**jetty中文教程**

参考：<http://orchome.com/jetty/index>

jetty的资料比较少，英文官网写的也是比较技术化的，没有一个比较通俗一点的中文教程。本次专题，直译官网，帮助各位减少学习时间。

由于专题维护能力有限，希望各位积极反馈想学习的部分，我们会着重整理这方面的资料。

# 关于jetty

**关于jetty**

  Jetty是一个纯粹的基于Java的网页服务器和Java Servlet容器。尽管网页服务器通常用来为人们呈现文档，但是Jetty通常在较大的软件框架中用于计算机与计算机之间的通信。Jetty作为Eclipse基金会的一部分，是一个自由和开源项目。该网页服务器被用在Apache ActiveMQ、Alfresco、Apache Geronimo、Apache Maven、Apache Spark、Google App Engine、Eclipse、FUSE、Twitter's Streaming API、Zimbra[9]等产品上。Jetty也是Lift、Eucalyptus、Red5、Hadoop、I2P等开源项目的服务器。Jetty支持最新的Java Servlet API（带JSP的支持），支持SPDY和WebSocket协议。

**Jetty的历史**

  Jetty最初是由软件工程师Greg Wilkins在悉尼的郊区巴尔曼开发的，原本是Mort Bay Server（莫特湾服务器）的一个HTTP服务器组件。

Jetty原名IssueTracker，而后更名为MBServler（Mort Bay SERVLet servER，Mort Bay Server的Servlet服务器），最终更名为Jetty。

Jetty于1995年开始开发，版本1.x和2.x由MortBay托管。从2000年到2005年，Jetty版本3.x、4.x和5.x由sourceforge.net托管。2005年，整个Jetty项目转移到了codehaus.org。。2009年，Jetty的核心部件已被转移到Eclipse.org，Codehaus.org继续提供Jetty版本7.x和8.x（不包含9.x）的集成、扩展和打包。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **版本** | **寄存于** | **Java版本** | **协议** | **Servlet版本** | **JSP版本** | **状态** |
| 9.1.x | Eclipse | 1.7 | HTTP/1.1,WebSocketJSR356, SPDY | 3.1 | 2.3 | 稳定（自2013年11月18日） |
| 9.0.x | Eclipse | 1.7 | HTTP/1.1,WebSocket, SPDY | 3.0（跟踪3.1版草案） | 2.2 | 稳定（自2013年03月08日） |
| 8.x | Eclipse,Codehaus | 1.6 | HTTP/1.1,WebSocket, SPDY | 3.0 | 2.1 | 稳定 |
| 7.x | Eclipse,Codehaus | 1.5, J2ME | HTTP/1.1,WebSocket, SPDY | 2.5 | 2.1 | 稳定 |
| 6.x | Codehaus | 1.4–1.5 | HTTP/1.1 | 2.5 | 2.0 | 不推荐 |
| 5.x | Sourceforge | 1.2–1.5 | HTTP/1.1 | 2.4 | 2.0 | 不推荐 |
| 4.x | Sourceforge | 1.2, J2ME | HTTP/1.1 | 2.3 | 1.2 | 古老的 |
| 3.x | Sourceforge | 1.2 | HTTP/1.1 RFC2068 | 2.2 | 1.1 | 化石态 |
| 2.x | Mortbay | 1.1 | HTTP/1.0 RFC1945 | 2.1 | 1.0 | 传奇态 |
| 1.x | Mortbay | 1.0 | HTTP/1.0 RFC1945 |  |  | 神话态 |

**Jetty的应用**

  Jetty在嵌入式的Java应用程序中提供Web服务，其已经是Eclipse IDE中的一个组成部分。它支持AJP、JASPI、JMX、JNDI、OSGi、WebSocket和其他的Java技术。

Apache Hadoop是Jetty应用在框架中的典型范例。 Hadoop在几个模块中使用Jetty作为Web服务器，其针对了不同的用途：

NameNode和JobTracker使用Jetty呈现管理页面。  
TaskTracker使用Jetty接收来自JobTracker的映射（Map），减少并洗牌（Shuffle）操作。  
Hadoop 0.23版后，TaskTracker的洗牌操作已从Jetty替换为Netty。

# 什么是Jetty

### 什么是Jetty?

Jetty、是一个开源项目提供HTTP服务器、HTTP客户端,以及servlet容器。Jetty是一个纯粹的基于Java的网页服务器和Java Servlet容器。尽管网页服务器通常用来为人们呈现文档，但是Jetty通常在较大的软件框架中用于计算机与计算机之间的通信。Jetty作为Eclipse基金会的一部分，是一个自由和开源项目。该网页服务器被用在Apache ActiveMQ、Alfresco、Apache Geronimo、Apache Maven、Apache Spark、Google App Engine、Eclipse、FUSE、Twitter's Streaming API、Zimbra[9]等产品上。Jetty也是Lift、Eucalyptus、Red5、Hadoop、I2P等开源项目的服务器。Jetty支持最新的Java Servlet API（带JSP的支持），支持SPDY和WebSocket协议。

### Jetty的历史

Jetty最初是由软件工程师Greg Wilkins在悉尼的郊区巴尔曼开发的，原本是Mort Bay Server（莫特湾服务器）的一个HTTP服务器组件。

Jetty原名IssueTracker，而后更名为MBServler（Mort Bay SERVLet servER，Mort Bay Server的Servlet服务器），最终更名为Jetty。

Jetty于1995年开始开发，版本1.x和2.x由MortBay托管。从2000年到2005年，Jetty版本3.x、4.x和5.x由sourceforge.net托管。2005年，整个Jetty项目转移到了codehaus.org。。2009年，Jetty的核心部件已被转移到Eclipse.org，Codehaus.org继续提供Jetty版本7.x和8.x（不包含9.x）的集成、扩展和打包。

### Jetty的应用

Jetty在嵌入式的Java应用程序中提供Web服务，其已经是Eclipse IDE中的一个组成部分。它支持AJP、JASPI、JMX、JNDI、OSGi、WebSocket和其他的Java技术。

Apache Hadoop是Jetty应用在框架中的典型范例。 Hadoop在几个模块中使用Jetty作为Web服务器，其针对了不同的用途：

NameNode和JobTracker使用Jetty呈现管理页面。  
TaskTracker使用Jetty接收来自JobTracker的映射（Map），减少并洗牌（Shuffle）操作。  
Hadoop 0.23版后，TaskTracker的洗牌操作已从Jetty替换为Netty。

