**配置Jetty Maven插件**

[快速入门：启动并运行](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/jetty-maven-plugin.html#get-up-and-running)

[支持的目标](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/jetty-maven-plugin.html#running-and-deploying)

[配置Jetty容器](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/jetty-maven-plugin.html#configuring-jetty-container)

[配置WebApp](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/jetty-maven-plugin.html#configuring-your-webapp)

[码头：运行](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/jetty-maven-plugin.html#jetty-run-goal)

[码头：运行战争](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/jetty-maven-plugin.html#running-assembled-webapp-as-war)

[码头：运行爆炸](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/jetty-maven-plugin.html#running-assembled-webapp-as-expanded-war)

[码头：部署战](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/jetty-maven-plugin.html#deploy-war-running-pre-assembled-war)

[码头：运行分叉](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/jetty-maven-plugin.html#jetty-run-forked-goal)

[码头：启动](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/jetty-maven-plugin.html#jetty-start-goal)

[码头：停止](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/jetty-maven-plugin.html#jetty-stop-goal)

[码头：有效的web-XML](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/jetty-maven-plugin.html#d0e17478)

[使用重叠战争](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/jetty-maven-plugin.html#using-overlaid-wars)

[配置安全设置](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/jetty-maven-plugin.html#configuring-security-settings)

[使用多个Webapp根目录](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/jetty-maven-plugin.html#using-multiple-webapp-root-directories)

[运行多个Webapp](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/jetty-maven-plugin.html#running-more-than-one-webapp)

[设置系统属性](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/jetty-maven-plugin.html#setting-system-properties)

Jetty Maven插件可用于快速开发和测试。您可以将其添加到根据常用Maven默认值构建的任何webapp项目。然后，插件可以定期扫描您的项目以进行更改，并自动重新部署Web应用程序（如果有）。这通过消除构建和部署步骤使开发周期更高效：您使用IDE对项目进行更改，并且正在运行的Web容器自动选择它们，允许您立即测试它们。

**重要**

您需要为此插件使用Maven 3和Java 1.7。

**快速入门：启动并运行**

首先，添加jetty-maven-plugin到您的 pom.xml定义：

|  |  |
| --- | --- |
| 1  2  3  4  五 | <plugin>    <groupId>org.eclipse.jetty</groupId>    <artifactId>jetty-maven-plugin</artifactId>    <version>9.2.21.v20170120</version>  </plugin> |

然后，从与root用户相同的目录中 pom.xml键入：

mvn码头：跑

这将启动Jetty并在http：// localhost：8080 /上提供您的项目。

Jetty继续运行直到你停止它。在运行时，它会定期扫描项目文件的更改，因此如果您保存更改并重新编译类文件，Jetty将重新部署您的Web应用程序，您可以立即测试您刚刚进行的更改。

您可以ctrl-c在运行它的终端窗口中使用a来终止插件。

**支持的目标**

Jetty Maven插件有许多不同的Maven目标。可以说最有用的是 run我们在快速入门部分看到的目标，该部分在未组装的webapp上运行Jetty。还有其他目标可以帮助您完成不同的任务。例如，您可能需要在Jetty的分叉实例中运行Web应用程序，而不是在运行Maven的进程中运行; 或者您可能需要对您希望部署Web应用程序的maven生命周期阶段进行更精细的控制。完成这些任务有不同的目标，以及其他几个任务。

要查看Jetty Maven插件支持的所有目标列表，请执行以下操作：

mvn码头：帮忙

要查看可针对特定目标配置的详细参数列表，请执行以下操作：

mvn jetty：help -Ddetail = true -Dgoal = goal-name

**配置Jetty容器**

这些配置元素设置了您的webapp执行的Jetty环境。它们与大多数目标相同：

**httpConnector**

可选的。如果未指定，Jetty将创建一个 侦听端口8080 的[ServerConnector](http://download.eclipse.org/jetty/stable-9/apidocs/org/eclipse/jetty/server/ServerConnector.html)实例。例如，可以使用jetty.port命令行上的系统属性更改此默认端口号mvn -Djetty.port=9999 jetty:run。或者，您可以使用此配置元素来设置ServerConnector的信息。以下是有效的配置子元素：

**港口**

要侦听的连接器的端口号。默认情况下为8080。

**主办**

连接器侦听的特定接口。默认情况下，所有接口。

**名称**

连接器的名称，用于[配置上下文以仅响应特定连接器](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/serving-webapp-from-particular-port.html)。

**空闲超时**

连接的最大空闲时间。

**soLinger**

套接字逗留时间。

您可以在标准的[jetty xml配置文件中](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/jetty-xml-config.html)配置连接器，并将其位置放入参数中。请注意，由于jetty-9.0不再可以直接在pom.xml中配置[https连接器](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/jetty-maven-plugin.html#maven-config-https)：您需要[使用jetty xml配置文件来执行此操作](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/jetty-maven-plugin.html#maven-config-https)。 [jettyXml](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/jetty-maven-plugin.html#jettyXml)

**jettyXml**

可选的。除了任何插件配置参数之外，以逗号分隔的要应用的文件位置列表。如果您要部署其他Web应用程序，处理程序，特定类型的连接器等，或者如果您有其他无法从插件配置的Jetty对象，则可以使用它。 [jetty xml](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/jetty-xml-config.html)

**scanIntervalSeconds**

扫描webapp以检查更改并在检测到任何热备份时自动进行热重新部署之间的暂停（以秒为单位）。 **默认情况下，此值为0，这将禁用热部署扫描。**大于0的数字启用它。

**重装**

默认值为“自动”，与非零结合使用会在检测到更改时自动进行热重新部署。设置为“手动”而不是通过在运行插件的控制台中键入换行来触发扫描。当您执行一系列要忽略的更改时，这可能很有用。在该用途中，使用参数。***scanIntervalSeconds***reload

