JBoss7 配置指南

1. jboss 各主要版本特性

1.1. jboss4 特性

JBoss4 包括 web 服务器(servlet/JSP 容器，HTML 服务器)、EJB2.0 容器。完整的纯 Java 的数据库引擎，(Java 消息服务)JMS，JavaMail,和 Java 事务处理 API/Java 事务处理服务（JTA/JTS）支持。早期的 JBoss 使用了 Apache Tomcat Web 服务器，但在 JBoss4.0 中已经吧 Apache Tomcat 内嵌到 JBoss 中了。后续又集成 Java 数据对象(JDO)，对于 JMS 多点传送机制支持的修补，对 J2EE1.4 的完全实现和分布式事务机制。

JBoss 的应用服务器控制和配置－JMX 机制，运行一次可以部署所有的组件和服务。资源属性和可配置参数可以通过 MBeans（可控制 beans）映射和更改，

这些控制可以在 JBoss 的控制台进行设置。一旦我们的 servlet-based 的应用程序被部署，JBoss 就自动安装一个部署 MBeans,这个 MBeans 会被添加到 JMX 控制台的导航菜单中。通过这个 MBean 就可以部署或卸载 WAR 应用程序，或查看应用程序相关的属性。

JBoss 4 基于 JBoss 3.2,在 J2EE 标准特性方面，主要的改进包括：

- JBoss 4 是业界第一家取得正式 J2EE 1.4 认证的应用服务器，完全符合规范的 J2EE 标准。
- 完全支持 J2EE web services（JAX-RPC 方式和 WS4EE 架构方式）和 SOA。
- 支持 AOP 模型,JBoss Aop 极大的提高了生产力。
- 与 Hibernate 紧密集成。
- 通过一个内建的 Caching 构架提升集群功能和分布式 Caching（TreeCache）。

JBoss4 完全遵循 J2EE1.4 标准，所以允许开发者在不同的应用服务器上重用 J2EE 组件（如 EJB 等），比如可以轻易的将部署在 Weblogic 或 Websphere 上的 EJB 迁移到 JBoss 上赖，JBoss4 比 JBoss3.2 实现了下面几个新的 J2EE 标准：

- JBoss4 支持 J2EE Web Services，包括 JAX-RPC 和 J2EE 架构的 Web Services，使用 EJB 提供安全的 Web Service 环境，它是基于 J2EE 的 SOA 实现。JBoss3.2 中旧的 JBoss.NET Web Services API 不再支持，新的 Web Service 实现是 WS BasicProfile-1.0 compliant。
- JBoss4 实现 JMS1.1 替代了 JBoss3.2 中的 JMS1.0
- JBoss4 实现了 JCA (Java Connector Architecture) 1.5 替代了 JBoss3.2 中的 JCA1.0
- JBoss4 实现了新的 Java Authorization Contract for Containers (JACC),JACC 是 JAVA2 一个基本的权限机制，为访问 EJB 方法和 web 资源赋予授权描述，即 J2EE 应用服务器和特定的授权认证服务器之间定义了一个连接的协约，新的实现在语法上基于 JBoss3.2，使用认证过的 Subject 声明 Roles，认证与 JAAS 的 authentication 保持一致。并且 security 配置，JBoss4 和 JBoss3.2 兼容。
- JBoss4 实现了 EJB2.1 规范.替代了 JBoss3.2 中的 EJB2.0 规范。

JBoss4 特性：

- JBoss4.2 必须需要安装 jdk5
- JBoss Ejb3 默认被安装
- JBoss 的 web 容器使用 JBoss Web v2.x (集成 tomcat6)
- deploy/jboss-web.deployer 目录替换了原先的 deploy/jbossweb-tomcat55.sar
- JBoss Transactions v4.2 为默认的事务管理器
- JBoss WS 提供 web service 功能
- JGroups/JBossCache 支持 channel multiplexing
- JBoss Remoting 更新到 stable 2.2.x，JBossMQ(JBoss4.0 使用)为默认 JMS 实现，但是可以使用 JBoss Messaging 替换。
- EJB 调用方式 由 rmi-invoker 替换为 JBoss Remoting 的 unified-invoker
- log4j 和 commons-logging 升级到新版本。

1.2. jboss5 特性

JBoss AS5 中，大部分显著的新特性添加都源自于要将所有主要的 JBoss 子系统带到下一个阶段去：

JBoss Messaging 1.4 现在取代了 JBossMQ，成为缺省的 JMS 提供者。除了透明的故障恢复和智能的消息重分发外，JBM 还支持即开即用的集群队列和主题。可以跨节点把消息复制到内存中，从而避免磁盘 I/O，或者能使用支持大消息的分页技术将消息持久化到任何流行的关系数据库中。JBM 证明，利用已完全出现的新的只扩展日志存储，原本就很卓越的性能和东西会变得更加优秀。

JBoss WebServices 3.0，完全支持 JAX-WS/JAX-RPC、XOP 和 SwA 的附件、还有一系列 WS-*标准。JBWS 转向了一个可插拔的架构，该架构允许更换底层的 WebServices 栈，所以你可以将 JBossWS-native 换成 Sun Metro 或 Apache CXF。这样的话，你就可以因地制宜，使用最合适 WebServices 栈。

为了改进可伸缩性和集群 Web 会话的钝化，AS5 中的集群支持 SFSB 的 Buddy 复制，以控制内存的使用。EJB3 Entity 和 Hibernate 缓存有了很大的改进，因为可以针对实体和查询使用不同的缓存，它们分别是失效缓存和复制缓存。在底层

的 JGroups 协议栈中，还有一些其它的性能优化。

JBoss Transactions 是 JBoss 5 默认的事务管理器。JBoss TS 已经与 JBoss 5 的 Servlet 容器——JBoss Web——一起在 AS 4.2 系列中进行了测试，JBoss Web 是基于 Apache Tomcat 的一个实现，支持原有的 APR-based 连接器，它在可伸缩性和性能上不但要达到，而且要超越 Apache Http 服务器的水平。

就 API 来说，AS5 是 Java EE 5 的实现，所有相关的 API 都会包含在内。对大部分 Java EE 5“新的”API 来说，比如 EJB3、JAX-WS、JPA 等，在 JBoss AS 4.2 系列中已经实现了，但由于 JBoss AS5 增加了 TCK 测试的覆盖范围，所以肯定会更为严格遵循规范。

JBoss5 应用服务器提供了大量的新功能：除了支持最新的 EJB 3.0 规范外，新版的 JBoss AOP 也正式发布。Web Services 方面，JBoss 现在支持全部的 J2EE Web Services，同时兼容 Microsoft.NET；Messaging 项目采用了完整的 JMS 1.1 实现，同时充分的改进了分布式目的单元格等功能的高可用性；JBoss Seam 中包括了一系列统一的革命性的组建设计模型和框架。同时 JBoss 5 中也集成了 Hibernate 3.2

JBoss AS 4.2 和企业应用平台的第一个版本（EAP 4.2）确实对 AS 5 造成了很大的影响。从零开始创建一个全新的内核、从 MBeans 转换到 POJO、在最底层集成 AOP、统一跨子系统的元数据处理、更改类加载系统、使部署器 Aspect 化，换句话说，就是改变内部架构、替换应用服务器的核心，同时还要保持与大部分已有服务的向后兼容性，为各种内部子系统引入合适的 SPI。长远看来这是好事，因为它允许最大的可插拔性，以及在不同的运行时环境中（比如独立的 EJB3 或嵌入到不同的应用服务器中）按需要选取使用各种 JBoss 项目。

JBoss AS5 不只是一个 Java EE 5 应用服务器。对下一代 JBoss 项目来说，它还寄托了成为最先进的服务器运行时环境的愿景。

1.3. jboss6 特性

JBoss AS6 最大亮点是对 Java EE 6 Web Profile 规范的支持，一份关于最流行的 Java EE 标准的报告中，排名前 5（JPA、JSP、EJB3、JSF 及 CDI）的都是 Java EE Web Profile 的必备组件。除了 Java EE 6 Web Profile 所需的这些组件外，AS 6

还提供了可选的经过认证的组件：RESTEasy 2.1.0——JAX-RS 1.1 规范的实现；HornetQ 2.1.2——JMS 1.1 规范的实现以及 JBoss Web Services CXF 栈——JAX-WS 2.2 规范的实现。

主要特性就是对 JBoss Injection 框架的完整实现。这对于满足 Java EE 6 平台规范所要求的 Resources、Naming 以及 Injection 是至关重要的。Infinispan v4.2.0 是个开源的数据网格平台，从 CR1 里程碑发布时就加入了，现在它也集成到了 JBoss AS 6 中，并且是默认的分布式缓存提供者。Infinispan 公开了一个兼容于 JSR-107 的 Cache 接口，你可以将对象存储其中。JBoss AS 6 服务器可以动态探测并注册到前端的 apache httpd 服务器。

对于性能来说，JBoss AS 5 与 6 之间有明显的变化。JBoss AS 6 对启动性能的提升很明显，现在的平均启动时间是 15 秒。用户能够感觉到这种改进，一定程度上是因为延迟了随 AS 一同发布的管理控制台应用的部署，转而以“按需”方式提供，同时还实现了 Timer Service 的延迟部署。Microcontainer（v2.2）的增强（包括新的注解扫描库的实现）极大降低了应用部署的时间。

1.4. jboss7 特性

JBoss AS7 可实现为云做好准备的架构，并可使启动时间缩短十倍，提供更快的部署速度并降低内在的占用。JBoss Enterprise Application Platform 6 的核心是 JBoss Application Server 7 的最新版本，该版本代表着 Java 应用服务器在从复杂和单一的形式转向更加轻便、模块化和敏捷的变革过程中的一个意义重大的里程碑。 该版本将使开发人员有重新思考如何开发和部署企业 Java 应用。

JBoss Application Server 7 构建于先前版本的良好基础之上，并提供更出色的性能、更低的内存占用率、分布式管理和 Java EE6 Web Profile 认证。它的新能力包括：

- Java Enterprise Edition（EE）6 Web Profile 认证，一种轻型的标准可移植 Java EE，专为开发和部署丰富的交换式 Web 应用而进行了优化。
- Java 上下文和依存关系插入（Java Context and Dependency Injection – CDI），这种标准化的统一框架支持类型安全的依存关系插入和定义完善的上下文生命周期，通过简化和优化代码的方式实现了代码的轻松编写、

测试和维护。

- Arquillian 测试，改善了对测试驱动式开发的支持，提供了远程和嵌入式组件测试，且不会生产完整企业 Java 容器所带来的不必要复杂性。
- 构建于轻型的高度优化的模块化服务容器和新型域模型基础上，使 JBoss Application Server 7 能够从最小的设备扩展至更大的关键任务集群。
- 通过基于 Eclipse 的 JBoss 工具来提供开发人员工具支持，改善了对 Java CDI、休眠、代表性状态传输和 Web 服务的支持。
- 全新的复杂域模型和丰富的管理 API，可实现强大的服务器和集群自动化。

2. 为什么 JBoss AS7 这么快

JBoss7 项目 lead Jason Green,最近在他的 blog 上，回答了这个问题. 以下是这篇 blog 的译文：

用 Ahmdahl 法则(高效并行)而不是 Moore 定律(等待硬件能有更高的时钟频率)来设计 JBoss7. 目前几乎每台台式机，笔记本和服务器都至少有两个 CPU 核心，而且多核的趋势还在继续。 CPU 时钟频率竞争的时代其实已经终结。所以软件也必须要适应这一趋势，充分利用硬件的计算能力。

JBoss AS 进行了一次重大的改变来获得这一关键的演进(evolution).我们重写了 AS7,使得它的整个架构是一个全新的，高性能的和可管理的。在令人惊谈的工程师的努力下，我们从在 github 上一个很小的原型，在一年多的时间里实现了今天看到的具有巨大工作量的遵循 Java EE Web Profile 标准的 J2EE 服务器(更不用说我们在这期间发布了 AS6,使得 JBoss 的用户可以早点得到关于 EE6 的新特性，新技术)。

在开始详细的解释以前，请允许我给出一些背景知识。应用服务器的核心问题是管理服务(service).在现代的应用服务器中几乎所有的部件都有生命周期，那就意味着在特定的时间点上，这个部件必须被启动，在以后的时间点，它必须被停止。 我们将所有具有生命周期的对象都看作是一个 service.另外一个 service 重要的属性是，service 和 service 的之间的依赖关系会影响到相应 service 的生命周期。举个例子，servlet 的 service 依赖于 web server.另外，如果这个 sevlet 使用

到其他的资源，比如数据库连接或者是 EJB,那么它也依赖于这些资源，这些依赖的资源可用以后自己才能启动。 但一个应用服务器启动或者部署时，它必须保证它能够按照正确的顺序将各种 service 启动。进一步，如果任何服务由于某种原因停止了，它必须能停止所有依赖于这个 service 的其他 sevice(以相对于启动相反的顺寻停止).这在单线程的环境下是一个简单的问题。

但 JBoss AS7 实在并行的启动和部署这些服务。这个复杂的问题通过我们全新的服务容器解决: JBoss Modula Service Container. MSC 是一个高级的并行状态机。它在运行中分析服务之间的依赖关系，并且在同一时间尽可能多的启动服务，但同时又遵循服务间的依赖关系。这就意味着不仅能够快速启动，而且能够并行的进行部署。

除了并行的 service,JBoss AS7 还有类模块化和并行的类加载技术。通过将类划分到恰当的类模块中，应用服务器可以自然地优化访问模式，仅仅查找一个点，就可以获得所需要的类。另外，由于限制了类模块之间的可见性，查找就没有没有那么大的开销。对于 JBoss Modules, 类模块解析和查找的时间复杂度是 O(1). 所有的这些操作都有很高的并行性，甚至大部分的类定义也是并行的。

AS7 对部署的处理也是高效的。一个主要的优化是我们通过快速扫描部分 class 来对 annotation 信息进行索引。为了取得更好的效率，我们允许模块预先生成空间效率指数(space efficient index)来更快的加载。另外一个部署时的优化，是我们谨慎的缓冲和再使用 relection data，因为 JVM 在这方面不是很有效率。

最后，另一个重要的原因我想强调的是我们已经并且会继续会守护 CPU 和内存在启动和部署方面的使用情况(尽可能的少占用内存和 CPU，做到尽可能的高效）。这就是在设计阶段就作出的好决定。一个有趣的例子是我们不再使用 JAXB(或者其他内省机制驱动的绑定器)来解析只读一次的配置文件。JAXB 或者其他采用内省机制的绑定器带来的是几乎用和真正做实际解析工作一样或者更到时间来处理如何解析。所有的这些意味着: 在 AS5 和 AS6 里处理 XML 的时间都比 AS7 的启动时间要长。

希望这些内容能够更好的从大的方面理解 AS7 如何获得高效性得以快速启动，为什么 JBoss7 与过去有很大的不同。这篇博客只是一个开始，继续关注我的下一个 blog,我将讨论 Jboss7 的 roadmap

3. JBoss AS7 中的新概念－域

JBoss AS7 新加入了域（domain）的概念并实现了相关功能。域的提出及实现，其目的是使得多台 JBoss AS 服务器的配置可以集中于一点，统一配置、统一部署，从而在管理多台 JBoss AS 服务器时，实现集中管理。本文详细介绍如何使用 AS7 的这一新特性。

3.1. 域(Domain)的概念及其与群集(Cluster)的区别

对于使用过 JBoss AS 过往版本的用户，可能对 AS 所提供的群集功能已经很熟悉了，在理解域的时候可能会遇到困难。那么域和群集有什么区别，用处上有什么不同呢？

总的来讲，JBoss 的群集的目的是提供：

- 负载平衡（Load Balance）
- 高可用（High Availablity）

而域的目的则是将多台服务器组成一个服务器组（Server Group），并为一个服务器组内的多台主机（Host）提供：

- 单点集中配置（通过一个域控制器，即 Domain Controller，实现组内主机的统一配置）
- 单点统一部署，通过域控制器将项目一次部署至组内全部主机。

