为负载均衡器。



注意

关于受支持的 Windows Server 和 IIS 的配置，请参考 [JBoss EAP 支持的配置](#)。

22.9.1. 配置 Microsoft IIS 使用 ISAPI Connector

从红帽客户门户网站下载 ISAPI connector：

1. 打开浏览器并登录红帽客户门户网站 [JBoss Software Downloads](#) 页面。
2. 从 **Product** 下拉菜单选择 **Web Connectors**。
3. 从 **Version** 下拉菜单里选择最新的 JBoss Core Services 版本。
4. 在列表里找到 **Red Hat JBoss Core Services ISAPI Connector**，然后点击 **Download** 链接。
5. 解压归档并复制 **sbin** 目录的内容至您的服务器。下面的说明假定这些内容被复制到了 **C:\connectors** 目录。

要用 IIS Manager (IIS 7) 配置 IIS Redirector：

1. 点击 **Start** → **Run** 并输入 **inetmgr** 来打开 IIS 管理者。
2. 在左侧的树型视图面板里，展开 **IIS 7**。
3. 双击 **ISAPI and CGI Registrations** 在新窗口里打开它。
4. 在 **Actions** 面板里，点击 **Add**。 **Add ISAPI or CGI Restriction** 窗口将被打开。
5. 指定下列值：
 - **ISAPI or CGI Path**: **C:\connectors\isapi_redirect.dll**
 - **Description** : **jboss**
 - **Allow extension path to execute** : 勾选复选框。
6. 点击 **OK** 关闭 **Add ISAPI or CGI Restriction** 窗口。
7. 定义 JBoss Native 虚拟目录
 - 右击 *Default Web Site* 并点击 *Add Virtual Directory*。 *Add Virtual Directory* 窗口将被打开。
 - 指定下列值来添加虚拟目录：
 - **Alias**: **jboss**
 - **Physical Path**: **C:\connectors**
 - 点击 **OK** 保存修改并关闭 **Add Virtual Directory** 窗口。
8. 定义 JBoss Native ISAPI 重定向过滤器
 - 在树型视图面板里，展开 **Sites** → **Default Web Site**。

- 在树型视图图面里，双击 **ISAPI Filters**。 **ISAPI Filters Features** 视图将显示。
- 在 **Actions** 面板里，点击 'Add'。 **Add ISAPI Filter** 窗口将显示。
- 在 **Add ISAPI Filter** 窗口里指定下列值：
 - Filter name:** jboss
 - Executable:** C:\connectors\isapi_redirect.dll
- 点击 **OK** 保存修改并关闭 **Add ISAPI Filters** 窗口。

9. 启用 ISAPI-dll 处理程序

- 在树型视图面里双击 **IIS 7**。 **IIS 7 Home Features View** 窗口将显示。
- 双击 **Handler Mappings**。 **Handler Mappings Features View** 窗口将显示。
- 在 **Group by** 组合框里，选择 **State**。 **Handler Mappings** 会在 **Enabled and Disabled Groups** 中显示。
- 找到 **ISAPI-dll**。如果它位于 **Disabled** 组，右击它并选择 **Edit Feature Permissions**。
- 启用下列权限：
 - Read
 - Script
 - Execute
- 点击 **OK** 保存修改并关闭 **Edit Feature Permissions** 窗口。

现在已配置 Microsoft IIS 使用 ISAPI 连接器了。

22.9.2. 配置 ISAPI Connector 发送客户请求至 JBoss EAP

这个任务配置一组 JBoss EAP 服务器从 ISAP 连接器接受请求。它没有包含对负载平衡或高可用性失效切换的配置。

这个配置是在 IIS 服务器上完成的，它假定您已配置 JBoss EAP [从外部 Web 服务器接受请求](#)。您也需要对 IIS 服务器的完整管理权限并[配置 IIS 所有 ISAPI 连接器](#)。

创建属性文件并设立重定向

- 创建一个目录来存储日志、属性文件和锁文件。
这个过程的剩余步骤假定您使用的是 **C:\connectors** 目录。如果您使用了不同的目录，请相应地修改说明。
- 创建 **isapi_redirect.properties** 文件。
创建一个名为 **C:\connectors\isapi_redirect.properties** 的新文件。复制下列内容到这个文件里。

```
# Configuration file for the ISAPI Connector
# Extension uri definition
extension_uri=/jboss/isapi_redirect.dll
```

```
# Full path to the log file for the ISAPI Connector
log_file=c:\connectors\isapi_redirect.log

# Log level (debug, info, warn, error or trace)
log_level=info

# Full path to the workers.properties file
worker_file=c:\connectors\workers.properties

# Full path to the uriworkermap.properties file
worker_mount_file=c:\connectors\uriworkermap.properties

#Full path to the rewrite.properties file
rewrite_rule_file=c:\connectors\rewrite.properties
```

如果您不想使用 **rewrite.properties** 文件，请用 # 注释最后一行。

3. 创建 **uriworkermap.properties** 文件

uriworkermap.properties 文件包含部署的应用程序 URL 和工作节点处理程序间的映射。下面的例子展示了这个文件的语法。请将您的 **uriworkermap.properties** 文件放入 **C:\connectors**。

```
# images and css files for path /status are provided by worker01
/status=worker01
/css/*=worker01

# Path /web-console is provided by worker02
# IIS (customized) error page is used for http errors with number
greater or equal to 400
# css files are provided by worker01
/web-console/*=worker02;use_server_errors=400
/web-console/css/*=worker01

# Example of exclusion from mapping, logo.gif won't be displayed
# /web-console/images/logo.gif=*

# Requests to /app-01 or /app-01/something will be routed to
worker01
/app-01|/*=worker01

# Requests to /app-02 or /app-02/something will be routed to
worker02
/app-02|/*=worker02
```

4. 创建 **workers.properties** 文件。

workers.properties 文件包含工作节点标签和服务器实例间的映射定义。这个文件遵循了用于 [Apache mod_jk 工作节点属性](#) 配置的相同的文件的语法。

下面是一个 **workers.properties** 文件示例。工作节点名称 **worker01** 和 **worker02** 必须匹配 [JBoss EAP undertow 子系统配置的 instance-id](#)。

将这个文件放入 **C:\connectors** 目录。

```
# An entry that lists all the workers defined
```

```

worker.list=worker01, worker02

# Entries that define the host and port associated with these
workers

# First JBoss EAP server definition, port 8009 is standard port for
AJP in EAP
worker.worker01.host=127.0.0.1
worker.worker01.port=8009
worker.worker01.type=ajp13

# Second JBoss EAP server definition
worker.worker02.host=127.0.0.100
worker.worker02.port=8009
worker.worker02.type=ajp13

```

5. 创建 **rewrite.properties** 文件。

rewrite.properties 文件包含用于特定应用程序的简单的 URL 重写规则。如下例所示，重写路径是用名-值对指定的。请将这个文件放入 **C:\connectors** 目录。

```

#Simple example
# Images are accessible under abc path
/app-01/abc/=/app-01/images/

```

6. 用 **net stop** 和 **net start** 命令重启您的 IIS 服务器。

```

C:\> net stop was /Y
C:\> net start w3svc

```

我们配置了 IIS 服务器发送客户请求至特定的 JBoss EAP 服务器。

22.9.3. 配置 ISAPI Connector 在多个 JBoss EAP 服务器间平衡客户请求

这个配置在您指定的 JBoss EAP 服务器间平衡客户请求。这个配置是在 IIS 服务器上完成的，它假定您已配置 JBoss EAP [从外部 Web 服务器接受请求](#)。您也需要对 IIS 服务器的完整管理权限并[配置 IIS 所有 ISAPI 连接器](#)。

平衡多个服务器间的客户请求

1. 创建一个目录来存储日志、属性文件和锁文件。

这个过程的剩余步骤假定您使用的是 **C:\connectors** 目录。如果您使用了不同的目录，请相应地修改说明。

2. 创建 **isapi_redirect.properties** 文件。

创建一个名为 **C:\connectors\isapi_redirect.properties** 的新文件。复制下列内容到这个文件里。

```

# Configuration file for the ISAPI Connector
# Extension uri definition
extension_uri=/jboss/isapi_redirect.dll

# Full path to the log file for the ISAPI Connector
log_file=c:\connectors\isapi_redirect.log

```

```
# Log level (debug, info, warn, error or trace)
log_level=info

# Full path to the workers.properties file
worker_file=c:\connectors\workers.properties

# Full path to the uriworkermapping.properties file
worker_mount_file=c:\connectors\uriworkermapping.properties

#OPTIONAL: Full path to the rewrite.properties file
rewrite_rule_file=c:\connectors\rewrite.properties
```

如果您不想使用 **rewrite.properties** 文件，请用 # 注释最后一行。

3. 创建 **uriworkermapping.properties** 文件。

uriworkermapping.properties 文件包含部署的应用程序 URL 和工作节点处理程序间的映射。下面的例子展示了带有负载均衡配置的文件语法。通配符 (*) 发送不同 URL 子目录的所有请求至负载均衡器调用的路由器。下一步骤将涵盖关于负载均衡器的配置。

将您的 **uriworkermapping.properties** 文件放入 **C:\connectors**。

```
# images, css files, path /status and /web-console will be
# provided by nodes defined in the load-balancer called "router"
/css/*=router
/images/*=router
/status=router
/web-console|/*=router

# Example of exclusion from mapping, logo.gif won't be displayed
# /web-console/images/logo.gif=*

# Requests to /app-01 and /app-02 will be routed to nodes defined
# in the load-balancer called "router"
/app-01|/*=router
/app-02|/*=router

# mapping for management console, nodes in cluster can be enabled or
# disabled here
/jkmanager|/*=status
```

4. 创建 **workers.properties** 文件。

workers.properties 文件包含工作节点标签和服务器实例间的映射定义。这个文件遵循了用于 [Apache mod_jk 工作节点属性](#) 配置的相同的文件的语法。

下面是一个 **workers.properties** 文件示例。负载均衡器在文件结尾进行配置，它由工作节点 **worker01** 和 **worker02** 组成。这些工作节点名称必须匹配 [JBoss EAP undertow 子系统配置](#) 的 **instance-id**。

将这个文件放入 **C:\connectors** 目录。

```
# The advanced router LB worker
worker.list=router,status

# First EAP server definition, port 8009 is standard port for AJP in
EAP
```

```
#
# lbfactor defines how much the worker will be used.
# The higher the number, the more requests are served
# lbfactor is useful when one machine is more powerful
# ping_mode=A - all possible probes will be used to determine that
# connections are still working

worker.worker01.port=8009
worker.worker01.host=127.0.0.1
worker.worker01.type=ajp13
worker.worker01.ping_mode=A
worker.worker01.socket_timeout=10
worker.worker01.lbfactor=3

# Second EAP server definition
worker.worker02.port=8009
worker.worker02.host=127.0.0.100
worker.worker02.type=ajp13
worker.worker02.ping_mode=A
worker.worker02.socket_timeout=10
worker.worker02.lbfactor=1

# Define the LB worker
worker.router.type=lb
worker.router.balance_workers=worker01,worker02

# Define the status worker for jkmanager
worker.status.type=status
```

5. 创建 **rewrite.properties** 文件。

rewrite.properties 文件包含用于特定应用程序的简单的 URL 重写规则。如下例所示，重写路径是用名-值对指定的。请将这个文件放入 **C:\connectors** 目录。

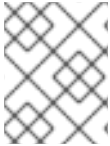
```
#Simple example
# Images are accessible under abc path
/app-01/abc/= /app-01/images/
Restart the IIS server.

Restart your IIS server by using the net stop and net start
commands.
C:\> net stop was /Y
C:\> net start w3svc
```

我们配置了 IIS 服务器发送客户请求至 **workers.properties** 文件里引用的 JBoss EAP 服务器，按 **1:3** 的比例将负载散布在服务器上。这个比例是从分配给每个服务器的负载平衡因子（lbfactor）衍生的。

22.10. ORACLE NSAPI CONNECTOR

Netscape Server API (NSAPI) 是 Oracle iPlanet Web Server（之前称为 Netscape Web Server）提供的 API，用于实现服务器的扩展。这些扩展被称为服务器插件。NSAPI 连接器用在 **nsapi_redirector.so** 里，它是根据 Oracle iPlanet Web Server 调整的 mod_jk 的扩展。NSAP 连接器让您配置 JBoss EAP 实例为工作节点，并将 Oracle iPlanet Web Server 作为负载均衡器。



注意

关于受支持的 Solaris 和 Oracle iPlanet Web Server 的配置，请参考 [JBoss EAP 支持的配置](#)。

22.10.1. 配置 Oracle iPlanet Web Server 使用 NSAPI Connector

先决条件

- JBoss EAP 在将作为工作节点的每台服务器上安装和配置。

从红帽客户门户网站下载 NSAPI 连接器：

1. 打开浏览器并登录红帽客户门户网站 [JBoss Software Downloads](#) 页面。
2. 从 **Product** 下拉菜单选择 **Web Connectors**。
3. 从 **Version** 下拉菜单里选择最新的 JBoss Core Services 版本。
4. 在列表里找到 **Red Hat JBoss Core Services NSAPI Connector**，确保您选择了正确的平台和架构，然后点击 **Download** 链接。
5. 解压位于 **lib/** 或 **lib64/** 目录里的 **nsapi_redirector.so** 文件到 **IPLANET_CONFIG/lib/** 或 **IPLANET_CONFIG/lib64/** 目录。

设置 NSAPI Connector：



注意

在这些说明里，**IPLANET_CONFIG** 指的是 Oracle iPlanet 配置目录，它通常是 **/opt/oracle/webserver7/config/**。如果您的 Oracle iPlanet 配置目录是不一样的，请相应地进行修改。

1. 禁用 servlet 映射。
打开 **IPLANET_CONFIG/default.web.xml** 文件并定位标题为 *Built In Server Mappings* 的部分。用 XML 注释字符（**<!--** 和 **-->**）来禁用对下列三个 Servlet 的映射。

- default
- invoker
- jsp

下面的配置示例展示了禁用的映射。

```
<!-- ===== Built In Servlet Mappings ===== -->
<!-- The servlet mappings for the built in servlets defined
above. -->
<!-- The mapping for the default servlet -->
<!--servlet-mapping>
  <servlet-name>default</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping-->
<!-- The mapping for the invoker servlet -->
<!--servlet-mapping>
  <servlet-name>invoker</servlet-name>
```

```
<url-pattern>/servlet/*</url-pattern>
</servlet-mapping-->
<!-- The mapping for the JSP servlet -->
<!--servlet-mapping>
  <servlet-name>jsp</servlet-name>
  <url-pattern>*.jsp</url-pattern>
</servlet-mapping-->
```

保存文件并退出。

2. 配置 iPlanet Web Server 加载 NSAP Connector 模块。

添加下列行至 **IPLANET_CONFIG/magnus.conf** 文件的结尾，根据您的配置修改文件路径。这些行定义 **nsapi_redirector.so** 模块及 **workers.properties** 文件的位置（这个文件列出了工作节点及它们的属性）

```
Init fn="load-modules" funcs="jk_init,jk_service"
shlib="/lib/nsapi_redirector.so" shlib_flags="(global|now)"

Init fn="jk_init"
worker_file="IPLANET_CONFIG/connectors/workers.properties"
log_level="info" log_file="IPLANET_CONFIG/connectors/nsapi.log"
shm_file="IPLANET_CONFIG/connectors/tmp/jk_shm"
```

上面的配置适用于 32 位架构。如果您使用 64 位 Solaris，请将 **lib/nsapi_redirector.so** 改为 **lib64/nsapi_redirector.so**。

保存文件并退出。

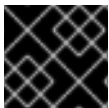
3. 配置 NSAPI Connector。

您可以对 NSAPI Connector 进行基本的配置，没有负载平衡或相关配置。在配置完成前选择下列选项之一。

- [配置 NSAPI Connector 发送客户请求至 JBoss EAP](#)
- [配置 NSAPI Connector 在多个 JBoss EAP 服务器间平衡客户请求](#)

22.10.2. 配置 NSAPI Connector 发送客户请求至 JBoss EAP

这个任务配置 NSAP Connector 重定向客户请求至没有负载平衡失效切换的 JBoss EAP 服务器。重定向是对每个部署（每个 URL）进行的。



重要

在继续这个任务前，您必须已[配置了 NSAPI Connector](#)。

设置基本的 HTTP Connector

1. 定义 URL 路径重定向至 JBoss EAP 服务器。



注意

在 **IPLANET_CONFIG/obj.conf** 里，每行开头是不允许空格的，除非这一行是上一行的继续。

编辑 **IPLANET_CONFIG/obj.conf** 文件。找到以 `<Object name="default">` 开始的部分，然后按照下例展示的格式添加要匹配的每个 URL。字符串 *jknsapi* 指的是下一步里要定义的 HTTP Connector。这个例子展示了用于模式匹配的通配符用法。

```
<Object name="default">
[... ]
NameTrans fn="assign-name" from="/status" name="jknsapi"
NameTrans fn="assign-name" from="/images(|/*)" name="jknsapi"
NameTrans fn="assign-name" from="/css(|/*)" name="jknsapi"
NameTrans fn="assign-name" from="/nc(|/*)" name="jknsapi"
NameTrans fn="assign-name" from="/jmx-console(|/*)" name="jknsapi"
</Object>
```

2. 定义提供每个路径的工作节点。

继续编辑 **IPLANET_CONFIG/obj.conf** 文件。直接在您已完成编辑的 `</Object>` 的结束标签后添加下列内容。

```
<Object name="jknsapi">
ObjectType fn=force-type type=text/plain
Service fn="jk_service" worker="worker01" path="/status"
Service fn="jk_service" worker="worker02" path="/nc(/*)"
Service fn="jk_service" worker="worker01"
</Object>
```