### 本指南分为2部分.

* 第一部分强调开始使用Jetty。它提供的信息下载Jetty,改变Jetyy端口运行,调整日志级别,和配置许多最常见的servlet容器特性如JNDI、JMX、会话管理。
* 第二部分描述了先进的Jetty使用,提供深度的具体功能,像我们的高度可伸缩的异步客户端,代理servlet配置,Jetty Maven插件,使用Jetty作为嵌入式服务器。先进的部分包括教程、howto、视频、和参考资料。

# jetty推荐版本

**我使用什么版本？**

Jetty9是最新版本的码头，并拥有很多改进了以前的版本。

本文档侧重于Jetty9。虽然许多人继续使用旧版本的Jetty，我们一般建议使用Jetty9，因为它代表了码头的版本，在未来的几年，我们会积极维护和改善。

# jetty下载

你可以在很多不同的方式下使用Jetty,从应用程序中嵌入Jetty, 推出不同的构建系统, 从基于不同的JVM, 或作为一个独立的部署。

# 下载Jetty

Eclipse基金会提供下载，下载地址如下。

<http://download.eclipse.org/jetty>

提供了zip和gzip2种格式格式，下载一个最适合自己的系统。 请注意，有许多其他的格式.sha 或.md5 这是效验文件.

### 下载内容

根目录的内容

|  |  |
| --- | --- |
| **Location** | **Description** |
| license-eplv10-aslv20.html | 许可文件 |
| README.txt | 有用的入门信息 |
| VERSION.txt | 版本信息 |
| bin/ | shell脚本程序，帮助jetty在unix系统中运行。 |
| demo-base/ | jetty的webapp例子。 |
| etc/ | 配置目录。 |
| lib/ | Jetty的jar包 |
| logs/ | 目录请求日志 |
| modules/ | 模块定义的目录 |
| notice.html | License information and exceptions |
| resources/ | 包含类路径更多的资源目录，通过配置激活 |
| start.d/ | 的目录中。ini文件包含命令行参数添加到有效(见start.ini) |
| start.ini | 文件包含命令行参数添加到有效(模块、属性和XML配置files) |
| start.jar | 通过jar运行jetty |
| webapps/ | 含webapps目录,JettyJetty的默认配置下运行 |

# 运行jetty

# 运行jetty

Jetty的默认端口为8080, 运行下面的命令启动jetty:

**> cd $HOME/jetty-distribution-9**

**> java -jar start.jar**

2015-02-09 04:01:36.970:WARN:oejs.HomeBaseWarning:main: This instance of Jetty is not running from a separate {jetty.base} directory, this is not recommended.  See documentation at http://www.eclipse.org/jetty/documentation/current/startup.html

2015-02-09 04:01:37.703:INFO:oejs.Server:main: jetty-9.2.7.v20150116

2015-02-09 04:01:37.751:INFO:oejdp.ScanningAppProvider:main: Deployment monitor [file:/home/hadoop/jetty-distribution-9.2.7.v20150116/webapps/] at interval 1

2015-02-09 04:01:37.801:INFO:oejs.ServerConnector:main: Started ServerConnector@14da173c{HTTP/1.1}{0.0.0.0:8080}

2015-02-09 04:01:37.801:INFO:oejs.Server:main: Started @3501ms

你可以在游览器里访问：http://localhost:8080.

### 演示

9.1版本以后，jetty没有部署任何演示web的应用程序，所以想看到更多有趣的演示服务器，需要运行从demo-base目录下运行，如下:

**> cd $HOME/jetty-distribution-9/demo-base/**

**> java -jar ../start.jar**

2015-02-09 04:42:53.617:INFO::main: Logging initialized @826ms

2015-02-09 04:42:54.167:WARN::main: demo test-realm is deployed. DO NOT USE IN PRODUCTION!

2015-02-09 04:42:54.173:INFO:oejs.Server:main: jetty-9.2.7.v20150116

2015-02-09 04:42:54.193:INFO:oejdp.ScanningAppProvider:main: Deployment monitor [file:/home/hadoop/jetty-distribution-9.2.7.v20150116/demo-base/webapps/] at interval 1

2015-02-09 04:42:54.782:WARN::main: test-jaas webapp is deployed. DO NOT USE IN PRODUCTION!

2015-02-09 04:42:55.334:INFO:oejsh.ContextHandler:main: Started o.e.j.w.WebAppContext@9a0e408{/test-jaas,file:/tmp/jetty-0.0.0.0-8080-test-jaas.war-\_test-jaas-any-3888906193790876917.dir/webapp/,AVAILABLE}{/test-jaas.war}

2015-02-09 04:42:55.916:INFO:oejsh.ContextHandler:main: Started o.e.j.w.WebAppContext@782283ad{/,file:/home/hadoop/jetty-distribution-9.2.7.v20150116/demo-base/webapps/ROOT/,AVAILABLE}{/ROOT}

2015-02-09 04:42:56.398:WARN::main: test-jndi webapp is deployed. DO NOT USE IN PRODUCTION!

2015-02-09 04:42:56.731:INFO:oejsh.ContextHandler:main: Started o.e.j.w.WebAppContext@4d74bd22{/test-jndi,file:/tmp/jetty-0.0.0.0-8080-test-jndi.war-\_test-jndi-any-5875179471315066596.dir/webapp/,AVAILABLE}{/test-jndi.war}

2015-02-09 04:42:56.747:INFO:oejsh.ContextHandler:main: Started o.e.j.s.h.MovedContextHandler@58e6c418{/oldContextPath,null,AVAILABLE}

2015-02-09 04:42:57.324:WARN::main: test-spec webapp is deployed. DO NOT USE IN PRODUCTION!

Calling TestListener.contextInitialized

2015-02-09 04:42:57.634:INFO:oejsh.ContextHandler:main: Started o.e.j.w.WebAppContext@57b0fd63{/test-spec,[file:/tmp/jetty-0.0.0.0-8080-test-spec.war-\_test-spec-any-8818760834781530071.dir/webapp/, jar:file:/tmp/jetty-0.0.0.0-8080-test-spec.war-\_test-spec-any-8818760834781530071.dir/webapp/WEB-INF/lib/test-web-fragment-9.2.7.v20150116.jar!/META-INF/resources],AVAILABLE}{/test-spec.war}

2015-02-09 04:42:58.321:INFO:oejsh.ContextHandler:main: Started o.e.j.w.WebAppContext@1e6e49a4{/proxy,file:/tmp/jetty-0.0.0.0-8080-xref-proxy.war-\_xref-proxy-any-3828059417986992849.dir/webapp/,AVAILABLE}{/xref-proxy.war}

2015-02-09 04:42:58.983:WARN::main: test webapp is deployed. DO NOT USE IN PRODUCTION!