简单来讲，群集的目标是让多台服务器分摊压力，当一台或多台服务器当机时，服务可以继续保持运转；而域的目标则是提供集中配置和管理多台服务器的能力。

在没有域的概念时，要想让群集内的多台服务器或几组服务器保持统一的配置，一个一个分别的去手工维护，是非常麻烦的事情，而域的引入解决了这一问题。

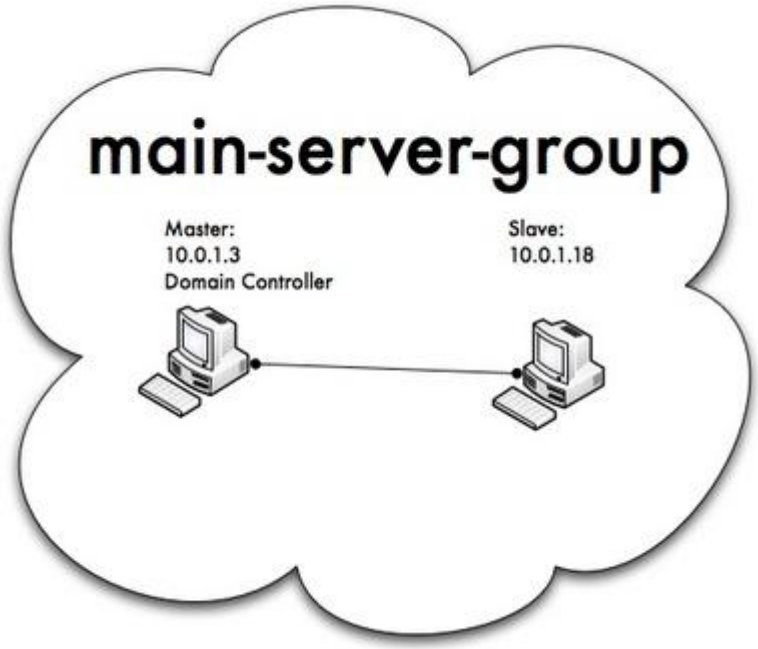
我们可以理解域和群集的相互关系是"正交（orthogonal）"的：通过一横一竖这两条轴，JBoss AS 为我们在运维方面提供了强大的可扩展能力。

3.2. 实验

熟悉了 AS7 中 Domain 的设计理念，接下来动手实际做个实验，看看 Domain 是如何在 AS7 中工作的。

1.1.1. 准备工作

使用两台电脑做为实验器材，两台电脑的 IP 分别为 10.0.1.3 及 10.0.1.18，分别运行 JBoss AS7，并组成一个服务器组（Server Group）。其中，使用 10.0.1.3 这台机器做为域控制器（Domain Controller）：



如上图所示，两台主机分别被命名为”master“及”slave“。通过配置，将 master 与 slave 组成一个服务器组，名为'main-server-group'，其中 master 将做为这个服务器组的域控制器。

需要说明一点的是，服务器组(Server Group)可以由多台服务器(Host)组成，并不一定只有两台，所以不要被 master 及 slave 这样的名字给迷惑了，以为一个服务器组仅支持一主一从两台 hosts。

本文中因为只使用两台服务器做为实验器材，因此出于方便角度将它们分别命名为 master 及 slave。

此外，在一个服务器组中，只有一台域控制器，在本实验中我们将使用 master

这台机器做为 domain controller。

1.1.2. 配置

AS7 由于经过了重新设计，因此在目录结构与配置文件上面与前一版本有了很大不同，对于熟悉了对 AS6 的配置和的人来讲，使用 AS7 会接触不少新概念和新思路。为了清楚表达，我会将一些与 AS6 及以前版本不同的地方做出必要的说明。

首先是 bin 目录中的内容：

Bash代码

```
liweinan@mg:~/projs/jboss-7.0.0.CR1/bin$ ls
domain.bat          jboss-admin.bat     standalone.conf     wsconsume.sh
domain.conf         jboss-admin.sh      standalone.conf.bat wsprovide.bat
domain.conf.bat     scripts             standalone.sh       wsprovide.sh
domain.sh           standalone.bat      wsconsume.bat
```

在 AS7 以前版本中，用来启动 JBoss 服务的 run.sh 不见了，取而代之的是 standalone.sh（独立运行模式）及 domain.sh（域运行模式）。我们稍后将使用 domain.sh 来运行 JBoss AS7，但首先要将两台 hosts 配置好，接下来讲解两台服务器的配置：

AS7 的目录结构和前一版本有很大不同，因为配置文件及其所在位置也有很大变动，下面是 AS7 的目录结构：

Bash代码

```
bin             docs            jboss-modules.jar standalone
bundles         domain          modules           welcome-content
```

可以看到有一个名为"domain"的目录，看一下这个 domain 目录里面的内容：

Bash代码

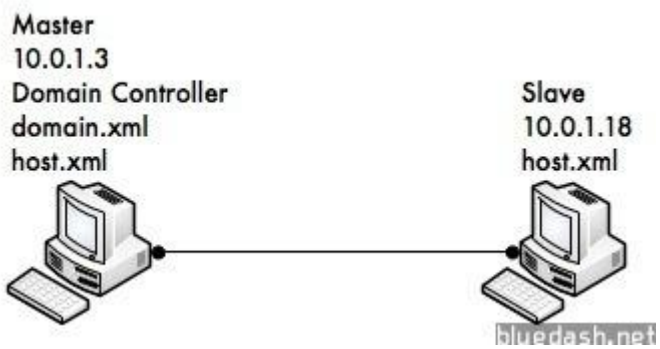
```
configuration content       lib         log         servers
```

这个目录中包含了 AS7 运行在 domain 模式下所需的配置及内容，其中名为“configuration”的目录里面含有我们所需要的配置文件：

Bash代码

```
liweinan@mba:~/projs/jboss-7.0.0.CR1/domain/configuration$ ls
domain-preview.xml    host.xml              mgmt-users.properties
domain.xml            host_xml_history
domain_xml_history    logging.properties
```

其中 domain.xml 和 host.xml 是我们需要关注的内容。我们需要对 master 及 slave 上面的配置文件分别进行配置：



从上图中可以看到，master 的 JBoss AS 中需要配置 domain.xml 及 host.xml 两个文件，其中 domain.xml 是做为域控制器必须配置的内容，host.xml 则是 master 及 slave 各自的 JBoss AS 都需要配置的文件。我们先从 master 上面的配置看起：

3.2.1.1. Master 上面的配置

3.2.1.1.1. domain.xml

Xml代码

```xml
<domain xmlns="urn:jboss:domain:1.0">

    <extensions>...

    <system-properties>...

    <profiles>
        <profile name="default">...

        <profile name="ha">...
    </profiles>

    <interfaces>...

    <socket-binding-groups>...

    <server-groups>
        <server-group name="main-server-group" profile="default">...
        <server-group name="other-server-group" profile="ha">...
    </server-groups>

</domain>
```

这个文件里面有几个部分是值得我们关注一下的：

extensions - 这一部分定义了域中服务器在启动时需要加载的模块。AS7 使用了全新设计的 JBoss Modules 来加载模块，大幅提高了服务器的启动。这一内容不是本文讲解重点，后续我会专门写一篇文章来介绍。目前了解到这一程度即可。

profiles - profiles 是 domain 中定义的一个核心概念，也是 domain 的核心组成部分。基于 profiles，AS7 便实现了域中各服务器的统一集中配置：用户可通过 profile 对各子系统（subsystem）进行配置，完成后将 profile 配置给某个或多个服务器组，各服务器组内的主机共用一份配置。

server groups - 服务器组的概念已经在前面的介绍中一再提及，也是 AS7 的域的设计中一个核心组成部分。在这里，AS7 默认定义了两个服务器组：main-server-group 及 other-server-group，它们分别使用'default' profile 及'ha' profile。在本实验中，我们将使用 main-server-group。

3.2.1.1.2. host.xml

Xml代码

```xml
<host xmlns="urn:jboss:domain:1.0" name="master">

    <management>
        <security-realms>...
          <management-interfaces>
          <native-interface interface="management" port="9999" />
          <http-interface interface="management" port="9990"/>
        </management-interfaces>
    </management>

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

    <interfaces>
        <interface name="management">
            <inet-address value="10.0.1.3"/>
        </interface>
        <interface name="public">
           <inet-address value="10.0.1.3"/>
        </interface>
    </interfaces>

    <jvms>...

    <servers>
        <server name="server-one" group="main-server-group">...
    </servers>
</host>
```

上面是一些 host.xml 中需要配置的关键内容，已经针对要做的测试做了一些配置上面的修改，以下是详细说明：

host name 按照我们的需要改成了"master"。

management - management 定义了服务器的管理端口，其中：9999 端口是所谓"native"二进制端口，后面的 jboss-admin.sh 管理命令会使用这个端口；9990 则提供基于 WEB 页面的管理端。我们等一下两种管理端口都会用到。

domain controller - 定义本服务器所需连接的 domain 控制器所在地址，因为 master 本身就是 domain controller，所以连接至本机 localhost 即可。

interfaces - management 及 public 接口服务所在的地址，我们要将其设为 slave 可以访问到的 IP 地址，保证 slave 可以连接至 host

servers - 一个物理主机实际上可以同时运行多台 JBoss AS7 的 Server，而每一台 Server 都可以配置到不同的服务器组去。在本实验中，我们使用最简的

情况，master 上面只跑一个 server-one，属于 main-server-group，把其它 AS7 默认设定的 server 可以都删掉，只留 server-one。

3.2.1.2. Slave 上面的配置

3.2.1.2.1. domain.xml

Slave 这台机器不作为域控制器而存在，因此不需要管它，也可以将 domain.xml 删掉或改名。

3.2.1.2.2. host.xml

Xml代码

```xml
<host xmlns="urn:jboss:domain:1.0" name="slave">

    <domain-controller>
       <remote host="10.0.1.3" port="9999"/>
    </domain-controller>

    <interfaces>
        <interface name="management">
            <inet-address value="10.0.1.18"/>
        </interface>
        <interface name="public">
           <inet-address value="10.0.1.18"/>
        </interface>
    </interfaces>

    <jvms>...

    <servers>
        <server name="server-one" group="main-server-group">...
    </servers>
</host>
```

上面的配置有几点需要说明：

* slave 里面，host name 指定为"slave"。
* domain-controller：指定为 master 的 IP：10.0.1.3，通过 9999 管理端口进行通讯。
* slave 上面，属于 main-server-group 的 server 也命名为"server-one"，这会和 master 上面的 server 冲突吗？实际上不会，因为两台同样名字的 server 运行在不同的 host 当中。

3.3. AS 7.1 的安全补充说明

从 JBoss AS 7.1 开始，对控制端口（9999 及 HTTP 端的 9990）的安全配置成为必须。因此需要补充下述安全配置方面的步骤：

首先要在 master 上面创建管理员的账号，在 bin 目录下有 add-user 工具可以帮我们创建账号密码并进行配置，我们创建一个管理员账号 admin，密码为 123123：

Bash代码

```
master:~/projs/as7/710/bin$ ./add-user.sh

Enter details of new user to add.
Realm (ManagementRealm) :
Username : admin
Password :
Re-enter Password :
About to add user 'admin' for realm 'ManagementRealm'
Is this correct yes/no? yes
Added user 'admin' to file 'master/as7/710/standalone/configuration/mgmt-users.properties'
Added user 'admin' to file 'master/as7/710/domain/configuration/mgmt-users.properties'
```

可以看到系统为我们分别在 domain 和 standalone 创建了 mgmt-users.properties，我们是用域模式运行，因此查看 domain/configuration/mgmt-users.properties 这个文件的最后一行：

Bash代码

```
admin=95333971266d87fbfa7d9963dd5e89d6
```

可以看到相关账号密码已经被创建。此时查看 host.xml：

Xml代码

```
<management>
    <security-realms>
        <security-realm name="ManagementRealm">
            <authentication>
                <properties path="mgmt-users.properties" relative-to="jboss.domain.config.dir"/>
            </authentication>
        </security-realm>
    </security-realms>
    <management-interfaces>
        <native-interface security-realm="ManagementRealm">...</native-interface>
        <http-interface security-realm="ManagementRealm">...</http-interface>
    </management-interfaces>
</management>
```

可以发现 host.xml 已经把安全配置应用起来了，使用 ManagementRealm 这个安全域进行认证。

同样地，我们需要在 slave 主机上进行一模一样的工作。

接下来，我们需要做一下 slave 对 master 的认证连接工作。slave 要想和 master 建立域控关系，需要知道 master 的管理端账号密码。在域控这一块，AS7 对认证有要求：需要在域控制器上以过来连接的主机 host 名为用户名添加账号。

我们在 slave 的 host.xml 文件中指定了 slave 的 host 名为 slave：

Xml代码

```xml
<host xmlns="urn:jboss:domain:1.0" name="slave">
```

因此，master 做为域控制器，需要在上面添加名为 slave 的管理员账号，密码仍为 123123：

Bash代码

```
master:~/projs/as7/710/bin$ ./add-user.sh

Enter details of new user to add.
Realm (ManagementRealm) :
Username : slave
Password :
Re-enter Password :
About to add user 'admin' for realm 'ManagementRealm'
Is this correct yes/no? yes
Added user 'slave' to file 'master/as7/710/standalone/configuration/mgmt-users.properties'
Added user 'slave' to file 'master/as7/710/domain/configuration/mgmt-users.properties'
```

这时再查看 mgmt-users.properties，可以看到多了 slave 账号：

Bash代码

```
admin=95333971266d87fbfa7d9963dd5e89d6
slave=f469d84edde53032bdac0a42bdedd810
```

接下来，我们要在 slave 主机的的 host.xml 做下认证配置，使用这个账号与 master 进行认证通信：

Xml代码

```xml
<management>
    <security-realms>
        <security-realm name="ManagementRealm">
                <server-identities>
                    <secret value="MTIzMTIz="/>
                </server-identities>
                <authentication>
                    <properties path="mgmt-users.properties" relative-to="jboss.domain.co
nfig.dir"/>
                </authentication>
            </security-realm>
    </security-realms>
    <management-interfaces>
        <native-interface security-realm="ManagementRealm">
            <socket interface="management" port="9999"/>
        </native-interface>
        <http-interface security-realm="ManagementRealm">
            <socket interface="management" port="9990"/>
        </http-interface>
    </management-interfaces>
</management>

<domain-controller>
   <remote host="10.0.2.1" port="9999" security-realm="ManagementRealm" />
   <!-- Alternative remote domain controller configuration with a host and port -->
   <!-- <remote host="192.168.100.1" port="9999"/> -->
</domain-controller>
```

上面的配置中有这些值得注意：

Xml代码

```xml
<domain-controller>
   <remote host="10.0.2.1" port="9999" security-realm="ManagementRealm" />
</domain-controller>
```

我们在认证域 ManagementRealm 中配置了 server-identities，这个认证域用在与域控制器 master 的连接方面。其中 secret value 是 domain 上对应 slave 主机名的那个账号的密码，用 base64 加密。我们在 master 上面配置的 slave 账号的密码为 123123，MTIzMTIz=则是 123123 的 base64 加密后的文字。这个配置用在连接 domain-controller 时进行认证：

有关更多的 AS7 的安全配置的信息，可查看官方文档：

Xml代码

```xml
https://docs.jboss.org/author/display/AS7/Securing+the+Management+Interfaces
```

关于 host.xml 的详细配置方法，也可参考 as7 目录下自带的 xsd 文档：

Bash代码

```
1. docs/schema/jboss-as-config_1_1.xsd
```

3.4. 部署

配置完成后，接下来便到了实际部署的阶段，我们将 master 和 slave 上面的 AS7 分别用 domain.sh 启动起来。

Bash代码

```
1. [Server:server-one] 21:17:14,491 INFO  [org.jboss.as] (Controller Boot Thread) JBoss A
   S 7.0.0.CR1 "White Rabbit" started in 6029ms - Started 109 of 163 services (54 services a
   re passive or on-demand)
2. [Host Controller] 21:18:02,635 INFO  [org.jboss.domain] (pool-3-thread-1) Registered remo
   te slave host slave
```

启动成功的话，应该可以在 master 上面看到上面的日志，slave 被成功的注册进来。

完成启动后，我们需要将待使用的 virtual-host 启动起来，当 AS7 以 domain 的方式启动时,默认是不启动任何 virtual server 的(在我目前使用的 7.0.0 CR1 White Rabbit 版本中是这样），我们可以在 domain.xml 中配置默认加载 virtual-host，也可以在服务器运行起来后，使用管理端命令动态的加载，在这里我准备使用后一种方式，从而讲解 AS7 管理端的使用方法。

在 AS7 的 bin 目录下面有一个 jboss-admin.sh， 这是 AS7 提供的全新的管理工具，我们使用这个工具，连接至 master：

Bash代码

```
1.  ./jboss-admin.sh
2. You are disconnected at the moment. Type 'connect' to connect to the server or 'help' fo
   r the list of supported commands.
3. [disconnected /] connect 10.0.1.3
4. Connected to domain controller at 10.0.1.3:9999
```

可以看到，我们已经连接到了 master 的 9999 管理端口。接下来可以查看"default"这个 profile 当中的 web 模块的运行情况：

Bash代码

```
[domain@10.0.1.3:9999 /] /profile=default/subsystem=web:read-children-names(child-type=connector)
{
    "outcome" => "success",
    "result" => ["http"]
}
```

可见 http 服务器已经启动，由于我们的"main-server-group"使用的是 default 这个 profile，因此，服务器组中的两台 host 的 web 模块接受 profile 的统一配置，都是已启动的。继续看一下 web 模块中的细节：

Bash代码

```
[domain@10.0.1.3:9999 /] /profile=default/subsystem=web/connector=http:read-resource(recursive=true)
{
    "outcome" => "success",
    "result" => {
        "protocol" => "HTTP/1.1",
        "scheme" => "http",
        "socket-binding" => "http",
        "ssl" => undefined,
        "virtual-server" => undefined
    }
}
```

注意到 virtual-server 的状态是未定义（undefined），我们要想将一个 web 项目部署进服务器组中的各个 host，就必须加载一个待部署的 virtual-server，因此我们使用命令来添加：