上面的例子将到 URL 路径 `/status` 的请求重定向到工作节点 *worker01*，将到 `/nc/` 下的所有 URL 路径的请求重定向到工作节点 *worker02*。第三行指定，所有与前面条件不匹配的、分配给 *jknsapi* 对象的 URL 都由 *worker01* 处理。

保存文件并退出。

3. 定义工作节点及其属性。

在 **IPLANET_CONFIG/connectors/** 目录里创建一个名为 **workers.properties** 的文件。粘贴下列内容到这个文件并按照需要进行修改。

```
# An entry that lists all the workers defined
worker.list=worker01, worker02

# Entries that define the host and port associated with these
workers
worker.worker01.host=127.0.0.1
worker.worker01.port=8009
worker.worker01.type=ajp13

worker.worker02.host=127.0.0.100
worker.worker02.port=8009
worker.worker02.type=ajp13
```

workers.properties 文件使用和 Apache `mod_jk` 相同的语法。

保存文件并退出。

4. 重启 iPlanet Web Server

执行下列命令来重启 iPlanet Web Server。

```
IPLANET_CONFIG/../../bin/stopserv
IPLANET_CONFIG/../../bin/startserv
```

现在，iPlanet Web Server 会把客户请求发送到您所配置的、部署在 JBoss EAP 上的 URL。

22.10.3. 配置 NSAPI Connector 在多个 JBoss EAP 服务器间平衡客户请求

这个任务配置 NSAPI Connector 发送客户请求至负载均衡配置里的 JBoss EAP 服务器。



重要

在继续这个任务前，您必须已[配置了 NSAPI Connector](#)。

配置连接器的负载均衡

1. 定义 URL 路径重定向至 JBoss EAP 服务器。



注意

在 **IPLANET_CONFIG/obj.conf** 里，每行开头是不允许空格的，除非这一行是上一行的继续。

编辑 **IPLANET_CONFIG/obj.conf** 文件。找到以 **<Object name="default">** 开始的部分，然后按照下例展示的格式添加要匹配的每个 URL。字符串 **jknsapi** 指的是下一步里要定义的 HTTP Connector。这个例子展示了用于模式匹配的通配符用法。

```
<Object name="default">
[...
NameTrans fn="assign-name" from="/status" name="jknsapi"
NameTrans fn="assign-name" from="/images(|/*)" name="jknsapi"
NameTrans fn="assign-name" from="/css(|/*)" name="jknsapi"
NameTrans fn="assign-name" from="/nc(|/*)" name="jknsapi"
NameTrans fn="assign-name" from="/jmx-console(|/*)" name="jknsapi"
NameTrans fn="assign-name" from="/jkmanager/*" name="jknsapi"
</Object>
```

2. 定义提供每个路径的工作节点。

继续编辑 **IPLANET_CONFIG/obj.conf** 文件。直接在上一步骤里修改的部分 (**</Object>**) 后面添加下列内容并按照需要进行修改：

```
<Object name="jknsapi">
ObjectType fn=force-type type=text/plain
Service fn="jk_service" worker="status" path="/jkmanager(/*)"
Service fn="jk_service" worker="router"
</Object>
```

这个 **jknsapi** 对象定义了用于提供每个映射至 **default** 对象里的 **name="jknsapi"** 的路径的工作节点。除了匹配 **/jkmanager/*** 的 URL 的所有内容都将重定向至名为 **router** 的工作节点。

3. 定义工作节点及其属性。

在 **IPLANET_CONFIG/connector/** 目录里创建一个名为 **workers.properties** 的文件。粘贴下列内容到这个文件并按照需要进行修改。

```
# The advanced router LB worker
# A list of each worker
worker.list=router,status

# First JBoss EAP server
# (worker node) definition.
# Port 8009 is the standard port for AJP
#

worker.worker01.port=8009
worker.worker01.host=127.0.0.1
worker.worker01.type=ajp13
worker.worker01.ping_mode=A
worker.worker01.socket_timeout=10
worker.worker01.lbfactor=3

# Second JBoss EAP server
worker.worker02.port=8009
worker.worker02.host=127.0.0.100
worker.worker02.type=ajp13
worker.worker02.ping_mode=A
worker.worker02.socket_timeout=10
worker.worker02.lbfactor=1

# Define the load-balancer called "router"
worker.router.type=lb
worker.router.balance_workers=worker01,worker02

# Define the status worker
worker.status.type=status
```

workers.properties 文件使用和 Apache mod_jk 相同的语法。

保存文件并退出。

4. 重启 iPlanet Web Server 7.0。

```
IPLANET_CONFIG/../../bin/stopserv
IPLANET_CONFIG/../../bin/startserv
```

iPlanet Web Server 重定向您配置的 URL 模式至负载均衡配置里的 JBoss EAP 服务器。

附录 A. 参考资料

A.1. 服务器运行时参数

应用程序服务器启动脚本在启动时可接受参数和选项。这允许服务器使用 `standalone.xml`、`domain.xml` 或 `host.xml` 配置文件之外的其他配置启动。

其他配置可能包括用其他套接字绑定集或次级配置来启动服务器。

可用的参数列表可以在启动时使用 `-h` 或 `--help` 选项来获取。

表 A.1. 运行时选项和参数

参数或选项	操作模式	描述
<code>--admin-only</code>	Standalone	设置服务器的运行类型为 ADMIN_ONLY 。这将导致它打开管理接口并接受管理请求，但不会启动其他运行时服务或接受最终用户请求。
<code>--admin-only</code>	Domain	设置主机控制器的运行类型为 ADMIN_ONLY ，这将导致它打开管理接口并接受管理请求，但不会启动服务器。如果这个主机控制器是域里的主控制器，它会从主机控制器接受转入连接。
<code>-b=<value></code> , <code>-b <value></code>	Standalone, Domain	设置系统属性 jboss.bind.address ，它用于配置公共接口的绑定地址。如果没有指定，默认值是 127.0.0.1 。关于设置其他接口的绑定地址，请参考 <code>-b<interface>=<value></code> 条目。
<code>-b<interface>=<value></code>	Standalone, Domain	设置系统属性 jboss.bind.address.<interface> 为给定的值。例如， <code>-bmanagement=IP_ADDRESS</code> 。
<code>--backup</code>	Domain	保留持久性域配置的备份，即使这个主机不是域控制器。
<code>-c=<config></code> , <code>-c <config></code>	Standalone	要使用的服务器配置文件的名称。默认值是 standalone.xml 。
<code>-c=<config></code> , <code>-c <config></code>	Domain	要使用的服务器配置文件的名称。默认值是 domain.xml 。
<code>--cached-dc</code>	Domain	如果主机不是域控制器且无法在引导时联系域控制器，则使用域配置的本地缓存备份进行引导。
<code>--debug [<port>]</code>	Standalone	用可选参数指定端口来激活调试模式。只有启动脚本支持它时才能使用。
<code>-D<name>[=<value>]</code>	Standalone, Domain	设置系统属性。

参数或选项	操作模式	描述
--domain-config=<config>	Domain	要使用的服务器配置文件的名称。默认值是 domain.xml 。
-h, --help	Standalone, Domain	显示帮助信息并退出。
--host-config=<config>	Domain	要使用的主机配置文件的名称。默认值是 host.xml 。
--interprocess-hc-address=<address>	Domain	主机控制器应该侦听进程控制器通讯的地址。
--interprocess-hc-port=<port>	Domain	主机控制器应该侦听进程控制器通讯的端口。
--master-address=<address>	Domain	设置系统属性 jboss.domain.master.address 为指定的值。在默认的从主机控制器配置上，它用于配置主主机控制器的地址。
--master-port=<port>	Domain	设置系统属性 jboss.domain.master.port 为指定的值。在默认的从主机控制器配置上，它用于配置主主机控制器的原生管理通讯的端口。
--read-only-server-config=<config>	Standalone	要使用的服务器配置的名称。这与 --server-config 和 -c 不同，原始文件不会被覆盖。
--read-only-domain-config=<config>	Domain	要使用的域配置的名称。这与 --domain-config 和 -c 不同，初始文件永不会被覆盖。
--read-only-host-config=<config>	Domain	要使用的主机配置文件的名称。这与 --host-config 不同，初始文件永不会被覆盖。
-P=<url>, -P <url>, --properties=<url>	Standalone, Domain	从给定 URL 加载系统属性。
--pc-address=<address>	Domain	进程控制器用来侦听它所控制的进程通讯的地址。
--pc-port=<port>	Domain	进程控制器侦听它控制的进程的通讯的端口。
-S<name>[=<value>]	Standalone	设置安全属性。
-secmgr	Standalone, Domain	运行安装了安全管理者的服务器。
--server-config=<config>	Standalone	要使用的服务器配置文件的名称。默认值是 standalone.xml 。

参数或选项	操作模式	描述
<code>-u=<value>, -u <value></code>	Standalone, Domain	设置系统属性 <code>jboss.default.multicast.address</code> ，它用于配置配置文件里的 <code>socket-binding</code> 元素里多点传送地址。如果未指定，它的默认值是 <code>230.0.0.4</code> 。
<code>-v, -V, --version</code>	Standalone, Domain	显示应用程序服务版本并退出。



警告

JBoss EAP 附带的配置文件被用来处理选项的行为（如 **`-b`**、**`-u`**）。如果您把配置文件修改为不再使用受这些选项控制的系统属性，那么在启动命令里添加它就不会起作用。

A.2. RPM 服务配置文件

相比 ZIP 或安装程序安装方式，JBoss EAP 的 RPM 安装包含两个额外的配置文件。服务启动脚本使用这些文件来指定 JBoss EAP 启动环境。对于 Red Hat Enterprise Linux 6 和 Red Hat Enterprise Linux 7，这些服务配置文件的位置是不同的。

表 A.2. 用于 Red Hat Enterprise Linux 6 的 RPM 配置文件

File	描述
<code>/etc/sysconfig/eap7-standalone</code>	Red Hat Enterprise Linux 6 上的独立 JBoss EAP 服务器专有的设置。
<code>/etc/sysconfig/eap7-domain</code>	Red Hat Enterprise Linux 6 上作为受管域运行的 JBoss EAP 服务器专有的设置。

表 A.3. 用于 Red Hat Enterprise Linux 7 的 RPM 配置文件

File	描述
<code>/etc/opt/rh/eap7/wildfly/eap7-standalone.conf</code>	Red Hat Enterprise Linux 7 上的独立 JBoss EAP 服务器专有的设置。
<code>/etc/opt/rh/eap7/wildfly/eap7-domain.conf</code>	Red Hat Enterprise Linux 7 上作为受管域运行的 JBoss EAP 服务器专有的设置。

A.3. RPM 服务配置文件属性

下表展示了 JBoss EAP RPM 服务的可用配置属性列表及其默认值。



注意

如果某个属性在 RPM 服务配置文件（如 `/etc/sysconfig/eap7-standalone`）和 JBoss EAP 启动配置文件（如 `EAP_HOME/bin/standalone.conf`）都有相同的名称，那么 JBoss EAP 启动配置文件里的值将优先使用。**JAVA_HOME** 就是这类属性中的一个。

表 A.4. RPM 服务配置文件属性

属性	描述
JAVA_HOME	Java Runtime Environment 的安装目录。 默认值： /usr/lib/jvm/jre
JAVAPATH	Java 可执行文件的安装路径。 默认值： \$JAVA_HOME/bin
WILDFLY_STARTUP_WAIT	在接收启动或重启命令后，init 脚本在确认服务器已启动成功前等待的时间（秒）。这个属性只适用于 Red Hat Enterprise Linux 6。 默认值： 60
WILDFLY_SHUTDOWN_WAIT	在接收停止或重启命令时，init 脚本在等待服务器关闭前等待的时间（秒）。这个属性只适用于 Red Hat Enterprise Linux 6。 默认值： 20
WILDFLY_CONSOLE_LOG	CONSOLE 日志处理程序将被重定向的文件。 对于独立服务器，默认值是 /var/opt/rh/eap7/log/wildfly/standalone/console.log ；对于受管域，默认值是 /var/opt/rh/eap7/log/wildfly/domain/console.log 。
WILDFLY_SH	用于启动 JBoss EAP 服务器的脚本。 对于独立服务器，默认值是 /opt/rh/eap7/root/usr/share/wildfly/bin/standalone.sh ；对于受管域，默认值是 /opt/rh/eap7/root/usr/share/wildfly/bin/domain.sh 。
WILDFLY_SERVER_CONFIG	要使用的服务器配置文件。 这个属性没有默认值。在启动时可以定义为 standalone.xml 或 domain.xml 。
WILDFLY_HOST_CONFIG	对于受管域，这个属性允许用户指定主机配置文件（如 host.xml ）。它没有默认值。
WILDFLY_MODULEPATH	JBoss EAP 模块目录的路径。 默认值： /opt/rh/eap7/root/usr/share/wildfly/modules

属性	描述
WILDFLY_BIND	设置系统属性 <code>jboss.bind.address</code> ，它用来配置公共接口的绑定地址。如果未指定，它的默认值是 <code>0.0.0.0</code> 。

A.4. JBOSS EAP 子系统概述

下表包括了 JBoss EAP 子系统的概述信息。

表 A.5. JBoss EAP 子系统

JBoss EAP 子系统	描述
batch-jberet	配置环境来运行批处理应用程序和管理批处理任务。
bean-validation	为验证 Java 对象数据来配置 bean 验证。
datasources	创建并配置数据源，管理 JDBC 数据库驱动。
deployment-scanner	配置部署扫描器来监测部署应用程序的特定位置。
ee	在 Java EE 平台中配置常规功能，如定义全局模块、启用基于描述符的属性替换、配置默认绑定。
ejb3	配置 Enterprise JavaBeans (EJBs)，包括会话和 message-driven bean。 ejb3 子系统的详细信息包括在 JBoss EAP 的 <i>Developing EJB Applications</i> 中。
iiop-openjdk	为 JTS 事务和其它 ORB 服务（包括安全性）配置 CORBA（Common Object Request Broker Architecture）。在 JBoss EAP 6 中，这个功能包括在 jacorb 子系统中。
infinispan	为 JBoss EAP 高可用性服务配置缓存功能。
io	定义被其它子系统使用的 worker 和缓冲池。
jaxrs	启用 JAX-RS 应用程序的部署，以及相关功能。
jca	为 JCA（Java EE Connector Architecture）容器和资源适配器配置常规设置。
jdr	启用用来帮助进行故障排除的诊断数据收集功能。JBoss EAP 的订阅用户在向红帽请求支持时，可以提供这些信息。
jgroups	配置协议堆栈，以及集群中的服务器间使用的通讯机制。
jmx	配置 JMX（Java Management Extensions）远程访问。

JBoss EAP 子系统	描述
jpa	<p>管理 JPA（Java Persistence API）2.1 容器管理的需要，并允许您部署持久性单元定义、注解和描述符。</p> <p>JBoss EAP 开发指南中包括了更多与 jpa 子系统相关的信息。</p>
jsf	管理 JSF（JavaServer Faces）的实现。
jsr77	提供 JSR-77 规格中定义的 Java EE 管理功能。
logging	通过 日志目录（log category） 和 日志 handler 配置系统级和应用程序级的日志功能。
mail	通过配置 mail 服务器属性 和 自定义 mail 传输 创建一个邮件服务，在 JBoss EAP 中部署的应用程序可以通过这个服务发送邮件。
messaging-activemq	<p>配置 JMS 目的地、连接工厂、Artemis 的其它配置，以及集成的消息厂商。在 JBoss EAP 6 中，消息功能包括在 messaging 子系统中。</p> <p>为 JBoss EAP 配置消息服务中包括了更多与 messaging-activemq 子系统相关的信息。</p>
modcluster	配置服务器方的 mod_cluster worker 节点 。
naming	把项绑定到全局 JNDI 命名空间，并配置远程 JNDI 接口。
picketlink-federation	<p>配置 PicketLink 基于 SAML 的单点登录功能（SSO）。</p> <p>JBoss EAP 如何使用 SAML v2 设置 SSO 中包括了更多与 picketlink-federation 子系统相关的信息。</p>
picketlink-identity-management	配置 PicketLink 身份管理服务。这个子系统不被支持。
pojo	启用部署包括被以前 JBoss EAP 版本支持的 JBoss Microcontainer 服务的应用程序。
remoting	为本地和 远程服务 配置入站（inbound）和出站（outbound）连接。
request-controller	为 安全挂起和关闭服务器 进行配置。
resource-adapters	配置和维护 资源适配器（resource adapter） 用来使用 JCA（Java Connector Architecture）规格进行 Java EE 应用程序和 EIS（Enterprise Information System）间的通讯。
rts	不被支持的 REST-AT 的实现。
sar	启用部署包括被以前 JBoss EAP 版本支持的 MBean 服务的 SAR 归档。

JBoss EAP 子系统	描述
安全	配置应用程序安全设置。 JBoss EAP 安全架构 中包括了与 security 子系统相关的详细信息。
security-manager	配置 Java Security Manager 使用的 Java 安全策略。 JBoss EAP 如何配置服务器安全 包括了更多与 security-manager 子系统相关的信息。
singleton	定义单例（singleton）策略以配置单例部署的行为，或创建单例 MSC 服务的行为。 JBoss EAP 开发指南 中包括了更多与 singleton 子系统相关的信息。
事务	配置 事务管理器选项 ，如超时值、事务日志，以及是否使用 JTS（Java Transaction Service）。
undertow	配置 JBoss EAP 的 web 服务器 和 servlet 容器设置。在 JBoss EAP 6 中，这个功能包括在 web 子系统中那。
webservices	为 web 服务厂商配置公布的端点地址和端点 handler 链、主机名、端口和 WSDL 地址。 JBoss EAP 开发 Web 服务应用程序 中包括了更多与 webservices 子系统相关的信息。
weld	为 JBoss EAP 配置 CDI（Contexts and Dependency Injection）。
xts	为在一个事务中协调 web 服务配置设置。