2015-02-09 04:42:59.708:INFO:oejsh.ContextHandler:main: Started o.e.j.w.WebAppContext@52bd157{/test,file:/tmp/jetty-0.0.0.0-8080-test.war-\_test-any-7950694189590271229.dir/webapp/,AVAILABLE}{/test.war}

2015-02-09 04:43:00.112:WARN::main: async-rest webapp is deployed. DO NOT USE IN PRODUCTION!

2015-02-09 04:43:00.163:INFO:oejsh.ContextHandler:main: Started o.e.j.w.WebAppContext@62f4b327{/async-rest,[file:/tmp/jetty-0.0.0.0-8080-async-rest.war-\_async-rest-any-1162777238280906246.dir/webapp/, jar:file:/tmp/jetty-0.0.0.0-8080-async-rest.war-\_async-rest-any-1162777238280906246.dir/webapp/WEB-INF/lib/example-async-rest-jar-9.2.7.v20150116.jar!/META-INF/resources],AVAILABLE}{/async-rest.war}

2015-02-09 04:43:00.179:INFO:oejs.ServerConnector:main: Started ServerConnector@2e3414dc{HTTP/1.1}{0.0.0.0:8080}

2015-02-09 04:43:00.193:INFO:oejs.ServerConnector:main: Started ServerConnector@51701830{SSL-http/1.1}{0.0.0.0:8443}

2015-02-09 04:43:00.194:INFO:oejs.Server:main: Started @7403ms

游览器访问 http://localhost:8080, 现在你就可以看到一个演示页面和测试页.

### 警告

演示的web应用程序不一定是安全的，不要部署在生产web服务器上。

### 创建一个新的Jetty部署

上述的demo-base目录是在Jetty9.1中添加到jetty的一个例子。Jetty允许服务器的配置和web应用程序单独存放在Jetty的部署，使升级可以以最小的中断来完成。Jetty的默认配置是基于两个属性：

jetty.home

定义Jetty分布的位置，它的库，默认模块和默认XML文件属性。(typically start.jar, lib, etc)

jetty.base

它定义了一个Jetty服务器的特定实例的位置，它的配置，日志和web应用程序的属性。 (typically start.ini, start.d, logs and webapps)

所述jetty.home和jetty.base性质可以显式地设置在命令行上，或者它们可以从环境推断，如果使用命令：:

**> cd $HOME/my-base**

**> java -jar $HOME/jetty-distribution-9/start.jar**

下面的命令创建一个新的基础目录，并启用它HTTP连接器和Web应用程序部署：

**> mkdir /tmp/mybase**

**> cd /tmp/mybase**

**> java -jar $HOME/jetty-distribution-9/start.jar**

WARNING: Nothing to start, exiting ... Usage: java -jar start.jar [options] [properties] [configs]

java -jar start.jar --help # for more information

> **java -jar $HOME/jetty-distribution-9/start.jar --add-to-startd=http,deploy**

http initialised in ${jetty.base}/start.d/http.ini (created)

server initialised in ${jetty.base}/start.d/server.ini (created)

deploy initialised in ${jetty.base}/start.d/deploy.ini (created)

MKDIR: ${jetty.base}/webapps

server initialised in ${jetty.base}/start.d/server.ini

**> java -jar $HOME/jetty-distribution-9/start.jar**

2013-09-06 14:59:32.542:INFO:oejs.Server:main: jetty-9-demo

2013-09-06 14:59:32.572:INFO:oejdp.ScanningAppProvider:main: Deployment monitor [file:/tmp/mybase/webapps/] at interval 1

2013-09-06 14:59:32.602:INFO:oejs.ServerConnector:main: Started ServerConnector@405a2273{HTTP/1.1}{0.0.0.0:8080}

[...]

### 改变Jetty端口

你可以通过在命令行中设置jetty.port改变端口：

**> cd $HOME/jetty-distribution-9/demo-base**

**> java -jar ../start.jar jetty.port=8081**

你可以将属性值添加到start.ini文件或者start.d/\*.ini文件中。默认情况下，jetty定义jetty.port属性在start.d/http.ini文件中，可以被修改。

### 笔记

通过以下配置属性：

* start.d/ http.ini文件是有效的命令行的一部分，包含--module= HTTP参数，它激活HTTP模块
* 模块/ http.mod文件定义指定了etc / jetty-http的http模块。它使用xml配置文件和模板ini属性
* 该jetty.port属性用于在etc/jetty.http.xml属性的XML元素的ServerConnector实例与端口注入

更多配置信息后面稍后介绍。

### 通过HTTPS运行

Https连接器添加到jetty配置，可以通过以下命令生成：

**> java -jar start.jar --add-to-startd=https**

https initialised in ${jetty.home}/start.d/https.ini (created)

ssl initialised in ${jetty.home}/start.d/ssl.ini (created)

server enabled in ${jetty.home}/start.ini

resources enabled in ${jetty.home}/start.ini

ext enabled in ${jetty.home}/start.ini

**> java -jar start.jar**

2013-09-06 13:52:43.326:INFO:oejs.Server:main: jetty-9...

[...]

--add-to-startd执行如下操作：

* 创建https.ini文件激活和配置https连接器模块。https模块增加etc/jetty-https.xml文件到命令行中。
* 创建start.d/ssl.ini文件激活和配置SSL keystore。ssl模块增加etc/jetty-ssl.xml文件到命令行。
* 检查etc/keystore文件是否存在，如果不存在，则下载一个示范keystore文件。

### 笔记

如果一个start.ini文件优先于个人start.d/\*.ini文件,然后选择——add-to-start =module 可用于附加模块激活到start.d ini文件,而不是创建一个文件。

#### 改变Jetty HTTPS端口

你可以配置SSL连接器不同的端口上运行通过设置https。命令如下：

**> cd $HOME/jetty-distribution-9/demo-base**

**> java -jar ../start.jar https.port=8444**

另外,属性值可以一开始就建立 .ini文件和start.d / \*.INI文件。如果你使用 —— add-to-startd 命令启用https，然后您可以编辑该属性在start.d/https.ini文件。

### 更多 start.jar 选项

该start.jar的工作就是解释命令行，start.ini和start.d建立属性和配置文件的Java类路径和列表传递给主类的Jetty的XML配置机制。该start.jar机构具有记录在启动管理部分，你可以通过使用命令看到他们在总结了许多选项：