Bash代码

```
[domain@10.0.1.3:9999 /] /profile=default/subsystem=web/virtual-server=other.com:add
{
    "outcome" => "success",
    "result" => {"server-groups" => [("main-server-group" => {
    "master" => {
        "host" => "master",
        "response" => {"outcome" => "success"}
    },
    "slave" => {
        "host" => "slave",
        "response" => {
            "outcome" => "success",
            "result" => undefined
        }
    }
})]}
}
```

可以看到，我们之前在 domain.xml 中配置的 "other.com" 这个 virtual host 被成功添加了。

接下来是部署 WEB 应用的环节，我们首先用 maven 制作一个最简单的 web 项目，仅包含一个欢迎页面：

Bash代码

```
mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-webapp -DarchetypeArtifactId=maven-archetype-webapp
```

生成的项目如下：

Bash代码

```
.
|-- pom.xml
`-- src
    `-- main
        |-- resources
        `-- webapp
            |-- WEB-INF
            |   `-- web.xml
            `-- index.jsp
```

使用如下命令将项目打成 WAR 包：

Bash代码

```
mvn package
```

得到 war：

Bash代码

```
target
`-- my-webapp.war
```

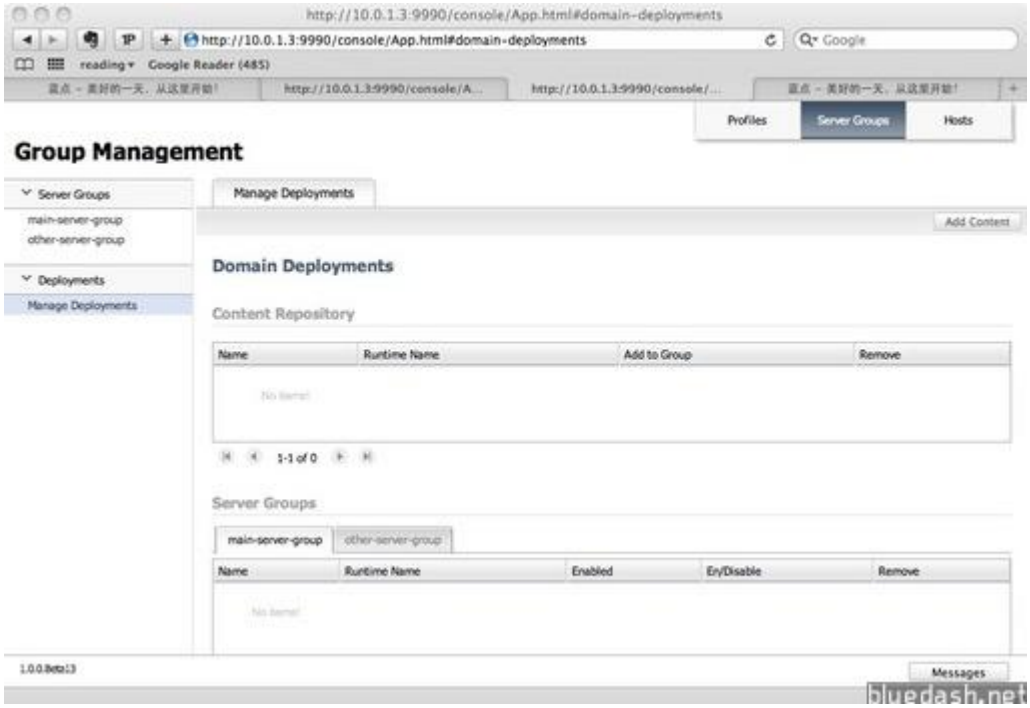
接下来是部署这个 war 包，对于本次实验来讲，关键的部分在于能否通过 domain 提供的 server group 管理功能，一次将一个项目部署进 server group 中的多台服务器。我们接下来就验证这点，顺便看下 AS7 提供的 WEB 管理功能，打开 WEB 浏览器，访问 master 的 HTTP 端口的管理地址：

Html代码

```
http://10.0.1.3:9990/console/App.html
```

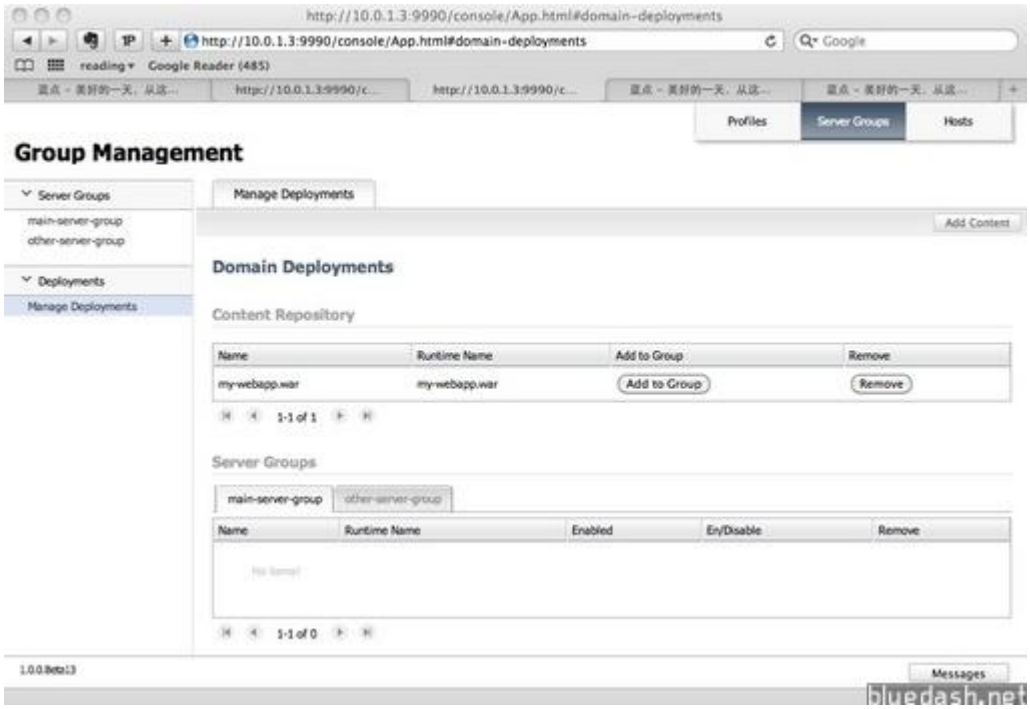
可以看到，管理页识别出 AS7 正运行在 domain 模式之下，并且目前共有两台主

机运行（左上角的列表分别列有 master 及 slave）。我们要关注的是 server-group：点击右上角的"Server Groups"，进入服务器组的管理页面，然后点击左边的"Manage Deployments"页面，进入部署功能页面：

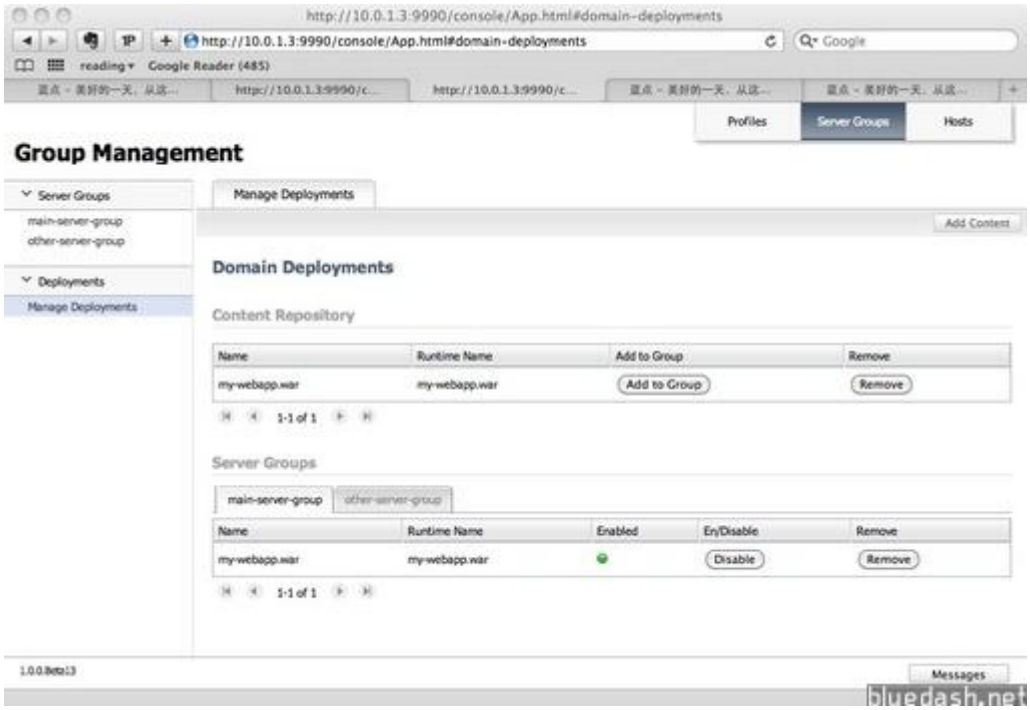


可以看到，目前还没有任何资源被加至服务器组，此时点击右边的"Add Content"功能，将 my-webpp.war 添加进 Content Repository（域控制器用于保存待部署资源的目录）。

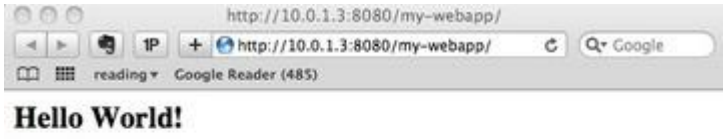
添加完成后如下图所示：



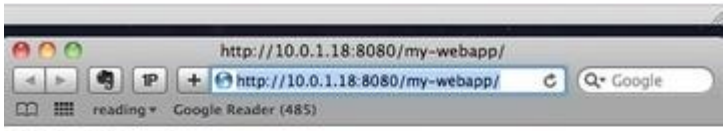
然后点击"Add To Group"将 my-webapp.war 添加至 "main-server-group"，并将其 enable，一切顺利的话结果如下所示：



此时我们预期的结果应该是 my-webapp.war 被同时部署至 master 及 slave 了，分别试着访问 master 及 slave 的 http 资源，看看是否都部署上 my-webapp 这个应用了：



Hello World!



Hello World!

结果如我们所预期的那样，两台服务器都可以访问到这个部署的资源。通过对一个点（Domain Controller）的配置与部署，我们实现了多 AS7 服务器的集中管理。

3.5.小结

通过域这个概念，实现了多服务器统一管理，统一配置，资源统一部署的目标。通过集中管理，我们可以在此基础上再进行群集的划分与部署，实现群集内多台服务器的单点配置与管理。可以说 AS7 的 Domain 概念的引入，与群集的概念组合在一起，通过一横一从两条轴，形成了完整的坐标系。

4. JBoss7 配置

4.1. 目标听众

这篇文档是为需要安装配置 JBoss Application Server(AS7)的人员编写。

4.1.1. 开始之前

你需要知道如何下载，安装和运行 JBoss Application Server7. 如果你还不了解这些信息， 请参考“入门指导"。

4.1.2. 手册中的示例

手册种大部分的例子会使用部分 XML 配置文件或者是 de-typed 的管理模型进行表示 。

4.2. 客户端

JBoss AS7 提供三种不同的方式对服务器进行配置和管理: web，命令行和 xml 配置文件形式。无论你选择什么样的配置方式，配置信息都会被同步到各个方式的管理界面上，并且被存储到 xml 配置文件中。

4.2.1. web 接口

web管理客户端是一个GWT的应用，它通过HTPP管理接口来管理域(domain)或者是单独运行(standalone)的服务器。

4.2.1.1. HTTP 管理接入点

基于 HTTP 协议的管理接入点负责接入 使用 http 协议与管理层进行交互 客户端。它负责接收使用 JSON 编解码的协议和 de-typed RPC 形式的的 api 来对可管理的域服务器或者单独运行服务器进行管理操作。web 控制台就是通过它来实

现的，但基于 HTTP 协议的管理接入点也可以与其他的管理终端进行集成，交互。

基于 HTTP 协议的管理点会运行在域控制器(domain controller)或者是单独运行服务器上，默认运行在 9990 端口上。

Xml代码

```xml
<management-interfaces>
 [...]
 <http-interface interface="management" port="9990"/>
<management-interfaces>
```

(参见 standalone/configuration/standalone.xml 或者 domain/configuration/host.xml)

基于 HTTP 协议的管理接入点运行在两个不同的 context 下。一个用于运行管理的操作 另外一个提供对 web 管理接口的访问。

- 域 API: http://<host>:9990/management
- Web 控制台: http://<host>:9990/console

4.2.1.2. 访问管理控制台

管理控制台和基于 HTTP 协议管理的 API 在统端口上运行，可以通过以下 URL 进行访问：

- http://<host>:9990/console

默认访问web管理页面的URL:http://localhost:9990/console.

4.2.1.3. 对管理控制台进行加密

web 管理控制台通过 HTTP 管理接口来对服务器进行通信。对于如何机密 HTTP 管理接口以及如何启用默认的安全域，请参考一下本文中关于“加密管理接口"章节。

4.2.2. 命令行接口

命令行方式的管理工具（CLI）提供了对域和单独运行服务器的管理。用户可以使用命令行来连接域服务器或者单独运行服务器,通过传输 de-typede 的管理模型来执行管理操作。

4.2.2.1. Native 管理接入点

Native 的管理接入点负责接入使用 AS 内部协议与管理层进行交互的客户端.它使用基于 java 对象来描述的管理操作、二进制协议和 RPC 形式的 API 来对域和单独运行服务器进行管理操作。命令行方式的管理工具使用它来实现对服务器的管理，单 Native 管理接入点也提供了极强的集成能力，可以和其他的客户端进行集成。

Nativeg 管理接入点运行在 host 控制器上或者是一个单独运行服务器上。如果使用命令行管理工具，Native 管理接入点必须被启用.默认 Native 管理接入点运行在 9999 端口上：

```
<management-interfaces>
<native-interface interface="management" port="9999"/>
[...]
<management-interfaces>
```

参见 standalone/configuration/standalone.xml 或者 domain/configuration/host.xml)

4.2.2.2. 运行命令行管理工具

根据操作系统，使用 JBossAS7 bin 目录下的 jboss-admin.sh 或者 jboss-admin.bat 来启动命令行管理工具.关于 AS7 目录的详细信息，请参考"入门指南"。

命令行工具启动以后的第一件事情就是连接被管理的 Jboss AS7 实例。我们通过命令 connect 进行：

Java代码

```
./bin/jboss-admin.sh
You are disconnected at the moment. Type 'connect' to connect to the server
or 'help' for the list of supported commands.
[disconnected /]

[disconnected /] connect
Connected to domain controller at localhost:9999

[domain@localhost:9999 /] quit
Closed connection to localhost:9999
```

localhost:9999 是 JBossAS7 域控制器客户端连接的默认主机和端口名.主机名和端口都是可选的参数，可以被单独或者一起指定。想要退出对话，可以键入 quit

命令来结束。

Xml代码

```
1.  jboss-admin脚本可以接收--cnnect参数: ./jboss-admin.sh --connenct
```

help 命令用来显示参考帮助文档：

Shell代码

```
1.  [domain@localhost:9999 /] help
2.  Supported commands:
3.
4.  cn (or cd)              - change the current node path to the argument;
5.  connect                 - connect to the specified host and port;
6.  deploy                  - deploy an application;
7.  help (or h)             - print this message;
8.  history                 - print or disable/enable/clear the history expansion.
9.  ls                      - list the contents of the node path;
10. pwn (or pwd)            - prints the current working node;
11. quit (or q)             - quit the command line interface;
12. undeploy                - undeploy an application;
13. version                 - prints the version and environment information.
14.
15. add-jms-queue           - creates a new JMS queue
16. remove-jms-queue        - removes an existing JMS queue
17. add-jms-topic           - creates a new JMS topic
18. remove-jms-topic        - removes an existing JMS topic
19. add-jms-cf              - creates a new JMS connection factory
20. remove-jms-cf           - removes an existing JMS connection factory
21.
22. data-source             - allows to add new, modify and remove existing data sources
23. xa-data-source          - allows to add new, modify and remove existing XA data sources
```

查看特定命令的详细帮助文档，需要在命令后加"--help"参数来获得。

4.2.2.3. 管理请求

管理请求允许与管理模型进行低级别的交互。它不同于高级别的命令(比如创建一个 jms 的 queue 命令:create-jms-queue),使用管理请求可以对服务器的配置像对直接对 xml 配置文件进行编辑而进行读和修改操作。整个配置用一个有地址的资源树进行表示，这个树上的每个节点提供一系列的操作供执行。

一个管理请求包含三个部分：地址，操作名和可选的操作参数

这是一个管理请求的规约：

规约代码

```
1.  [/node-type=node-name (/node-type=node-name)*] : operation-name [( [parameter-name=parame
    ter-value (,parameter-name=parameter-value)*] )]
```

举个例子：

Shell代码

```
/profile=production/subsystem=threads/bounded-queue-thread-pool=pool1:write-core-threads (count=0, per-cpu=20)
```

Tab补全代码

```
Tab补全:
所有的命令和参数都支持tab补全。比如结点的类型，名字，操作名和参数名。我们正考虑增加别名在tab补全时可以显示更少的选项，然后在后台翻译成相应的管理请求。
```

管理请求字符串之间的空格是不敏感的。

4.2.2.3.1. 管理资源的地址

管理请求可以不含有地址信息和参数,比如::read-resource, 可以列出当前Node 下的所有节点类型。

在管理命令中，为了消除歧义需要以下几个前缀：

- " :" --- 在当前节点上执行操作，比如：

  [subsystem=web] :read-resource(recursive="true")

- " ./" ---- 在当前节点的子节点上执行操作,如：

  [subsystem=web] ./connector=http:read-resource
  这个操作的全路径地址是: subsystem=web,connector=http.