**dumpOnStart**

可选的。默认值为false。如果为true，则jetty将在启动时转储服务器结构。

**loginServices**

可选的。[org.eclipse.jetty.security.LoginService](http://download.eclipse.org/jetty/stable-9/apidocs/org/eclipse/jetty/security/LoginService.html) 实现列表。请注意，没有默认领域。如果你在你的领域使用领域，web.xml你可以在这里指定一个相应的领域。您可以在jetty xml文件中配置登录服务，并将其位置添加到 jettyXml参数中。

**requestLog**

可选的。的的实现[org.eclipse.jetty.server.RequestLog](http://download.eclipse.org/jetty/stable-9/apidocs/org/eclipse/jetty/server/RequestLog.html) 请求日志接口。可以使用尊重NCSA格式的实现org.eclipse.jetty.server.NCSARequestLog。还有其他三种配置RequestLog的方法：

* 在jetty xml配置文件中，如jettyXml参数中指定的那样 。
* 在上下文xml配置文件中，如contextXml参数中指定的那样 。
* 在webApp元素中。

*See*[*Configuring Request Logs*](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/configuring-jetty-request-logs.html)*for more information.*

**服务器**

自jetty-9.3.1以来可选。这将配置[org.eclipse.jetty.server.Server](http://git.eclipse.org/c/jetty/org.eclipse.jetty.project.git/plain/jetty-server/src/main/java/org/eclipse/jetty/server/Server.java) 插件的实例，但通常不需要配置它，因为插件会自动为您配置一个。特别是，如果使用jettyXml元素，那么通常不希望定义此元素，因为您可能正在使用jettyXml文件来配置具有特殊构造函数参数的服务器，例如自定义线程池。如果同时定义服务器元素并使用jettyXml元素，该元素指向具有类似“<Configure id =”Server“class =”org.eclipse.jetty.server.Server“>”的行的配置文件

**stopPort**

可选的。侦听停止命令的端口。与[停止](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/jetty-maven-plugin.html#jetty-stop-goal) 或[分叉](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/jetty-maven-plugin.html#jetty-run-forked-goal) 目标结合使用很有用。

**STOPKEY**

可选的。与stopPort一起用于停止码头。与stop或run-forked目标一起使用时很有用。

**系统属性**

可选。允许您配置系统属性以执行插件。有关更多信息，请参阅[设置系统属性](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/jetty-maven-plugin.html)。

**systemPropertiesFile**

可选的。包含要为插件执行设置的系统属性的文件。默认情况下，您在此处**进行的**设置 **不会**覆盖已在命令行，JVM或POM via中设置的任何系统属性systemProperties。读取[设置系统属性](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/jetty-maven-plugin.html)以了解如何强制覆盖。

**跳跃**

默认值为false。如果为true，则插件的执行退出。与 -Djetty.skip在命令行上设置SystemProperty相同。在集成测试期间配置Jetty以执行并且您想跳过测试时，这非常有用

**useProvidedScope**

默认值是false。如果为true，则将依赖项 <scope>provided</scope>放在*容器类路径中*。请注意，这不是webapp类路径，因为“提供”表示通常期望这些依赖项由容器提供。你应该很少需要使用它。相反，您应该将提供的依赖项复制为相反的显式依赖项plugin。

**excludedGoals**

可选的。jetty插件目标名称列表，它将导致插件打印信息性消息并退出。如果您想阻止用户执行您知道无法使用项目的目标，则非常有用。

**配置Https连接器**

要配置https连接器，您需要使用jetty xml配置文件。此示例使用直接从jetty分发etc /目录复制的文件，但您当然可以组成自己的xml文件。我们将使用以下文件：

**的jetty.xml**

设置[org.eclipse.jetty.server.Server](http://git.eclipse.org/c/jetty/org.eclipse.jetty.project.git/plain/jetty-server/src/main/java/org/eclipse/jetty/server/Server.java) 要使用的插件的实例的各种特征。重要的是，它设置了[org.eclipse.jetty.server.HttpConfiguration](http://git.eclipse.org/c/jetty/org.eclipse.jetty.project.git/plain/jetty-server/src/main/java/org/eclipse/jetty/server/HttpConfiguration.java) 我们可以在配置连接器的后续xml文件中引用的元素。这是相关部分：

|  |  |
| --- | --- |
| 1  2  3  4  五  6  7  8  9  10  11  12  13  14  15  16 | <New id="httpConfig" class="org.eclipse.jetty.server.HttpConfiguration">    <Set name="secureScheme">https</Set>    <Set name="securePort"><Property name="jetty.secure.port" default="8443" /></Set>    <Set name="outputBufferSize">32768</Set>    <Set name="requestHeaderSize">8192</Set>    <Set name="responseHeaderSize">8192</Set>    <Set name="sendServerVersion">true</Set>    <Set name="sendDateHeader">false</Set>    <Set name="headerCacheSize">512</Set>      <!-- Uncomment to enable handling of X-Forwarded- style headers    <Call name="addCustomizer">      <Arg><New class="org.eclipse.jetty.server.ForwardedRequestCustomizer"/></Arg>    </Call>    -->  </New> |