A.5. ADD-USER 工具参数

下表描述了 **add-user.sh** 或 **add-user.bat**（这是添加可立即验证的新用户至属性文件的工具）的可用参数。

表 A.6. Add-User 命令参数

命令行参数	描述
-a	在应用程序域（realm）里创建用户。如果没有使用，默认是在管理域里创建一个用户。
-dc <value>	包含属性文件的域配置目录。如果没有使用，默认目录是 EAP_HOME/domain/configuration/ 。
-sc <value>	包含属性文件的替代独立服务器配置目录。如果没有使用，默认目录是 EAP_HOME/standalone/configuration/ 。

命令行参数	描述
-up, --user-properties <value>	替代的用户属性文件的名称。它可以是绝对路径，也可以是和指定替代配置目录的 -sc 或 -dc 参数联合使用的文件名称。
-g, --group <value>	分配给这个用户的一个用逗号隔开的组的列表。
-gp, --group-properties <value>	替代的组属性文件的名称。它可以是绝对路径，也可以是和指定替代配置目录的 -sc 或 -dc 参数联合使用的文件名称。
-p, --password <value>	用户的密码。
-u, --user <value>	用户名称。有效字符是字母、数字和下列字符：, ./=@\.
-r, --realm <value>	用户保护管理接口的区名。如果忽略，默认值是 ManagementRealm 。
-s, --silent	不带控制台输出地运行 add-user 脚本。
-e, --enable	启用用户。
-d, --disable	禁用用户。
-cw, --confirm-warning	在交互模式里自动确认警告。
-h, --help	显示 add-user 脚本的使用信息。

A.6. 管理审计日志属性

表 A.7. Logger 属性

属性	描述
enabled	是否启用审计日志。
log-boot	服务器引导时是否应该为操作记录日志。
log-read-only	是否应该为没有修改配置的操作或任何运行时服务记录日志。

表 A.8. 日志格式器属性

属性	描述
compact	如果为 true ，它将在一行里产生 JSON 数据。因为仍有一些值可能会包含换行符，所以如果将整个记录放在一行里很重要，则需要把 escape-new-line 或 escape-control-characters 设置为 true 。

属性	描述
date-format	<code>java.text.SimpleDateFormat</code> 可理解的日期格式。如果 include-date 为 false ，这将被忽略。
date-separator	日期和格式化的日志消息的剩余部分间的分割符。如果 include-date 为 false ，这将被忽略。
escape-control-characters	如果为 true ，它将转义所有的控制字符（十进制值大于 32 的 ASCII 条目）。例如，换行符转换为 #012 。如果为 true ，它将覆盖 escape-new-line=false 。
escape-new-line	如果为 true 它，它将用八进制 ASCII 代码 #012 转义换行符。
include-date	是否在格式化的日志记录里包含日期。

表 A.9. 文件处理程序属性

属性	描述
disabled-due-to-failure	这个处理程序是否由于失败的日志记录而已被禁用（只读属性）。
failure-count	在处理程序初始化后，记录日志失败的次数（只读属性）。
formatter	用于格式化日志消息的 JSON 格式器。
max-failure-count	在禁用这个日志处理程序前记录日志失败的最多次数。
路径	审计日志文件的路径。
relative-to	一个之前命名的路径或系统提供的标准路径之一的名称。如果提供了 relative-to, path 属性将作为这个属性指定的相对路径处理。

表 A.10. Syslog 处理程序属性

属性	描述
app-name	如 <i>RFC-5424</i> 的 6.2.5 章节定义的，添加至 syslog 记录的应用程序名称。如果未指定，它默认是产品的名称。
disabled-due-to-failure	这个处理程序是否由于失败的日志记录而已被禁用（只读属性）。
facility	如 <i>RFC-5424</i> 的 6.2.1 章节和 <i>RFC-3164</i> 的 4.1.1 章节所定义的用于 syslog 日志的工具。
failure-count	在处理程序初始化后，记录日志失败的次数（只读属性）。

属性	描述
formatter	用于格式化日志消息的 JSON 格式器。
max-failure-count	在禁用这个日志处理程序前记录日志失败的最多次数。
max-length	所允许的日志消息（包括头部）的最大长度（字节）。如果未定义，当 syslog-format 为 RFC3164 时，它的默认值为 1024 字节；当 syslog-format 为 RFC5424 时，它的默认值为 2048 字节。
protocol	用于 syslog 处理程序的协议。必须是 udp 、 tcp 或 tls 之一。
syslog-format	syslog 格式： RFC5424 或 RFC3164 。
truncate	如果消息（包括头部信息）的长度（字节数）大于 max-length 的值，是否应该截断消息。如果为 false ，消息将被分段，并用相同的头信息进行发送。



注意

Syslog 服务器的实现各有不同，所以不是所有的设置都适用于所有的 Syslog 服务器。我们已使用 *rsyslog* syslog 进行了测试。

这个表只列出高级别的属性。每个属性都有配置参数，一些还有子配置参数。

A.7. 接口属性

表 A.11. 接口属性和值

接口元素	描述
any	指定接口的选择标准的一部分应该满足至少一个（但不需要全部）嵌套标准集。
any-address	空元素表示使用这个接口的套接字应该绑定至通配符地址。除非将 java.net.preferIPv4Stack 系统属性设置为 true ，IPv6 通配符地址（ ::: ）将被使用，此时使用的 IPv4 通配符地址是 0.0.0.0 。如果套接字绑定至双栈主机上的 IPv6 anylocal 地址，它可以接受 IPv6 以及 IPv4 通讯；如果套接字绑定至 IPv4 (IPv4-mapped) anylocal 地址，那它只接受 IPv4 通讯。
inet-address	IPv6 或 IPv4 用句点分隔的 IP 地址，或是可解析为 IP 地址的主机名。
link-local-address	指定接口的选择标准的一部分是否应该是与 link-local 关联的地址。
loopback	指定接口的选择标准的一部分是否应该是 loopback 接口。

接口元素	描述
loopback-address	可能不会在主机的 loopback 接口实际配置的 loopback 地址。它与 inet-address 类型不一样，即使未找到关联 IP 地址的 NIC，给定的值都将被使用。
multicast	指定接口的选择标准的一部分是否应该支持多点传送。
nic	网络接口的名称（如 eth0、eth1、lo）。
nic-match	表示主机上可用的网络接口的常规表达式，匹配它来寻找可接受的接口。
not	指定接口的选择标准的一部分不匹配任何嵌套的标准集。
point-to-point	指定接口的选择标准的一部分是否应该是 point-to-point 接口。
public-address	指定接口的选择标准的一部分是否应该具有可公共路由的地址。
site-local-address	指定接口的选择标准的一部分是否应该是与 site-local 关联的地址。
subnet-match	网络 IP 地址，以及地址的网络前缀的位数（以斜杠格式表示，如：192.168.0.0/16）。
up	指定接口的选择标准的一部分当前是否应该上线。
virtual	指定接口的选择标准的一部分是否应该是虚拟接口。

A.8. 套接字绑定属性

表 A.12. 套接字绑定属性

属性	描述
client-mappings	指定这个套接字绑定的客户映射。连接至这个套接字的客户应该使用匹配其转出接口的映射里的目的地地址。这允许使用网络地址转换的高级网络拓扑结构，或者在多个网络接口上绑定。每个映射应该以声明的顺序进行求值，第一个成功的匹配用来确定目的地。
fixed-port	端口值是否应该保持固定，即使已应用数字偏移量至套接字组里的其他套接字。
interface	套接字应该绑定的接口的名称，或者对于多点传送套接字来说，它应该侦听的接口。这应该是声明的接口中的一个。如果未定义，套接字绑定组里的 default-interface 属性将被使用。

属性	描述
multicast-address	套接字应该接收多点传送通讯的多点传送地址。如果未指定，套接字将不会接收多点传送。
multicast-port	套接字应该接收多点传送通讯的多点传送端口，如果配置了 'multicast-address' 则必须配置它。
name	套接字的名称。需要访问套接字配置信息的服务将用这个名称进行查找。这个属性是必需的。
port	套接字应该绑定的端口号码。请注意如果服务器对所有端口应用了端口偏移量进行增和减，这个值可被覆盖。

A.9. 默认的套接字绑定

表 A.13. 默认的套接字绑定

名称	端口	多点传送 端口	描述	套接字绑定组
ajp	8009		Apache JServ 协议。用于 HTTP 集群和负载平衡。	standard-sockets、ha-sockets、full-sockets、full-ha-sockets
http	8080		部署的 Web 应用程序的默认端口。	standard-sockets、ha-sockets、full-sockets、full-ha-sockets
https	8443		部署的 Web 应用程序和客户间的用 SSL 加密的连接。	standard-sockets、ha-sockets、full-sockets、full-ha-sockets
iiop	3528		用于 JTS 事务和依赖 ORB 的服务的 CORBA 服务。	full-sockets, full-ha-sockets
iiop-ssl	3529		SSL 加密的 CORBA 服务。	full-sockets, full-ha-sockets
jgroups-mping		45700	多点传送。用于发现 HA 集群里的初始成员资格。	ha-sockets, full-ha-sockets
jgroups-tcp	7600		HA 集群里使用 TCP 的单播端点发现。	ha-sockets, full-ha-sockets

名称	端口	多点传送 端口	描述	套接字绑定组
jgroups-tcp-fd	57600		用于 TCP 上的 HA 故障检测。	ha-sockets, full-ha-sockets
jgroups-udp	55200	45688	HA 集群里使用 UDP 的多点传送端点发现。	ha-sockets, full-ha-sockets
jgroups-udp-fd	54200		用于 UDP 上的 HA 故障检测。	ha-sockets, full-ha-sockets
management-http	9990		用于管理层的 HTTP 通讯。	standard-sockets、ha-sockets、full-sockets、full-ha-sockets
management-https	9993		用于管理层的 HTTPS 通讯。	standard-sockets、ha-sockets、full-sockets、full-ha-sockets
modcluster		23364	用于 JBoss EAP 和 HTTP 负载均衡器间的多点传送端口。	ha-sockets, full-ha-sockets
txn-recovery-environment	4712		JTA 事务恢复管理者。	standard-sockets、ha-sockets、full-sockets、full-ha-sockets
txn-status-manager	4713		JTA / JTS 事务管理者。	standard-sockets、ha-sockets、full-sockets、full-ha-sockets

A.10. 部署扫描器的 MARKER 文件

部署扫描器使用 Marker 文件来标记 JBoss EAP 服务器实例的 deployment 目录里的应用程序的状态。Marker 文件和部署的名称相同，其后缀表示应用程序部署的状态。

例如，成功的 **test-application.war** 部署会有名为 **test-application.war.deployed** 的 marker 文件。

下表列出了可用的 marker 文件类型及含义。

表 A.14. Marker 文件类型

文件名称后缀	来源	描述
.deployed	系统生成的	指示内容已被部署。如果删除这个文件，内容将被卸载。

文件名称后缀	来源	描述
.dodeploy	用户生成的	表示内容应该部署还是重新部署。
.failed	系统生成的	表示部署失败。Marker 文件包含和失败原因相关的信息。如果删除这个 marker 文件，内容将可以再次自动部署。
.isdeploying	系统生成的	表示部署正在进行。完成后这个 marker 文件将被删除。
.isundeploying	系统生成的	由删除 .deployed 文件触发，这表示内容正被卸载。完成后这个 marker 文件将被删除。
.pending	系统生成的	表示部署扫描器承认部署内容的需要，但目前有一个问题阻止了自动部署（例如内容正在复制）。这个 marker 文件充当全局的部署障碍，表示当这个 marker 文件存在时，扫描器不会指示服务器部署或卸载任何内容。
.skipdeploy	用户生成的	禁用应用程序的自动部署。它作为阻止展开内容的自动部署的临时方法，可以避免推送正在编辑的不完整的内容。它也可以用于压缩的内容，但扫描器会检测正在对压缩内容的修改并等待其完成。
.undeployed	系统生成的	表示内容已被卸载。删除这个 marker 文件不会影响内容的重新部署。

A.11. 部署扫描器属性

部署扫描器包含下列可配置属性。

表 A.15. 部署扫描器属性

名称	Default	描述
auto-deploy-exploded	false	允许展开的内容自动部署而无需 .dodeploy marker 文件。只推荐用于基本的部署场景，在开发人员或操作系统进行修改时阻止展开的应用程序部署。
auto-deploy-xml	true	允许 XML 内容自动部署而无需 .dodeploy marker 文件。
auto-deploy-zipped	true	允许压缩的内容自动部署而无需 .dodeploy marker 文件。
deployment-timeout	600	部署扫描器在取消前允许尝试部署的时间（以秒为单位）。

名称	Default	描述
路径	deployments	要扫描的实际文件系统路径。它被当作绝对路径处理，除非指定了 relative-to 属性。
relative-to	jboss.server.base.dir	到一个文件系统的路径由服务器配置中的 path 设置。
runtime-failure-causes-rollback	false	部署的运行时失败是否导致部署及作为扫描操作一部分的其他所有部署（可能并不相关）的回滚。
scan-enabled	true	允许以 scan-interval 间隔在启动时自动扫描应用程序。
scan-interval	5000	应该扫描仓库的变动的的时间间隔（毫秒）。小于 1 的值会导致扫描只在初始启动时进行扫描。

A.12. ROOT LOGGER 属性

表 A.16. Root Logger 属性

属性	描述
filter	定义简单的过滤器类型。 <i>已弃用，转而使用 filter-spec。</i>
filter-spec	定义过滤器的表达式。下面的表达式定义了排除不匹配某个模式的日志条目的过滤器： not(match("WFLY.*"))
handlers	Root Logger 使用的日志处理程序列表。
level	Root Logger 记录的日志消息的最低级别。



注意

其他处理程序不会继承为 Root Logger 指定的 **filter-spec**。相反您必须为每个处理程序指定一个 **filter-spec**。

A.13. 日志类别属性

表 A.17. 日志类别属性

属性	描述
category	捕获的日志消息所在的日志类别。
filter	定义简单的过滤器类型。 <i>已弃用，转而使用 filter-spec。</i>

属性	描述
filter-spec	定义过滤器的表达式。下面的表达式定义了不匹配某个模式的过滤器： <code>not(match("WFLY.*"))</code>
handlers	和 Logger 相关联的日志处理程序列表。
level	日志类别记录的日志消息的最低级别。
use-parent-handlers	如果设置为 true ，除了其他分配的处理程序，这个类别将使用 Root Logger 的日志处理程序。

A.14. 日志处理程序属性

表 A.18. 控制台日志处理程序属性

属性	描述
autoflush	如果设置为 true ，日志消息将被发送至处理程序接收时立即分配的文件。
enabled	如果设置为 true ，处理程序被启用且正常工作。如果设置为 false ，处理日志消息时处理程序将被忽略。
encoding	用于输出的字符编码模式。
filter	定义简单的过滤器类型。 <i>已弃用，转而使用 filter-spec。</i>
filter-spec	定义过滤器的表达式。下面的表达式定义了不匹配某个模式的过滤器： <code>not(match("WFLY.*"))</code>
formatter	这个日志处理程序使用的日志格式器。
level	日志处理程序记录的日志消息的最低级别。
name	日志处理程序的名称。 <i>已弃用（处理程序的地址里已包含这个名称）</i>
named-formatter	处理程序使用的格式器的名称。
target	日志处理程序输出所在的系统输出流。对于系统错误流和标准输出流，它分别是 System.err 和 System.out。

表 A.19. 文件日志处理程序属性

属性	描述
----	----

属性	描述
append	如果设置为 true ，这个处理程序写入的所有消息都将附加到这个文件（如果存在）。如果设置为 false ，每次应用服务器启动时都会创建一个新的文件。
autoflush	如果设置为 true ，日志消息将被发送至处理程序接收时立即分配的文件。
enabled	如果设置为 true ，处理程序被启用且正常工作。如果设置为 false ，处理日志消息时处理程序将被忽略。
encoding	用于输出的字符编码模式。
file	代表这个日志处理程序输出写入的文件的对象。它有两个配置属性： relative-to 和 path 。
filter	定义简单的过滤器类型。 <i>已弃用，转而使用 filter-spec。</i>
filter-spec	定义过滤器的表达式。下面的表达式定义了不匹配某个模式的过滤器： not(match("WFLY.*"))
formatter	这个日志处理程序使用的日志格式器。
level	日志处理程序记录的日志消息的最低级别。
name	日志处理程序的名称。 <i>已弃用（处理程序的地址里已包含这个名称）</i>
named-formatter	处理程序使用的格式器的名称。

表 A.20. 定期日志处理程序属性

属性	描述
append	如果设置为 true ，这个处理程序写入的所有消息都将附加到这个文件（如果存在）。如果设置为 false ，每次应用服务器启动时都会创建一个新的文件。
autoflush	如果设置为 true ，日志消息将被发送至处理程序接收时立即分配的文件。
enabled	如果设置为 true ，处理程序被启用且正常工作。如果设置为 false ，处理日志消息时处理程序将被忽略。
encoding	用于输出的字符编码模式。
file	代表这个日志处理程序输出写入的文件的对象。它有两个配置属性： relative-to 和 path 。
filter	定义简单的过滤器类型。 <i>已弃用，转而使用 filter-spec。</i>