> java -jar start.jar --help

# jetty部署Web应用程序

# 部署web应用

在jetty目录中的webaaps目录中，只要放标准的WAR文件和jetty的配置文件，则jetty会自动热部署你的web应用。

* 一个标准的web应用，子目录要有WEB-INF/,否者则被部署为静态内容，访问的路径是http://localhost:8080/example，如果是ROOT（不区分大小写）目录，则上下文地址是 / ，如果目标结尾是”.d”,它会被忽视（通过配置使用）
* 如果部署一个example.war，访问的上下文路径就是/example（即：http://localhost:8080/example），如果命名为ROOT(不区分大小写)，则上下文路径就是 /,如果example.war和example存在，只有WAR会部署。
* 如果是XML部署，必须在XML中配置上下文地址，如果example.war和example存在，只会部署XML。

如果你有标准的web应用，你可以复制到jetty中webapps目录中。

### jetty web应用示例

下面是我们提供的示例webapps:

**ROOT/**

      静态内容目录，访问用 / ,jetty的欢迎页就在里面。

**test.xml**

      用于配置和部署test.war,还包括配置在test.d目录中的上下文路径以及设置额外的描述符。  
**test.war**  
      应用示例，上面的test.xml配置和部署的。  
**test.d**

      目录里，是通过test.xml使用其他的配置文件去注入额外的配置生成的test.war

**async-rest.war**

    异步REST的web应用示例到eBay,自动部署到基于文件名 /async-rest上。

**test-jaas.war**

      web应用示例，利用JAAS做验证。

**test-jaas.xml**  
      test-jass.war的一个上下文配置文件，其他配置包括LoginService，是为身份认证和授权。  
**test-jndi.war**

      展示了使用jNDI的一个web应用示例。

**test-jndi.xml**  
      test-jndi.war的配置文件，其他配置包括从webapp引用的命名空间中定义的对象。  
**test-spec.war**  
      展示使用annotations，fragments,ServletContainerInitializers 和其他的 Servlet Specification 3.0/3.1功能的web应用示例，  
**xref-proxy.war**

      使用透明的代理从Eclipse网站去服务Jetty src外部参考的web应用示例。

**example-moved.xml**

      展示如何使用MovedContextHandler从一个地址重定向到另一个地址的web应用示例。

# 在maven中配置jetty

# Maven中找到jetty

### Maven的坐标

jetty几乎在maven成立以来就支持了，只是一直在不同的项目下，当时jetty是基于SourceForge,Codehaus它位于org.mortbay.jetty这个groupid下，随着jetty7搬到Eclipse基金会后，就使用了新的groupid。

<dependency>

<groupId>org.eclipse.jetty</groupId>

<artifactId>jetty-project</artifactId>

<version>${project.version}</version>

</dependency>

### 中央的更新日志

jetty的版本之间的变化跟踪记录在 VERSIONS.txt 文件中，是源码控制下释放生成的，这些生成的POM文件在释放过程中也会上传到Maven仓库里，你可以找到它们。

<http://repo2.maven.org/maven2/org/eclipse/jetty/jetty-project/9.0.0.RC0/jetty-project-9.0.0.RC0-version.txt>

<dependency>

  <groupId>org.eclipse.jetty</groupId>

  <artifactId>jetty-project</artifactId>

  <version>${project.version}</version>

  <classifier>version</classifier>

  <type>txt</type>

</dependency>

# jetty的配置介绍

# 第3章. Jetty配置介绍

### 如何配置jetty

要明白jetty的配置，你需要先明白“如何做”和“怎么做”，这一节介绍如何配置jetty在什么样的机制下去执行配置，下一节介绍你如何配置这些机制。

### jetty POJO配置

jetty的核心组件是简单传统的JAVA对象（POJO），配置jetty的过程其实就是实例化对象的过程。

* 编写JAVA代码直接实例化装配到jetty对象，这被称为嵌入jetty。
* 使用XML配置，这种是控制反转（Ioc）框架，在XML中实例化和装配jetty对象。etc/jetty.xml是主的Jetty XML配置文件，但也很多其他的etc/feature.xml 文件包含在jetty中。
* 使用第三方的Ioc框架，像Spring，实例化和装配jetty对象到Spring bean.

因为jetty配置文件是通过Ioc完成的，Jetty API文档是最标准的配置参考。

### Jetty启动配置文件

Jetty通过以下配置文件实例化，通过start.jar注射和启动服务器。

**ini files**  
jetty启动采用命令行的方式。 $JETTY\_BASE/start.ini 或 $JETTY\_BASE/start.d/\*.ini 文件去创建有效参数的命令行。参数可能是：

* 在jetty Ioc（或spring）XML格式中的XML文件。
* 模块激活形式 --module=name
* 属性的格式 name=value，用参数表示Jetty Ioc XML
* 一个标准的JAVA属性文件包含附加的启动属性。
* 其他的start.jar选项（参见java -jar start.jar --help）
* 一些JVM选项

ini文件位于jetty的安装地址（如果jetty home不同），通过编辑去更改配置（例如，修改端口）。

**mod文件**

 $JETTY\_HOME/modules/\*.mod文件包含模块的定义，可以通过- -module=name激活使用，每个模块文件的定义如下:

* 订购和激活模块依赖关系
* 模块所需要的库添加到classpath。
* 模块所需要的XML文件添加到有效的命令行。
* 通过激活模块所需要的文件。
* 用—add-to-start=name选项激活时，要使用模板ini文件。

通常模块文件很少编辑，只有重大的结构性变化才会修改。\*.mod文件通常放在$JETTY\_HOME/modules/，但是额外的和编辑过的模块可能在$JETTY\_BASE/module。如果需要模块改动，最好在修改之前，从$JETTY\_HOME/modules/复制到$JETTY\_BASE/modules/目录，在修改。

**XML files**

Jetty IoC XML格式或spring IoC格式的XML文件有2种，在ini文件或通过模块定义增加命令行。XML文件和Java对象组成了服务器，连接器和环境。因为Jetty IoC XML文件使用属性，最常见的配置可以在不修改这些XML文件来完成，而是可以通过编辑相应的ini文件属性来实现。

这些XML文件通常位于 $JETTY\_HOME/etc/,额外或编辑的XML文件可以添加到 $JETTY\_BASE/etc/.如果配置需要改变，最好先从$JETTY\_HOME/etc/ 复制到 $JETTY\_BASE/etc/，在进行修改。