**码头，ssl.xml**

设置将由https连接器使用的ssl。这是jetty-ssl.xmljetty-distribution中的文件：

|  |  |
| --- | --- |
| 1  2  3  4  五  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  三十  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78 | <?xml version="1.0"?>  <!DOCTYPE Configure PUBLIC "-//Jetty//Configure//EN" "<http://www.eclipse.org/jetty/configure_9_0.dtd>">    <!-- ============================================================= -->  <!-- Base SSL configuration                                        -->  <!-- This configuration needs to be used together with 1 or more   -->  <!-- of jetty-https.xml or jetty-http2.xml                         -->  <!-- ============================================================= -->  <Configure id="Server" class="org.eclipse.jetty.server.Server">      <!-- =========================================================== -->    <!-- Add a SSL Connector with no protocol factories              -->    <!-- =========================================================== -->    <Call  name="addConnector">      <Arg>        <New id="sslConnector" class="org.eclipse.jetty.server.ServerConnector">          <Arg name="server"><Ref refid="Server" /></Arg>          <Arg name="acceptors" type="int"><Property name="ssl.acceptors" default="-1"/></Arg>          <Arg name="selectors" type="int"><Property name="ssl.selectors" default="-1"/></Arg>          <Arg name="factories">            <Array type="org.eclipse.jetty.server.ConnectionFactory">              <!-- uncomment to support proxy protocol          <Item>                <New class="org.eclipse.jetty.server.ProxyConnectionFactory"/>              </Item>-->            </Array>          </Arg>            <Set name="host"><Property name="jetty.host" /></Set>          <Set name="port"><Property name="ssl.port" default="443" /></Set>          <Set name="idleTimeout"><Property name="ssl.timeout" default="30000"/></Set>          <Set name="soLingerTime"><Property name="ssl.soLingerTime" default="-1"/></Set>          <Set name="acceptorPriorityDelta"><Property name="ssl.acceptorPriorityDelta" default="0"/></Set>          <Set name="selectorPriorityDelta"><Property name="ssl.selectorPriorityDelta" default="0"/></Set>          <Set name="acceptQueueSize"><Property name="ssl.acceptQueueSize" default="0"/></Set>        </New>      </Arg>    </Call>      <!-- ============================================================= -->    <!-- Create a TLS (SSL) Context Factory  for later reuse           -->    <!-- ============================================================= -->    <New id="sslContextFactory" class="org.eclipse.jetty.util.ssl.SslContextFactory">      <Set name="KeyStorePath"><Property name="jetty.base" default="." />/<Property name="jetty.keystore" default="etc/keystore"/></Set>      <Set name="KeyStorePassword"><Property name="jetty.keystore.password" default="OBF:1vny1zlo1x8e1vnw1vn61x8g1zlu1vn4"/></Set>      <Set name="KeyManagerPassword"><Property name="jetty.keymanager.password" default="OBF:1u2u1wml1z7s1z7a1wnl1u2g"/></Set>      <Set name="TrustStorePath"><Property name="jetty.base" default="." />/<Property name="jetty.truststore" default="etc/keystore"/></Set>      <Set name="TrustStorePassword"><Property name="jetty.truststore.password" default="OBF:1vny1zlo1x8e1vnw1vn61x8g1zlu1vn4"/></Set>      <Set name="EndpointIdentificationAlgorithm"></Set>      <Set name="NeedClientAuth"><Property name="jetty.ssl.needClientAuth" default="false"/></Set>      <Set name="WantClientAuth"><Property name="jetty.ssl.wantClientAuth" default="false"/></Set>      <Set name="ExcludeCipherSuites">       <Array type="String">        <Item>SSL\_RSA\_WITH\_DES\_CBC\_SHA</Item>        <Item>SSL\_DHE\_RSA\_WITH\_DES\_CBC\_SHA</Item>        <Item>SSL\_DHE\_DSS\_WITH\_DES\_CBC\_SHA</Item>        <Item>SSL\_RSA\_EXPORT\_WITH\_RC4\_40\_MD5</Item>        <Item>SSL\_RSA\_EXPORT\_WITH\_DES40\_CBC\_SHA</Item>        <Item>SSL\_DHE\_RSA\_EXPORT\_WITH\_DES40\_CBC\_SHA</Item>        <Item>SSL\_DHE\_DSS\_EXPORT\_WITH\_DES40\_CBC\_SHA</Item>       </Array>      </Set>    </New>      <!-- =========================================================== -->    <!-- Create a TLS specific HttpConfiguration based on the        -->    <!-- common HttpConfiguration defined in jetty.xml               -->    <!-- Add a SecureRequestCustomizer to extract certificate and    -->    <!-- session information                                         -->    <!-- =========================================================== -->    <New id="sslHttpConfig" class="org.eclipse.jetty.server.HttpConfiguration">      <Arg><Ref refid="httpConfig"/></Arg>      <Call name="addCustomizer">        <Arg><New class="org.eclipse.jetty.server.SecureRequestCustomizer"/></Arg>      </Call>    </New>    </Configure> |

**码头，https.xml**

使用HttpConfiguration jetty.xml和来自 jetty-ssl.xml以下的ssl配置设置https连接器 ：

|  |  |
| --- | --- |
| 1  2  3  4  五  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | <?xml version="1.0"?>  <!DOCTYPE Configure PUBLIC "-//Jetty//Configure//EN" "<http://www.eclipse.org/jetty/configure_9_0.dtd>">    <!-- ============================================================= -->  <!-- Configure a HTTPS connector.                                  -->  <!-- This configuration must be used in conjunction with jetty.xml -->  <!-- and jetty-ssl.xml.                                            -->  <!-- ============================================================= -->  <Configure id="sslConnector" class="org.eclipse.jetty.server.ServerConnector">      <Call name="addIfAbsentConnectionFactory">      <Arg>        <New class="org.eclipse.jetty.server.SslConnectionFactory">          <Arg name="next">http/1.1</Arg>          <Arg name="sslContextFactory"><Ref refid="sslContextFactory"/></Arg>        </New>      </Arg>    </Call>      <Call name="addConnectionFactory">      <Arg>        <New class="org.eclipse.jetty.server.HttpConnectionFactory">          <Arg name="config"><Ref refid="sslHttpConfig" /></Arg>        </New>      </Arg>    </Call>    </Configure> |

现在你需要让插件知道应用上面的文件：

|  |  |
| --- | --- |
| 1  2  3  4  五  6  7  8 | <plugin>    <groupId>org.eclipse.jetty</groupId>    <artifactId>jetty-maven-plugin</artifactId>    <version>9.2.21.v20170120</version>    <configuration>      <jettyXml>jetty.xml,jetty-ssl.xml,jetty-https.xml</jettyXml>    </configuration>  </plugin> |