属性	描述
filter-spec	定义过滤器的表达式。下面的表达式定义了不匹配某个模式的过滤器： <code>not(match("WFLY.*"))</code> 。
formatter	这个日志处理程序使用的日志格式器。
level	日志处理程序记录的日志消息的最低级别。
name	日志处理程序的名称。 <i>已弃用（处理程序的地址里已包含这个名称）</i>
named-formatter	处理程序使用的格式器的名称。
suffix	这个字符串包含在轮换日志的后缀里。 suffix 的格式是句点（.）后面跟着 SimpleDateFormat 可以解析的日期字符串。

表 A.21. Size 日志处理程序属性

属性	描述
append	如果设置为 true ，这个处理程序写入的所有消息都将附加到这个文件（如果存在）。如果设置为 false ，每次应用服务器启动时都会创建一个新的文件。
autoflush	如果设置为 true ，日志消息将被发送至处理程序接收时立即分配的文件。
enabled	如果设置为 true ，处理程序被启用且正常工作。如果设置为 false ，处理日志消息时处理程序将被忽略。
encoding	用于输出的字符编码模式。
file	代表这个日志处理程序输出写入的文件的对象。它有两个配置属性： relative-to 和 path 。
filter	定义简单的过滤器类型。 <i>已弃用，转而使用 filter-spec。</i>
filter-spec	定义过滤器的表达式。下面的表达式定义了不匹配某个模式的过滤器： <code>not(match("WFLY.*"))</code>
formatter	这个日志处理程序使用的日志格式器。
level	日志处理程序记录的日志消息的最低级别。

属性	描述
max-backup-index	<p>轮换日志文件保留的最大数目。当到达这个数目时，最旧的日志文件将被重用。默认为 1。</p> <p>如果使用了 suffix 属性，轮换日志文件的后缀将包含在轮算法里。当轮换日志文件时，以 name+suffix 开始的最旧的文件将被删除，剩余的轮换日志文件的数字型后缀都会增加，而最近轮换的日志文件的后缀将是 1。</p>
name	日志处理程序的名称。 <i>已弃用（处理程序的地址里已包含这个名称）</i>
named-formatter	处理程序使用的格式器的名称。
rotate-on-boot	如果设置为 true ，新的日志文件将服务器重启时创建。默认是 false 。
rotate-size	日志文件在轮换前可到达的最大尺寸。数字后的单个字符表示单位： b 为字节、 k 为千字节、 m 为兆字节，而 g 为千兆字节。例如 50m 表示 50MB。
suffix	这个字符串包含在轮换日志的后缀里。 suffix 的格式是句点（.）后面跟着 SimpleDateFormat 可以解析的日期字符串。

表 A.22. Periodic Size 日志处理程序属性

属性	描述
append	如果设置为 true ，这个处理程序写入的所有消息都将附加到这个文件（如果存在）。如果设置为 false ，每次应用服务器启动时都会创建一个新的文件。
autoflush	如果设置为 true ，日志消息将被发送至处理程序接收时立即分配的文件。
enabled	如果设置为 true ，处理程序被启用且正常工作。如果设置为 false ，处理日志消息时处理程序将被忽略。
encoding	用于输出的字符编码模式。
file	代表这个日志处理程序输出写入的文件的对象。它有两个配置属性： relative-to 和 path 。
filter-spec	定义过滤器的表达式。下面的表达式定义了不匹配某个模式的过滤器： not(match("WFLY.*"))
formatter	这个日志处理程序使用的日志格式器。
level	日志处理程序记录的日志消息的最低级别。

属性	描述
max-backup-index	<p>轮换日志文件保留的最大数目。当到达这个数目时，最旧的日志文件将被重用。默认为 1。</p> <p>如果使用了 suffix 属性，轮换日志文件的后缀将包含在轮换乘法里。当轮换日志文件时，以 name+suffix 开始的最旧的文件将被删除，剩余的轮换日志文件的数字型后缀都会增加，而最近轮换的日志文件的后缀将是 1。</p>
name	日志处理程序的名称。 <i>已弃用（处理程序的地址里已包含这个名称）</i>
named-formatter	处理程序使用的格式器的名称。
rotate-on-boot	如果设置为 true ，新的日志文件将服务器重启时创建。默认是 false 。
rotate-size	日志文件在轮换前可到达的最大尺寸。数字后的单个字符表示单位： b 为字节、 k 为千字节、 m 为兆字节，而 g 为千兆字节。例如 50m 表示 50MB。
suffix	这个字符串包含在轮换日志的后缀里。 suffix 的格式是句点（.）后面跟着 SimpleDateFormat 可以解析的日期字符串。

表 A.23. Syslog 处理程序属性

属性	描述
app-name	当以 RFC5424 格式格式化消息时使用的应用程序名称。默认的应用程序名称是 java 。
enabled	如果设置为 true ，处理程序被启用且正常工作。如果设置为 false ，处理日志消息时处理程序将被忽略。
facility	RFC-5424 和 RFC-3164 定义的工具。
hostname	发送消息的主机的名称。例如，运行应用服务器的主机的名称。
level	日志处理程序记录的日志消息的最低级别。
port	syslog 服务器侦听的端口。
server-address	syslog 服务器的地址。
syslog-format	按照 RFC 规格格式化日志消息。

表 A.24. 自定义日志处理程序属性

属性	描述
class	要使用的日志处理程序类。
enabled	如果设置为 true ，处理程序被启用且正常工作。如果设置为 false ，处理日志消息时处理程序将被忽略。
encoding	用于输出的字符编码模式。
filter	定义简单的过滤器类型。 <i>已弃用，转而使用 filter-spec。</i>
filter-spec	定义过滤器的表达式。下面的表达式定义了不匹配某个模式的过滤器： not(match("WFLY.*"))
formatter	这个日志处理程序使用的日志格式器。
level	日志处理程序记录的日志消息的最低级别。
module	日志处理程序依赖的模块。
name	日志处理程序的名称。 <i>已弃用（处理程序的地址里已包含这个名称）</i>
named-formatter	处理程序使用的格式器的名称。
properties	用于日志处理程序的属性。

表 A.25. Async 日志处理程序属性

属性	描述
enabled	如果设置为 true ，处理程序被启用且正常工作。如果设置为 false ，处理日志消息时处理程序将被忽略。
filter	定义简单的过滤器类型。 <i>已弃用，转而使用 filter-spec。</i>
filter-spec	定义过滤器的表达式。下面的表达式定义了不匹配某个模式的过滤器： not(match("WFLY.*"))
level	日志处理程序记录的日志消息的最低级别。
name	日志处理程序的名称。 <i>已弃用（处理程序的地址里已包含这个名称）</i>
overflow-action	当超过队列长度时这个处理程序如何应对。它可以设置为 BLOCK 或 DISCARD 。 BLOCK 让日志应用程序等待直至队列有可用空间。这和非异步日志处理程序是相同的行为。 DISCARD 允许日志处理程序继续，但会删除日志消息。
queue-length	在等待子处理程序响应时，这个处理程序保持的日志消息的最大数目。

属性	描述
subhandlers	这个异步处理程序将日志消息传入的日志处理程序列表。

A.15. 数据源连接 URL

表 A.26. 数据源连接 URL

数据源	连接 URL
IBM DB2	jdbc:db2://SERVER_NAME:PORT/DATABASE_NAME
MariaDB	jdbc:mariadb://SERVER_NAME:PORT/DATABASE_NAME
Microsoft SQL 服务器	jdbc:sqlserver://SERVER_NAME:PORT;DatabaseName=DATABASE_NAME
MySQL	jdbc:mysql://SERVER_NAME:PORT/DATABASE_NAME
Oracle	jdbc:oracle:thin:@SERVER_NAME:PORT:ORACLE_SID
PostgreSQL	jdbc:postgresql://SERVER_NAME:PORT/DATABASE_NAME
Sybase	jdbc:sybase:Tds:SERVER_NAME:PORT/DATABASE_NAME

A.16. 数据源参数

表 A.27. 数据源参数

参数	数据源类型	描述
allocation-retry	Non-XA, XA	在抛出异常前分配连接应该尝试的次数。默认是 0 ，表示在第一次失败就抛出异常。
allocation-retry-wait-millis	Non-XA, XA	重试连接分配间等待的时间（毫秒）。默认值是 5000 。
allow-multiple-users	Non-XA, XA	是否有多个用户将通过 getConnection(user, password) 方法访问数据源，且这个行为是否包括内部池类型。
background-validation	Non-XA, XA	连接在后台线程还是在使用前进行检验。后台检验通常不会和 validate-on-match 一起使用，或者有冗余的检查。通过后台检验，连接可能在校验间损坏并交给客户，所以应用程序必须考虑这种可能性。
background-validation-millis	Non-XA, XA	后台校验运行的频率（毫秒）。

参数	数据源类型	描述
blocking-timeout-wait-millis	Non-XA, XA	在抛出异常前等待连接时阻塞的最长时间（毫秒）。请注意，这只有在等待连接锁时才会阻塞，而且如果创建新的连接耗费很长时间也不会抛出异常。
capacity-decrementer-class	Non-XA, XA	定义递减池连接策略的类。
capacity-decrementer-properties	Non-XA, XA	在定义递减池连接策略的类里注入的属性。
capacity-incrementer-class	Non-XA, XA	定义递增池连接策略的类。
capacity-incrementer-properties	Non-XA, XA	在定义递增池连接策略的类里注入的属性。
check-valid-connection-sql	Non-XA, XA	检查池连接有效性的 SQL 语句。当从池里获取受管连接时可以调用它。
connectable	Non-XA, XA	启用 CMR 的使用，表示本地资源可以可靠地参与 XA 事务。
connection-listener-class	Non-XA, XA	指定继承 <code>org.jboss.jca.adapters.jdbc.spi.listener.ConnectionListener</code> 的类。这个类侦听连接激活和钝化以在连接返回至应用程序或池之前执行操作。被指定的类必须用两个资源 JAR 和 JDBC 驱动捆绑在一个模块（ Install a JDBC Driver as a Core Module ）或单独的全局模块（ Define Global Modules ）里。
connection-listener-property	Non-XA, XA	要注入到 <code>connection-listener-class</code> 里指定的类的属性。注入的属性和 JavaBeans 规则兼容。例如，如果您指定名为 <code>foo</code> 的属性，那么连接 Listener 类需要有接受 <code>String</code> 参数的 <code>setFoo</code> 方法。
connection-properties	仅用于 Non-XA	传入 <code>Driver.connect(url, props)</code> 方法的任意字符串名/值对连接属性。
connection-url	仅用于 Non-XA	JDBC 驱动连接 URL。
datasource-class	仅用于 Non-XA	JDBC 数据源类的全限定名。
driver-class	仅用于 Non-XA	JDBC 驱动类的全限定名。
driver-name	Non-XA, XA	定义数据源应该使用的 JDBC 驱动。这是对应安装的驱动的名称的符号名称。如果驱动被部署为 JAR，这个名称就是部署的名称。

参数	数据源类型	描述
enabled	Non-XA, XA	是否应该启用数据源。
enlistment-trace	Non-XA, XA	是否应该记录 enlistment 跟踪信息。
exception-sorter-class-name	Non-XA, XA	提供检验异常是否应该广播错误的方法的 org.jboss.jca.adapters.jdbc.ExceptionSorter 实例。
exception-sorter-properties	Non-XA, XA	exception sorter 的属性。
flush-strategy	Non-XA, XA	指定在发生错误时如何冲刷池。有效值是： <ul style="list-style-type: none"> FailingConnectionOnly (<i>default</i>) IdleConnections EntirePool
idle-timeout-minutes	Non-XA, XA	连接在关闭前可以空闲的最长时间（分钟）。实际的最长时间也取决于 IdleRemover 扫描时间，它是任何池的最小 idle-timeout-minutes 的二分之一。
initial-pool-size	Non-XA, XA	池应该保持的连接的初始数量。
interleaving	只适用于 XA	是否对 XA 连接启用 interleaving
jndi-name	Non-XA, XA	数据源的唯一 JNDI 名称。
jta	仅用于 Non-XA	启用 JTA 集成。
max-pool-size	Non-XA, XA	池可以保持的最大数量。
mcp	Non-XA, XA	ManagedConnectionPool 实现。例如 org.jboss.jca.core.connectionmanager.pool.mcp.SemaphoreArrayListManagedConnectionPool 。
min-pool-size	Non-XA, XA	池可以保持的连接的最小数目。
new-connection-sql	Non-XA, XA	每当连接被添加至连接池时要执行的 SQL 语句。
no-recovery	只适用于 XA	连接池是否应该从恢复排除。

参数	数据源类型	描述
no-tx-separate-pool	只适用于 XA	是否为每个上下文创建独立的子池。某些 Oracle 数据源可能要求这样做，这可能不允许 XA 连接既用于 JTA 事务的内部又用于其外部。使用这个选项将导致您的池大小两倍于 max-pool-size ，因为实际将创建两个池。
pad-xid	只适用于 XA	是否覆盖 XID。
password	Non-XA, XA	创建新连接时使用的密码。
pool-fair	Non-XA, XA	定义池是否是公平的。这个设置是在 JCA 里用于管理连接池的 Semaphore 类的一部分，它在某些不要求租借连接的顺序的情况下提供了性能优势。
pool-prefill	Non-XA, XA	是否应该预填充池。
pool-use-strict-min	Non-XA, XA	是否应该严格考虑 min-pool-size 。
prepared-statements-cache-size	Non-XA, XA	在 Least Recently Used (LRU) 缓存里每个连接的 prepared 语句的数量。
query-timeout	Non-XA, XA	查询超时时间（秒）。默认是不会超时。
reauth-plugin-class-name	Non-XA, XA	重新验证物理连接的重验证插件实现的全限定类名。
reauth-plugin-properties	Non-XA, XA	重验证插件的属性。
recovery-password	只适用于 XA	连接用于恢复的资源时使用的密码。
recovery-plugin-class-name	只适用于 XA	恢复插件实现的全限定类名。
recovery-plugin-properties	只适用于 XA	恢复插件的属性。
recovery-security-domain	只适用于 XA	连接用于恢复的资源时使用的安全域。
recovery-username	只适用于 XA	连接用于恢复的资源时使用的用户名。
same-rm-override	只适用于 XA	javax.transaction.xa.XAResource.isSameRM(XAResource) 类返回 true 还是 false 。
security-domain	Non-XA, XA	处理验证的 JAAS security-manager 的名称。这个名称对应 JAAS 登录配置的 application-policy/name 属性。

参数	数据源类型	描述
set-tx-query-timeout	Non-XA, XA	是否根据剩余时间设置查询超时直至事务超时。如果没有事务存在，将使用任何已配置的查询超时。
share-prepared-statements	Non-XA, XA	Whether JBoss EAP should cache, instead of close or terminate, the underlying physical statement when the wrapper supplied to the application is closed by application code. The default is false .
spy	Non-XA, XA	启用 JDBC 驱动上的 spy 功能。这将所有 JDBC 通讯登记至数据源。请注意， logging 子系统里的日志类别 jboss.jdbc.spy 也必须设置为日志级别 DEBUG 。
stale-connection-checker-class-name	Non-XA, XA	提供 isStaleConnection(SQLException) 方法的 org.jboss.jca.adapters.jdbc.StaleConnectionChecker 实例。如果这个方法返回 true ，那么异常会打包在 org.jboss.jca.adapters.jdbc.StaleConnectionException 里。
stale-connection-checker-properties	Non-XA, XA	过时连接检查器属性。
statistics-enabled	Non-XA, XA	是否启用运行时统计。默认值是 false 。
track-statements	Non-XA, XA	当连接返回至池且连接返回至预备语句缓存时是否检查未关闭的语句。如果为 false ，语句不会被跟踪。它的有效值为： <ul style="list-style-type: none"> • true：语句和结构都会被跟踪，如果未关闭则会发出警告。 • false：既不会跟踪语句也不会跟踪结果。 • nowarn：语句被跟踪但不会发出警告（默认）。
tracking	Non-XA, XA	是否跨事务边界跟踪事务句柄。

参数	数据源类型	描述
transaction-isolation	Non-XA, XA	<p>java.sql.Connection 事务隔离级别。有效值是：</p> <ul style="list-style-type: none"> • TRANSACTION_READ_UNCOMMITTED • TRANSACTION_READ_COMMITTED • TRANSACTION_REPEATABLE_READ • TRANSACTION_SERIALIZABLE • TRANSACTION_NONE
url-delimiter	Non-XA, XA	高可用性数据源的 connection-url 里的分隔符。
url-property	只适用于 XA	xa-datasource-property 里的 URL 属性。
url-selector-strategy-class-name	Non-XA, XA	实现 org.jboss.jca.adapters.jdbc.URLSelectorStrategy 的类。
use-ccm	Non-XA, XA	启用缓存的连接管理者。
use-fast-fail	Non-XA, XA	如果为 true，如果连接无效，连接分配在第一次尝试就会失败。如果为 false，会一直尝试直至池耗尽。
use-java-context	Non-XA, XA	是否将数据源绑定至全局 JNDI。
use-try-lock	Non-XA, XA	内部锁的超时时间。会试图在配置的时间（秒）之内获取锁，而不是在锁不可用时立即失败。请使用 tryLock() 而不是 lock() 。
user-name	Non-XA, XA	创建新连接时使用的用户名。
valid-connection-checker-class-name	Non-XA, XA	提供 SQLException.isValidConnection(Connection e) 方法来检验连接的 org.jboss.jca.adapters.jdbc.ValidConnectionChecker 实现。异常抛出表示连接被销毁。这覆盖了 check-valid-connection-sql 参数（如果存在）。
valid-connection-checker-properties	Non-XA, XA	有效连接检查器属性