- " /" --- 在根节点上执行操作,如：

  [subsystem=web] /:read-resource 或子节点 : [subsystem=web] /subsystem=logging:read-resource

4.2.2.3.2. 操作类型和操作描述列表

操作的类型可以分为在任何节点上的通用操作和在特殊节点上的特殊操作(如:subsystem).通用的操作包括：

操作列表代码

```
add
remove
read-attribute
write-attribute
read-children-names
read-children-resources
read-children-types
read-operation-description
read-operation-names
read-resource
read-resource-description
```

对于特殊操作列表（比如在 logging 子系统上可以进行的特殊操作),可以通过管理的节点进行查询。比如，查询一个单独运行服务器上 logging 子系统上所支持的操作：

Shell代码

```
[[standalone@localhost:9999 /] /subsystem=logging:read-operation-names
{
   "outcome" => "success",
   "result" => [
       "add",
       "change-root-log-level",
       "read-attribute",
       "read-children-names",
       "read-children-resources",
       "read-children-types",
       "read-operation-description",
       "read-operation-names",
       "read-resource",
       "read-resource-description",
       "remove-root-logger",
       "set-root-logger",
       "write-attribute"
   ]
}
```

可以看出，logging 支持三个额外特殊的操作:change-root-log-level ，set-root-logger and remove-root-logger

进一步关于被管理节点描述或者被管理节点上操作的描述，可以通过一下命令查询：

Shell代码

```
[standalone@localhost:9999 /] /subsystem=logging:read-operation-description(name=change-root-log-level)
{
   "outcome" => "success",
   "result" => {
       "operation-name" => "change-root-log-level",
       "description" => "Change the root logger level.",
       "request-properties" => {"level" => {
           "type" => STRING,
           "description" => "The log level specifying which message levels will be logged by this logger.
                            Message levels lower than this value will be discarded.",
           "required" => true
       }}
   }
}
```

Java代码

```
<STRONG>递归模式查看全部信息输入::read-resource(recursive=true).</STRONG>
```

4.2.2.4. 命令行历史信息

命令行(和操作请求)历史信息默认是开启的。历史信息在内存中和硬盘文件中都有保存,并且命令行历史信息在命令行对话之间保存。

命令行历史信息文件信息保存在名为.jboss-cli-history 的文件中，这个文件会在用户的 home 目录下自动创建。当启动命令行模式时，这个文件会被读入内存中来对初始化命令行历史信息。

在命令行对话中，你可以使用上下键来向前和向后查阅命令行历史信息。

命令行历史可以通过 history 命令进行操作。如果 history 命令执行时不带参数，它会将内存中所有的历史命令和操作打印出来(取决于历史信息的最大个数，默认 500). history 命令支持 3 个可选的参数：

Shell代码

```
1. disable-关闭历史记录功能(但不会清除已经记录的历史信息)
2. enable-开启历史记录功能(从上次关闭历史记录前的最后一条记录开始)
3. clear-清除内存种的历史记录(但不会清除文件中的信息)
```

4.2.2.5. 批处理

批处理模式允许用户以将一组命令和操作按照原子的方式执行。如果一个命令或者操作失败，那么在批处理中成功执行的子命令将会被回滚。

不是所有的命令都可以批处理种执行。比如: cd, ls, help 等不能被转换成操作请求的就不可以在批处理种执行。 只有可以转换成为操作请求的命令才可以在批处理种执行。批处理的命令实际上是以组合操作请求的方式执行的。

执行 batch 命令进入批处理模式：

Shell代码

```
1. [standalone@localhost:9999 /] batch
2. [standalone@localhost:9999 / #] /subsystem=datasources/data-source="java\:\/H2DS":enabl
   e
3. [standalone@localhost:9999 / #] /subsystem=messaging/jms-queue="newQueue":add
```

run-batch 执行一个批处理：

Shell代码

```
[standalone@localhost:9999 / #] run-batch
The batch executed successfully.
```

退出批处理编辑模式并且不丢失更改：

Shell代码

```
[standalone@localhost:9999 / #] holdback-batch
[standalone@localhost:9999 /]
```

稍后重新激活批处理：

Shell代码

```
[standalone@localhost:9999 /] batch
Re-activated batch
#1 /subsystem=datasources/data-source=java:/H2DS:\/H2DS:enable
```

还有一些比较重要的批处理命令(使用 tab 补全来查看以下列表)：

Shell代码

```
clear-batch
edit-batch-line (e.g. edit-batch line 3 create-jms-topic name=mytopic)
remove-batch-line (e.g. remove-batch-line 3)
move-batch-line (e.g. move-batch-line 3 1)
discard-batch
```

4.2.3. 配置文件

域管理和单服务器的 xml 配置可以在 configuration 子目录下找到：

Shell代码

```
domain/configuration/domain.xml
domain/configuration/host.xml
standalone/configuration/standalone.xml
```

一个被管理的域有两种类型的配置:一种是对整个域的配置(domain.xml)另外一种是对每个加入到域里主机(host)的配置(host.xml).关于如何配置域拓详细信息请参考"域配置"章节。xml 配置是核心可靠的配置源。任何通过 web 接口或者命令行方式对配置的更改都持久化到 XML 配置文件中.如果一个域或者单独服务器离线,xml 配置文件也可以进行手动更改，任何更改都在下一次启动时生效。

但是，我们鼓励用户使用 web 接口或者命令行方式更改配置文件，而不是采用离线编辑的方式对配置文件进行更改。对正在处理的配置文件进行的外部更改将不会被探测到，从而有可能会被覆盖。

4.3. 核心管理概念

4.3.1. 运行模式

JBoss Application Server 7 可以被启动到两个不同的模式.域模式可以用来运行和管理多个 jboss 服务器的拓扑， 或者是单服务器模式，仅运行一个服务器的实例

4.3.1.1. 单服务器模式

对于大多数的使用来说，通过管理域实现的中心管理能力是不需要的。对于这些使用场景，一个 jboss7 的实例可以被运行成单服务器模式。一个单服务器的实例是一个独立的进程，像 JBoss3 ,4,5 或 6 的实例，可以通过 standalone.sh 或者 standalone.bat 进行启动。

如果需要多个服务器的实例或者多服务器的管理，那么就需要用户来协调管理多个服务器。比如:在所有的单服务器上部署一个相同的应用，用户需要在每台服务器上进行操作。更为可能，用户会启动多个单独运行的服务器来组成高可用的集群，就像是使用 JBoss 3, 4 5 和 6 那样。

4.3.1.2. 管理域

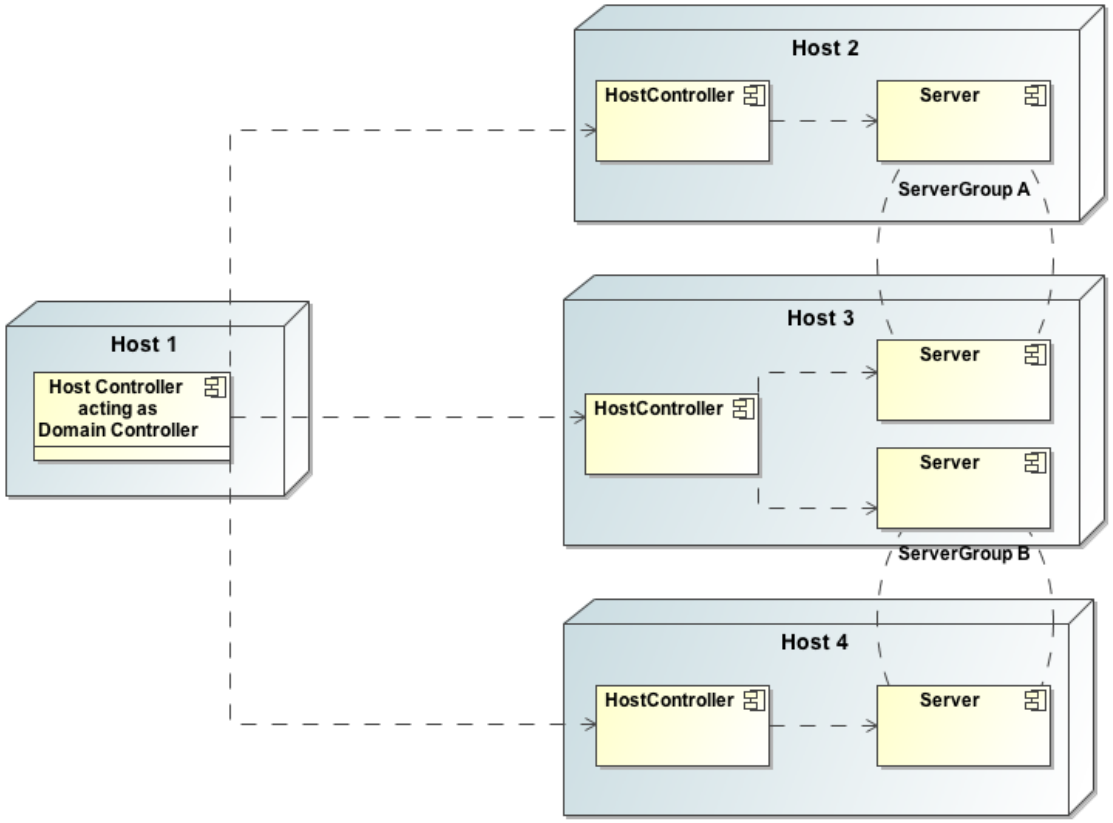
JBoss7 一个最重要的新特性就是可以通过一个管理点来管理多个 JBoss7 服务器实例.一个域包含一个 DomainController 进程（域的中心管理点），这些被管的服务器是这个域的成员。

在这个域里所有的服务器实例共享统一的管理策略，域控制器来保证每个服务器都使用这一管理策略来配置。域可以横跨几个物理（或者虚拟)机器，所有的 JBoss7 服务实例运行在一个受 HostController 进程控制的给定主机（host)上。一个 HostController 的实例会被配置成为中心的 DomainController.在每个主机上的 HostController 可以与 DomainController 进行交互来控制运行在自己主机(host)上服务器实例的生命周期，并且帮助 DomainController 来管理。

当你通过 domain.sh 或者 domian.bat 来在一个主机上运行 JBoss7 管理域时，

那么你的意图是去运行一个 HostController，并且一般是至少运行一个 JBoss7 的服务器实例,而且在主机上的 HostController 应该被配置来充当 DomainController.

下图是一个管理域的拓扑：



4.3.1.2.1. Host(主机)

上图中的每个 Host 方框代表一个物理或者虚拟的主机。一个物理的主机可以包含零个，一个或者多个服务器实例(server instance)。

4.3.1.2.2. 主机控制器(HostController)

当 domain.sh 或者 domain.bat 在主机上运行时，一个叫 HostController 的进程也会被启动。HostContoller 只负责管理服务器实例；它不会处理服务器实例的日常工作。HostController 负责在自己的主机上启动停止单个服务器实例的进程，并且与 DomainController 进行交互来管理这些服务器实例。

HostController 默认在主机文件系统中 JBoss7 安装目录下读取 domain/configuration/host.xml 文件。host.xml 文件主要包含特定主机的信息：

主要是：

- 在安装要运行的 JBoss7 服务器的实例名。
- 关于 HostContraller 如何与 DomainController 联系，并且注册到 DomainController 种来获取配置的信息。这个配置信息可以是如何查找联系一个远端的 DomainController,或者是告诉 HostController 本身就可以充当 DomainController
- 特定于本地安装的各种配置信息，如在 domain.xml 里(见以下内容)中 interface 的配置信息可以被映射成 host.xml 中的 IP 地址信息。在 domain.xml 中的抽象路径信息可以被映射成 host.xml 的真实文件系统信息。

4.3.1.2.3. Domain Controller(域控制器）

一个 HostController 的实例被配置成整个域的中心管理点，就成为了 DomainController.DomainController 的主要负责维护域的管理策略，保证所有的 HostController 能够获取目前的配置信息，以及协同 HostController 来保证运行的服务器实例都根据当前的策略来配置。中心管理策略默认存储 DomainController 安装主机的 domain/configuration/domain.xml 中。

domain.xml 必须位于运行 Domain Controller Jboss7 安装目录下的 domain/configuration. 如果不作为 Domain Controller 运行的 AS7 则不需要这个文件;比如运行连接到远程 DomainController 的 HostController 的服务器。但不作为 DomainController 运行 HostController 的 AS7 安装中存在这个文件，也不会有影响。

domain.xml 文件包括各种配置，在 Domain 下的 JBoss7 的实例可以配置各种 profile 去运行。一个 profile 的配置包含各种组成 profile 的 subsystem 的详细配置信息(比如，一个集成的 jboss web 实例就是一个 subsystem,一个 JBoss TS 的事务

管理器也是一个 subsystem）.Domain 的配置信息也包括在 subsystem 需要用到的 socket 组的定义。Domain 配置信息也包含 Server group 的定义。

4.3.1.2.4. Server Group (服务器组)

Server group 是一组被统一管理和配置的服务器实例。在一个管理域里，每个服务器实例都是服务器组的一个成员(即使是一个组里只有一个服务器，这个服务器仍然是一个组的成员)。Domain Controller 和 Host Controller 会保证在一个 server group 里所有的 server 具有一致的配置。这些服务器被配置成同样的 profile, 并且部署相同的内容。

一个 domain 可以有多个 server group.上面的图示种给出了两个 server group："ServerGroupA"和“ServerGroupB"。不同的 Server group 可以被配置成不同的 profile 和部署不同的内容:比如在一个 domain 在不同层的服务器来提供不同的服务。不同的 Server group 也可以运行同样的 profile，部署相同的内容；比如对应用进行升级时候，为了避免整个服务不可用，可以首先对一个 server group 中应用进行升级,然后再对另外一个 sever group 升级。

sever group 定义的例子如下：

```
<server-group name="main-server-group" profile="default">
    <socket-binding-group ref="standard-sockets"/>
    <deployments>
        <deployment name="foo.war_v1" runtime-name="foo.war" />
        <deployment name="bar.ear" runtime-name="bar.ear" />
    </deployments>
</server-group>
```

一个 server-group 的配置包含以下不可缺省的属性：

- name -- server group 名
- profile -- server 运行在的 profile 名

另外，还有以下可选配置：

- socket-binding-group -- 定义在 server group 中用到的默认的 socket binding group 名， 可以在 host.xml 里覆盖。如果在 server-group 里没有定义，那么必须在每个 server 的 host.xml 里定义。
- deployments -- 在组服务器要部署的内容。
- system-properties -- 组服务器种要设置的所有的 system properties
- jvm -- 在组服务器种默认的 jvm 设置。Host Controller 将会合并在 host.xml 里提供的属性，并且用这些属性来启动服务器的 JVM.详细配置信息选项请参考 JVM 配置。

4.3.1.2.5. Server (服务器）

在上述图表中的 server 表示一个实际的应用服务器实例。Server 运行于独立域 HostController 的 JVM 进程中。Host Controller 负责启动这一 JVM 进程。(在管理域里，终端用户不能直接从命令行里启动一个 server 进程)。

HostController 合并整理 domain.xml 里的域配置信息和从 host.xml 上获得的主机配置信息。

4.3.1.3. 决定运行在单独服务器或者管理域上

什么用例适合管理域，什么适合单独服务器(standalone server)？管理域协调多个服务器的管理--通过 JBoss7 提供的中心节点，用户可以管理多个服务器,通过域管理的丰富功能来统一服务器的配置，通过协调方式将服务器配置变更(包括部署内容）在各个服务器上生效。

重要的是要理解选择管理域和单独服务器要根据你的服务器是如何管理的,而不是根据为终端用户请求提供什么样服务的能力。管理域和单独服务器的差别对于高可用集群也是十分重要的。理解高可用性对于运行的单独服务器和管理域是正交的。也就是说,一组单独服务器可以被配置成为高可用性集群。管理域和