**警告**

就像已安装的Jetty发行版一样，xml文件的排序非常重要。

您还可以使用jetty xml文件为要使用的插件配置http连接器。这里我们使用jetty-http.xmlJetty发行版中的相同 文件：

|  |  |
| --- | --- |
| 1  2  3  4  五  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  三十  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51 | <?xml version="1.0"?>  <!DOCTYPE Configure PUBLIC "-//Jetty//Configure//EN" "<http://www.eclipse.org/jetty/configure_9_0.dtd>">    <!-- ============================================================= -->  <!-- Configure the Jetty Server instance with an ID "Server"       -->  <!-- by adding a HTTP connector.                                   -->  <!-- This configuration must be used in conjunction with jetty.xml -->  <!-- ============================================================= -->  <Configure id="Server" class="org.eclipse.jetty.server.Server">      <!-- =========================================================== -->    <!-- Add a HTTP Connector.                                       -->    <!-- Configure an o.e.j.server.ServerConnector with a single     -->    <!-- HttpConnectionFactory instance using the common httpConfig  -->    <!-- instance defined in jetty.xml                               -->    <!--                                                             -->    <!-- Consult the javadoc of o.e.j.server.ServerConnector and     -->    <!-- o.e.j.server.HttpConnectionFactory for all configuration    -->    <!-- that may be set here.                                       -->    <!-- =========================================================== -->    <Call name="addConnector">      <Arg>        <New id="httpConnector" class="org.eclipse.jetty.server.ServerConnector">          <Arg name="server"><Ref refid="Server" /></Arg>          <Arg name="acceptors" type="int"><Property name="http.acceptors" default="-1"/></Arg>          <Arg name="selectors" type="int"><Property name="http.selectors" default="-1"/></Arg>          <Arg name="factories">            <Array type="org.eclipse.jetty.server.ConnectionFactory">              <!-- uncomment to support proxy protocol          <Item>                <New class="org.eclipse.jetty.server.ProxyConnectionFactory"/>              </Item>-->              <Item>                <New class="org.eclipse.jetty.server.HttpConnectionFactory">                  <Arg name="config"><Ref refid="httpConfig" /></Arg>                </New>              </Item>            </Array>          </Arg>          <Set name="host"><Property name="jetty.host" /></Set>          <Set name="port"><Property name="jetty.port" default="80" /></Set>          <Set name="idleTimeout"><Property name="http.timeout" default="30000"/></Set>          <Set name="soLingerTime"><Property name="http.soLingerTime" default="-1"/></Set>          <Set name="acceptorPriorityDelta"><Property name="http.acceptorPriorityDelta" default="0"/></Set>          <Set name="selectorPriorityDelta"><Property name="http.selectorPriorityDelta" default="0"/></Set>          <Set name="acceptQueueSize"><Property name="http.acceptQueueSize" default="0"/></Set>        </New>      </Arg>    </Call>    </Configure> |

现在我们将它添加到要应用的插件的配置列表中：

|  |  |
| --- | --- |
| 1  2  3  4  五  6  7  8 | <plugin>    <groupId>org.eclipse.jetty</groupId>    <artifactId>jetty-maven-plugin</artifactId>    <version>9.2.21.v20170120</version>    <configuration>      <jettyXml>jetty.xml,jetty-http.xml,jetty-ssl.xml,jetty-https.xml</jettyXml>    </configuration>  </plugin> |

或者，您可以使用pom内的[**httpConnector**](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/jetty-maven-plugin.html#maven-http-connector)配置元素，如上所述。

**配置WebApp**

这些配置参数适用于您的webapp。它们几乎是所有目标的共同点。

**Web应用程序**

表示类的扩展[org.eclipse.jetty.webapp.WebAppContext](http://download.eclipse.org/jetty/stable-9/apidocs/org/eclipse/jetty/webapp/WebAppContext.hml)。您可以使用此对象上的任何setter方法来配置您的webapp。以下是一些最有用的：

**的contextPath**

您的webapp的上下文路径。默认情况下，此设置为/。

**描述**

该路径web.xml为你的web应用文件。

**defaultsDescriptor**

在webdefault.xml将之前应用于webapp 的文件 的路径web.xml。如果您不提供，Jetty使用烘焙到的默认文件 jetty-webapp.jar。

**overrideDescriptor**

web.xml读取后Jetty应用的文件的路径 web.xml。您可以使用它来替换或添加配置。

**tempDirectory**

当您的webapp运行时，Jetty可用于扩展或复制jar和jsp编译的目录的路径。默认是 ${project.build.outputDirectory}/tmp。

**baseResource**

Jetty提供静态资源的路径。默认为 src/main/webapp。

**resourceBases**

baseResource如果您有多个目录要从中提供静态内容，请使用而不是。这是一组目录名称。

**baseAppFirst**

默认为“true”。控制是否在webapp的原始基础资源之前或之后添加任何重叠的战争。有关更多信息，请参阅有关[重叠战争](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/jetty-maven-plugin.html#using-overlaid-wars)的部分。

**contextXml**

在 webApp元素之后应用于webapp的上下文xml文件的路径。

**码头：运行**

在run目标上没有被内置到WAR web应用程序运行。相反，Jetty从其源代码部署webapp。它在Maven默认项目位置中查找webapp的组成部分，但您可以在插件配置中覆盖这些部分。例如，默认情况下它会查找：

* 资源 ${project.basedir}/src/main/webapp
* 课程 ${project.build.outputDirectory}
* web.xml 在 ${project.basedir}/src/main/webapp/WEB-INF/

该插件会自动确保在部署之前重建类并使其保持最新。如果更改类的源并且IDE在后台自动编译它，则插件会选择更改的类。

您无需将Web应用程序组装到WAR中，从而在开发周期中节省了时间。调用后，您可以将插件配置为连续运行，扫描项目中的更改并在必要时自动执行热重新部署。您所做的任何更改都会立即反映在Jetty的运行实例中，让您快速从编码跳转到测试，而不是经历以下循环：代码，编译，重组，重新部署，测试。

这是一个小例子，它每十秒开启扫描一次更改，并将webapp上下文路径设置为 /test：

|  |  |
| --- | --- |
| 1  2  3  4  五  6  7  8  9  10  11  12 | <plugin>    <groupId>org.eclipse.jetty</groupId>    <artifactId>jetty-maven-plugin</artifactId>    <version>9.2.21.v20170120</version>    <configuration>      <scanIntervalSeconds>10</scanIntervalSeconds>      <webApp>        <contextPath>/test</contextPath>      </webApp>    </configuration>  </plugin> |