**web.xml**  
Servlet规范定义了定义和配置Web应用程序使用的过滤器，servlet和资源web.xml部署描述符。Jetty WebAppContext组件使用这种XML格式：

* web应用上下文的默认配置。
* 解释在WEB-INF/web.xml文件的web应用程序所提供的特定的配置。
* 解释描述符代码段，包含Jar文件中的WEB-INF/lib的META-INF目录。

Normally the web.xml file for a web application is found in the WEB-INF/web.xml location within the war file/directory or as web.xml fragments with jar files found in WEB-INF/lib. Jetty also supports multiple web.xml files so that a default descriptor may be applied before WEB-INF/web.xml (typically set to etc/webdefault.xml by the deploy module) and an override descriptor may be applied after WEB-INF/web.xml (typically set by a context XML file see test.xml)

**Property Files**

标准的Java属性文件也用于jetty配置

* 通过使用属性元素的jetty IoC XML的参数。
* 要配置默认的日志记录机制（StdErrLog),你也可以引入其他日志记录框架，并且还可以使用属性文件。（例如,log4j）
* 作为一个简单的数据库登录用户名和凭据。

### Jetty IoC XML format

要明白Jetty Ioc XML的格式，下面是一个嵌入式的jetty服务器和配置在Java里的例子。

package org.eclipse.jetty.embedded;

import org.eclipse.jetty.server.Connector;

import org.eclipse.jetty.server.Handler;

import org.eclipse.jetty.server.Server;

import org.eclipse.jetty.server.ServerConnector;

import org.eclipse.jetty.server.handler.DefaultHandler;

import org.eclipse.jetty.server.handler.HandlerCollection;

import org.eclipse.jetty.servlet.ServletContextHandler;

public class ExampleServer {

public static void main(String[] args) throws Exception {

Server server = new Server();

ServerConnector connector = new ServerConnector(server);

connector.setPort(8080);

server.setConnectors(new Connector[] { connector });

ServletContextHandler context = new ServletContextHandler();

context.setContextPath("/");

context.addServlet(HelloServlet.class, "/hello");

context.addServlet(AsyncEchoServlet.class, "/echo/\*");

HandlerCollection handlers = new HandlerCollection();

handlers.setHandlers(new Handler[] { context, new DefaultHandler() });

server.setHandler(handlers);

server.start();

server.join();

}

}

Jetty ioc XML格式允许你在XML中实例化和配置完全相同的服务器，而无需编写任何Java代码：

<?xml version="1.0"?>

<!DOCTYPE Configure PUBLIC "-//Jetty//Configure//EN" "http://www.eclipse.org/jetty/configure\_9\_3.dtd">

<Configure id="ExampleServer" class="org.eclipse.jetty.server.Server">

<Set name="connectors">

<Array type="org.eclipse.jetty.server.Connector">

<Item>

<New class="org.eclipse.jetty.server.ServerConnector">

<Arg><Ref refid="ExampleServer"/></Arg>

<Set name="port">8080</Set>

</New>

</Item>

</Array>

</Set>

<New id="context" class="org.eclipse.jetty.servlet.ServletContextHandler">

<Set name="contextPath">/hello</Set>

<Call name="addServlet">

<Arg>org.eclipse.jetty.embedded.HelloServlet</Arg>

<Arg>/</Arg>

</Call>

</New>

<Set name="handler">

<New class="org.eclipse.jetty.server.handler.HandlerCollection">

<Set name="handlers">

<Array type="org.eclipse.jetty.server.Handler">

<Item>

<Ref refid="context" />

</Item>

<Item>

<New class="org.eclipse.jetty.server.handler.DefaultHandler" />

</Item>

</Array>

</Set>

</New>

</Set>

</Configure>

在实践中，最常用的是在etc目录中的标准XML文件，因此，配置jetty通常只需要修改存在的XML文件和修改它们注入的属性值。

# jetty部署

## 第二部分 第四章：jetty部署

本章讨论部署jetty的各种方式，讨论的范畴从部署绑定到部署第三方产品，还包含调度管理，提供web应用程序和叠加覆盖部署。

# 剖析jetty的Web应用程序

Web应用可动态的部署集合（servlets, filters, jsps, etc..）和静态的内容、支持库和描述元数据。

是按照Servlet的规范定义的格式和布局。你可以查看官方Servlet规范文档进行更详细的了解。

Web应用程序可以打包成一个Web Archive(WAR文件)或作为一个目录树。

* /WEB-INF/

特殊的 Servlet API 定义用来存储任何不属于公共访问的 Web 应用的目录。不会被Web游览器直接访问到。

* /WEB-INF/web.xml

定义你的Web应用的行为。

* /WEB-INF/classes/

JAVA类位置

* /WEB-INF/lib/

JAR文件目录（库）

jetty的WebAppClassloader首先会从/WEB-INF/classes/加载类。然后在/WEB-INF/lib/中加载jar文件。

# jetty自动部署web应用

部署web应用的最简单的方法是把你的WAR文件或WAR的目录文件放到${jetty.home}/webapps/目录，让Jetty扫描到它，并根据同名的上下文地址部署。

只有遵循web应用程序的布局方式，jetty会进行检测和部署。

指向你的WAR包的文件名（或目录名）。

##### 例如：文件或目录名对应的上下文

| **文件 或 目录名** | **指定的上下文地址** |
| --- | --- |
| /webapps/footrope.war | <http://host/footrope/> |
| /webapps/baggywrinkle-1.0.war | <http://host/baggywrinkle-1.0/> |
| /webapps/lazaret-2.1.3-SNAPSHOT.war | <http://host/lazaret-2.1.3-SNAPSHOT/> |
| /webapps/belaying-pins/WEB-INF/web.xml | <http://host/belaying-pins/> |
| /webapps/root.war(special name) | <http://host/> |
| /webapps/root/WEB-INF/web.xml(special name) | <http://host/> |

# 部署特定的web应用 - jetty

自动 Web 应用程序部署非常的快速简便，但有时候你可能需要调整某些部署的属性（例如，你想调整上下文地址，不基于文件名称，或者你想为web应用定义一个特殊的数据库连接池）。你可以使用Jetty Deployable Descriptor XML File来完成这种调整。

## Jetty部署描述符XML文件

jetty支持通过XML文件构建一个ContextHandler的实例部署web应用。

## 使用基本的描述符文件

默认安装的jetty，会自动扫描它的 $JETTY\_HOME/webapps 目录，若要部署web应用，则需将文件放在该目录中即可。

部署描述符文件本身就是一个配置了 WebAppContext类的XML文件，在默认安装的情况下，你只需设置2个属性：

* war  
  web应用程序的文件系统路径(或目录)
* contextPath  
  上下文路径使用的web应用程序

例如：这是一个描述符文件，项目/opt/myapp/myapp.war指定上下文路径为/wiki:

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

<!DOCTYPE Configure PUBLIC "-//Jetty//Configure//EN" "http://www.eclipse.org/jetty/configure\_9\_0.dtd">

<Configure class="org.eclipse.jetty.webapp.WebAppContext">

<Set name="contextPath">/wiki</Set>

<Set name="war">/opt/myapp/myapp.war</Set>

</Configure>

##### 注意

你可以使用系统属性和属性元素，如果你设置系统属性myapp.home=/opt/myapp,你可以重写前面的例子。

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

<!DOCTYPE Configure PUBLIC "-//Jetty//Configure//EN" "http://www.eclipse.org/jetty/configure\_9\_0.dtd">

<Configure class="org.eclipse.jetty.webapp.WebAppContext">

<Set name="contextPath">/wiki</Set>

<Set name="war"><SystemProperty name="myapp.home"/>/myapp.war</Set>

</Configure>

如果你需要修改home路径，你可以简单的改变系统属性，如果你经常为一个app在多版本之间切换。这就非常有用了。

配置先进的描述符文件  
如果你看过WebAppContext类的文档，你会发现不仅仅是只有上面提到的2个属性，下面这些例子，是介绍高级描述符文件的：

第一个例子告诉Jetty，在部署的时候不展开WAR文件。这可以让大家知道，不应该更改这个临时解压的WAR,因为这种变化不持续，因此不适用用于web应用程序部署。

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

<!DOCTYPE Configure PUBLIC "-//Jetty//Configure//EN" "http://www.eclipse.org/jetty/configure\_9\_0.dtd">

<Configure class="org.eclipse.jetty.webapp.WebAppContext">

<Set name="contextPath">/wiki</Set>

<Set name="war"><SystemProperty name="myapp.home"/>/myapp.war</Set>

<Set name="extractWAR">false</Set>

</Configure>

下一个例子是检索Java EE Servlet的上下文，并设置它的初始化参数。您还可以使用setAttribute方法来设置一个Servlet上下文属性。但是，由于Web应用程序的web.xml文件中的部署描述符之后处理，所以web.xml值会覆盖同名的描述符属性。

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

<!DOCTYPE Configure PUBLIC "-//Jetty//Configure//EN" "http://www.eclipse.org/jetty/configure\_9\_0.dtd">

<Configure class="org.eclipse.jetty.webapp.WebAppContext">

<Set name="contextPath">/wiki</Set>

<Set name="war"><SystemProperty name="myapp.home"/>/myapp.war</Set>

<Get name="ServletContext">

<Call name="setInitParameter">

<Arg>myapp.config</Arg>

<Arg><SystemProperty name="myapp.home">/config/app-config.xml</Arg>

</Call>

</Get>

</Configure>

下面的示例设置一个特殊的web.xml覆盖描述符。该描述符是在web应用程序的web.xml之后处理的，因此它可以覆盖相同名称的属性。如果你想添加的参数或附加的Servlet映射没有破开一个打包WAR文件，那此功能是非常有用的。

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

<!DOCTYPE Configure PUBLIC "-//Jetty//Configure//EN" "http://www.eclipse.org/jetty/configure\_9\_0.dtd">

<Configure class="org.eclipse.jetty.webapp.WebAppContext">

<Set name="contextPath">/wiki</Set>

<Set name="war"><SystemProperty name="myapp.home"/>/myapp.war</Set>

<Set name="overrideDescriptor">/opt/myapp/overlay-web.xml</Set>

</Configure>

接下来的例子不仅是web应用的上下文配置，又是一个数据库连接池（参见数据源的例子，如果web.xml不包含这个数据源的引用，可以使用覆盖描述符的机制（前面的列子）我们的应用程序就可以使用），将其列入：

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

<!DOCTYPE Configure PUBLIC "-//Jetty//Configure//EN" "http://www.eclipse.org/jetty/configure\_9\_0.dtd">

<Configure class="org.eclipse.jetty.webapp.WebAppContext">

<Set name="contextPath">/wiki</Set>

<Set name="war"><SystemProperty name="myapp.home"/>/myapp.war</Set>

<New id="DSTest" class="org.eclipse.jetty.plus.jndi.Resource">

<Arg></Arg>

<Arg>jdbc/DSTest</Arg>

<Arg>

<New class="org.apache.commons.dbcp.BasicDataSource">

<Set name="driverClassName">org.some.Driver</Set>

<Set name="url">jdbc.url</Set>

<Set name="username">jdbc.user</Set>

<Set name="password">jdbc.pass</Set>

</New>

</Arg>

</New>

</Configure>

还有很多其他的设置，你可以在WebAppContext上进行改变，可以参考javadoc，在war在部署期间，也避免了zip文件异常，无论是自动解压或是否将 web 应用程序的某些部分复制到一个临时位置的 WebAppContext 设置说明，请参阅文档。

# WebAppContexts的部署处理 - jetty

# WebAppContexts 的部署处理

进入服务前，web应用程序需要一定的处理：可能需要解压，为jar创建一个特殊的类加载器，web.xml和web-fragment.xml描述符处理，扫描类注解以及很多其他的东西，随着web应用程序变得更加复杂，我们增加方法来帮助你扩大或减少在部署时所做的处理量，在本节中，我们将看看这个处理，以及如何调节它。

相反，如果你正在寻找如何配置特定的WebAppContext - 如它的上下文路径配置，是否应该解压或不解压。那你应该看下一节。

### 类配置

web应用部署，关于一系列的org.eclipse.jetty.webapp.Configuration类都适用，每一个执行一个特定的功能，这些配置的顺序具有重要意义，后面的配置倾向于构建在信息提取或安装在上述配置， 这些默认的列表，命令，按顺序的应用到每 org.eclipse.jetty.webapp.WebAppContext；