参数	数据源类型	描述
validate-on-match	Non-XA, XA	当连接工厂试图匹配受管连接时，是否执行连接校验。当客户在使用连接前必须进行校验时，它应该被使用。Validate-on-match 通常不会和 background-validation 一起使用，否则将会有冗余的检查。
wrap-xa-resource	只适用于 XA	是否将 XAResource 打包在 org.jboss.tm.XAResourceWrapper 实例里。
xa-datasource-class	只适用于 XA	javax.sql.XADataSource 实现类的全限定名。
-properties	只适用于 XA	XA 数据源属性的字符串名/值对。
xa-resource-timeout	只适用于 XA	如果非零，这个值将被传入 XAResource.setTimeout 方法。

A.17. 数据源统计

表 A.28. 核心的池统计信息

名称	描述
ActiveCount	活动连接的数量。每个连接要么被应用程序使用，要么位于池里可用。
AvailableCount	池里有效连接的数量。
AverageBlockingTime	阻塞获取池的排斥锁所消耗的平均时间。这个值的单位是毫秒。
AverageCreationTime	创建连接花费的平均时间（单位为毫秒）。
AverageGetTime	获取连接平均花费的时间。
AverageUsageTime	使用连接的平均时间。
BlockingFailureCount	试图获取连接的失败次数。
CreatedCount	创建的连接数量。
DestroyedCount	销毁的连接数量。
IdleCount	当前空闲的连接数量。

名称	描述
InUseCount	当前使用的连接数量。
MaxCreationTime	创建连接花费的最长时间（单位为毫秒）。
MaxGetTime	获取连接花费的最长时间。
MaxPoolTime	池里连接的最长时间。
MaxUsageTime	使用连接的最长时间。
MaxUsedCount	使用的连接的最大数量。
MaxWaitCount	同一时间等待连接的请求的最大数量。
MaxWaitTime	等待池里排他锁所消耗的最长时间。
TimedOut	超时连接的数量。
TotalBlockingTime	等待池的排他锁所消耗的总共时间。这个值的单位是毫秒。
TotalCreationTime	创建连接花费的总共时间（单位为毫秒）。
TotalGetTime	获取连接花费的总共时间。
TotalPoolTime	池里连接花费的总共时间。
TotalUsageTime	使用连接花费的总共时间。
WaitCount	必须等待获取连接的请求的数量。
XACommitAverageTime	XAResource commit 调用的平均时间。
XACommitCount	XAResource commit 调用的次数。
XACommitMaxTime	XAResource commit 调用的最长时间。
XACommitTotalTime	所有 XAResource commit 调用的总共时间。
XAEndAverageTime	XAResource end 调用的平均时间。
XAEndCount	XAResource end 调用的次数。
XAEndMaxTime	XAResource end 调用的最长时间。
XAEndTotalTime	所有 XAResource end 调用的总共时间。

名称	描述
XAForgetAverageTime	XAResource forget 调用的平均时间。
XAForgetCount	XAResource forget 调用的次数。
XAForgetMaxTime	XAResource forget 调用的最长时间。
XAForgetTotalTime	所有 XAResource forget 调用的总共时间。
XAPrepareAverageTime	XAResource prepare 调用的平均时间。
XAPrepareCount	XAResource prepare 调用的次数。
XAPrepareMaxTime	XAResource prepare 调用的最长时间。
XAPrepareTotalTime	所有 XAResource prepare 调用的总共时间。
XARecoverAverageTime	XAResource recover 调用的平均时间。
XARecoverCount	XAResource recover 调用的次数。
XARecoverMaxTime	XAResource recover 调用的最长时间。
XARecoverTotalTime	所有 XAResource recover 调用的总共时间。
XARollbackAverageTime	XAResource rollback 调用的平均时间。
XARollbackCount	XAResource rollback 调用的次数。
XARollbackMaxTime	XAResource rollback 调用的最长时间。
XARollbackTotalTime	所有 XAResource rollback 调用的总共时间。
XAStartAverageTime	XAResource start 调用的平均时间。
XAStartCount	XAResource start 调用的次数。
XAStartMaxTime	XAResource start 调用的最长时间。
XAStartTotalTime	所有 XAResource start 调用的总共时间。

表 A.29. JDBC 统计信息

名称	描述
PreparedStatementCacheAccessCount	语句缓存被访问的次数。
PreparedStatementCacheAddCount	添加至语句缓存的语句数量。
PreparedStatementCacheCurrentSize	目前缓存在语句缓存里的预备和可调用语句的数量。
PreparedStatementCacheDeleteCount	从缓存丢弃的语句数量。
PreparedStatementCacheHitCount	缓存里语句被使用的次数。
PreparedStatementCacheMissCount	语句请求无法满足缓存里的请求的次数。

A.18. 事务管理者的配置选项

表 A.30. 事务管理者的配置选项

选项	描述
default-timeout	默认的事务超时时间。默认值是 300 秒。您可以在程序里为每个事务覆盖这个值。
enable-statistics	<i>已过时。由 statistics-enabled 替代。</i>
enable-tsm-status	是否启用事务状态管理者（Transaction Status Manager, TSM）服务，这个服务用于进程外的恢复。这个选项不被支持，因为运行进程外恢复管理者从不同的进程来联系 ActionStatusService 是不受支持的（通常是通过内存联系）。
hornetq-store-enable-async-io	<i>已过时。由 journal-store-enable-async-io 替代。</i>
jdbc-action-store-drop-table	JDBC 操作存储是否应该丢弃表。默认值为 false 。
jdbc-action-store-table-prefix	表可以使用可选的前缀在配置的 JDBC 操作存储中写事务日志。
jdbc-communication-store-drop-table	JDBC 通讯存储是否应该丢弃表。默认值为 false 。
jdbc-communication-store-table-prefix	表可以使用可选的前缀在配置的 JDBC 通讯存储中写事务日志。
jdbc-state-store-drop-table	JDBC 状态存储是否应该丢弃表。默认值为 false 。
jdbc-state-store-table-prefix	表可以使用可选的前缀在配置的 JDBC 状态存储中写事务日志。

选项	描述
jdbc-store-datasource	非 XA 数据源使用的 JNDI 名。数据源应该在 datasources 子系统中定义。
journal-store-enable-async-io	是否对日志库启用 AsyncIO 。默认值是 false 。使其生效需要重启服务器。
jts	是否使用 Java Transaction Service (JTS) 事务。默认值是 false ，表示只使用 JTA 事务。
node-identifier	<p>事务管理者的节点标识符。如果没有设置这个选项，您将会在服务器启动时看到一个警告信息。下列情况下需要这个选项：</p> <ul style="list-style-type: none"> • 对于 JTS - JTS 通讯 • 当两个事务管理者访问共享的资源管理者 • 当两个事务管理者访问共享的对象存储 <p>对于每个事务管理者来说，节点标识符必须是唯一的，因为在恢复期间要求数据的完整性。节点标识符对于 JTA 也必须是唯一的，因为多个节点可能与相同的资源管理者交互或共享事务对象库。</p>
object-store-path	事务管理者对象库存储数据的相对或绝对文件系统路径。默认是相对于 object-store-relative-to 的路径。如果 object-store-relative-to 被设置为一个空字符串，这个值将作为绝对路径对待。
object-store-relative-to	域模型里对全局路径配置的引用。默认值是 JBoss EAP 的数据目录，即 jboss.server.data.dir 属性的值。对于受管域是 EAP_HOME/domain/data/ ，而对于独立服务器是 EAP_HOME/standalone/data/ 。对象库的 object-store-path 事务管理者属性是相对于这个路径的值。如果把这个属性设置为空字符串，则 object-store-path 会被认为是一个绝对路径。
process-id-socket-binding	如果事务管理者应该使用基于套接字的进程 ID，这是套接字绑定配置的名称。如果 process-id-uuid 为 true ，它将是 undefined ，否则必须进行设置。
process-id-socket-max-ports	<p>事务管理者为每个事务日志创建一个唯一的标识符。系统为生成唯一的标识符提供了两个不同的机制：基于套接字的机制和基于进程标识符的机制。</p> <p>在使用基于套接字的标识符时，套接字将被打开且其端口号被用于标识符。如果端口已被使用，将探测下一个端口，直至找到空闲的端口。process-id-socket-max-ports 代表事务管理器将尝试直至失败的最大套接字数。默认值是 10。</p>
process-id-uuid	设置为 true 以使用进程标识符来为每个事务创建唯一的标识符。否则，基于套接字的机制将被使用。默认值为 true 。更多信息请参考 process-id-socket-max-ports 。要启用 process-id-socket-binding ，请将 process-id-uuid 设置为 false 。

选项	描述
recovery-listener	事务恢复过程是否应该侦听网络套接字。默认为 false 。
socket-binding	当 recovery-listener 被设置为 true 时，指定事务定期恢复 listener 使用的套接字绑定。
statistics-enabled	是否启用统计数据功能。
status-socket-binding	指定用于事务状态管理者的套接字绑定。这个配置选项不受支持。
use-hornetq-store	<i>已过时。由 use-journal-store 替代。</i>
use-jdbc-store	使用 JDBC 存储写事务日志。设为 true 启用，设为 false 来使用默认的日志存储类型。
use-journal-store	对于事务日志使用 Apache ActiveMQ Artemis 日志存储机制而不是基于文件的存储。默认这是被禁用的，但它可以提高 I/O 性能。对于单独事务管理者上的 JTS 事务，我们不推荐使用它。当修改这个选项时，服务器必须使用 shutdown 命令重启以使其生效。

A.19. 资源适配器属性

下表描述了资源适配器属性。

表 A.31. 主要属性

属性	描述
archive	资源适配器归档
beanvalidationgroups	需要使用的 bean 验证组。
bootstrap-context	使用的 bootstrap 上下文的唯一名称。
config-properties	自定义的配置属性。
module	资源适配器从哪个模块加载。
statistics-enabled	是否启用运行时统计数据。
transaction-support	资源适配器的事务支持级别。
wm-security	为资源适配器切换启用或禁用 wm.security 。如果禁用，包括默认参数在内的所有 wm-security-* 参数都会被忽略。
wm-security-default-groups	添加到使用的 Subject 实例的默认组列表。

属性	描述
wm-security-default-principal	添加到使用的 Subject 实例的 principal 名。
wm-security-domain	需要使用的安全域的名称。
wm-security-mapping-groups	组映射列表。
wm-security-mapping-required	指定安全凭证是否需要一个映射。
wm-security-mapping-users	用户映射列表。

表 A.32. admin-objects Attributes

属性	描述
class-name	一个管理对象的完全限定类名。
enabled	指定是否需要启用管理对象。
jndi-name	管理对象的 JNDI 名。
use-java-context	如果需要把对象绑定到全局 JNDI，把它设置为 false。

表 A.33. connection-definitions 属性

属性	描述
allocation-retry	指定在抛出一个异常前，尝试分配一个连接的次数。
allocation-retry-wait-millis	重试连接分配间等待的时间（毫秒）。
background-validation	指定连接应该在一个后台线程中验证，还是在使用前进行验证。修改这个设置需要重启服务器。
background-validation-millis	后台验证运行的时间（以毫秒为单位）。修改这个值需要重启服务器。
blocking-timeout-wait-millis	在抛出异常前等待连接时阻塞的最长时间（毫秒）。请注意，这只有在等待连接锁时才会阻塞，而且如果创建新的连接耗费很长时间也不会抛出异常。
capacity-decrementer-class	定义递减池连接策略的类。
capacity-decrementer-properties	注入到用于定义减少池中的连接策略的类的属性。
capacity-incrementer-class	定义递增池连接策略的类。

属性	描述
capacity-incrementer-properties	注入到用于定义增加池中的连接策略的类的属性。
class-name	受管连接工厂或管理对象的完全限定类名。
connectable	启用 CMR 的使用。它代表本地资源可以可靠地参与 XA 事务。
enabled	指定是否启用资源适配器。
enlistment	指定在资源适配器支持的情况下，是否使用 lazy enlistment 功能。
enlistment-trace	指定 JBoss EAP 或 IronJacamar 是否应该记录 enlistment 跟踪信息。
flush-strategy	在发生错误时如何冲刷池。有效值是： FailingConnectionOnly （默认）、 IdleConnections 、 EntirePool 。
idle-timeout-minutes	连接在关闭前可以空闲的最长时间（分钟）。实际的最长时间也取决于 IdleRemover 扫描时间，它是任何池的最小 idle-timeout-minutes 值一半。修改这个值需要重启服务器。
initial-pool-size	池应该保持的连接的初始数量。
interleaving	指定是否对 XA 连接启用 interleaving。
jndi-name	连接工厂的 JNDI 名。
max-pool-size	池里最大的连接数。在每个子池里创建的连接都不能超过这个值。
mcp	ManagedConnectionPool 的实现。例如 org.jboss.jca.core.connectionmanager.pool.mcp.SemaphoreArrayListManagedConnectionPool 。
min-pool-size	一个池的最少连接数量。
no-recovery	指定连接池是否应该从恢复中排除。
no-tx-separate-pool	Oracle 不希望 XA 连接既在 JTA 事务内部又在外部使用。要绕过这个问题，您可以为不同的上下文创建独立的子池。
pad-xid	指定 Xid 是否需要被填充。
pool-fair	指定池是否应该公平使用。
pool-prefill	指定池是否应该被预先填充。修改这个值需要重启服务器。
pool-use-strict-min	指定 min-pool-size 是否是严格的。

属性	描述
recovery-password	恢复所使用的密码。
recovery-plugin-class-name	恢复插件实现的完全限定类名。
recovery-plugin-properties	恢复插件的属性。
recovery-security-domain	用于恢复的安全域。
recovery-username	用于恢复的用户名。
same-rm-override	无条件地设置 <code>javax.transaction.xa.XAResource.isSameRM(XAResource)</code> 是否返回 <code>true</code> 或 <code>false</code> 。
security-application	表示应用程序提供的参数（如 <code>getConnection(user, pw)</code> ）被用来区分池里的连接。
security-domain	定义用来区分池中的连接的 <code>javax.security.auth.Subject</code> 的安全域。
security-domain-and-application	表示应用程序提供的参数，如 <code>getConnection(user, pw)</code> 或 <code>Subject</code> （来自于安全域），被用来区分池里的连接。
sharable	启用共享连接功能，这将允许在支持的情况下启用 lazy association 功能。
tracking	指定 IronJacamar 是否应该跟踪跨事务边界的连接句柄。
use-ccm	允许使用一个缓存的连接管理器。
use-fast-fail	如果为 <code>true</code> ，在第一次尝试时，如果无效就使连接分配失败；如果为 <code>false</code> ，则会在尝试失败后继续尝试，直到已检查了所有可能的连接。
use-java-context	把它设置为 <code>false</code> 会把这个对象绑定到全局 JNDI。
validate-on-match	指定当一个连接工厂试图匹配一个受管的连接时，是否需要进行连接验证。这通常只适用于使用后台验证。
wrap-xa-resource	指定 <code>XAResource</code> 实例是否应该打包在一个 <code>org.jboss.tm.XAResourceWrapper</code> 实例中。
xa-resource-timeout	这个值（以秒为单位）传递给 <code>XAResource.setTimeout()</code> 。它的默认值是 <code>0</code> 。

资源适配器模式可以在 `EAP_HOME/docs/schema/wildfly-resource-adapters_4_0.xsd` 里找到。

A.20. 资源适配器统计

表 A.34. 资源适配器统计

名称	描述
ActiveCount	活动连接的数量。每个连接要么被应用程序使用，要么位于池里可用。
AvailableCount	池里有效连接的数量。
AverageBlockingTime	阻塞获取池的排斥锁所消耗的平均时间。这个值的单位是毫秒。
AverageCreationTime	创建连接花费的平均时间（单位为毫秒）。
CreatedCount	创建的连接数量。
DestroyedCount	销毁的连接数量。
InUseCount	当前使用的连接数量。
MaxCreationTime	创建连接花费的最长时间（单位为毫秒）。
MaxUsedCount	使用的连接的最大数量。
MaxWaitCount	同一时间等待连接的请求的最大数量。
MaxWaitTime	等待池里排他锁所消耗的最长时间。
TimedOut	超时连接的数量。
TotalBlockingTime	等待池的排他锁所消耗的总共时间。这个值的单位是毫秒。
TotalCreationTime	创建连接花费的总共时间（单位为毫秒）。
WaitCount	必须等待连接的请求的数量。

A.21. UNDERTOW 子系统属性

表 A.35. undertow 属性

属性	Default	描述
default-security-domain	other	web 部署使用的默认安全域。
default-server	default-server	用于部署的默认服务器。
default-servlet-container	default	用于部署的默认 servlet 容器。

属性	Default	描述
default-virtual-host	default-host	用于部署的默认虚拟主机。
instance-id	<code>\${boss.node.name}</code>	集群实例 ID。
statistics-enabled	false	是否启用统计数据

缓冲缓存属性

表 A.36. **buffer-cache** 属性

属性	Default	描述
buffer-size	1024	缓冲的大小。更小的缓冲允许更有效地利用空间。
buffers-per-region	1024	每个区的缓冲数量。
max-regions	10	区的最大数量。它控制可用于缓存的最大内存数量。

Servlet 容器属性

Servlet 容器组件具有下列结构：

- [servlet-container](#)
 - [mime-mapping](#)
 - [welcome-file](#)
 - [crawler-session-management \(part of settings\)](#)
 - [jsp \(part of settings\)](#)
 - [persistent-sessions \(part of settings\)](#)
 - [session-cookie \(part of settings\)](#)
 - [websockets \(part of settings\)](#)

servlet-container 属性

表 A.37. **servlet-container** 属性

属性	Default	描述
allow-non-standard-wrappers	false	是否可以使用没有扩展标准 Wrapper 类的请求和资源 Wrapper。
default-buffer-cache	default	用于缓存静态资源的缓冲缓存。