**组态**

除了大多数目标共有的元素外， 目标还支持：**webApp**jetty:run

**classesDirectory**

您为webapp编译的类的位置。您应该很少需要设置此参数。相反，你应该 build outputDirectory在你的 pom.xml。

**testClassesDirectory**

您的webapp的已编译测试类的位置。默认情况下这是 ${project.build.testOutputDirectory}。

**useTestScope**

如果为true，则testClassesDirectory 范围“test” 的类和依赖项首先放在类路径上。默认情况下，这是错误的。

**webAppSourceDirectory**

默认情况下，此设置为 ${project.basedir}/src/main/webapp。如果静态源位于其他位置，请相应地设置此参数。

**jettyEnvXml**

可选的。一个位置jetty-env.xml 的文件，它允许你创建JNDI绑定并满足 env-entry，resource-env-ref以及 resource-ref连接在web.xml 该范围限定只对Web应用程序，而不是与您可能在同一时间被部署其他的web应用共享（例如，通过使用 jettyConfig文件）。

**scanTargets**

可选的。除了插件自动扫描的文件和目录之外，还要定期扫描的文件和目录列表。

**scanTargetPatterns**

可选的。如果你有你想要扫描的额外文件一个长长的清单，这是更方便的使用模式匹配表达式来指定它们，而不是用列举他们 scanTargetsList的scanTargetPatterns，每个组成的目录，以及包括和/或排除参数指定文件匹配模式。

这是一个例子：

|  |  |
| --- | --- |
| 1  2  3  4  五  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  三十  31  32  33  34  35  36  37  38 | <project>  ...    <plugins>  ...      <plugin>        <groupId>org.eclipse.jetty</groupId>        <artifactId>jetty-maven-plugin</artifactId>        <version>9.2.21.v20170120</version>        <configuration>          <webAppSourceDirectory>${project.basedir}/src/staticfiles</webAppSourceDirectory>          <webApp>            <contextPath>/</contextPath>            <descriptor>${project.basedir}/src/over/here/web.xml</descriptor>            <jettyEnvXml>${project.basedir}/src/over/here/jetty-env.xml</jettyEnvXml>          </webApp>          <classesDirectory>${project.basedir}/somewhere/else</classesDirectory>          <scanTargets>            <scanTarget>src/mydir</scanTarget>            <scanTarget>src/myfile.txt</scanTarget>          </scanTargets>          <scanTargetPatterns>            <scanTargetPattern>              <directory>src/other-resources</directory>              <includes>                <include>\*\*/\*.xml</include>                <include>\*\*/\*.properties</include>              </includes>              <excludes>                <exclude>\*\*/myspecial.xml</exclude>                <exclude>\*\*/myspecial.properties</exclude>              </excludes>            </scanTargetPattern>          </scanTargetPatterns>        </configuration>      </plugin>    </plugins>  </project> |

无论出于何种原因，如果您无法在未组装的Web应用程序上运行，那么目标run-war和 run-exploded工作在未组装的Web应用程序上。

**码头：运行战争**

此目标首先将您的webapp打包为WAR文件，然后将其部署到Jetty。如果设置非零 scanInterval，Jetty会监视您pom.xml和WAR文件; 如果有任何变化，它会重新部署战争。

**组态**

**战争**

构建的WAR文件的位置。默认为 ${project.build.directory}/${project.build.finalName}.war。如果这还不够，请将其设置为您的自定义位置。

以下是设置方法：

|  |  |
| --- | --- |
| 1  2  3  4  五  6  7  8  9  10  11  12  13  14  15 | <project>  ...    <plugins>  ...      <plugin>        <groupId>org.eclipse.jetty</groupId>        <artifactId>jetty-maven-plugin</artifactId>        <version>9.2.21.v20170120</version>        <configuration>          <war>${project.basedir}/target/mycustom.war</war>        </configuration>      </plugin>    </plugins>  </project> |

**码头：运行爆炸**

运行爆炸的目标首先将您的webapp组装成爆炸的WAR文件，然后将其部署到Jetty。如果设置非零 scanInterval，Jetty会监视您的pom.xml, WEB-INF/lib, WEB-INF/类以及 WEB-INF/web.xml更改并在必要时重新部署。

**组态**

**战争**

爆炸WAR的位置。默认为 ${project.build.directory}/${project.build.finalName}，但您可以通过设置此参数来覆盖默认值。

以下是设置方法：

|  |  |
| --- | --- |
| 1  2  3  4  五  6  7  8  9  10  11  12  13  14  15 | <project>  ...    <plugins>  ...      <plugin>        <groupId>org.eclipse.jetty</groupId>        <artifactId>maven-jetty-plugin</artifactId>        <version>9.2.21.v20170120</version>        <configuration>          <war>${project.basedir}/target/myfunkywebapp</war>        </configuration>      </plugin>    </plugins>  </project> |

**码头：部署战**

这基本上是相同的jetty:run-war，但是如果没有组装当前模块的WAR，你可以指定任何战争的位置来运行。与run-war此不同，此插件执行的阶段不受“包”阶段的约束 - 您可以将其绑定到任何阶段以使用它。

**组态**

**战争**

WAR文件的位置。默认为 ${project.build.directory}/${project.build.finalName}，但您可以通过设置此参数来覆盖默认值。

**守护进程**

如果为true，则此插件将启动jetty，但让构建继续。如果要在特定阶段将jetty作为执行绑定启动，然后在另一个阶段停止执行绑定，这将非常有用。或者，您可以将此参数设置为false，在这种情况下，jetty将阻止，您将需要使用cntrl-c来阻止它。

这是配置：

|  |  |
| --- | --- |
| 1  2  3  4  五  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  三十  31  32  33 | <project>  ...    <plugins>  ...    <plugin>      <groupId>org.eclipse.jetty</groupId>      <artifactId>jetty-maven-plugin</artifactId>      <version>9.2.21.v20170120</version>      <configuration>        <war>/opt/special/some-app.war</war>        <stopKey>alpha</stopKey>        <stopPort>9099</stopPort>      </configuration>      <executions>        <execution>          <id>start-jetty</id>          <phase>test-compile</phase>          <goals>            <goal>deploy-war</goal>          </goals>        </execution>        <execution>          <id>stop-jetty</id>          <phase>test</phase>          <goals>            <goal>stop</goal>          </goals>        </execution>       </executions>      </plugin>    </plugins>  </project> |