**表4.1。默认配置类**

|  |  |
| --- | --- |
| [org.eclipse.jetty.webapp.WebInfConfiguration](http://download.eclipse.org/jetty/stable-9/apidocs/org/eclipse/jetty/webapp/WebInfConfiguration.html) | 提取 war, jars 和 定义的类路径 |
| [org.eclipse.jetty.webapp.WebXmlConfiguration](http://download.eclipse.org/jetty/stable-9/apidocs/org/eclipse/jetty/webapp/WebXmlConfiguration.html) | 处理 WEB-INF/web.xml 文件 |
| [org.eclipse.jetty.webapp.MetaInfConfiguration](http://download.eclipse.org/jetty/stable-9/apidocs/org/eclipse/jetty/webapp/MetaInfConfiguration.html) | 寻找容器和web应用中的jar，为 META-INF/resources 和 META-INF/web-fragment.xml |
| [org.eclipse.jetty.webapp.FragmentConfiguration](http://download.eclipse.org/jetty/stable-9/apidocs/org/eclipse/jetty/webapp/FragmentConfiguration.html) | 处理所有找到的 META-INF/web-fragment.xml 文件 |
| [org.eclipse.jetty.webapp.JettyWebXmlConfiguration](http://download.eclipse.org/jetty/stable-9/apidocs/org/eclipse/jetty/webapp/JettyWebXmlConfiguration.html) | 处理 WEB-INF/jetty-web.xml 文件 |

### 解剖配置类

WebAppContext的5种不同的阶段的生命周期，  
**preConfigure**

        作为WebAppContext启动前执行的这个阶段，发现任何在随后的阶段中将需要用到的资源。  
**configure**  
        这个阶段就是类的工作,通常使用的资源在preConfigure阶段发现的。  
**postConfigure**  
        这一阶段允许配置明确下创建的任何资源，可能是在前面的2阶段不需要的。  
**deconfigure**

        这个阶段每当发生WebAppContext被停止，并允许撤销任何创建的资源/元数据。WebAppContext应该能够不占用资源的干净的启动/停止多次。  
**destroy**  
当WebAppContext实际上从服务中移除了。例如,与之关联的war文件从$JETTY\_HOME/webapps目录中删除。

按顺序列出的每个阶段，例如，用我们默认的配置类作为例子，preConfigure()将调用WebInfConfiguration，WebXmlConfiguration, MetaInfConfiguration,FragmentConfiguration和JettyWebXmlConfiguration.  循环重新开始configure()和再一次postConfigure()，在相反的顺序循环重复的deconfigure()和最后的destroy()阶段；

# 通过创建额外配置扩展容器支持

正如我们所看到的，有一个支持web应用的基础部署配置的默认设置，你会发现，我们还没提到JavaEE的功能，如JNDI，也没有高级servlet规范功能，如注解。这是因为jetty的理念是允许用户根据自己的需要调整容器，如果你不需要这些功能，你可以不使用它们，一个重要的考虑因素是因为例如 注解需要大量且费时的去扫描WEB-INF/lib的jar，作为有思想的web应用可能具有这些jar的分值，也可以通过有效的部署延迟，我们将在其他单元看到另一个 jetty为减少花费分析jar的webapp，

jetty利用了配置的灵活性，使JNDI和注释支持可插拔。

首先，让我们看看如何配置帮助启用 JNDI。  
  
JNDI查找在web应用程序中需要容器到容器的环境的web应用程序中定义的连接资源。为实现这一目标，我们使用 2 额外配置 ︰  
  
表4.2。 JNDI配置类

|  |  |
| --- | --- |
| [org.eclipse.jetty.plus.webapp.EnvConfiguration](http://download.eclipse.org/jetty/stable-9/apidocs/org/eclipse/jetty/plus/webapp/EnvConfiguration.html) | 为 webapp，创建 java:comp/env, 适用于 WEB-INF/jetty-env.xml 文件 |
| [org.eclipse.jetty.plus.webapp.PlusConfiguration](http://download.eclipse.org/jetty/stable-9/apidocs/org/eclipse/jetty/plus/webapp/PlusConfiguration.html) | JNDI进程相关的 WEB-INF/web.xml 和 钩子命名条目 |

这些配置必须严格上面的顺序添加。并应紧接着之前的配置列表中org.eclipse.jetty.webapp.JettyWebXmlConfiguration类插入。为了充分支持JNDI，你需要做一些其他的事情，你可以在这里找到其中的全部细节。

### 如何设置配置列表

你有很多的选项,如何使Jetty使用不同的配置列表。

### 直接在WebAppContext设置列表

如果你只有一个web应用，你想要影响，这可能是最简单的选择，但是，要么需要一个你自己的web应用程序上下文的xml，或者你需要在代码中调用相应的。让我们看看如何添加JNDI和注释的配置的示例。

<?xml version="1.0"?>

<!DOCTYPE Configure PUBLIC "-//Jetty//Configure//EN" "http://www.eclipse.org/jetty/configure\_9\_0.dtd">

<Configure class="org.eclipse.jetty.webapp.WebAppContext">

<Set name="war"><SystemProperty name="jetty.home" default="."/>/webapps/my-cool-webapp</Set>

<Set name="configurationClasses">

<Array type="java.lang.String">

<Item>org.eclipse.jetty.webapp.WebInfConfiguration</Item>

<Item>org.eclipse.jetty.webapp.WebXmlConfiguration</Item>

<Item>org.eclipse.jetty.webapp.MetaInfConfiguration</Item>

<Item>org.eclipse.jetty.webapp.FragmentConfiguration</Item>

<Item>org.eclipse.jetty.plus.webapp.EnvConfiguration</Item>

<Item>org.eclipse.jetty.plus.webapp.PlusConfiguration</Item>

<Item>org.eclipse.jetty.annotations.AnnotationConfiguration</Item>

<Item>org.eclipse.jetty.webapp.JettyWebXmlConfiguration</Item>

</Array>

</Set>

</Configure>

当然，你也可以用这种方法来减少应用于特定WebAppContext配置。

### 设置列表部署所有的webapps

你可以设置配置类的列表上的WebAppProvider。这样就将应用于每个部署人员部署的 WebAppContext：

<?xml version="1.0"?>

<!DOCTYPE Configure PUBLIC "-//Jetty//Configure//EN" "http://www.eclipse.org/jetty/configure\_9\_0.dtd">

<Configure id="Server" class="org.eclipse.jetty.server.Server">

<Call name="addBean">

<Arg>

<New id="DeploymentManager" class="org.eclipse.jetty.deploy.DeploymentManager">

<Set name="contexts">

<Ref refid="Contexts" />

</Set>

<Call id="webappprovider" name="addAppProvider">

<Arg>

<New class="org.eclipse.jetty.deploy.providers.WebAppProvider">

<Set name="monitoredDirName"><Property name="jetty.home" default="." />/webapps</Set>