属性	Default	描述
default-encoding		用于所有部署的应用程序的默认编码。
default-session-timeout	30	用于容器里部署的所有应用程序的默认会话超时时间（以分钟为单位）。
directory-listing		是否应该为默认的 servlet 启用目录列表。
disable-caching-for-secured-pages	true	是否设置头部来为安全页面禁用缓存。禁用它可能导致安全问题，因为敏感页面可能会被中介缓存。
eager-filter-initialization	false	是否在部署启动，而不是第一次被请求时调用过滤器的 init()。
ignore-flush	false	忽略 Servlet 输出流上的冲刷。多数情况下这些操作只会影响到性能。
max-sessions		在同一时刻可以处于活动状态的最大会话数量。
proactive-authentication	false	是否使用主动验证。如果为 true ，出现凭证时用户将总会被验证。
session-id-length	30	生成的会话 ID 的长度。越长的会话 ID 越安全。
stack-trace-on-error	local-only	在出现错误时是否应该生成带有跟踪栈信息的错误页面。其值为 all、none 和 local-only。
use-listener-encoding	false	使用 Listener 上定义的编码

mime-mapping 属性

表 A.38. mime-mapping 属性

属性	Default	描述
value		这个映射的 MIME 类型。

welcome-file 属性

定义欢迎文件，它没有选项。

crawler-session-management 属性

为 crawler bot 配置特殊会话处理。



注意

当使用管理 CLI 管理 **crawler-session-management** 元素时，它在 **servlet-container** 元素的 **settings** 中可用。例如：

```
/subsystem=undertow/servlet-container=default/setting=crawler-session-management:add
/subsystem=undertow/servlet-container=default/setting=crawler-session-management:read-resource
```

表 A.39. crawler-session-management 属性

属性	Default	描述
session-timeout		crawler 所拥有会话的超时时间（以秒为单位）。
user-agents		用来匹配一个 crawler 的用户代理的正则表达式。

jsp 属性



注意

当使用管理 CLI 管理 **jsp** 元素时，它在 **servlet-container** 元素的 **settings** 中可用。例如：

```
/subsystem=undertow/servlet-container=default/setting=jsp:read-resource
```

表 A.40. jsp 属性

属性	Default	描述
check-interval	0	用后台线程检查 JSP 更新的间隔。
development	false	启用开放模式，它可以在运行时重载 JSP。
disabled	false	启用 JSP 容器。
display-source-fragment	true	当发生运行错误时，试图显示相应的 JSP 源码片段。
dump-smap	false	写入 SMAP 数据至文件。
error-on-use-bean-invalid-class-attribute	false	在 useBean 使用了错误的类时显示错误。
generate-strings-as-char-arrays	false	将字符串常量生成为字符数组。
java-encoding	UTF8	生成用于 Java 源码的编码。

属性	Default	描述
keep-generated	true	保持生成的 Servlet。
mapped-file	true	映射至 JSP 源码。
modification-test-interval	4	两次更新测试间的最小时间间隔（秒）。
optimize-scriptlets	false	是否应该优化 JSP scriptlets 来删除字符串连接。
recompile-on-fail	false	在每次请求时重试失败的 JSP 编译。
scratch-dir		指定不同的工作目录。
smap	true	启用 SMAP。
source-vm	1.8	用于编译的源虚拟机级别。
tag-pooling	true	启用标签池。
target-vm	1.8	用于编译的目标虚拟机级别。
trim-spaces	false	从生成的 Servlet 里删除空格。
x-powered-by	true	启用 x-powered-by 里的 JSP 引擎的广告。

persistent-sessions 属性



注意

当使用管理 CLI 管理 **persistent-sessions** 元素时，它在 **servlet-container** 元素的 **settings** 中可用。例如：

```
/subsystem=undertow/servlet-
container=default/setting=persistent-sessions:add
/subsystem=undertow/servlet-
container=default/setting=persistent-sessions:read-resource
```

表 A.41. **persistent-sessions** 属性

属性	Default	描述
路径		持久性会话数据目录的路径。如果它为 null，会话将保存在内存里。
relative-to		这个目录相对的路径

session-cookie Attributes



注意

当使用管理 CLI 管理 **session-cookie** 元素时，它在 **servlet-container** 元素的 **settings** 中可用。例如：

```
/subsystem=undertow/servlet-container=default/setting=session-cookie:add
/subsystem=undertow/servlet-container=default/setting=session-cookie:read-resource
```

表 A.42. session-cookie Attributes

属性	Default	描述
comment		Cookie comment.
domain		Cookie 域。
http-only		cookie 是否只是 http。
max-age		cookie 最长有效时间。
name		cookie 的名称
secure		cookie 是否安全。

websockets 属性



注意

当使用管理 CLI 管理 **websockets** 元素时，它在 **servlet-container** 元素的 **settings** 中可用。例如：

```
/subsystem=undertow/servlet-container=default/setting=websockets:read-resource
```

表 A.43. websockets 属性

属性	Default	描述
buffer-pool	default	用于 websocket 部署的缓冲池。
dispatch-to-worker	true	回调是否应该被分发至 worker 线程。如果为 false ，那么它们将运行在 IO 线程，这会更快速，但必须小心，不要执行阻塞操作。

属性	Default	描述
worker	default	用于 websocket 部署的 worker

过滤器属性

custom-filter 过滤器

表 A.44. custom-filter 属性

属性	Default	描述
class-name		HttpHandler 的类名。
module		可以从中加载类的模块的名称。
parameters		过滤器参数

error-page 过滤器

错误页面

表 A.45. error-page 属性

属性	Default	描述
code		错误页代码。
路径		错误页路径。

expression-filter 过滤器

解析 Undertow 表达式语言的过滤器。

表 A.46. expression-filter 过滤器

属性	Default	描述
expression		定义过滤器的表达式。
module		加载过滤器定义所使用的模块。

gzip 过滤器

定义 gzip 过滤器，它没有属性。

mod-cluster 过滤器

mod-cluster 过滤器组件具有下列结构：

- [mod-cluster](#)
 - [balancer](#)

- [load-balancing-group](#)
- [node](#)
 - [context](#)

表 A.47. mod-cluster 属性

属性	Default	描述
advertise-frequency	10000	mod-cluster 在网络里广告自己的频率（以毫秒为单位）。
advertise-path	/	注册 mod_cluster 的路径。
advertise-protocol	http	使用中的协议。
advertise-socket-binding		用来广告的多播组。
broken-node-timeout	60000	在断开的节点从表里删除前必须等待的时间。
cached-connections-per-thread	5	将无限期保持活动的连接数量。
connection-idle-timeout	60	连接在关闭前可以处于空闲状态的时间。如果池大小降至配置的最小值（由 cached-connections-per-thread 配置），连接将不会超时。
connections-per-thread	10	每个 IO 线程将保持在后台服务器的连接数量。
enable-http2	false	负载均衡器是否应该试图升级后台连接至 HTTP2。如果不支持 HTTP2，将使用 HTTP 或 HTTPS。
health-check-interval	10000	ping 后台节点进行“健康检查”的频率。
management-access-predicate		predicate 应用于转入请求，它确定是否可以执行 mod_cluster 管理命令。它在限制对源自 management-socket-binding 的请求的管理的安全性之上提供了额外的安全性。
management-socket-binding		mod_cluster 管理端口的套接字绑定。在使用 mod_cluster 时应该定义两个 HTTP Listener，一个共用的处理请求，另外一个绑定到内部网络来处理 mod_cluster 命令。这套接字绑定应该对应内部的 Listener，不应该公共访问。
max-request-time	-1	对后台节点的请求在被终止前可消耗的最长时间。
request-queue-size	10	如果连接池已满，在请求被以 503 错误拒绝前可以排队请求的数量。

属性	Default	描述
security-key		用于 mod_cluster 组的安全密钥。所有的成员都必须使用相同的安全密钥。
security-realm		提供 SSL 配置的安全域。
use-alias	false	是否执行别名检查。
worker	default	用于发送广告通知的 XNIO 工作节点。

表 A.48. balancer 属性

属性	Default	描述
max-attempts		尝试发送请求至后台服务器的次数。
sticky-session		是否启用粘性会话。
sticky-session-cookie		会话 Cookie 的名称。
sticky-session-force		如果为 true ，则如果请求无法路由至粘性节点，那将返回错误。否则，请求将被路由至其他节点。
sticky-session-path		粘性会话 Cookie 的路径。
sticky-session-remove		如果请求没有路由至正确的主机则删除会话 Cookie。
wait-worker		等待可用工作节点的秒数。

load-balancing-group 属性

定义没有选项的负载均衡组。

表 A.49. node 属性

属性	Default	描述
aliases		节点别名。
cache-connections		无限期保持活动的连接数量。
elected		elected 的数量。
flush-packets		是否立即冲刷接收到的数据。

属性	Default	描述
load		这个节点的当前负载。
load-balancing-group		这个节点所属的负载均衡组。
max-connections		每个 IO 线程最大的连接数量。
open-connections		当前打开的连接的数量。
ping		节点 ping。
queue-new-requests		如果接收到请求但没有立即可用的工作节点，是否应该进行排队。
read		从节点读的字节数。
request-queue-size		请求队列的大小。
status		这个节点的当前状态。
timeout		请求超时。
ttl		如果连接数量大于 cache-connections ，没有请求时连接在关闭前将保持活动的时间。
uri		负载均衡器用来连接节点的 URI。
written		传输至节点的字节数。

表 A.50. context 属性

属性	Default	描述
requests		对这个上下文的请求数量。
status		这个上下文的状态。

request-limit 过滤器

表 A.51. request-limit 属性

属性	Default	描述
max-concurrent-requests		并发请求的最大数量。
queue-size		请求被拒绝前排队的请求数量。

response-header 过滤器

response-header 过滤器允许您添加自定义头部信息。

表 A.52. response-header 属性

属性	Default	描述
header-name		头名称。
header-value		头的值。

rewrite 过滤器

表 A.53. rewrite 属性

属性	Default	描述
redirect	false	是否重定向而不是重写。
target		定义目标的表达式。如果您重定向至固定的目标，请为值加上单引号。

Handler 属性**file** 属性

表 A.54. file 属性

属性	Default	描述
cache-buffer-size	1024	缓冲的大小。
cache-buffers	1024	缓冲的数量。
case-sensitive	true	是否使用区分大小写的文件处理方式。请注意，只有在底层文件系统不区分大小写时，把这个值设置为 false 才有效。
directory-listing	false	是否启用列出目录功能。
follow-symlink	false	启用下面的符号链接。
路径		文件处理程序将充当资源所在的文件系统的路径。
safe-symlink-paths		对于符号链接目标安全的路径。

对静态资源使用 **WebDAV**

之前的 JBoss EAP 版本允许将 WebDAV 和 **web** 子系统一起使用（**WebdavServlet**）来容纳静态资源，并启用额外的 HTTP 方法以访问和操作这些文件。在 JBoss EAP 7 里，**undertow** 子系统提供了通过文件处理程序服务静态文件的机制。但 **undertow** 子系统不支持 WebDAV。如果您想在 JBoss EAP 7

里使用 WebDAV，可以编写自定义的 WebDAV servlet。

reverse-proxy 属性

reverse-proxy 处理程序组件具有下列结构：

- [reverse-proxy](#)
 - [host](#)

表 A.55. reverse-proxy 属性

属性	Default	描述
cached-connections-per-thread	5	将无限期保持活动的连接数量。
connection-idle-timeout	60	连接在关闭前可以处于空闲状态的时间。如果池大小降至配置的最小值（如 cached-connections-per-thread），连接将不会超时。
connections-per-thread	10	每个 IO 线程将保持在后台服务器的连接数量。
max-request-time	-1	代理请求被终止前可以保持活动的最长时间。默认为无限制。
problem-server-retry	30	在试图重新连接下线的服务器前等待的时间（秒）。
request-queue-size	10	如果连接池已满，在请求被以 503 错误拒绝前可以排队请求的数量。
session-cookie-names	JSESSIONID	逗号分开的会话 Cookie 名称列表。通常它仅为 JSESSIONID。

表 A.56. host 属性

属性	Default	描述
instance-id		用来启用粘性会话的实例 ID 或 JVM 路由。
outbound-socket-binding		这个主机的转出套接字绑定。
路径	/	如果主机使用非根资源的可选路径。
scheme	http	使用的 scheme 类型。
security-realm		为到主机的连接提供 SSL 配置的安全区。

服务器属性

这服务器组件具有下列结构：

- [server](#)

- [http-listener](#)
- [https-listener](#)
- [ajp-listener](#)
- [host](#)
 - [access-log \(part of settings\)](#)
 - [single-sign-on \(part of settings\)](#)
 - [filter-ref](#)
 - [location](#)
 - [filter-ref](#)

server 属性

表 A.57. server 属性

属性	Default	描述
default-host	default-host	服务器默认的虚拟主机。
servlet-container	default	服务器默认的 Servlet 容器。

http-listener 属性

表 A.58. http-listener 属性

属性	Default	描述
allow-encoded-slash	false	如果请求中包括经过编码的字符（如 %2F），是否需要对其进行解码。
allow-equals-in-cookie-value	false	是否允许不加引号的 Cookie 值里出现非转义的等号字符。不加引号的 Cookie 值可能没有包含等号，如果有等号，值将到等号为止，后面的部分将被忽略。
always-set-keep-alive	true	是否在响应里加入 Connection: keep-alive 头部，即使规格并没有严格要求它。
buffer-pipelined-data	true	是否应该缓冲管线请求。
buffer-pool	default	Listener 的缓冲池。
certificate-forwarding	false	是否启用证书转发。如果启用，listener 将从 SSL_CLIENT_CERT 属性获取证书。只有位于代理后面才应该启用它，且应该配置代理总是设置这些头部。

属性	Default	描述
decode-url	true	解析器是否用所选的字符编码（默认是 UTF-8）解码 URL 和查询参数。如果为 false，则不会进行解码。这将允许之后的处理程序将它们解码至所要的字符集。
disallowed-methods	["TRACE"]	逗号分开的不允许的 HTTP 方法列表。
enable-http2	false	是否为这个 Listener 启用 HTTP2 支持。
enabled	true	是否启用 listener。
http2-enable-push	true	是否为这个连接启用服务器推送。
http2-header-table-size		用于 HPACK 压缩的头部表的大小（字节）。用于压缩的内存数量将为每个连接进行分配。越大的值使用越多的内存，但可能提供越好的压缩性能。
http2-initial-window-size		控制客户发送数据至服务器的速度的流控制窗口的大小。
http2-max-concurrent-streams		在单个连接上任何时刻都保持活动的 HTTP/2 流的最大数量。
http2-max-frame-size		最大的 HTTP/2 帧大小。
http2-max-header-list-size		服务准备接受的请求头部的最大尺寸。
max-buffered-request-size	16384	缓冲请求的最大尺寸（字节）。请求通常不会被缓冲，最常见的情形是为 POST 请求执行 SSL 重新协商时，必须缓冲完整的 post 数据来执行重新协商。
max-connections		并发连接的最大数量
max-cookies	200	将被解析的 Cookie 的最大数量。这被用于保护 Hash 漏洞。
max-header-size	1048576	HTTP 请求头部的最大尺寸（字节）。
max-headers	200	将被解析的头部的最大数量。这被用于保护 Hash 漏洞。
max-parameters	1000	将被解析的参数的最大数量。这被用于保护 Hash 漏洞。它适用于两个查询参数和 POST 数据，但不是累积的。例如，您可以有最多参数 * 2 的总共参数。
max-post-size	10485760	将被接受的 POST 数据的最大尺寸。

属性	Default	描述
no-request-timeout		在连接被容器关闭之前可以处于空闲状态的时间（毫秒）。
proxy-address-forwarding	false	是否启用 x-forwarded-host 和类似的头部并设置远程的 IP 地址和主机名
read-timeout		配置套接字读超时（毫秒）。如果超过给定的时间而没有成功读取，套接字的下一次读取将抛出 {@link ReadTimeoutException}。
receive-buffer		接收缓冲大小。
record-request-start-time	false	是否记录请求的开始时间，允许记录请求时间。这对性能的影响虽小但却是明显的。
redirect-socket		如果这个 Listener 支持非 SSL 请求，而请求通过要求 SSL 的传输接收的，Undertow 将自动重定向请求至这里指定的套接字绑定端口。
request-parse-timeout		解析请求所花费的最长时间（毫秒）。
resolve-peer-address	false	启用主机 DNS 查找
send-buffer		发送缓冲大小。
socket-binding		这个 Listener 的套接字绑定。
tcp-backlog		用指定的 backlog 配置服务器。
tcp-keep-alive		配置频道以依赖于实现的方式发送 TCP keep-alive 消息。
url-charset	UTF-8	URL 字符集。
worker	default	listener 的 XNIO 工作节点。
write-timeout		配置套接字写超时（毫秒）。如果超过给定的时间而没有成功写入，套接字的下一次写入将抛出 {@link WriteTimeoutException}。