**码头：运行分叉**

此目标允许您在新JVM中启动Web应用程序，可选择将参数传递给新JVM。此目标支持与目标相同的配置参数，以及jetty:run一些额外的帮助配置分叉进程。

**组态**

除了jetty:run目标之外，可用的配置参数是：

**jvmArgs**

可选的。一个字符串，表示要传递给分叉JVM的任意参数。

**waitForChild**

默认是true。这会导致父进程等待分叉进程退出。在这种情况下，您可以使用 cntrl-c终止两个进程。将其设置为更有用false，在这种情况下父进程终止，同时保持子进程运行。您使用jetty:stop目标来停止子进程。

**maxStartupLines**

默认是50。这是父进程从子进程读取的最大行数，以接收子进程已启动的指示。如果子进程在stdout上产生过多的输出，则可能需要增加此数字。

某些容器配置参数**不**适用于此目标：

**scanIntervalSeconds**

不支持。分叉码头不会监控和重新部署webapp。

**重装**

不支持。分叉码头不会重新部署webapp。

**httpConnector**

不支持。要定义自定义连接器，请使用jetty xml文件。

**loginServices**

不支持。要定义LoginServices，请改用jetty xml或context xml文件。

**requestLog**

不支持。要定义RequestLog设置，请改用jetty xml或context xml文件。

**系统属性**

不支持。使用该jvmArgs参数将系统属性传递给分叉进程。

要将未组装的Web应用程序部署到在新JVM中运行的Jetty：

mvn jetty：run-forked

Jetty继续执行，直到你：

* cntrl-c在终端窗口中按下以停止插件，这也会停止分叉的JVM（仅当您开始使用时 waitForChild=true）
* 使用jetty：stop停止分叉的JVM，它也会停止插件。

**注意**

如果要为jetty连接器设置自定义端口，则需要在jetty xml文件中指定它，而不是设置连接器和端口标记。您可以jetty.xml使用jettyXml 参数指定使用该位置的位置。

**码头：启动**

此目标适用于您的执行绑定 pom.xml。它类似于jetty:run 目标，但它不会首先执行构建，直到“测试 - 编译”阶段，以确保已生成webapp的所有必需的类和文件。当你想通过你的执行绑定来控制Jetty的启动和停止时，这是最有用的 pom.xml。

例如，您可以将插件配置为在单元测试开始时启动Web应用程序，并在结束时停止。为此，您需要execution为Jetty插件设置几个方案。您使用pre-integration-test和 post-integration-testMaven构建阶段来触发Jetty的执行和终止：

|  |  |
| --- | --- |
| 1  2  3  4  五  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  三十 | <plugin>    <groupId>org.eclipse.jetty</groupId>    <artifactId>jetty-maven-plugin</artifactId>    <version>9.2.21.v20170120</version>    <configuration>      <scanIntervalSeconds>10</scanIntervalSeconds>      <stopKey>foo</stopKey>      <stopPort>9999</stopPort>    </configuration>    <executions>      <execution>        <id>start-jetty</id>        <phase>pre-integration-test</phase>        <goals>          <goal>start</goal>        </goals>        <configuration>          <scanIntervalSeconds>0</scanIntervalSeconds>        </configuration>      </execution>      <execution>        <id>stop-jetty</id>        <phase>post-integration-test</phase>         <goals>           <goal>stop</goal>         </goals>       </execution>    </executions>  </plugin> |

**码头：停止**

停止目标停止正在运行的码头实例。要使用它，您需要使用特殊的端口号和密钥配置插件。启动码头的其他目标也将使用相同的端口号和密钥。

**stopPort**

jetty的端口号，用于侦听以接收停止命令以使其关闭。

**STOPKEY**

发送到的字符串值stopPort以验证stop命令。

**stopWait**

插件等待确认jetty已停止的最长时间（以秒为单位）。如果为false或未指定，则插件不会等待确认，而是在发出stop命令后退出。

这是一个配置示例：

|  |  |
| --- | --- |
| 1  2  3  4  五  6  7  8  9  10  11 | <plugin>    <groupId>org.eclipse.jetty</groupId>    <artifactId>jetty-maven-plugin</artifactId>    <version>9.2.21.v20170120</version>    <configuration>      <stopPort>9966</stopPort>      <stopKey>foo</stopKey>      <stopWait>10</stopWait>    </configuration>  </plugin> |

然后，当Jetty运行时（在另一个窗口中），键入：

mvn码头：停下来

的stopPort必须是自由的，你正在运行的机器上。如果不是这种情况，则在“Started SelectedChannelConnector ...”消息之后会出现“Address in in use”错误消息。

**码头：有效的web-XML**

此目标根据Servlet规范的规则计算合成web.xml（“有效web.xml”），同时考虑应用程序中Web组件的所有可发现配置源：描述符（webdefault.xml，web.xml， web-fragment.xmls，web-override.xml）和发现的注释（@WebServlet，@ WebFilter，@ WebListener）。生成来自这些组合源的有效web.xml并显示为maven日志输出。作为分析的一部分生成的有关您的webapp的其他有用信息也作为上下文参数存储在effective-web.xml中。effective-web.xml可以与[Quickstart](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/quickstart-webapp.html)一起使用 快速启动webapp的功能（请注意，Quickstart不适合mvn jetty目标）。