<Set name="configurationClasses">

<Array type="java.lang.String">

<Item>org.eclipse.jetty.webapp.WebInfConfiguration</Item>

<Item>org.eclipse.jetty.webapp.WebXmlConfiguration</Item>

<Item>org.eclipse.jetty.webapp.MetaInfConfiguration</Item>

<Item>org.eclipse.jetty.webapp.FragmentConfiguration</Item>

<Item>org.eclipse.jetty.plus.webapp.EnvConfiguration</Item>

<Item>org.eclipse.jetty.plus.webapp.PlusConfiguration</Item>

<Item>org.eclipse.jetty.annotations.AnnotationConfiguration</Item>

<Item>org.eclipse.jetty.webapp.JettyWebXmlConfiguration</Item>

</Array>

</Set>

</New>

</Arg>

</Call>

</New>

</Arg>

</Call>

</Configure>

不用列举完整的列表,您可以简单地指定您想要添加的类，并指定你希望它们插入列表中的位置。让我们看看用这种方法在JNDI配置支持添加的一个列子 - 像以前一样，你可以在xml文件或通过代码。此示例使用XML文件，实际上它是$JETTY\_HOME/etc/jetty-plus.xm。

<?xml version="1.0"?>

<!DOCTYPE Configure PUBLIC "-//Jetty//Configure//EN" "http://www.eclipse.org/jetty/configure\_9\_0.dtd">

<Configure id="Server" class="org.eclipse.jetty.server.Server">

<!-- =========================================================== -->

<!-- Add plus Configuring classes to all webapps for this Server -->

<!-- =========================================================== -->

<Call class="org.eclipse.jetty.webapp.Configuration$ClassList" name="setServerDefault">

<Arg><Ref refid="Server" /></Arg>

<Call name="addAfter">

<Arg name="afterClass">org.eclipse.jetty.webapp.FragmentConfiguration</Arg>

<Arg>

<Array type="String">

<Item>org.eclipse.jetty.plus.webapp.EnvConfiguration</Item>

<Item>org.eclipse.jetty.plus.webapp.PlusConfiguration</Item>

</Array>

</Arg>

</Call>

</Call>

</Configure>

org.eclipse.jetty.webapp.Configuration.ClassList类提供这些方法插入：  
  
**addAfter**  
        在给定的配置类名称后插入提供的配置类名称的列表。  
**addBefore**  
        在给定配置类名称之前插入配置类名称提供的列表。

# 其他配置

**org.eclipse.jetty.server.webapp.ContainerIncludeJarPattern**  
  
这是一个上下文属性,可以设置在org.eclipse.jetty.webapp.WebAppContext去控制容器的类路径的哪些部分应该处理注释、META-INF/resources, META-INF/web-fragment.xml和META-INF里的tld。  
  
通常情况下，容器类路径没有包括处理。不过，有时你需要包括一些，例如，你可能需要共享一些libraries，因此你把它们放进$JETTY\_HOME/lib目录中。libraries包含注解,因此必须扫描。  
  
下面是一个例子：

<?xml version="1.0"?>

<!DOCTYPE Configure PUBLIC "-//Jetty//Configure//EN" "http://www.eclipse.org/jetty/configure\_9\_0.dtd">

<Configure class="org.eclipse.jetty.webapp.WebAppContext">

<Call name="setAttribute">

<Arg>org.eclipse.jetty.server.webapp.ContainerIncludeJarPattern</Arg>

<Arg>.\*/foo-[^/]\*\.jar$|.\*/bar-[^/]\*\.jar$|.\*/classes/.\*</Arg>

</Call>

</Configure>

注意：模式的顺序限定了jar或类目录的扫描顺序。

**org.eclipse.jetty.server.webapp.WebInfIncludeJarPattern**

类似于前面的上下文属性,该属性控制处理jar，像注释、META-INF/resources, META-INF/web-fragment.xml 和META-INF数据的tld。不管怎样，从web应用的类路径的这个属性控制（通常是WEB-INF/lib）进行处理，当你有几个jar包的时候，但是你只需要扫描一些jar包，这就特别有用了。  
  
例子：

<?xml version="1.0"?>

<!DOCTYPE Configure PUBLIC "-//Jetty//Configure//EN" "http://www.eclipse.org/jetty/configure\_9\_0.dtd">

<Configure class="org.eclipse.jetty.webapp.WebAppContext">

<Call name="setAttribute">

<Arg>org.eclipse.jetty.server.webapp.WebInfIncludeJarPattern</Arg>

<Arg>.\*/spring-[^/]\*\.jar$</Arg>

</Call>

</Configure>

注意：模式的顺序限定了jar或类目录的扫描顺序。

# jetty热部署

**热部署**

jetty让你通过监测目录变化将任意上下文或 web 应用程序的部署。如果你向目录中添加一个web 应用程序或上下文描述符，jetty的DeploymentManager(DM) 部署新的上下文。如果你touch或更新上下文描述符，DM 停止、 重新配置和重新部署它的上下文。如果您删除一个上下文，DM停止它，并从服务器上删除。

为了控制这种行为，你可以配置WebAppProvider属性：

**monitoredDirName**  
        该目录扫描可能部署的web应用（或部署描述符的XML文件）  
  
**scanInterval**  
        提供 monitoredDirName 扫描之间的秒数。  
        值为 0 将禁用连续热部署扫描，只在启动时部署Web应用程序。  
  
这个配置的默认位置在${jetty.home}/etc/jetty-deploy.xml。

<?xml version="1.0"?>

<!DOCTYPE Configure PUBLIC "-//Jetty//Configure//EN" "http://www.eclipse.org/jetty/configure\_9\_0.dtd">

<Configure id="Server" class="org.eclipse.jetty.server.Server">

<Call name="addBean">

<Arg>

<New id="DeploymentManager" class="org.eclipse.jetty.deploy.DeploymentManager">

<Set name="contexts">

<Ref refid="Contexts" />

</Set>

<Call name="setContextAttribute">

<Arg>org.eclipse.jetty.server.webapp.ContainerIncludeJarPattern</Arg>

<Arg>.\*/servlet-api-[^/]\*\.jar$</Arg>

</Call>

<Call id="webappprovider" name="addAppProvider">

<Arg>

<New class="org.eclipse.jetty.deploy.providers.WebAppProvider">

<Set name="monitoredDirName"><Property name="jetty.home" default="." />/webapps</Set>

<Set name="defaultsDescriptor"><Property name="jetty.home" default="." />/etc/webdefault.xml</Set>

<Set name="scanInterval">1</Set>

<Set name="extractWars">true</Set>

</New>

</Arg>

</Call>

</New>

</Arg>

</Call>

</Configure>

这个配置的默认位置在${jetty.home}/etc/jetty-deploy.xml。