单服务器模式决定服务器是如何管理的，而不是他们所提供的功能：

所以,考虑到以上原因：

- 如果只有一个服务器，运行在域模式下不会有更多的好处,运行在单服务器模式下是更好的选择。
- 对于多服务器生产环境，运行在域模式和单服务器模式下取决于用户是否想使用管理域提供的中心管理。某些企业已经开发出自有成熟的多服务器管理方式，并且能够轻松协调管理多个单独服务器。读于这些企业，由多个单独运行服务器组成的多服务器架构是一个好的选择。
- 单服务器更适合与大多数的开发环境。任何在管理域下运行的服务器的配置都可以在单服务器模式运行的服务器获得,因此即使应用是将来要运行在域管理模式的生产环境中，很多(几乎是全部)开发都可以在单服务器模式下进行。
- 运行管理域对于一些高级的开发是有帮助的。比如:需要多 JBoss7 服务器实例之间进行交互的开发。开发人员会发现配置多个服务器作为域成员是进行多服器集群的有效方式。

4.3.2. 通用的配置概念

以下通用的配置概念对于管理域模式和单服器模式都适用：

4.3.2.1. Extensions (扩展)

一个扩展(是一个能扩展服务器功能的模块）. JBoss 7 的内核是简单清量级的。所有用户需要用到应用服务器的功能都是通过扩展提供的。一个扩展被打包成为在 modules 目录下的一个模块。用户如果需要一个特别的扩展，则需要在 domain.xml 或者 standalone.xml 里加入<extension/> xml 元素来指明这个模块名。

```
<extensions>
    [...]
    <extension module="org.jboss.as.transactions"/>
    <extension module="org.jboss.as.web" />
```

```
    <extension module="org.jboss.as.webservices" />
    <extension module="org.jboss.as.weld" />
</extensions>
```

4.3.2.2. Profile 和 subsystem(子系统)

在 domain.xml 和 standalone.xml 配置中最重要的部分是一个(在 standalone.xml)或者多个(在 domain.xml 里)profile 的配置。一个 profile 是一个命名的子系统集合。一个子系统是使用一个扩展添加到和服务器核心的一组功能(参考以上的扩展)。一个子系统可以提供处理 servlet 的功能;一个子系统可以提供 EJB 容器，一个子系统可以提供 JTA，等等。一个 profile 是命名的子系统的列表,并且包含各个子系统详细的配置信息。 一个服务器拥有大量子系统的 profile 会提供丰富的功能.一个拥有数量少并且功能专注的子系统提供的功能相应减少，但是具有更少的内存消耗。

domain.xml 和 standalone.xml 里关于 profile 的配置看上去大致相同，唯一的不同是 standalone.xml 只允许有一个 profile 的 xml 元素(服务器运行的 proifle),但 domain.xml 可以有多个 profile,每一个 profile 可以映射到一个或者多个服务器组。

domain.xml 和 standalone.xml 里关于子系统的配置是相同的。

4.3.2.3. Paths(路径)

路径是一个文件系统路径的逻辑名。在 doamin.xml,host.xml 和 standalone.xml 配置种都包含用来来声明路径的部分。其他的配置可以通过逻辑名来引用这些路径，而不需要包含路径的所有全部信息(在不同的机器都不相同).比如: logging 子系统的配置包含对 jboss.server.log.dir 路径的引用来指向 server 的 log 目录:

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

JBoss7 自动提供一系列的标准路径，而不需要用户在配置文件中配置.

jboss.home - JBossAS 安装的跟目录

user.home - 用户的 home 目录

user.dir - 用户当前的工作路径

java.home - java 安装路径

jboss.server.base.dir - 一个服务器实例的跟目录

jboss.server.data.dir - 服务器存储数据的目录

jboss.server.log.dir - 服务器日志文件目录

jboss.server.tmp.dir - 服务器存储临时文件目录

jboss.domain.servers.dir -host Controller 在此目录为服务器实例创建的工作区(仅在管理域模式下)

用户可以通过在配置文件中使用<path>xml 元素来增加自己的路径或者覆盖除了上面前五个路径的配置。

```
<path name="example" path="example"
relative-to="jboss.server.data.dir"/>
```

Path 的属性：

name -- 路径名

path -- 实际的物理文件系统名,如果没有 relative-to 的定义，将会被处理成为绝对路径.

relative-to -- (可选)前面定义的路径名，或者系统提供的标准路径。domain 里<path>配置不可以包含 path 和 relative-to 属性,只需要 name 属性。

```
<path name="x"/>
```

上面这个配置的示例简单的说明:有意个叫"x"的路径，在 domain.xml 配置的其他部分可以引用。指向 x 的实际文件系统的路径是主机相关的，需要在每个机器的 host.xml 里定义。如果这种在 domain.xml 里定义 path 的方式被使用，那么在每个机器上 host.xml 里都需要 path 来有定义实际的文件系统路径:

<path name="x" path="/var/x" />

在 standalone.xml 里<path/>都必须包含实际文件系统的路径信息。

4.3.2.4. nterfaces (接口)

接口就是对 socket 可以绑定到的一个物理接口,IP 地址或者主机名的逻辑命名。domain.xml,host.xml 和 standalone.xml 的配置信息种都包行有接口声明的部分。其他部分的配置可以根据这些逻辑名来引用这些接口,而不需要包含这些接口全部详细的信息(这些接口信息在不同的机器上不尽相同)。一个接口的配置包含接口的逻辑名，也包含解析这个接口名成为真实物理地址的信息。详细信息请参考接口和端口部分。

在 domain.xml 里<interface/>元素只需要 name 属性,不需要包含任何真实 IP 地址的信息:

<interface name="internal"/>

这样一个配置简单的表明:"有一个叫 internal 的接口在 domain.xml 的其他部分可以引用。指向 internal 的 IP 地址是和主机相关的，地址信息将会在每台机器的 host.xml 里指定。"如果使用这一方法，那么在每台机器上的 host.xml 里必须有 interface 元素来指定 IP 地址:

```
<interface name="internal">
   <nic name="eth1"/>
</interface>
```

在 standalone.xml 里的<interface/>元素必须包含 IP 地址信息。

4.3.2.5. socket binding(socket 绑定)和 socket binding group(socket 绑定组)

socket 绑定是对一个 socket 命名的配置。
在 doamin.xml,和 standalone.xml 配置种都包含用来来声明 socket 命名的部分。其他的配置可以通过逻辑名来引用这些 socket，而不需要包含 socket 的所有全部信息(在不同的机器都不相同).请参考 interfaces 和 ports 部分。

4.3.2.6. System Properties(系统属性)

系统属性值可以在 domain.xml, host.xml 和 standalone.xml 里的多个地方设置.standalone.xml 里设置的值会成为 server 启动进程的一部分。在 domain.xml 和 host.xml 设置的值将在 serer 启动时生效。

当在 domain.xml 或者 host.xml 里设置的一个系统属性,它是否能够最终被应用生效取决于它在什么地方被配置。如果系统属性作为在 domain.xml 里根节点下的一个子孙节点被设置，那么它将在所有的 server 上生效。如果在 domain.xml 中<server-group/>里的<system-property/>设置，那么它将在这个组里所有的 server 生效。在 host.xml 里根节点下作为一个子节点设置系统属性，那么它将在这个主机的 host controller 控制的所有 server 上生效。最后,在 host.xml 中<server/>里的<system-property>设置，那么它将在只在那个 sever 上生效。同样的属性可以被配置在多个地方:<server/> 中的值要优先于在 host.xml 根节点中直接定义的值，host.xml 里定义的值要优先于任何 domain.xml 里的值，<server-group/>里定义的值要优先于通过 domain.xml 里根节点里定义的值。

4.3.3. Management resources(管理资源)

当 JBOss7 在启动的时候解析配置文件，或者当使用 AS7 的管理接口的时候，都是在增加，删除或者修改 AS7 内部管理模型的管理资源。AS7 的管理资源具有以下特征:

4.3.3.1. Address (地址)

所有 JBossAS 的管理资源都以树的结构进行组织。指向一个管理资源树节点的路径就是管理资源的地址。每个资源地址的片段都是键值对:

键是在双亲上下文中资源的类型.因此，单服务器模式运行的服务器根资源就有以下类型的子孙:子系统，接口，socket 绑定等等。提供 AS7web 服务器能力的子系统就有类型是 connector 和 virtual-server 的子孙。提供 AS7 消息服务器的子系统就有 jms-queue 和 jms-topic 类型的子孙节点。

值给定类型特定资源的名字。比如:子系统里的 web 或者 messaging, 子系统 connector 里的 http 或者 https.

管理资源的全路径是一个排好序的键值对的列表,这个地址可以指从资源数的根指向这个资源。地址中使用"/"来分割地址元素，使用"="来分割键和值:

```
/subsystem=web/connector=http
/subsystem=messaging/jms-queue=testQueue
/interface=public
```

如果使用 HTTP API,那么"/:来分割键和值而不是"=":

```
http://localhost:9990/management/subsystem/web/connector/http
http://localhost:9990/management/subsystem/messaging/jms-queue/testQueue
http://localhost:9990/management/interface/public
```

4.3.3.2. operations(操作)

查询更改一个管理资源的状态要通过一个操作。一个操作具有一下特征:

- 字符串名:
- 零个或者多个命名的参数.每个参数都有一个字符串名和一个类型是 org.jboss.drm.ModelNode 值(或者当通过 CLI 调用时，ModelNode 用文本内容表示，通过 HTTP APi 调用时候，model node 用 JSON 对象表示)。参数是可选的:
- 返回值也是一个类型是 org.jboss.dmr.ModelNode 的值(或者当通过 CLI 调用时，ModelNode 用文本内容表示，通过 HTTP APi 调用时候，model node 用 JSON 对象表示)。
- 除了根节点的资源，每个资源应该有一个 remove 操作("这里应该是因为在 AS7.0 时很多资源都没有)。add 操作的参数会根据资源而不同。remove 操作没有参数:

全局的操作都适用于所有的资源.详细内容请参考全局操作部分。

一个资源所支持的操作可以通过调用这个资源本身的一个操作获得:read-operation-names.一旦知道了操作的名,关于操作的参数和返回的详细信息就可以通过调用 read-operation-description 来获得。比如，在单独运行服务器上获取根节点资源所支持的操作名,然后得到一个操作全部的详细信息，在 CLI 中可以通过以下来获得:

```
[standalone@localhost:9999
/] :read-operation-names
{
    "outcome" => "success",
    "result" => [
        "add-namespace",
        "add-schema-location",
        "delete-snapshot",
        "full-replace-deployment",
        "list-snapshots",
        "read-attribute",
        "read-children-names",
        "read-children-resources",
        "read-children-types",
        "read-config-as-xml",
        "read-operation-description",
```

```
        "read-operation-names",
        "read-resource",
        "read-resource-description",
        "reload",
        "remove-namespace",
        "remove-schema-location",
        "replace-deployment",
        "shutdown",
        "take-snapshot",
        "upload-deployment-bytes",
        "upload-deployment-stream",
        "upload-deployment-url",
        "validate-address",
        "write-attribute"
    ]
}
[standalone@localhost:9999
/] :read-operation-description(name=upload-deployment-url)
{
    "outcome" => "success",
    "result" => {
        "operation-name" => "upload-deployment-url",
        "description" => "Indicates that the deployment content
available at the included URL should be added to the deployment content
repository. Note that this operation does not indicate the content should
be deployed into the runtime.",
        "request-properties" => {"url" => {
            "type" => STRING,
            "description" => "The URL at which the deployment content is
available for upload to the domain's or standalone server's deployment
content repository.. Note that the URL must be accessible from the target
of the operation (i.e. the Domain Controller or standalone server).",
            "required" => true,
            "min-length" => 1,
            "nillable" => false
        }},
        "reply-properties" => {
            "type" => BYTES,
            "description" => "The hash of managed deployment content that
has been uploaded to the domain's or standalone server's deployment
content repository.",
            "min-length" => 20,
            "max-length" => 20,
            "nillable" => false
```

```
        }
    }
}
```

如何获取一个资源所支持的操作请参考以下 Description 部分。

4.3.3.3. Attributes(属性)

管理资源将它们的状态暴露成为属性。属性有 string 类型名，和一个类型是 org.jboss.drm.ModelNode 值(或者当通过 CLI 调用时，ModelNode 用文本内容表示，通过 HTTP APi 调用时候，model node 用 JSON 对象表示)。

属性可以是只读或者是可读写的。读写属性值可以通过全局的 read-attribute 和 write-attribute 操作来进行。

read-attribute 操作仅有一个"name"参数，它的值是这个 atrribute 的名。比如在 sorkcet-binding 资源里通过 CLI 来读 port 属性:

```
[standalone@localhost:9999 /]
/socket-binding-group=standard-sockets/socket-binding=https:read-attr
ibute(name=port)
{
    "outcome" => "success",
    "result" => 8443
}
```

如果一个属性是可写的，资源的状态可以通过 write-attribute 操作来改变。这个操作接受两个参数:

name - 属性名

value - 属性值

比如在 sorkcet-binding 资源里通过 CLI 来设置 port 属性:

```
[standalone@localhost:9999 /]
/socket-binding-group=standard-sockets/socket-binding=https:write-att
ribute(name=port,value=8444)
{"outcome" => "success"}
```

属性可以有两种存储类型:

CONFIGURATION - 表示属性值保存在持久化的配置中，比如管理资源配置要从这些文件读取的:domain.xml,host.xml 或者 standalone.xml.

RUNTIME - 表示属性之日仅仅保存在运行的服务器中而不是持久化的配置中。 比如一个 metric(如已经处理的请求数)十一个 RUNTIME 属性一个典型的例子。

管理资源暴露出的所有属性值可以通过"read-resource"操作再加上“include-runtime=true"的参数来获得，比如通过 CLI:

```
[standalone@localhost:9999 /]
/subsystem=web/connector=http:read-resource(include-runtime=true)
{
    "outcome" => "success",
    "result" => {
        "bytesReceived" => "0",
        "bytesSent" => "0",
        "errorCount" => "0",
        "maxTime" => "0",
        "processingTime" => "0",
        "protocol" => "HTTP/1.1",
        "requestCount" => "0",
        "scheme" => "http",
```

```
        "socket-binding" => "http",
        "ssl" => undefined,
        "virtual-server" => undefined
    }
}
```

省略 include-runtime(或者设置成 false)来限制仅仅存储在持久化配置上的值才被输出。

```
[standalone@localhost:9999 /]
/subsystem=web/connector=http:read-resource

{
    "outcome" => "success",
    "result" => {
        "protocol" => "HTTP/1.1",
        "scheme" => "http",
        "socket-binding" => "http",
        "ssl" => undefined,
        "virtual-server" => undefined
    }
}
```

参考一下 Description 相关部分来获取更多关于一个特定资源暴露出属性的信息。

4.3.3.4. Children（子节点)

管理资源可以支持子资源。一个资源孩子节点的类型(比如 web 子系统资源的 connector 子节点)可以通过查询资源的 description 来获得(参考以下

Description 部分)或者通过调用"read-children-types"操作。当你知道了子节点的类型，你就可以通过全局"read-children-types"操作和已知节点类型来获得全部子节点名。这个操作仅接受一个参数类型:child-type,它的值即使已知的子节点类型。比如，一个表示 socketbinding 组的资源有孩子节点。使用 CLI 来获得这些子节点的类型和给定类型所有的资源:

```
[standalone@localhost:9999 /]
/socket-binding-group=standard-sockets:read-children-types
{
    "outcome" => "success",
    "result" => ["socket-binding"]
}
[standalone@localhost:9999 /]
/socket-binding-group=standard-sockets:read-children-names(child-type
=socket-binding)
{
    "outcome" => "success",
    "result" => [
        "http",
        "https",
        "jmx-connector-registry",
        "jmx-connector-server",
        "jndi",
        "osgi-http",
        "remoting",
        "txn-recovery-environment",
        "txn-status-manager"
    ]
}
```

4.3.3.5. Descriptions(描述)

所有的管理资源都暴露用于描述管理资源属性，操作和子节点类型的元数据(metadata).通过调用一个或者多个被管理资源所支持的全局操作来获取.以上我们给出了 read-operation-names, read-operation-description, read-children-types 和 read-children-names 操作的例子。

read-resource-description 操作用来获得一个资源属性，子节点的详细信息。比如,通过 CLI:

```
[standalone@localhost:9999 /]
/socket-binding-group=standard-sockets:read-resource-description
{
    "outcome" => "success",
    "result" => {
        "description" => "Contains a list of socket configurations.",
        "head-comment-allowed" => true,
        "tail-comment-allowed" => false,
        "attributes" => {
            "name" => {
                "type" => STRING,
                "description" => "The name of the socket binding group.",
                "required" => true,
                "head-comment-allowed" => false,
                "tail-comment-allowed" => false,
                "access-type" => "read-only",
                "storage" => "configuration"
            },
            "default-interface" => {
                "type" => STRING,
                "description" => "Name of an interface that should be
```

```
used as the interface for any sockets that do not explicitly declare one.",
                "required" => true,
                "head-comment-allowed" => false,
                "tail-comment-allowed" => false,
                "access-type" => "read-write",
                "storage" => "configuration"
            },
            "port-offset" => {
                "type" => INT,
                "description" => "Increment to apply to the base port
values defined in the socket bindings to derive the runtime values to use
on this server.",
                "required" => false,
                "head-comment-allowed" => true,
                "tail-comment-allowed" => false,
                "access-type" => "read-write",
                "storage" => "configuration"
            }
        },
        "operations" => {},
        "children" => {"socket-binding" => {
            "description" => "The individual socket configurtions.",
            "min-occurs" => 0,
            "model-description" => undefined
        }}
    }
}
```