以下配置参数允许您保存文件：

**deleteOnExit**

默认情况下这是true。如果设置为 false，则生成有效的web.xml到effective-web.xmlbuild target目录中调用的文件中 。

**effectiveWebXml**

您希望生成有效Web xml的文件的完整路径名。

请注意，不会考虑servlet，过滤器和侦听器的编程声明。

**使用重叠战争**

如果您的webapp依赖于其他war文件，则[jetty：run](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/jetty-maven-plugin.html#jetty-run-goal)和[jetty：run-forked](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/jetty-maven-plugin.html#jetty-run-forked-goal)目标可以合并所有这些目标中的资源。它可以根据[maven-war-plugin](http://maven.apache.org/plugins/maven-war-plugin/)的设置来实现，或者如果你的项目没有使用[maven-war-plugin](http://maven.apache.org/plugins/maven-war-plugin/) 来处理覆盖，它可以回退到一个简单的算法来确定资源的排序。

**使用maven-war-plugin**

maven-war-plugin具有丰富的资源合并功能。jetty：run和jetty：run-forked目标能够解释大部分目标并在执行未组装的webapp时应用它们。通过查看具体示例可能最好地看到这一点。

假设您的webapp依赖于以下战争：

|  |  |
| --- | --- |
| 1  2  3  4  五  6  7  8  9  10  11 | <dependency>    <groupId>com.acme</groupId>    <artifactId>X</artifactId>    <type>war</type>  </dependency>  <dependency>    <groupId>com.acme</groupId>    <artifactId>Y</artifactId>    <type>war</type>  </dependency> |

*Containing:*

|  |  |
| --- | --- |
| 1  2  3  4  五  6  7  8  9  10  11  12  13 | WebAppX:     /foo.jsp   /bar.jsp   /WEB-INF/web.xml    WebAppY:     /bar.jsp   /baz.jsp   /WEB-INF/web.xml   /WEB-INF/special.xml |

它们是为[maven-war-plugin配置的](http://maven.apache.org/plugins/maven-war-plugin/overlays.html)：

|  |  |
| --- | --- |
| 1  2  3  4  五  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | <plugin>    <groupId>org.apache.maven.plugins</groupId>    <artifactId>maven-war-plugin</artifactId>    <version>9.2.21.v20170120</version>    <configuration>      <overlays>        <overlay>          <groupId>com.acme</groupId>          <artifactId>X</artifactId>          <excludes>            <exclude>bar.jsp</exclude>          </excludes>        </overlay>        <overlay>          <groupId>com.acme</groupId>          <artifactId>Y</artifactId>          <excludes>            <exclude>baz.jsp</exclude>          </excludes>        </overlay>        <overlay>        </overlay>      </overlays>    </configuration>  </plugin> |

然后执行jetty：run将产生以下资源排序：com.acme.X.war : com.acme.Y.war: ${project.basedir}/src/main/webapp。请注意，由于maven-war-plugin中的空<overlay />元素，当前项目的资源位于排序的最后。您可以使用它，也可以将<baseAppFirst>false</baseAppFirst>参数指定 给jetty-maven-plugin。

此外，由于exclusions上面指定的， bar.jsp只有从排除的com.acme.Y.war.类似地满足对资源baz.jsp的请求，对它的请求将导致404错误。

**没有maven-war-plugin**

该算法相当简单，基于从属战争的声明顺序，并且不支持排除。配置参数<baseAppFirst>（有关详细信息，请参阅[配置Web应用程序](https://www.eclipse.org/jetty/documentation/9.2.22.v20170531/jetty-maven-plugin.html#configuring-your-webapp)一节）可用于控制Webapp的资源在运行时是首先放在资源路径上还是最后放在资源路径上。

例如，假设我们的webapp依赖于这两场战争：

|  |  |
| --- | --- |
| 1  2  3  4  五  6  7  8  9  10 | <dependency>    <groupId>com.acme</groupId>    <artifactId>X</artifactId>    <type>war</type>  </dependency>  <dependency>    <groupId>com.acme</groupId>    <artifactId>Y</artifactId>    <type>war</type>  </dependency> |

假设webapps包含：

|  |  |
| --- | --- |
| 1  2  3  4  五  6  7  8  9  10  11  12 | WebAppX:     /foo.jsp   /bar.jsp   /WEB-INF/web.xml    WebAppY:     /bar.jsp   /baz.jsp   /WEB-INF/web.xml   /WEB-INF/special.xml |

然后我们的webapp提供了以下额外资源：

|  |  |
| --- | --- |
| 1  2  3  4  五 | /foo.jsp (X)  /bar.jsp (X)  /baz.jsp (Y)  /WEB-INF/web.xml (X)  /WEB-INF/special.xml (Y) |

**配置安全设置**

您可以在插件中配置LoginServices。以下是为webapp设置HashLoginService的示例：

|  |  |
| --- | --- |
| 1  2  3  4  五  6  7  8  9  10  11  12  13  14  15  16  17 | <plugin>    <groupId>org.eclipse.jetty</groupId>    <artifactId>jetty-maven-plugin</artifactId>    <version>9.2.21.v20170120</version>    <configuration>      <scanIntervalSeconds>10</scanIntervalSeconds>      <webApp>        <contextPath>/test</contextPath>      </webApp>      <loginServices>        <loginService implementation="org.eclipse.jetty.security.HashLoginService">          <name>Test Realm</name>          <config>${project.basedir}/src/etc/realm.properties</config>        </loginService>      </loginServices>    </configuration>  </plugin> |

**使用多个Webapp根目录**

如果您希望在执行Web应用程序时包含外部资源，但未将其组装到WAR中，则不能使用上述重叠的战争方法，但您可以告诉Jetty这些外部资源所在的目录位于。在运行时，当Jetty收到资源请求时，它会搜索所有位置以检索资源。这很像战争覆辙，但没有战争。这是一个配置示例：

|  |  |
| --- | --- |
| 1  2  3  4  五  6  7  8 | <configuration>    <webApp>      <contextPath>/${build.finalName}</contextPath>      <baseResource implementation="org.eclipse.jetty.util.resource.ResourceCollection">        <resourcesAsCSV>src/main/webapp,/home/johndoe/path/to/my/other/source,/yet/another/folder</resourcesAsCSV>      </baseResource>    </webApp>  </configuration> |

**运行多个Webapp**

您可以使用jetty.xml文件来配置要部署的额外（预编译）Web应用程序，也可以使用<contextHandlers>配置元素执行此操作。如果要在同一个Jetty实例中部署webapp A和webapps B和C：