以下属性是只读的，并只在 **undertow** 子系统启用了统计功能时有效：

表 A.59. http-listener 度量属性

属性	Default	描述
bytes-received		这个 Listener 已接收了字节数量。
bytes-sent		这个 Listener 已发送出去的字节数量。
error-count		这个 Listener 已发送的 500 响应的数量。
max-processing-time		这个 Listener 上处理请求的最长时间。
processing-time		这个 Listener 处理所有请求的总共时间。
request-count		这个 Listener 已服务的请求的数量。

https-listener 属性

表 A.60. https-listener 属性

属性	Default	描述
allow-encoded-slash	false	如果请求中包括经过编码的字符（如 %2F），是否需要对其进行解码。
allow-equals-in-cookie-value	false	是否允许不加引号的 Cookie 值里出现非转义的等号字符。不加引号的 Cookie 值可能没有包含等号，如果有等号，值将到等号为止，后面的部分将被忽略。
always-set-keep-alive	true	是否在响应里加入 Connection: keep-alive 头部，即使规格并没有严格要求它。
buffer-pipelined-data	true	是否应该缓冲管线请求。
buffer-pool	default	Listener 的缓冲池。
decode-url	true	解析器是否用所选的字符编码（默认是 UTF-8）解码 URL 和查询参数。如果为 false，则不会进行解码。这将允许之后的处理程序将它们解码至所要的字符集。
disallowed-methods	["TRACE"]	逗号分开的不允许的 HTTP 方法列表。
enable-http2	false	为这个 Listener 启用 HTTP2 支持。
enable-spdy	false	为这个 Listener 启用 SPDY 支持。
enabled	true	是否启用 listener。
enabled-cipher-suites		配置已启用 SSL 的密码。

属性	Default	描述
enabled-protocols		配置 SSL 协议。
http2-enable-push	true	是否为这个连接启用服务器推送。
http2-header-table-size		用于 HPACK 压缩的头部表的大小（字节）。用于压缩的内存数量将为每个连接进行分配。越大的值使用越多的内存，但可能提供越好的压缩性能。
http2-initial-window-size		控制客户发送数据至服务器的速度的流控制窗口的大小。
http2-max-concurrent-streams		在单个连接上任何时刻都保持活动的 HTTP/2 流的最大数量。
http2-max-frame-size		最大的 HTTP/2 帧大小。
http2-max-header-list-size		服务准备接受的请求头部的最大尺寸。
max-buffered-request-size	16384	缓冲请求的最大尺寸（字节）。请求通常不会被缓冲，最常见的情形是为 POST 请求执行 SSL 重新协商时，必须缓冲完整的 post 数据来执行重新协商。
max-connections		并发连接的最大数量
max-cookies	100	将被解析的 Cookie 的最大数量。这被用于保护 Hash 漏洞。
max-header-size	1048576	HTTP 请求头部的最大尺寸（字节）。
max-headers	200	将被解析的头部的最大数量。这被用于保护 Hash 漏洞。
max-parameters	1000	将被解析的参数的最大数量。这被用于保护 Hash 漏洞。它适用于两个查询参数和 POST 数据，但不是累积的。例如，您可以有最多参数 * 2 的总共参数。
max-post-size	10485760	将被接受的 POST 数据的最大尺寸。
no-request-timeout		在连接被容器关闭之前可以处于空闲状态的时间（毫秒）。
read-timeout		配置套接字读超时（毫秒）。如果超过给定的时间而没有成功读取，套接字的下一次读取将抛出 {@link ReadTimeoutException}。
receive-buffer		接收缓冲大小。

属性	Default	描述
record-request-start-time	false	是否记录请求的开始时间，允许记录请求时间。这对性能的影响虽小但却是明显的。
request-parse-timeout		解析请求所花费的最长时间（毫秒）。
resolve-peer-address	false	启用主机 DNS 查找
security-realm		listener 的安全区。
send-buffer		发送缓冲大小。
socket-binding		Listener 的套接字绑定。
ssl-session-cache-size		活动的 SSL 会话的最大数量。
ssl-session-timeout		SSL 会话的超时时间（秒）。
tcp-backlog		用指定的 backlog 配置服务器。
tcp-keep-alive		配置频道以依赖于实现的方式发送 TCP keep-alive 消息。
url-charset	UTF-8	URL 字符集。
verify-client	NOT_REQUESTED	用于 SSL 频道的 SSL 客户验证模式。
worker	default	listener 的 XNIO 工作节点。
write-timeout		配置套接字写超时（毫秒）。如果超过给定的时间而没有成功写入，套接字的下一次写入将抛出 {@link WriteTimeoutException}。

以下属性是只读的，并只在 **undertow** 子系统启用了统计功能时有效：

表 A.61. https-listener 度量属性

属性	Default	描述
bytes-received		这个 Listener 已接收了字节数量。
bytes-sent		这个 Listener 已发送出去的字节数量。
error-count		这个 Listener 已发送的 500 响应的数量。

属性	Default	描述
max-processing-time		这个 Listener 上处理请求的最长时间。
processing-time		这个 Listener 处理所有请求的总共时间。
request-count		这个 Listener 已服务的请求的数量。

ajp-listener 属性

表 A.62. ajp-listener 属性

属性	Default	描述
allow-encoded-slash	false	如果请求中包括经过编码的字符（如 %2F），是否需要对其进行解码。
allow-equals-in-cookie-value	false	是否允许不加引号的 Cookie 值里出现非转义的等号字符。不加引号的 Cookie 值可能没有包含等号，如果有等号，值将到等号为止，后面的部分将被忽略。
always-set-keep-alive	true	是否在响应里加入 Connection: keep-alive 头部，即使规格并没有严格要求它。
buffer-pipelined-data	true	是否缓冲管线请求（pipelined request）。
buffer-pool	default	AJP Listener 的缓冲池。
decode-url	true	如果为 true，解析器将用所选的字符编码（默认是 UTF-8）解码 URL 和查询参数。如果为 false，则不会进行解码。这将允许之后的处理程序将它们解码至所要的字符集。
disallowed-methods	["TRACE"]	逗号分开的不允许的 HTTP 方法列表。
enabled	true	是否启用 listener。
max-ajp-packet-size		支持的 AJP 数据包的最大尺寸。如果它被修改，在负载均衡器和后台服务器上都应增大。
max-buffered-request-size	16384	缓冲请求的最大尺寸（字节）。请求通常不会被缓冲，最常见的情形是为 POST 请求执行 SSL 重新协商时，必须缓冲完整的 post 数据来执行重新协商。
max-connections		并发连接的最大数量
max-cookies	200	将被解析的 Cookie 的最大数量。这被用于保护 Hash 漏洞。

属性	Default	描述
max-header-size	1048576	HTTP 请求头部的最大尺寸（字节）。
max-headers	200	将被解析的头部的最大数量。这被用于保护 Hash 漏洞。
max-parameters	100	将被解析的参数的最大数量。这被用于保护 Hash 漏洞。它适用于两个查询参数和 POST 数据，但不是累积的。例如，您可以有最多参数 * 2 的总共参数。
max-post-size	10485760	将被接受的 POST 数据的最大尺寸
no-request-timeout		在连接被容器关闭之前可以处于空闲状态的时间（毫秒）。
read-timeout		配置套接字读超时（毫秒）。如果超过给定的时间而没有成功读取，套接字的下一次读取将抛出 {@link ReadTimeoutException}。
receive-buffer		接收缓冲大小。
record-request-start-time	false	是否记录请求的开始时间，允许记录请求时间。这对性能的影响虽小但却是明显的。
redirect-socket		如果这个 Listener 支持非 SSL 请求，而请求通过要求 SSL 的传输接收的，Undertow 将自动重定向请求至这里指定的套接字绑定端口。
request-parse-timeout		解析请求所花费的最长时间（毫秒）。
resolve-peer-address	false	启用主机 DNS 查找
scheme		listener 的模式，它可以是 HTTP 或 HTTPS。在默认情况下，这个模式将从转入的 AJP 请求里获取。
send-buffer		发送缓冲大小。
socket-binding		AJP Listener 的套接字绑定。
tcp-backlog		用指定的 backlog 配置服务器。
tcp-keep-alive		配置频道以依赖于实现的方式发送 TCP keep-alive 消息。
url-charset	UTF-8	URL 字符集。
worker	default	listener 的 XNIO 工作节点。

属性	Default	描述
write-timeout		配置套接字写超时（毫秒）。如果超过给定的时间而没有成功写入，套接字的下一次写入将抛出 {@link WriteTimeoutException}。

以下属性是只读的，并只在 **undertow** 子系统启用了统计功能时有效：

表 A.63. **ajp-listener** 度量属性

属性	Default	描述
bytes-received		这个 Listener 已接收了字节数量。
bytes-sent		这个 Listener 已发送出去的字节数量。
error-count		这个 Listener 已发送的 500 响应的数量。
max-processing-time		这个 Listener 上处理请求的最长时间。
processing-time		这个 Listener 处理所有请求的总共时间。
request-count		这个 Listener 已服务的请求的数量。

host 属性

表 A.64. **host** 属性

属性	Default	描述
alias		用逗号分开的主机列表。
default-response-code	404	如果设置，这将在请求的上下文不存在时送回响应代码。
default-web-module	ROOT.war	默认的 Web 模块。
disable-console-redirect	false	如果为 true，不会为这个主机启用控制台重定向。

filter-ref 属性

表 A.65. **filter-ref** 属性

属性	Default	描述
----	---------	----

属性	Default	描述
predicate		Predicate 提供根据交换进行 true/false 决定的简单途径。许多处理程序都要求有条件地进行应用，而 Predicate 则提供了指定条件的通用途径。
priority	1	定义过滤器顺序，它应该为 1 或更大的数字。较大的数字会比相同上下文里的内容较早地包含在处理程序链里。

access-log 属性



注意

当使用管理 CLI 管理 **access-log** 元素时，它在 **host** 元素的 **settings** 中可用。例如：

```
/subsystem=undertow/server=default-server/host=default-host/setting=access-log:add  
/subsystem=undertow/server=default-server/host=default-host/setting=access-log:read-resource
```

表 A.66. access-log 属性

属性	Default	描述
directory	\${jboss.server.log.dir}	保存日志的目录。
extended	false	日志是否使用扩展的日志文件格式。
pattern	common	访问日志模式。
predicate		确定是否应该记录请求的 Predicate。
prefix	access_log	日志文件名称的前缀。
relative-to		这个目录相对的路径
rotate	true	是否每天轮换访问日志。
suffix	.log	日志文件名称的后缀。
use-server-log	false	日志是否应该写入到服务器日志，而不是单独的文件。
worker	default	用于日志的工作节点的名称。

single-sign-on 属性



注意

当使用管理 CLI 管理 **single-sign-on** 元素时，它在 **host** 元素的 **settings** 中可用。例如：

```
/subsystem=undertow/server=default-server/host=default-host/setting=single-sign-on:add
/subsystem=undertow/server=default-server/host=default-host/setting=single-sign-on:read-resource
```



重要

虽然从应用程序的角度来看，分布式的单点登录和之前的 JBoss EAP 里的版本没什么不同，在 JBoss EAP 7 里，验证信息的缓存和分布的处理是不同的。对于 JBoss EAP 7，当运行 *HA* 配置集时，每个主机默认都有自己的 *infinispan* 缓存，用以保存相关的会话的 SSO Cookie 信息。这个缓存基于 Web 缓存容器的默认缓存。JBoss EAP 也将处理所有主机的单独缓存之间的信息传播。

表 A.67. single-sign-on 属性

属性	Default	描述
cookie-name	JSESSIONIDS SO	cookie 的名称
domain		将被使用的 Cookie 域。
http-only	false	设置 Cookie httpOnly 属性。
路径	/	Cookie 路径。
secure	false	设置 Cookie 安全属性。

location 属性

表 A.68. location 属性

属性	Default	描述
handler		这个位置的默认处理程序。

A.22. HTTP 方法的默认行为

和之前的 JBoss EAP 版本里的 **web** 子系统相比，JBoss EAP 7.0 里的 **undertow** 子系统具有不同的 HTTP 方法的默认行为。下表列出了 JBoss EAP 7.0 里的默认行为。

表 A.69. HTTP 方法的默认行为

HTTP 方法	JSP	Servlet	静态的 HTML
GET	OK	取决于它的实现	OK
POST	OK	取决于它的实现	NOT_ALLOWED
HEAD	OK	取决于它的实现	OK
PUT	NOT_ALLOWED	取决于它的实现	NOT_ALLOWED
TRACE	NOT_ALLOWED	NOT_ALLOWED	NOT_ALLOWED
DELETE	NOT_ALLOWED	取决于它的实现	NOT_ALLOWED
OPTIONS	NOT_ALLOWED	取决于它的实现	OK

A.23. IO 子系统属性

表 A.70. worker 属性

属性	Default	描述
io-threads		要使用的 IO 线程的数量
stack-size	0	栈的大小
task-keepalive	60	任务的 Keep-alive 时间
task-max-threads		任务的最大线程数量

表 A.71. buffer-pool 属性

属性	Default	描述
buffer-size		缓冲的大小
buffers-per-slice		每个 slice 有多少个缓冲
direct-buffers		缓冲池是否使用直接缓冲

A.24. REMOTING 子系统属性

表 A.72. remoting 属性

属性	Default	描述
worker-read-threads	1	为远程工作节点创建的读线程数量
worker-task-core-threads	4	用于远程工作节点任务线程池的核心线程的数量。
worker-task-keepalive	60	保持非核心远程工作节点任务线程活动的时间（毫秒）。
worker-task-limit	16384	在拒绝前允许远程工作节点任务的最大数量。
worker-task-max-threads	16	用于远程工作节点任务线程池的线程的最大数量。
worker-write-threads	1	为远程工作节点创建的写线程数量



重要

remoting 元素的上述属性已被启用。现在应该配置这些属性使用 **io** 子系统。

表 A.73. 端点属性

属性	Default	描述
auth-realm		如果没有指定验证 CallbackHandler 时使用的验证区。
authentication-retries	3	指定关闭连接前允许客户重试验证的次数。
authorize-id		SASL 验证 ID。它被用作如果没有指定验证 CallbackHandler 时使用的验证用户名，所选的 SASL 机制要求用户名。
buffer-region-size		分配的缓冲区的大小。
heartbeat-interval	2147483647	用于连接 Heartbeat 的间隔（毫秒）。如果在这段时间内，这个连接在转出方向是空闲的，Ping 消息将被发送，这将触发一个相应的回复信息。
max-inbound-channels	40	频道上并发的转入消息的最大数量。
max-inbound-message-size	9223372036854775807	允许的转入消息的最大尺寸。超过这个尺寸的消息在读取端和写入端都将抛出异常。
max-inbound-messages	80	连接支持的转入频道的最大数量。
max-outbound-channels	40	频道上并发的转出消息的最大数量。

属性	Default	描述
max-outbound-message-size	9223372036854775807	发送的转出消息的最大尺寸。大于这个值的消息不会被传送，试图这样做会在写入端抛出异常。
max-outbound-messages	65535	连接支持的转出频道的最大数量。
receive-buffer-size	8192	这个端点将通过连接接受的最大缓冲的尺寸。
receive-window-size	131072	连接频道的接收方向的最大窗口大小（字节）。
sasl-protocol	remoting	创建 SaslServer 或 SaslClient 的地方，可用来覆盖默认指定的协议 <i>remoting</i> 。
send-buffer-size	8192	这个端点将通过连接传输的最大缓冲的尺寸。
server-name		连接的服务器端在初始 greeting 时将它的名称传给客户，默认这个名称将从连接的本地地址自动发现，或者它可以用这个属性进行覆盖。
transmit-window-size	131072	连接频道的传输方向的最大窗口大小（字节）。
worker	default	要使用的工作节点



注意

当使用管理 CLI 来更新 *endpoint* 元素时，它位于 *remoting* 元素下的 *configuration*，例如：`/subsystem=remoting/configuration=endpoint/`。

连接器属性

连接器组件具有下列结构：

- connector
 - property
 - security
 - sasl
 - property
 - sasl-policy
 - policy

表 A.74. 连接器属性

属性	Default	描述
authentication-provider		<i>authentication-provider</i> 元素包含用于转入连接的验证提供者的名称。
sasl-protocol	<i>remote</i>	传入用于验证的 SASL 机制的协议。
security-realm		用于这个连接器的验证的相关安全区。
server-name		在初始消息交换里发送的、用于基于 SASL 验证的服务器名称。
socket-binding		套接字绑定附属的名称。

表 A.75. **property** 属性

属性	Default	描述
value		属性的值。

Security 属性

security 组件允许您配置连接器的安全性，但不包含直接的配置属性。它可以用嵌套组件进行配置，如 [sasl](#)。

表 A.76. **sasl** 属性

属性	Default	描述
include-mechanisms		可选的嵌套 <i>include-mechanisms</i> 元素包含一个允许的 SASL 机制名称的白名单。这个列表里没有出现的机制将不被允许。
qop		可选的嵌套 <i>qop</i> 元素包含一个降序排列的 quality-of-protection 值的列表。
reuse-session	false	可选的嵌套 <i>reuse-session</i> 布尔型元素指定服务器是否应该尝试重用之前验证的会话信息。这个机制可能不支持这样的重用，而其他因素也可能会阻止它。
server-auth	false	可选的嵌套 <i>server-auth</i> 布尔型元素指定服务器是否应该验证客户。不是所有的机制都支持这个设置。
strength		可选的嵌套 "strength" 元素包含一个降序排列的密码强度值的列表。

sasl-policy 属性

sasl-policy 组件允许您指定可选的策略以缩小可用的机制集合，但它不包含直接的配置属性。它可以用嵌套组件进行配置，如 [policy](#)。

表 A.77. *policy* 属性

属性	Default	描述
forward-secrecy	true	可选的嵌套 <i>forward-secrecy</i> 元素包含一个布尔值，它指定是否要求实现在会话间转发秘密的机制。转发秘密意味着插入某个会话不会自动提供插入以后的会话的信息。
no-active	true	可选的嵌套 <i>no-active</i> 元素包含一个布尔值，它指定是否允许易受主动（非目录）攻击的机制。 <i>false</i> 为允许，而 <i>true</i> 为拒绝。
no-anonymous	true	可选的嵌套 <i>no-anonymous</i> 元素包含一个布尔值，它指定是否允许接受匿名登录的机制。 <i>false</i> 为允许，而 <i>true</i> 为拒绝。
no-dictionary	true	可选的嵌套 <i>no-dictionary</i> 元素包含一个布尔值，它指定是否允许易受被动目录攻击的机制。 <i>false</i> 为允许，而 <i>true</i> 为拒绝。
no-plain-text	true	可选的嵌套 <i>no-plain-text</i> 元素包含一个布尔值，它指定是否允许易受简单被动攻击（如 <i>PLAIN</i> ）的机制。 <i>false</i> 为允许，而 <i>true</i> 为拒绝。
pass-credentials	true	可选的嵌套 <i>pass-credentials</i> 元素包含一个布尔值，它指定是否要求传递客户凭证的机制。