注意在上述例子中的输入:"operations" => {}"。如果命令行已经包含参数

operation=true,(比如
/socket-binding-group=standard-sockets:read-resource-description(operations=true))，那么操作的结果会包含这个资源所支持的每个操作的描述。

关于被管理资源上运行 read-resource-description 和其他全局操作所支持的其他参数的详细信息，请参考全局操作部分。

4.3.3.6. 和 JMX Beans 相比

JBossAS 管理的资源在概念上和 Open MBeans 极为相似。他们有一下主要的几点不同:

- JBossAS 的管理资源通过树形结构进行组织。资源地址的键值顺序是不同的，因为它定义了被管理资源在数形结构上的位置,而 JMX ObjectName 的 key 属性顺序不是很重要。

- Open MBean 属性的值，操作参数的值和操作的返回值，必须是 JDK 规定的简单类型(String,Boolen,Integer 等等) 或者实现了 javax.management.openmbean.CompositeData 或者 javax.management.openmbean.TabularData 的对象。JBossAS 管理资源的属性值，操作参数值和操作的返回值都是 org.jboss.dmr.ModelNode 类型。

4.3.3.7. 管理资源树的基本结构(management resource trees)

如同以上提到的，被管理资源以树形结构组织。数的结构决定于你运行在单服务器模式还是管理域模式。

4.3.3.7.1. 单服务器模式(Standalone server)

单服务器模式下管理树的树形结构和 standalone.xml 配置文件的十分相似:

根节点资源:

extension - 在服务器上安装的扩展。

path - 服务器上可用的路径。

system-property - 配置文件中设置的系统属性 (如不在命令行种设置的属性)

core-service=management - 服务器中核心的管理服务。

core-service=service-container - JBoss7 的最核心资源 JBoss MSC ServiceContainer

ubsystem - server 上安装的子系统.大部分的管理模型都是子系统类型的子节点。

interface - 接口配置

socket-binding-group - server 上 socket 绑定组的中心资源

socket-binding - 单个 socket 绑定的配置

deployment - server 上已经部署的内容

4.3.3.7.2. 管理域模式 (managed domain)

在管理域模式下，管理资源树会跨越整个域，包含域范围的配置(如在 domain.xml 上定义的配置),和主机相关的配置(如在 host.xml 配置的内容)以及每个运行应用服务器暴露出的管理资源。在管理域中 Host Controller 进程提供对整个资源树的访问.如果 Host Controller 是主 Domain Controller，那么资源树中对于每个主机相关信息都是可得到的,如果 Host Controller 是远程 Domain Controller 的从属(slave)，那么仅与 Host Controller 所在的 host 相关部分的资源树的是可以访问的。

整个域的根资源如下。这些资源包括它的子孙，除了 host 类型都会被持久化到 domain.xml 中。

extension - 域模式上运行的扩展。

path - 在域中可用的路径。

system-property - 配置文件中设置的系统属性 (如不在命令行种设置

的属性),可以在整个域里使用

profile - 一组子系统的设置，可以分配给 server group

subsystem - 子系统的设置,可以组成 profile.

interface - 接口配置

socket-binding-group - socket 绑定组设置,可以被 sever group 使用。

socket-binding - 单个 socket 绑定的配置

deployment - 可用的部署内容，可以分配给 sever group. deployments available for assignment to server groups

server-group - server group 配置

host - 单独的 Host Controller.每个 host 类型的节点都代表是特定主机的根资源。host 和 host 的子孙节点的配置会被持久化存储到主机的 host.xml 文件中。

path - 主机服务器上可用的路径

system-property - 主机服务器配置文件中设置的系统属性

core-service=management - Host Controller 的核心管理服务。

interface - 可以被 Host Controller 和主机上服务器使用的接口配置。

jvm - 用来启动服务器的 JVM 设置

server-config - 配置 HostController 如何启动 sever;配置使用什么 server group，和覆盖在其他资源上定义的和服务器相关的配置项。

server - server 的根资源.sever 和一下资源不会直接被持久化； 在域范围和主机级别的持久化的 yuan ,组成了 server 的配置。

extension - 在服务器上安装的扩展。

path - 服务器上可用的路径。

system-property - 配置文件中设置的系统属性（如不在命令行种设置的属性）

core-service=management - 服务器中核心的管理服务。

core-service=service-container - JBoss7 的最核心资源 JBoss MSC ServiceContainer

subsystem - server 上安装的子系统.大部分的管理模型都是子系统

类型的子节点。

interface - 接口配置

socket-binding-group - server 上 socket 绑定组的中心资源

socket-binding - 单个 socket 绑定的配置

deployment - server 上已经部署的内容

4.4. 管理任务

4.4.1. 网络接口和端口

4.4.1.1. 网络接口声明

JBoss AS 7 在整个配置文件中都引用命名的接口。一个网络接口通过指定一个逻辑名和选择一个物理接口来声明。

```
[standalone@localhost:9999 /] :read-children-names(child-type=interface)
{
  "outcome" => "success",
  "result" => [
    "management",
    "public"
  ]
}
```

以上操作意味着 server 声明了两个接口:一个可以使用”management”进行引用，另外一个可以用”public”引用。管理层（比如 HTTP 管理点)需要用到的所有组件和服务都可以使用”management”接口。与网络通讯有关的应用(如 Web, Message 等等)都可以使用”public”接口.接口的名字没有任何特别的要求；可以用任何名字声明接口。配置的其他部分可以用逻辑名来引用这些接口，而不用包含接口的所有详细信息（在管理域里服务器上的这些信息随着机器不同而不同).

domain.xml, host.xml 和 standalone.xml 都包含声明接口的部分。但我们看这些在 xml 文件中接口声明时，就会发现接口的选择条件(selection criteria)。接口选

择的条件有两种类型:一种是单独的 xml 元素，接口绑定到通配符地址；另外一种是接口或者地址有一个或者多个特征值需要满足。下面是一个接口条件选择的例子，每个接口都有特定的 IP 地址:

```
<interfaces>
  <interface name="management">
   <inet-address value="127.0.0.1"/>
  </interface>
  <interface name="public">
   <inet-address value="127.0.0.1"/>
  </interface>
</interfaces>
```

另外一些使用通配符的例子:

```
<interface name="global">
   <!-- 使用任何地址 -->
   <any-address/>
</interface>

<interface name="ipv4-global">
   <!--使用任何 IPV4 的例子-->
   <any-ipv4-address/>
</interface>

<interface name="ipv6-global">
   <!-- 使用任何 IPV6 的例子 -->
   <any-ipv6-address/>
</interface>

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

```
<interface name="default">
  <!-- 匹配下面子网地址，而且支持 multicats 不是点对点的地址-->
  <subnet-match value="192.168.0.0/16"/>
  <up/>
  <multicast/>
  <not>
    <point-to-point/>
  </not>
</interface>
```

4.4.1.2. Socket Binding Groups

AS7 中 socket 的配置类似于 interface 的声明,Sockets 用一个逻辑名来声明，可以在整个配置中引用。 多个 Sockets 声明可以用一个特定的名字声明成为一个组。这样在配置一个在管理域里的 server group 时可以方便的引用一个特定的 socket binding group. Socket binding group 通过 interface 逻辑名来引用 interface:

```
<socket-binding-group name="standard-sockets"
default-interface="public">
   <socket-binding name="jndi" port="1099"/>
   <socket-binding name="jmx-connector-registry" port="1090"/>
   <socket-binding name="jmx-connector-server" port="1091"/>
   <socket-binding name="http" port="8080"/>
   <socket-binding name="https" port="8443"/>
   <socket-binding name="jacorb" port="3528"/>
   <socket-binding name="jacorb-ssl" port="3529"/>
   <socket-binding name="osgi-http" port="8090"/>
   <socket-binding name="remoting" port="4447"/>
   <socket-binding name="txn-recovery-environment" port="4712"/>
```

```
    <socket-binding name="txn-status-manager" port="4713"/>
    <socket-binding name="messaging" port="5445"/>
    <socket-binding name="messaging-throughput" port="5455"/>
</socket-binding-group>
```

一个 socket binding 包含一下信息:

- name - socket 配置的逻辑名，可以在配置的其他任何地方引用。
- port - 这个配置中 socket 要绑定到的基础端口（注意 server 可以通过配置增减所有端口值来覆盖这一配置）
- interface （可选） - 配置中 socket 要绑定接口的逻辑名（参考 上面的接口声明 ）.如果没有指定， socket binding group 配置元素中的 default-interface 属性值将会被使用。
- multicast-address (可选) --如果 socket 用于多播，将会使用这个多播地址。
- multicast-port （可选） - 如果 socket 用于多播，将会使用这个多播端口
- fixed-port (可选, 默认是 false) - 如果是 true, 端口值将一直使用这个值，这个值不会被使用增减端口值而覆盖。

4.4.2. 管理接口的安全性

除了在运行服务器或者服务器组上的各种服务，JBoss7 还提供了两个管理接口允许远程的客户端可以管理 JBoss AS7.这个章节中介绍如何使用这些接口，以及如何对这些接口进行加密。

这两个管理接口被暴露成一个 HTTP 接口和一个 Native 接口。HTTP 接口既用来提供基于 GWT 的管理控制台(admin console)使用，也提供给使用 JSON 编码协议和 de-typed RPC API 各种管理操作使用。当运行在单独运行服务器(standalone)时候，Native 接口允许管理操作通过私有的二进制协议访问。这种使用二进制协议类型的操作可以通过 AS7 提供的命令行工具，也可以通过使用 AS7jar 文件的远程客户端进行交互。

在管理域下使用这些接口稍有些负责。在每一个主机上都有一个 host

controller 的进程。在主机上的 host controller 会配置成为 domain controller.在管理域中可以用同样的方式来使用 HTTP 接口; HTTP 接口允许基于 GWT 的管理控制台(admin console)运行在主 domain controller，也允许任何基于 HTTP 和 JSON 的管理控制客户端在任何 host controller 上执行管理操作。然而其他的一些客户端则使用 Natvie 接口：一旦 host controller 启动真正的应用服务器实例，这些应用服务器则通过 native 接口与 host controoler 后台建立连接；从 host controller 则使用 native 接口与主 domain controller 在后台建立连接来获取 domain 模型的拷贝，并随后接收主 domain cotroller 的操作请求。

4.4.2.1. 初始化设置

单独运行服务器的接口配置在 standalone.xml 里定义，在管理域里运行服务器的接口配置在 host.xml 中。在两个文件中，接口配置都有相同的结构:

```
<management>
   ...
   <management-interfaces>
      <native-interface interface="management" port="9999" />
      <http-interface interface="management" port="9990"/>
   </management-interfaces>
</management>
...
<interfaces>
   <interface name="management">
      <inet-address value="127.0.0.1"/>
   </interface>
   <interface name="public">
      <inet-address value="127.0.0.1"/>
   </interface>
</interfaces>
```

navtive 接口默认监听 9999 端口，http 接口监听 9990.管理接口同时与一个命名为

“management”的网络接口(network inteface)相关联。虽然 management 网络接口(network interface)的配置和 public 网络接口的默认配置相同，但我们推荐不要合并这两个配置。managment 和 public 的网络接口分开配置可以保证任何将应用服务器中服务更为公开的配置更改，不会无意识的公开本不需要公开的管理接口。

4.4.2.2. 快速配置

在本章节剩下的部分我们讲更为详细的讲述安全域的配置-但是如果你想快速的启用安全域并且完善安全配置来满足需求，默认的配置包含一个预先定义的安全域，它基于一个 property 文件和一个可以通过命令行来启用的脚本。

安全域定义在 standalone.xml 或者 host.xml 文件中<management>元素. 默认的安全域:

```
<management>
   <security-realms>
      <security-realm name="PropertiesMgmtSecurityRealm">
         <authentication>
            <properties path="mgmt-users.properties"
relative-to="jboss.server.config.dir" />
         </authentication>
      </security-realm>
   </security-realms>
   ...
</management>
```

默认安全域通过调用在 configuratiion 目录下的 mgmt-user.properties 来校验连接的用户。property 文件默认没有任何用户，因此新的用户要用 username=password 格式添加到文件中:

手动启用两个接口配置好的管理域:

```
<management>
   ...
   <management-interfaces>
```

```
      <native-interface interface="management" port="9999"
security-realm="PropertiesMgmtSecurityRealm" />
      <http-interface interface="management" port="9990"
security-realm="PropertiesMgmtSecurityRealm"/>
   </management-interfaces>
</management>
```

这将为 Http interface 启用 Http Digest authentication,并且在 Native interface 启用 Digest SASL-这也意味着对于原始密码不会在客户端和服务器端进行传输验证。

使用脚本来启用安全域,首先要编辑“mgmt-users.properties”,因为配置会马上生效。你需要至少定义一个用户，并且执行以下命令:

对于单独运行的服务器:

```
./jboss-admin.sh --connect --file=scripts/secure-standalone-mgmt.cli
```

对于在管理域的服务器:

```
./jboss-admin.sh --connect
--file=scripts/secure-host-controller-mgmt.cli
```

注意这个脚本只能运行在默认配置为 master 的 host 上。如果创建了其他具有不同名称的 host，那么需要更新这个脚本或者手动对这个新的配置实施安全性。并且还要注意，这个脚本仅仅改变它要运行的名为 master 的 host,如果有多个 host controller,这个脚本需要使用他们所有正确的 host 名字运行去更改。同时，请阅读这个章节的其他部分关于如何配置从 host controller 连接主 host controller 的校验。

禁用 JMX 远程访问

除了以上的 JBoss 管理协议，还有允许 JDK 和应用管理操作的远程 JMX 连接。为了安全性，可以通过删除远程连接配置来禁止这一服务，或者删除整个 subsystem.

```
<subsystem xmlns="urn:jboss:domain:jmx:1.0">
      <!-- Delete the following line to disable remote access -->
      <jmx-connector registry-binding="jmx-connector-registry"
```

server-binding="jmx-connector-server" />

</subsystem>

4.4.2.3. 详细配置

管理接口的配置在<management>下的三个节点中:

<management>

<security-realms />

<outbound-connections />

<management-interfaces />

</management>

<security-realms /> - 配置一个或者多个安全域来定义远程用户如何连接到服务器进行验证，并且定义服务器上的身份(identity)。

<outbound-connections /> -有时候安全域的配置需要连接到一个外部的资源；这些连接在这里配置。

<management-interfaces /> - 这里定义 Http interface 和 Native interface，正如我们在简介里描述的那样。

4.4.2.3.1. 管理接口

对于单个管理接口的配置是最简单的。仅仅需要配置管理接口的"security-realm"属性，来指定使用安全域的名字。因为管理接口启动安全域时，要查询安全域所提供的功能，并且启动安全相依的传输:比如用户的密码如果可以从安全域中获得，Http interface 会尝试使用 Digest 验证，如果用户密码不能从安全域中获取，http interface 会转而支持 Basic 验证。

<management> ...

<management-interfaces>

<native-interface ... security-realm="PropertiesMgmtSecurityRealm" />

<http-interface ... security-realm="PropertiesMgmtSecurityRealm"/>

</management-interfaces>

</management>

管理接口可以使用同样的安全域，但这不是必须的。如果需要，不同的管理接口可以使用不同的安全域。

4.4.2.3.2. 安全域

<security-realms /> 元素用来配置一个或者多个安全域。安全域的配置具有以下结构:

```
<management>
  <security-realms>
    <security-realm name="SampleRealm">
      <server-identities />
      <authentication />
    </security-realm>
  </security-realms>
  ...
</management>
```

<server-identities />元素定义 server 的身份信息。目前可以配置一个 SSL 身份(identity)来定义服务器如何从一个 keystore 取得身份信息。也可以配置一个加密的身份-服务器使用什么样的命令或密码和其他的服务器进行通信。

<authentication /> 定义如何验证连接到服务器的用户

4.4.2.3.2.1 Authentication(验证)

最初，AS7 支持三种机制来验证连接到服务器的用户:

LDAP - 使用 LDAP 服务器来验证用户的额身份信息。

Users - 定义在 domain model 里的用户名和密码信息,这仅作为简单测试使用。

Properties - 用户名和密码定义在一个服务器安装文件目录的 property 文件中。

下表概括了管理接口支持的验证机制，用来对终端用户在传输级别上进行验证:

Authentication Mechanism	HTTP Interface	Native Interface
LDAP	HTTP BASIC	Not Supported[1]
Users	HTTP DIGEST	SASL DIGEST
Properties	HTTP DIGEST	SASL DIGEST

1 - 将被增加到 AS7-1167
HTTP Basic 和 SASL Plain(实现以后)在一个表单里传输用户密码，很容易被破解。
下面的章节阐述如何配置这些验证机制:

4.4.2.3.2.1.1 LDAP

LDAP 验证操作首先要建立一个和远程目录服务器的连接。然后使用用户提通的用户名去执行查找区别用户的识别名(distinguished name)。最后验证器和目录服务器建立一个新的连接，使用查找到的识别名和用户提供的密码来验证是否是合法用户。
这是一个使用 LDAP 验证的安全域配置:

```
<security-realm name="TestRealm">
   <authentication>
      <ldap connection="ldap_connection"
base-dn="CN=Users,DC=mydomain,DC=aslab"
username-attribute="sAMAccountName"  />
   </authentication>
</security-realm>
```

ldap 元素可以配置以下属性:
connection - 定义在 <outbound-connections>的连接来连接到 LDAP 目录服务器。
base-dn - 开始搜索用户的上下文中的识别名(基准识别名).
Username-attribute - 目录中的用户名的属性，用来匹配提供的用户名
recursive (default - false) - 是否需要迭代查找
user-dn (default - dn) - 用户中存放识别名的属性， 用来校验用户信息

4.4.2.3.2.1.2 User

User 校验器是一个对存储在 domain model 里用户名和密码进行验证的简单校验器。校验器仅用作简单的测试使用:
这是一个使用 User 验证器的例子:

```
<security-realm name="TestRealm">
   <authentication>
      <users>
```

```
            <user username="TestUser">
               <password>TestUserPassword</password>
            </user>
         </users>
      </authentication>
</security-realm>
```
在这个配置中，每个用户都用<user>进行定义，用户名使用”username” 属性定义，password 定义在 user 下的<password>中。

4.4.2.3.2.1.3 Properties

Properties 校验器和 User 校验器类似，除了用户名和密码定义在一个 properties 文件中。比起 User 校验的优点是 password 不必在 domain model 中暴露。
这是一个使用 properties 验证器配置安全域的一个例子:
```
<security-realm name="TestRealm">
   <authentication>
      <properties path="users.properties"
relative-to="jboss.server.config.dir" />
   </authentication>
</security-realm>
```
Properties 文件通过简单定义”path”属性来指定文件的路径 和 ”relative-to”属性来引用定义好的路径和 path 属性相对的路径。在这个例子中，user.properties 在存放 stadnalone.xml 文件相同的目录下。 如果”relateive-to”属性没有指定，那么 path 属性的之必须是一个绝对路径。

4.4.2.3.2..2 Server Identities（服务器身份）

<server-identities>用于配置在多种场景中服务器辨别自己身份的信息。目前在 HTTP interface 中可以定义一个 SSL indentiy 并且使用这一 indentity 来启用 SSL，另外一个 Secret identity 可以存放一个密码，当 host controller 和远程的 domain controller 建立连接时，使用这一个定义好的 Secret indentity.

- SLL

SSL identity 的配置目前需要从本地文件系统中加载一个静态的 keystore.以后会增强这一个功能来允许多种类型的 keystore:
一个 SSL indentity 的配置示例如下:
```
<security-realm name="TestRealm">
   <server-identities>
      <ssl>
         <keystore path="server.keystore"
```

```
relative-to="jboss.server.config.dir" password="keystore_password" />
      </ssl>
   </server-identities>
</security-realm>
```
keystore 的路径信息和 properites 验证器中 properties 文件信息相同，使用一个路径指定 keystore 和一个可选的 relative-to 属性来指定 path 属性相对于一个已知的路径。

- Secret

从 domain controller 连接到一个加密的主 domain controller 时，需要配置 Secret identity.
为了实现连接加密的主 domain controoler,下面是在从 domain controller 中增加的配置：

```
<host xmlns="urn:jboss:domain:1.0"
      name="slave">

   <management>
      <security-realms>
         <security-realm name="TestRealm">
            <server-identities>
               <secret value="c2xhdmVfcGFzc3dvcmQ=" />
            </server-identities>
         </security-realm>
       </security-realms>
       ...
    </management>

    <domain-controller>
       <remote host="127.0.0.1" port="9999" security-realm="TestRealm"
/>
    </domain-controller>


    ...
</host>
```
这里<remote>定义了 domain controller 引用了一个定义好的安全域。，这个引用意味着这个安全域会被用来加载客户端的配置(以后这将会扩展使得域也同样可以为客户端的连接定义 SSL)
secret 是密码采用 Base64 编码，连接会使用 host 名(在这个示例中是'slave')和从 secret 中得到的密码进行验证。
AS7-1102 列出了密码的处理将会被增强，来更好的保护密码的配置。如采用密码混淆，加密方式以及使用外部的 security provider, smart card 或者使用 PKCS#11 的硬件加密模块。

4.4.2.3.3. Outbound connections(外部连接)

如前面所述，外部连接用来连接一个远程的服务器，目前仅支持 LDAP 连接，以后会增加数据库连接来支持对存储在数据库中的信息进行验证。

- LDAP

下面是一个连接 LDAP 服务器的例子:

```
<outbound-connections>
   <ldap name="ldap_connection" url="ldap://127.0.0.1"
search-dn="CN=AS7 Test Server,CN=Users,DC=mydomain,DC=aslab"
search-credential="AS_Password" />
</outboundconnections>
```

<ladp>可以配置以下属性:

name - 连接名，ladp 验证其会使用这个名字来引用这个连接。

url - 连接目录服务器的 URL.

search-dn - 用户初始化搜索的识别名

search-credential - 连接进行搜索的密码

initial-context-factory (default - com.sun.jndi.ldap.LdapCtxFactory) - 用来建立连接的 initial context factory

4.4.2.4. 问题

Application server 如何连接到 host controller 的 native interface 上-是如何进行验证的?

当 JBossAS7 进程启动时会创建一个随机的 key 并且将这个 key 传输到启动的服务器实例，applicaiotn server 使用这个 key 来验证 native interface 的连接。

4.4.3. JVM 设置

管理域和单独运行服务器的 JVM 设置是不相同的。在管理域中， domain controller 组件会负责停止和启动服务器进程，因此由它来决定 JVM 的设置。在单独运行服务器中，由启动服务器的进程 (比如通过命令行参数)负责 JVM 的设置。

4.4.3.1. 管理域

在管理域里， JVM 设置可以在不同的作用域上声明 :比如在特定的服务器组，一个主机或者一个特别的服务器。如果没有显式声明， JVM 设置从父作用域继承。这样可以在不同的层次上允许定制或者继承 JVM 设置。

我们来看一下对一个服务器组 JVM 的声明 :

```
<server-groups>
       <server-group name="main-server-group" profile="default">
           <jvm name="default">
               <heap size="64m" max-size="512m"/>
           </jvm>
           <socket-binding-group ref="standard-sockets"/>
       </server-group>
       <server-group name="other-server-group" profile="default">
           <jvm name="default">
               <heap size="64m" max-size="512m"/>
           </jvm>
           <socket-binding-group ref="standard-sockets"/>
       </server-group>
</server-groups>
```

(参见 domain/configuration/domain.xml)

在这个例子里，服务器组 "main-server-group" 的 jvm 设置成 64m 的 heap size 和 最大是 512m 的 heap size.任何属于这个组的服务器都会集成这些 JVM 设置。你可以改变整个组，或者一个特定服务器，主机的 JVM 设置 :

```
<servers>
       <server name="server-one" group="main-server-group"
auto-start="true">
           <jvm name="default"/>
       </server>
       <server name="server-two" group="main-server-group"
auto-start="true">
           <jvm name="default">
               <heap size="64m" max-size="256m"/>
           </jvm>
           <socket-binding-group ref="standard-sockets"
port-offset="150"/>
       </server>
```

```
        <server name="server-three" group="other-server-group"
auto-start="false">
            <socket-binding-group ref="standard-sockets"
port-offset="250"/>
        </server>
</servers>
```

(参考 domain/configuration/host.xml)

在这个例子中， *server-two* 属于 *main-server-group*， 因此会继承名字为 default 的 JVM 设置,但是它在 server-two 服务器上声明了一个较低的 maxium heap size。

```
[domain@localhost:9999 /]
/host=local/server-config=server-two/jvm=default:read-resource
{
    "outcome" => "success",
    "result" => {
        "heap-size" => "64m",
        "max-heap-size" => "256m",
    }
}
```

4.4.3.2. 单独运行服务器

对于单独运行的服务器，则需要在执行 $JBOSS_HOME/bin/standalone.sh 脚本时使用命令行参数来设置 JVM,或者在 $JBOSS_HOME/bin/standalone.conf 声明。 (对于 windows 用户，需要执行 %JBOSS_HOME%/bin/standalone.bat 和设置

%JBOSS_HOME%/bin/standalone.conf.bat.)

4.4.4. 命令行参数

启动 JBoss AS7 的管理域，需要执行 :$JBOSS_HOME/bin/domain.sh 脚本，启动单独运行的服务器需要执行 $JBOSS_HOME/bin/standalone.sh . 使用这两个脚本启动时，将会使用默认的设置。以下内容，我们讲介绍如何通过额外的命令行参数来覆盖这些默认的设置。

4.4.4.1. 系统属性

单服务器和管理域模式都使用用来设置标准位置 (如 jboss.home.dir,jboss.server.config.dir)的默认设置， B 这小节中介绍这些系统属性的默认值。每个系统属性，都可以通过标准的 JVM 设置方式 -Dkey=value 覆盖：

```
$JBOSS_HOME/bin/standalone.sh
-Djboss.home.dir=some/location/AS7/jboss-as \

-Djboss.server.config.dir=some/location/AS7/jboss-as/custom-standalon
e
```

以上的命令行启动一个不是标准的 AS home 目录，并且使用一个特定的配置文件路径 . 具体系统属性的含义将在以下内容中介绍。

同时，你也可以使用一个 properties 文件通过下面任何一种方式来覆盖配置默认的系统属性 :

```
$JBOSS_HOME/bin/domain.sh
--properties=/some/location/jboss.properties
$JBOSS_HOME/bin/domain.sh -P=/some/location/jboss.properties
```

这个 properties 文件是一个标准的包含 key=value 对的标准 Java property 文件 :

```
jboss.home.dir=/some/location/AS7/jboss-as
jboss.domain.config.dir=/some/location/AS7/custom-domain
```

4.4.4.2. 单独运行模式（ Standalone)

属性名	说明	默认值
java.ext.dirs	指定 JDK extension 路径	null
jboss.home.dir	JBoss AS 7 安装的根目录	standalone.sh 设置为 $JBOSS_HOME
jboss.server.base.dir	server 的 base 目录	jboss.home.dir /standalone
jboss.server.config.dir	base configuration 目录	jboss.server.base.dir /configuration
jboss.server.data.dir	用于存放持久	jboss.server.base.dir /data

	化数据的目录	
jboss.server.log.dir	存放 server.log 的目录	jboss.server.base.dir /log
jboss.server.temp.dir	临时文件目录	jboss.server.base.dir /tmp
jboss.server.deploy.dir	部署目录	jboss.server.data.dir /content

4.4.4.3. 管理域模式 (Managed Domain)

属性名	说明	默认值
jboss.home.dir	The root directory of the JBoss AS 7 installation.	domain.sh 设置为 $JBOSS_HOME
jboss.domain.base.dir	domain 的 base 目录	jboss.home.dir /domain
jboss.domain.config.dir	base configuration 目录	jboss.domain.base.dir /configuration
jboss.domain.data.dir	用于存放持久化数据的目录 .	jboss.domain.base.dir /data
jboss.domain.log.dir	存放 host-controller.log 和 process-controller.log 文件的目录	jboss.domain.base.dir /log
jboss.domain.temp.dir	临时文件目录	jboss.domain.base.dir /tmp
jboss.domain.deployment.dir	部署目录	jboss.domain.base.dir /content
jboss.domain.servers.dir	被管服务器输出存放的目录	jboss.domain.base.dir /log

4.4.4.4. 其他命令行参数

第一种接收参数的格式是 :

--name=value

比如 :

$JBOSS_HOME/bin/standalone.sh --server-config=standalone-ha.xml

如果参数名是一个单词，那么使用一个” -”前缀，而不是两个” --”:

-x=value

比如 :

$JBOSS_HOME/bin/standalone.sh -P=/some/location/jboss.properties

下面说明在单服务器和管理域模式下可用的的命令行参数 :

4.4.4.4.1. 单服务器模式（ Standalone）

参数名	缺省的默认值	值
--server-config	jboss.server.config.dir /standalone.xml	一个相对于 jboss.server.config.dir 的或者是一个绝对路径

4.4.4.4.2. 管理域模式 (Managed Domain)

参数名	缺省的默认值	值
--domain-config	jboss.domain.config.dir/domain.xml	一个相对于 jboss.domain.config.dir 的路径或者是一个绝对路径
--host-config	jboss.domain.config.dir/host.xml	一个相对于 jboss.domain.config.dir 的路径或者是一个绝对路径

下面的参数不需要指定值，并且只能被用于 host controller.(比如被配置连接到远程 domain controller 的主机)

参数	功能
--backup	使从 host controller 创建和维护一个域配置的本地拷贝
--cached-dc	如果从 (slave)host controller 在启动时不能连接主 domain controller 取得配置信息，那么通过 -bakcup 创建的本地拷贝将会被使用。同时 slave host controller 不会改变任何 domain 的配置，仅启动服务器。

4.4.4.4.3. 通用参数 (Common parameters)

这些没有值的参数既适用于单服务器模式也适用于管理域模式。下表介绍这些参数的使用 :

参数	功能
--version	打印 JBossAS 的版本信息，并且退出 JVM。

-V
--help
-h　　打印各参数的帮助信息，并且退出 JVM

4.4.5. 子系统配置

以下章节中将集中介绍通过 CLI 和 web 接口进行操作的高级管理用例 .对于每个子系统详细的配置属性，请参考每个子系统的参考文档。

配置的 schema 文件都在目录 $JBOSS_HOME/docs/schema

4.4.5.1. 数据源 (Data sources)

Datasources 在通过子系统进行配置。声明一个新的数据源，需要两个步骤 : 提供一个 JDBC 驱动，然后定义一个使用这个 JDBC 驱动的数据源。

4.4.5.1.1. JDBC 驱动安装

在应用服务器中安装 JDBC 驱动推荐使用一个常规的 jar 进行部署。因为当在域模式下运行应用服务器时，部署的内容会自动传送到要部署的所有服务器上，因此使用 jar 文件将利用这一特性而不需要关心额外的事情。

任何符合 JDBC4 的启动将会被自动识别并且按照名字和版本安装到系统中。JDBC jar 使用 Java server provider 机制进行识别。 Jar 文件中需要包含一个文件名是 META-INF/services/java.sql.Driver 的文本文件 ,这个文件中包含在这个 jar 里的驱动类的名称。如果你的 JDBC 驱动 jar 不符合 JDBC 规范，我们通过其他方式也可以部署这样的驱动。

修改 Jar 文件

最直接的方式是简单的修改 Jar 文件添加缺失的文件。你可以通过一下命令添加 :

The most straightforward solution is to simply modify the JAR and add the missing file. You can do

1. 改变路径到或者创建一个空的临时文件夹 .
2. 创建一个 META-INF 子目录
3. 创建一个 META-INF/services 子目录
4. 创建 一个只包含一行内容 :JDBC 驱动类的全名的文件
   META-INF/services/java.sql.Driver .
5. 使用 jar 命令来跟新这个 jar 文件 :

```
jar \-uf jdbc-driver.jar META-INF/services/java.sql.Driver
```

如何部署

JDBC4 驱动

jar 文件，请参考”应用部署 “章节。

4.4.5.1.2. 数据源定义 (Datasource Definitions)

```
subsystem xmlns="urn:jboss:domain:datasources:1.0">
    <datasources>
        <datasource jndi-name="java:jboss/datasources/ExampleDS"
pool-name="ExampleDS">

<connection-url>jdbc:h2:mem:test;DB_CLOSE_DELAY=-1</connection-url>
            <driver>h2</driver>
            <pool>
                <min-pool-size>10</min-pool-size>
                <max-pool-size>20</max-pool-size>
                <prefill>true</prefill>
            </pool>
            <security>
                <user-name>sa</user-name>
                <password>sa</password>
            </security>
        </datasource>

<xa-datasource jndi-name="java:jboss/datasources/ExampleXADS" pool-na
me="ExampleXADS">
           <driver>h2</driver>
           <xa-datasource-property name="URL">jdbc:h2:mem:test</xa-da
tasource-property>
           <xa-pool>
                <min-pool-size>10</min-pool-size>
                <max-pool-size>20</max-pool-size>
                <prefill>true</prefill>
```

```
            </xa-pool>
            <security>
                 <user-name>sa</user-name>
                 <password>sa</password>
            </security>
        </xa-datasource>
        <drivers>
            <driver name="h2" module="com.h2database.h2">

<xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-datasource-class
>
            </driver>
        </drivers>
  </datasources>

</subsystem>
```

(参见 standalone/configuration/standalone.xml)

如以上示例所示，数据源通过逻辑名来引用 JDBC 驱动 .通过命令行 (CLI)可以很方便的查询同样的信息 :