将配置放在webapp A中pom.xml：

|  |  |
| --- | --- |
| 1  2  3  4  五  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | <plugin>    <groupId>org.eclipse.jetty</groupId>    <artifactId>jetty-maven-plugin</artifactId>    <version>9.2.21.v20170120</version>    <configuration>      <scanIntervalSeconds>10</scanIntervalSeconds>      <webApp>        <contextPath>/test</contextPath>      </webApp>      <contextHandlers>        <contextHandler implementation="org.eclipse.jetty.maven.plugin.JettyWebAppContext">          <war>${project.basedir}../../B.war</war>          <contextPath>/B</contextPath>        </contextHandler>        <contextHandler implementation="org.eclipse.jetty.maven.plugin.JettyWebAppContext">          <war>${project.basedir}../../C.war</war>          <contextPath>/B</contextPath>        </contextHandler>      </contextHandlers>    </configuration>  </plugin> |

**重要**

如果要部署的ContextHandler是webapp，则 **必须**使用 org.eclipse.jetty.maven.plugin.JettyWebAppContext 实例而不是标准org.eclipse.jetty.webapp.WebAppContext实例。只有前者才允许webapp在maven环境中正常运行。

或者，将jetty.xml文件添加到webapp A. jetty.xml从jetty发行版中复制文件，然后为其他2个webapps添加WebAppContexts：

|  |  |
| --- | --- |
| 1  2  3  4  五  6  7  8  9  10  11  12  13  14  15  16  17  18 | <Ref refid="Contexts">    <Call name="addHandler">      <Arg>        <New class="org.eclipse.jetty.maven.plugin.JettyWebAppContext">          <Set name="contextPath">/B</Set>          <Set name="war">../../B.war</Set>        </New>      </Arg>    </Call>    <Call>      <Arg>        <New class="org.eclipse.jetty.maven.plugin.JettyWebAppContext">          <Set name="contextPath">/C</Set>          <Set name="war">../../C.war</Set>        </New>      </Arg>    </Call>  </Ref> |

然后将此jetty.xml 文件的位置配置到webapp A的jetty插件中：

|  |  |
| --- | --- |
| 1  2  3  4  五  6  7  8  9  10  11  12 | <plugin>    <groupId>org.eclipse.jetty</groupId>    <artifactId>jetty-maven-plugin</artifactId>    <version>9.2.21.v20170120</version>    <configuration>      <scanIntervalSeconds>10</scanIntervalSeconds>      <webApp>        <contextPath>/test</contextPath>      </webApp>      <jettyXml>src/main/etc/jetty.xml</jettyXml>    </configuration>  </plugin> |

对于这些解决方案中的任何一个，必须已经构建了其他Web应用程序，并且不会自动监视它们的更改。您可以参考预构建的webapps的压缩WAR文件或扩展的等效文件。

**设置系统属性**

您可以指定Jetty设置的属性名称/值对作为执行插件的系统属性。此功能对于整理命令行并节省大量输入非常有用。

但是，**有时无法使用此功能来设置系统属性** - 有时使用System属性的软件组件已经在maven运行时初始化（在这种情况下，您需要在命令行上提供System属性） ，或者到码头运行的时候。在后一种情况下，您可以使用[maven属性插件](http://mojo.codehaus.org/properties-maven-plugin/)来定义系统属性。这是一个将logback日志记录系统配置为jetty记录器的示例：

|  |  |
| --- | --- |
| 1  2  3  4  五  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | <plugin>    <groupId>org.codehaus.mojo</groupId>    <artifactId>properties-maven-plugin</artifactId>    <version>1.0-alpha-2</version>    <executions>      <execution>        <goals>          <goal>set-system-properties</goal>        </goals>        <configuration>          <properties>            <property>              <name>logback.configurationFile</name>              <value>${project.baseUri}/resources/logback.xml</value>            </property>          </properties>        </configuration>      </execution>    </executions>  </plugin> |

请注意，如果已设置System属性（例如，从命令行或JVM本身），则默认情况下这些已配置的属性不会覆盖它们（请参阅下面的<force>参数的使用）。

**在POM中指定系统属性**

以下是如何在POM中指定系统属性的示例：

|  |  |
| --- | --- |
| 1  2  3  4  五  6  7  8  9  10  11  12  13  14  15 | <plugin>    <groupId>org.eclipse.jetty</groupId>    <artifactId>jetty-maven-plugin</artifactId>    <configuration>      <systemProperties>        <systemProperty>          <name>fooprop</name>          <value>222</value>        </systemProperty>      </systemProperties>      <webApp>        <contextPath>/test</contextPath>      </webApp>    </configuration>  </plugin> |

要更改默认行为以使这些系统属性覆盖命令行上的那些，请使用< force> 参数：

|  |  |
| --- | --- |
| 1  2  3  4  五  6  7  8  9  10  11  12  13  14  15  16 | <plugin>    <groupId>org.eclipse.jetty</groupId>    <artifactId>jetty-maven-plugin</artifactId>    <configuration>      <systemProperties>        <force>true</force>        <systemProperty>         <name>fooprop</name>         <value>222</value>       </systemProperty>      </systemProperties>      <webApp>        <contextPath>/test</contextPath>      </webApp>    </configuration>  </plugin> |

**在文件中指定系统属性**

您还可以在文件中指定系统属性。以这种方式指定的系统属性不要覆盖在命令行，JVM或直接在POM via中设置的系统属性 systemProperties。

假设我们有一个名为的文件mysys.props，其中包含以下内容：

|  |  |
| --- | --- |
| 1 | fooprop=222 |

这可以在插件上配置，如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  五  6  7  8  9  10 | <plugin>    <groupId>org.eclipse.jetty</groupId>    <artifactId>jetty-maven-plugin</artifactId>    <configuration>      <systemPropertiesFile>${project.basedir}/mysys.props</systemPropertiesFile>      <webApp>        <contextPath>/test</contextPath>      </webApp>    </configuration>  </plugin> |

您可以通过 jetty.systemPropertiesFile在命令行上设置System属性（！）来指定文件。