HTTP 连接器属性

http-connector 组件具有下列结构：

- [http-connector](#)
 - [property \(same as connector\)](#)
 - [security \(same as connector\)](#)
 - [sasl \(same as connector\)](#)
 - [property \(same as connector\)](#)
 - [sasl-policy \(same as connector\)](#)
 - [policy \(same as connector\)](#)

表 A.78. *http-connector* 属性

属性	Default	描述
----	---------	----

属性	Default	描述
authentication-provider		<i>authentication-provider</i> 元素包含用于转入连接的验证提供者的名称。
connector-ref		在 undertow 子系统中连接到的连接器名称。
sasl-protocol	<i>remote</i>	传入用于验证的 SASL 机制的协议。
security-realm		用于这个连接器的验证的相关安全区。
server-name		在初始消息交换里发送的、用于基于 SASL 验证的服务器名称。

出站连接属性

outbound-connection 组件具有下列结构：

- [outbound-connection](#)
 - [property](#)

表 A.79. outbound-connection 属性

属性	Default	描述
uri		出站连接的连接 URL。

表 A.80. property 属性

属性	Default	描述
value		属性的值。



注意

上面的 **property** 属性和将在连接创建过程中使用的 XNIO 选项相关。

远程出站连接

remote-outbound-connection 组件具有下列结构：

- [remote-outbound-connection](#)
 - [property \(same as outbound-connection\)](#)

表 A.81. remote-outbound-connection 属性

属性	Default	描述
outbound-socket-binding-ref		用来确定连接的目的地址和端口的 outbound-socket-binding 的名称。
protocol	<i>http-remoting</i>	用于远程连接的协议。默认是 http-remoting 。
security-realm		对用于获取密码和 SSL 配置的安全区的引用。
username		对远程服务器进行验证时使用的用户名。

本地出站连接属性
local-outbound-connection 组件具有下列结构：

- [local-outbound-connection](#)
 - [property \(same as outbound-connection\)](#)

表 A.82. local-outbound-connection 属性

属性	Default	描述
outbound-socket-binding-ref		用来确定连接的目的地址和端口的 outbound-socket-binding 的名称。

A.25. APACHE HTTP SERVER 的 MOD_CLUSTER 指令

mod_cluster 连接器是一个基于 Apache HTTP Server 的负载平衡器。它使用一个通讯频道来转发来自 Apache HTTP Server 的请求至一系列应用服务器节点中的一个。下列指令可以用来配置 mod_cluster。



注意
使用 *ProxyPass* 指令是没有必要的，因为 mod_cluster 会自动配置必须转发至 Apache HTTP Server 的 URL。

表 A.83. mod_cluster 指令

指令	描述	值
----	----	---

指令	描述	值
CreateBalancers	定义在 Apache HTTP Server VirtualHost 里如何创建平衡器。它允许这样的指令： ProxyPass /balancer://mycluster1/ 。	<ul style="list-style-type: none"> • 0：创在 Apache HTTP Server 里定义的所有虚拟主机 • 1：不创建平衡器（要求至少一个 <i>ProxyPass</i> 或 <i>ProxyMatch</i> 来定义平衡器名称） • 2：只创建主要的服务器（默认）
UseAlias	点击对应服务器名称的别名。	<ul style="list-style-type: none"> • 0：忽略别名（默认） • 1：检查别名
LBstatusRecalTime	负载均衡逻辑重新计算节点状态的时间间隔（秒）。	默认值：5 秒
WaitBeforeRemove	已删除的节点被 HTTPD 遗忘之前的时间（秒）。	默认：10 秒
ProxyPassMatch/ProxyPass	ProxyPassMatch 和 ProxyPass 都是 <i>mod_proxy</i> 的指令，当使用 ! （而不是 back-end URL）时，它阻止路径里的 reverse-proxy。它可以允许 Apache HTTP Server 提供静态内容。例如， ProxyPassMatch ^(/.*\..gif)\$! ，这个例子允许 Apache HTTP Server 直接提供 .gif 文件。	

如果所有其他的节点都已下线，*mod_cluster* 逻辑里的 hot-standby 节点是所有请求都去往的最后的应急节点。这和 *mod_proxy* 里的 hot-standby 逻辑类似。

要配置 hot-standby 节点，请用 **simple-load-provider** 替换 *modcluster* 子系统里的 *dynamic-load-provider*（将 factor 设置为 **0**），例如：

```
<subsystem xmlns="urn:jboss:domain:modcluster:2.0">
  <mod-cluster-config advertise-socket="modcluster" connector="ajp">
    <dynamic-load-provider>
      <load-metric type="cpu"/>
    </dynamic-load-provider>
    <simple-load-provider factor="0"/>
  </mod-cluster-config>
</subsystem>
```

在 *mod_cluster-manager* 控制台里，节点显示为 **OK** 状态和 **Load: 0**。

例如，有三个节点：

- Node A, Load: 10
- Node B, Load: 10
- Node C, Load: 0

负载将在节点 A 和 B 间进行平衡。如果两者都不可用，那么节点 C 将承担负载。

mod_manager

除了另有说明，mod_manager 指令的上下文总是 VirtualHost。server config 上下文暗示指令必须位于 VirtualHost 配置的外面。否则将显示错误信息且 Apache HTTP Server 不会启动。

表 A.84. mod_manager 指令

指令	描述	值
EnableMCPMReceive	允许 VirtualHost 从节点接收 MCPM。在 Apache HTTP Server 配置里包含 <i>EnableMCPMReceive</i> 以允许 mod_cluster 正常运行。将其保存在 VirtualHost 里您配置广告的地方。	
MemManagerFile	<i>mod_manager</i> 用来存储配置、生成用于共享内存或锁定文件的密钥的基础名称。它必须是一个绝对路径名称，目录根据需要来创建。我们推荐将这些文件放入本地磁盘而不是 NFS 共享位置。	\$server_root/logs/
Maxcontext	mod_cluster 支持的最大上下文数量	默认值： 100
Maxnode	mod_cluster. Context: server config 支持的最大节点数量	默认值： 20
Maxhost	mod_cluster 支持的最大主机（别名）数量。它也包含平衡器的最大数量。	默认值： 20
Maxsessionid	提供 <i>mod_cluster-manager</i> 处理程序里活动会话数量的活动 <i>sessionid</i> 的数量。当 mod_cluster 在 5 分钟内没有从会话接收任何信息时，会话就是非活动的。这个字段仅用于演示和调试目的。	0 ：逻辑未激活。
MaxMCMPMaxMessSize	来自其他 Max 指令的 MCMP 消息的最大尺寸	从其他 Max 指令计算的最小值： 1024

指令	描述	值
ManagerBalancerName	当 JBoss EAP 实例没有提供平衡器名称时使用的平衡器名称。	<i>mycluster</i>
PersistSlots	告诉 <i>mod_slotmem</i> 将节点、编码和上下文持久化至文件里。	Off
CheckNonce	当使用 <i>mod_cluster-manager</i> 处理程序时切换 <i>nonce</i> 的检查。	on/off 默认 : on - 已检查 Nonce
AllowDisplay	切换 <i>mod_cluster-manager</i> 主页上的其他显示。	on/off 默认 : off - 只显示版本
AllowCmd	允许使用 <i>mod_cluster-manager</i> URL 的命令。	on/off 默认 : on - 允许的命令
ReduceDisplay	减少 <i>mod_cluster-manager</i> 主页上显示的信息，从而可以显示更多的节点。	on/off 默认 : off - 显示完整信息
SetHandler mod_cluster-manager	<p>显示 <i>mod_cluster</i> 从集群获得的节点信息。这些信息包含普通信息以及活动会话的计数。</p> <pre><Location /mod_cluster- manager> SetHandler mod_cluster-manager Require ip 127.0.0.1 </Location></pre>	on/off 默认 : off



注意

当访问 **httpd.conf** 里定义的位置时：

- Transferred : 对应发送至后台服务器的 POST 数据。
- Connected : 对应当 *mod_cluster* 状态页面被请求时已处理的请求数量。
- Num_sessions : 对应 *mod_cluster* 报告为活动（过去 5 分钟内有请求）的会话数量。当 Maxsessionid 为 0 时，这个字段不会出现。它仅用于演示和调试目的。

A.26. MODCLUSTER 子系统属性

modcluster 子系统具有以下结构：

- [mod-cluster-config](#)
 - [dynamic-load-provider](#)
 - [custom-load-metric](#)
 - [load-metric](#)
 - [ssl](#)

表 A.85. mod-cluster-config 配置选项

属性	Default	描述
advertise	true	是否使用 Advertise 逻辑。
advertise-security-key		包含用于 Advertise 逻辑的安全密钥的字符串。
advertise-socket		用于 Advertise 套接字的套接字绑定的名称。
auto-enable-contexts	true	即使 mod_cluster_manager (httpd) 里禁用也启用上下文。
balancer		平衡器的名称。
connector		mod_cluster 反向代理将连接的 Undertow listener 的名称。
excluded-contexts		mod_cluster 应该忽略的上下文列表，它是用逗号隔开的格式化字符串。
flush-packets	false	启用/禁用 HTTPD 里的数据包冲刷。
flush-wait	-1	冲刷 HTTPD 里的数据包前等待的时间。最大值是 2,147,483,647 。
load-balancing-group		loadBalancingGroup 的名称。
max-attempts	1	处理幂等请求 (idempotent request) 的最多尝试次数。允许的值介于 -1 和 2,147,483,647 之间。
node-timeout	-1	节点的代理连接的超时时间 (秒)。允许的值介于 -1 和 2,147,483,647 之间。
ping	10	等待对 ping 的 pong 回复的时间 (秒)。
proxies		mod_cluster 用 socket-binding-group 里定义的 outbound-socket-binding 注册的代理的列表。


属性	Default	描述
proxy-list		<p>代理的列表。格式为用逗号隔开的字符串（hostname:port）。</p> <div>  <div> <p>注意</p> <p>proxy-list 属性已被弃用，转而使用 proxies 属性（对 outbound-socket-binding 引用的列表）。</p> </div> </div>
proxy-url	/	MCMP 请求的基础 URL。
session-draining-strategy	DEFAULT	卸载 Web 应用程序时的会话消耗策略。
simple-load-provider		简单的负载提供者。允许的值介于 0 和 2, 147, 483, 647 之间。
smax	-1	HTTPD 里的软空闲连接计数的最大值。允许的值介于 -1 和 2, 147, 483, 647 之间。
socket-timeout	20	等待 HTTPD 响应 MCMP 消息的超时时间。允许的值介于 1 和 2, 147, 483, 647 之间。
status-interval	10	STATUS 消息从应用范围器发送至反向代理花费的时间（秒）。允许的值介于 1 和 2, 147, 483, 647 之间。
sticky-session	true	对请求使用粘性会话。
sticky-session-force	false	不要用会话信息对请求进行失效切换。
sticky-session-remove	false	在失效切换时删除会话信息。
stop-context-timeout	10	等待上下文处理待定请求的最长时间。允许的值介于 1 和 2, 147, 483, 647 之间。
ttl	-1	空闲连接超过 smax 的存活时间（秒）。允许的值介于 -1 和 2, 147, 483, 647 之间。
worker-timeout	-1	HTTPD 里等待可用工作节点处理请求的超时时间。允许的值介于 -1 和 2, 147, 483, 647 之间。

表 A.86. dynamic-load-provider 配置选项

属性	Default	描述
decay	2	延迟。

属性	Default	描述
history	9	历史。

表 A.87. custom-load-metric 属性选项

属性	Default	描述
capacity	1.0	度量的容量。
class		自定义度量的类名。
property		度量的属性。
weight	1	度量的权重。

表 A.88. load-metric 属性选项

属性	Default	描述
capacity	1.0	度量的容量。
property		度量的属性。
type		度量的类型。
weight	1	度量的权重。

表 A.89. ssl 属性选项

属性	Default	描述
ca-certificate-file		证书机构。
ca-revocation-url		证书认证机构的撤销列表。
certificate-key-file	\${user.home}/.keystore	证书的密钥文件。
cipher-suite		允许的密码套件。
key-alias		密钥别名。
password	changeit	密码。
protocol	TLS	已启用的 SSL 协议。

属性	Default	描述
----	---------	----

A.27. MOD_JK 工作节点属性

`workers.properties` 文件定义 `mod_jk` 传入客户请求的工作节点的行为。`workers.properties` 文件定义不同的应用服务器所在的位置，以及如何在它们之间进行负载平衡。

属性的常用结构是 `worker.WORKER_NAME.DIRECTIVE`。`WORKER_NAME` 是必须匹配 JBoss EAP [undertow](#) 子系统里配置的 `instance-id` 的唯一名称。`DIRECTIVE` 是应用至工作节点的设置。

用于 Apache `mod_jk` 负载平衡器的配置引用

模板指定默认的 `per-load-balancer` 设置。您可以在 `load-balancer` 设置里覆盖这个模板。

表 A.90. 全局属性

属性	描述
<code>worker.list</code>	用逗号隔开的将被 <code>mod_jk</code> 使用的工作节点列表。

表 A.91. 强制的指令

属性	描述
<code>type</code>	工作节点的类型。默认类型是 <code>ajp13</code> 。其他可能的值还有 <code>ajp14</code> 、 <code>lb</code> 、 <code>status</code> 。关于这些指令的更多信息，请参考 https://tomcat.apache.org/connectors-doc/reference/workers.html 上的《Apache Tomcat Connectors Reference》。

表 A.92. 负载平衡指令

属性	描述
<code>balance_workers</code>	指定负载平衡器必须管理的工作节点。您可以对相同的负载平衡器多次使用这个指令。它由一个用逗号隔开的工作节点名称列表组成。
<code>sticky_session</code>	指定来自相同会话的请求是否总被路由至相同的工作节点。默认值是 <code>1</code> ，表示启用粘性会话。要禁用粘性会话，请将其设置为 <code>0</code> 。通常应该启用粘性会话，除非所有的请求都是完全无状态的。

表 A.93. 连接指令

属性	描述
<code>host</code>	后台服务器的主机名或 IP 地址。后台服务器必须支持 <code>ajp</code> 协议栈。默认值是 <code>localhost</code> 。

属性	描述
port	后台服务器实例侦听定义的协议请求的端口。默认值是 8009，这是 AJP13 工作节点的默认侦听端口。AJP14 工作节点的默认值是 8011。
ping_mode	<p>检测连接的网络状态依据的条件。检测使用空的 AJP13 数据包进行 CPing，期待 CPong 响应。使用指令标记组合来指定条件。这些标记不是用逗号或其他任何空格隔开的。ping_mode 可以是 C、P、I 和 A 的任意组合。</p> <ul style="list-style-type: none"> • C - Connect。在连接服务器后探测一次连接。用 connect_timeout 指定超时时间。否则将使用 ping_timeout。 • P - Prepost。在发送每个请求至服务器前探测连接。用 prepost_timeout 指令指定超时时间。否则将使用 ping_timeout。 • I - Interval。按 connection_ping_interval 指定的时间间隔探测连接，否则将使用 ping_timeout。 • A - 全部。C、P、I 的快捷方式，它表示使用所有的连接探测。
ping_timeout, connect_timeout, prepost_timeout, connection_ping_interval	上面的连接探测设置的超时时间。单位为毫秒，ping_timeout 的默认值是 10000。
lbfactor	<p>指定单独的后台服务器实例的负载平衡因子。为功能越强的服务器赋予越多的负载是很有用的。要赋予某个工作节点 3 倍于默认的负载，请将其设置为 3： worker.my_worker.lbfactor=3。</p>

下面的例子演示了用两个侦听端口 8009 工作节点（node1 和 node2）间的粘性会话进行负载平衡。

workers.properties 文件示例

```
# Define list of workers that will be used for mapping requests
worker.list=loadbalancer,status

# Define Node1
# modify the host as your host IP or DNS name.
worker.node1.port=8009
worker.node1.host=node1.mydomain.com
worker.node1.type=ajp13
worker.node1.ping_mode=A
worker.node1.lbfactor=1

# Define Node2
# modify the host as your host IP or DNS name.
worker.node2.port=8009
worker.node2.host= node2.mydomain.com
worker.node2.type=ajp13
worker.node2.ping_mode=A
worker.node2.lbfactor=1

# Load-balancing behavior
```

```
worker.loadbalancer.type=lb
worker.loadbalancer.balance_workers=node1,node2
worker.loadbalancer.sticky_session=1

# Status worker for managing load balancer
worker.status.type=status
```

关于 Apache mod_jk 的进一步配置超出了本文档的范畴，您可以在[Apache 文档](#)里找到它们。

A.28. 安全管理者子系统属性

security-manager 子系统自身没有可配置属性，但它有一个带有可配置属性的子资源：**deployment-permissions=default**。

表 A.94. 默认的配置选项

属性	Default	描述
maximum-permissions		赋予部署或 JAR 的最大权限集。
minimum-permissions		赋予部署或 JAR 的最小权限集。

Revised on 2018-01-12 05:20:26 EST