```
[standalone@localhost:9999 /]
/subsystem=datasources:read-resource(recursive=true)
{
    "outcome" => "success",
    "result" => {
        "data-source" => {"java:/H2DS" => {
            "connection-url" => "jdbc:h2:mem:test;DB_CLOSE_DELAY=-1",
            "jndi-name" => "java:/H2DS",
            "driver-name" => "h2",
            "pool-name" => "H2DS",
            "use-java-context" => true,
            "enabled" => true,
            "jta" => true,
            "pool-prefill" => true,
            "pool-use-strict-min" => false,
            "user-name" => "sa",
            "password" => "sa",
            "flush-strategy" => "FailingConnectionOnly",
            "background-validation" => false,
            "use-fast-fail" => false,
            "validate-on-match" => false,
            "use-ccm" => true
        }},
```

```
        "xa-data-source" => undefined,
        "jdbc-driver" => {"h2" => {
            "driver-name" => "h2",
            "driver-module-name" => "com.h2database.h2",
            "driver-xa-datasource-class-name" =>
"org.h2.jdbcx.JdbcDataSource"
        }}
    }
}


[standalone@localhost:9999 /]
/subsystem=datasources:installed-drivers-list
{
    "outcome" => "success",
    "result" => [{
        "driver-name" => "h2",
        "deployment-name" => undefined,
        "driver-module-name" => "com.h2database.h2",
        "module-slot" => "main",
        "driver-xa-datasource-class-name" =>
"org.h2.jdbcx.JdbcDataSource",
        "driver-class-name" => "org.h2.Driver",
        "driver-major-version" => 1,
        "driver-minor-version" => 2,
        "jdbc-compliant" => true
    }]
}
```

使用 web 控制台和命令行可以极大的简化 JDBC 驱动的部署和数据源的创建。

命令行方式提供了一些列的命令来创建和更改数据源 ：

```
[standalone@localhost:9999 /] help
Supported commands:

 [...]

 data-source              - allows to add new, modify and remove existing
data sources
 xa-data-source           - allows add new, modify and remove existing XA
data sources
```

特定命令的详细描述请使用”

-b”参数查询。

4.4.5.1.3. 参考

datasource 子系统由 IronJacamar 项目提供。更多关于配置属性和属性的详细介绍请参考项目文档 ：

- IronJacamar 主页 ： http://www.jboss.org/ironjacamar
- 项目文档 ： http://www.jboss.org/ironjacamar/docs
- Schema 描述 ： http://docs.jboss.org/ironjacamar/userguide/1.0/en-US/html/deployment.html#deployingds_descriptor

4.4.5.2. 消息 (Messaging)

JMS 服务器需要通过 messaging 子系统进行配置。在本章节中，我们将概括介绍常用的配置项。其他详细的介绍，请参考 HornetQ 用户指南 (参见 ″参考″).

4.4.5.2.1. Connection Factories

JMS connection factories 可以分为两类 ： In-VM connection factory 和被远程客户端使用的 connections factories. 每个 connecton factory 都引用一个 connector ， 每个

connector 都关联到一个 socket binding. Connection Factory 的 entry 定义 factory 被暴露的 JNDI name.

```
<subsystem xmlns="urn:jboss:domain:messaging:1.0">
   [...]
<connectors>
   <in-vm-connector name="in-vm" server-id="0"/>
   <netty-connector name="netty" socket-binding="messaging"/>
   <netty-connector name="netty-throughput"
socket-binding="messaging-throughput">
       <param key="batch-delay" value="50"/>
   </netty-connector>
</connectors>
```

```
   [...]
<jms-connection-factories>
   <connection-factory name="InVmConnectionFactory">
       <connectors>
           <connector-ref connector-name="in-vm"/>
       </connectors>
       <entries>
           <entry name="java:/ConnectionFactory"/>
       </entries>
   </connection-factory>
   <connection-factory name="RemoteConnectionFactory">
       <connectors>
           <connector-ref connector-name="netty"/>
       </connectors>
       <entries>
           <entry name="RemoteConnectionFactory"/>
       </entries>
   </connection-factory>
   <pooled-connection-factory name="hornetq-ra">
       <connectors>
           <connector-ref connector-name="in-vm"/>
       </connectors>
       <entries>
           <entry name="java:/JmsXA"/>
       </entries>
   </pooled-connection-factory>
</jms-connection-factories>
[...]
</subsystem>
```

(参见 standalone/configuration/standalone.xml)

4.4.5.2.2. Queues and Topics

Queues 和 topics 是 messaging 子系统的子资源 .每个 Entry 指定一个 queue 或者 topic 的 JNDI 名 :

```
<subsystem xmlns="urn:jboss:domain:messaging:1.0">
   [...]
   <jms-destinations>
       <jms-queue name="testQueue">
           <entry name="queue/test"/>
       </jms-queue>
       <jms-topic name="testTopic">
```

```
            <entry name="topic/test"/>
        </jms-topic>
    </jms-destinations>
</subsystem>
```

(参见 standalone/configuration/standalone.xml)

JMS endpoints 通过命令行方式可以很容易的创建 :

```
[standalone@localhost:9999 /] add-jms-queue --name=myQueue
--entries=queues/myQueue

[standalone@localhost:9999 /]
/subsystem=messaging/jms-queue=myQueue:read-resource
{
    "outcome" => "success",
    "result" => {"entries" => ["queues/myQueue"]},
    "compensating-operation" => undefined
}
```

JbossAS 同时也提供了其他很多的维护

JMS 子系统的命令

```
:
[standalone@localhost:9999 /] help
Supported commands:
[...]
add-jms-queue           - creates a new JMS queue
remove-jms-queue        - removes an existing JMS queue
add-jms-topic           - creates a new JMS topic
remove-jms-topic        - removes an existing JMS topic
add-jms-cf              - creates a new JMS connection factory
remove-jms-cf           - removes an existing JMS connection factory
```

获取更多命令行的详细信息，请使用”

--help”参数获取。

4.4.5.2.3. Dead Letter 和 Redelivery

有些设置可以在通配符地址上生效，而不是一个特别的 messaging destination.Dead letter queue 和 redelivery 设置就可以使用通配符地址 :

```
<subsystem xmlns="urn:jboss:domain:messaging:1.0">
[...]
<address-settings>
   <address-setting match="#">
       <dead-letter-address>
           jms.queue.DLQ
       </dead-letter-address>
       <expiry-address>
           jms.queue.ExpiryQueue
       </expiry-address>
       <redelivery-delay>
           0
       </redelivery-delay>
       [...]
   </address-setting>
</address-settings>
[...]
</subsystem>
```

(参见 standalone/configuration/standalone.xml)

4.4.5.2.4. 安全性

安全性的设置也可以使用通配符地址生效,如同 DLQ 和 redelivery 设置一样 :

```
<subsystem xmlns="urn:jboss:domain:messaging:1.0">
[...]
<security-settings>
   <security-setting match="#">
       <permission type="send" roles="guest"/>
       <permission type="consume" roles="guest"/>
       <permission type="createNonDurableQueue" roles="guest"/>
       <permission type="deleteNonDurableQueue" roles="guest"/>
   </security-setting>
</security-settings>
[...]
</subsystem>
```

(参见 standalone/configuration/standalone.xml)

4.4.5.2.5. 参考

Messaging 子系统由 Hornetq 项目提供。详细的关于可用的配置项信息，请查询 hornetq 项目文档。

- HornetQ 主页 ： http://www.jboss.org/hornetq
- 项目文档 ： http://www.jboss.org/hornetq/docs

4.4.5.3. Web

Web 子系统的配置由三个部分组成 :JSP, connectors 和 virtural servers。高级特性如 :负载均衡， failover 等将在高”可用性指南”中介绍。默认配置对于大多数的用例都可以提供合理的性能。

需要的扩展 ：

```
<extension module="org.jboss.as.web" />
```

基本子系统配置的例子 ：

```
  <subsystem xmlns="urn:jboss:domain:web:1.0"
default-virtual-server="default-host">
      <connector name="http" scheme="http" protocol="HTTP/1.1"
socket-binding="http"/>
      <virtual-server name="default-host" enable-welcome-root="true">
         <alias name="localhost" />
         <alias name="example.com" />
      </virtual-server>
   </subsystem>
```

依赖于其他子系统 ： 无 .

4.4.5.3.1. 容器设置 (Container configuration)

JSP 设置 (JSP Configuration)

这里的”配置“包含了所有关于 servlet engine 自身的设置。详细的关于配置属性的介绍，请参考 JBossWeb 有关文档.

```
[standalone@localhost:9999 /]
/subsystem=web:read-resource
{
    "outcome" => "success",
    "result" => {
        "configuration" => {
            "static-resources" => {
                "sendfile" => 49152,
                "max-depth" => 3,
                "read-only" => true,
                "webdav" => false,
                "listings" => false,
                "disabled" => false
            },
            "jsp-configuration" => {
                "development" => false,
                "keep-generated" => true,
                "recompile-on-fail" => false,
                "check-interval" => 0,
                "modification-test-interval" => 4,
                "display-source-fragment" => true,
                "error-on-use-bean-invalid-class-attribute" => false,
                "java-encoding" => "UTF8",
                "tag-pooling" => true,
                "generate-strings-as-char-arrays" => false,
                "target-vm" => "1.5",
                "dump-smap" => false,
                "mapped-file" => true,
                "disabled" => false,
                "source-vm" => "1.5",
                "trim-spaces" => false,
                "smap" => true
            }
        },
        "connector" => {"http" => undefined},
        "virtual-server" => {"localhost" => undefined}
    }
}
```

(参见 standalone/configuration/standalone.xml)

4.4.5.3.2. Connector 设置 (Connector configuration)

Connecotrs 是 web 子系统的子资源。每个 connector 都引用一个特定的 socket binding:

```
[standalone@localhost:9999 /]
/subsystem=web:read-children-names(child-type=connector)
{
    "outcome" => "success",
    "result" => ["http"]
}

[standalone@localhost:9999 /]
/subsystem=web/connector=http:read-resource(recursive=true)
{
    "outcome" => "success",
    "result" => {
        "protocol" => "HTTP/1.1",
        "scheme" => "http",
        "socket-binding" => "http",
        "ssl" => undefined,
        "virtual-server" => undefined
    }
}
```

创建一个 connector 需要先声明一个 socket binding:

```
[standalone@localhost:9999 /]
/socket-binding-group=standard-sockets/socket-binding=custom:add(port
=8181)
```

新创建的没有被使用的

socket binding 可以用来创建一个新的

connector 配置

:

```
[standalone@localhost:9999 /]
/subsystem=web/connector=test-connector:add(
               socket-binding=custom, scheme=http, protocol="HTTP/1.1",
enabled=true
               )
```

web 子系统可以配置三种类型的 connecotr:

HTTP Connectors

默认的 connector,通常运行在 8080 端口。配置请参考以上内容

HTTPS Connectors

HTTPS connectors 是 web 子系统的子资源。默认使用 JSSE.每个 connector 引用一个特定的 socket binding:

```
[standalone@localhost:9999 /]
/subsystem=web:read-children-names(child-type=connector)
{
    "outcome" => "success",
    "result" => [
        "ajp",
        "http",
        "https"
    ]
}
[standalone@localhost:9999 /]
/subsystem=web/connector=https:read-resource(recursive=true)
{
    "outcome" => "success",
    "result" => {
        "protocol" => "HTTP/1.1",
        "scheme" => "https",
        "secure" => true,
        "socket-binding" => "https",
        "ssl" => {},
        "virtual-server" => undefined
    }
}
```

创建一个新的 connector 首先需要声明一个新的 socket binding:

```
[standalone@localhost:9999 /]
/socket-binding-group=standard-sockets/socket-binding=https:add(port=
8443)
```

新创建的，没有使用的

socket binding 可被用来设置新创建的

connecotr：

```
[standalone@localhost:9999 /]
/subsystem=web/connector=test-connector:add(socket-binding=https,
scheme=https, protocol="HTTP/1.1", enabled=true, ssl = {})
```

默认 SSL 使用” tomcat”别名和” changit”密码。可以使用 keytool 来创建相应的 keystore:

```
keytool -genkey -alias tomcat -keyalg RSA
```

当然需要指定值是” changeit”的密码。

AJP Connectors

AJP Connectors 是 web 子系统的子资源。它和前段 apache httpd 的 mod_jdk,mode_proxy 和 mod_cluster 一起使用。

每个 connecotor 都引用一个特定的 socket binding:

```
[standalone@localhost:9999 /]
/subsystem=web:read-children-names(child-type=connector)
{
    "outcome" => "success",
    "result" => [
        "ajp",
        "http"
    ]
}

[standalone@localhost:9999 /]
/subsystem=web/connector=ajp:read-resource(recursive=true)
{
    "outcome" => "success",
    "result" => {
        "protocol" => "AJP/1.3",
        "scheme" => "http",
        "socket-binding" => "ajp",
        "ssl" => undefined,
        "virtual-server" => undefined
    }
}
```

创建一个新的 connector 首先需要声明一个新的 socket binding:

```
[standalone@localhost:9999 /]
/socket-binding-group=standard-sockets/socket-binding=ajp:add(port=80
09)
```

新创建的，没有使用的

socket binding 可被用来设置新创建的

connecotr：

```
[standalone@localhost:9999 /] /subsystem=web/connector=ajp:add(
               socket-binding=ajpm, protocol="AJP/1.3", enabled=true
               )
```

Native Connectors

Native connectors 是基于 Tomcat native 的高性能的 connectors.如果 nativie 模块安装的话，就可以使用 native connectors 。

目前很多发布已经包含 jboss web native(如果你还没有试用过 JBoss web native)。

在配置层面，由于使用 OpenSSL,只有 SSL 部分需要被不同的配置。

```
[standalone@localhost:9999 /]
/subsystem=web/connector=https:read-resource(recursive=true)
{
    "outcome" => "success",
    "result" => {
        "protocol" => "HTTP/1.1",
        "scheme" => "https",
        "secure" => true,
        "socket-binding" => "https",
        "ssl" => {
            "certificate-file" =>
"/home/jfclere/CERTS/SERVER/newcert.pem",
            "certificate-key-file" =>
"/home/jfclere/CERTS/SERVER/newkey.pem",
            "password" => "xxxxxxx"
        },
        "virtual-server" => undefined
    }
}
```

4.4.5.3.3. Virtual-server 配置（Virtual-Server configuration）

和 connector 类似， virtual server 声明 web 子系统的子资源。可以通过使用别名来引用 virtual server,

同时 virtual server 也可以指定默认的 web 应用来充当 root web context 。

```
[standalone@localhost:9999 /]
/subsystem=web:read-children-names(child-type=virtual-server)
{
    "outcome" => "success",
    "result" => ["localhost"]
}

[standalone@localhost:9999 /]
/subsystem=web/virtual-server=default-host:read-resource
{
    "outcome" => "success",
    "result" => {
        "access-log" => undefined,
        "alias" => ["example.com"],
        "default-web-module" => undefined,
        "enable-welcome-root" => true,
        "rewrite" => undefined
    }
}
```

增加一个 virtual server 的声明可以通过默认的 add 操作 :

```
[standalone@localhost:9999 /]
/subsystem=web/virtual-server=example.com:add

[standalone@localhost:9999 /]
/subsystem=web/virtual-server=example.com:remove
```

在 configuration tree 上任意一个节点上都可以执行增加和删除操作

4.4.5.3.4. 参考

Web 子系统部件由 jboss web 项目提供。关于 web 子系统可配置的属性的详细介绍，请参考 JBoss Web 文档 ：

- JBoss Web 配置和参考 ： http://docs.jboss.org/jbossweb/7.0.x/config/index.html
- JBossWeb 主页 ： http://www.jboss.org/jbossweb
- 项目文档 ： http://docs.jboss.org/jbossweb/7.0.x/

4.4.5.4. Web services

Web service endpoint 通过包含有 webservice endpoint 实现的部署来提供因此他们可以通过部署资源进行查询。

进一步的信息，请参考”应用部署”章节。每个 webservice endpoint 都需要指定一个 web context 和一个 wsdl 的 URL ：

```
[standalone@localhost:9999 /]
/deployment="*"/subsystem=webservices/endpoint="*":read-resource
{
   "outcome" => "success",
   "result" => [{
       "address" => [
           ("deployment" => "jaxws-samples-handlerchain.war"),
           ("subsystem" => "webservices"),
           ("endpoint" => "jaxws-samples-handlerchain:TestService")
       ],
       "outcome" => "success",
       "result" => {
           "class" =>
"org.jboss.test.ws.jaxws.samples.handlerchain.EndpointImpl",
           "context" => "jaxws-samples-handlerchain",
           "name" => "TestService",
           "type" => "JAXWS_JSE",
           "wsdl-url" =>
"http://localhost:8080/jaxws-samples-handlerchain?wsdl"
       }
   }]
}
```

4.4.5.4.1. 参考

Webservice 子系统由 JBossWS 项目提供。关于 websevice 子系统可配置的属性的详细介绍，请参考 JBoss WS 文档 ：

- JBossWS 主页 ： http://www.jboss.org/jbossws
- 项目文档 ： https://docs.jboss.org/author/display/JBWS