CONTENTS INCLUDE:
- Jetty Architecture
- Configuring Jetty
- Basic Usage
- Advanced Usage
- Using with Web Frameworks
- Running Standalone
- Integration with Maven and more...

Jetty

A Lightweight, Open-Source Web Server and Servlet Container

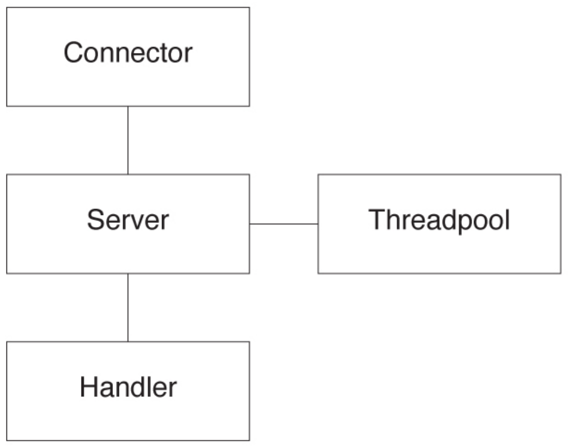
By Jos Dirksen

ABOUT JETTY

Jetty, an open-source web server hosted by the Eclipse foundation, is a full-fledged HTTP server and Servlet container that can be easily configured to serve static and dynamic content. You can very easily embed Jetty into your own applications using Java or XML-based configuration or run Jetty directly from Maven. Additionally, you can use Jetty in many different high-demand areas such as Yahoo Hadoop Cluster, Google AppEngine, and Yahoo! Zimbra.
This RefCard refers primarily to Jetty 8; however, most of the configuration discussed will work on Jetty 7.

JETTY ARCHITECTURE

It's important to understand the architecture behind Jetty. The main architecture of Jetty consists out of the following four components.



Server

The server is the container class for all the other components of the architecture. A server contains a number of connectors that listen on a specific port and accepts connections (e.g., HTTP, HTTPS, or AJP13) from clients. A server also contains a set of handlers that process the client requests from the connectors and produce responses that are returned to the client. Threads retrieved from the configured threadpool do this processing.

Connector

Jetty provides a number of connectors that accept requests from clients to be processed by the handlers. Jetty comes with the following set of default connectors.

Connector	Description and Usage
SocketConnector	Uses blocking IO and normal JRE sockets. One thread is allocated per connection. Only use when NIO connector isn't available.
BlockingChannelConnector	Uses NIO buffers with a blocking thread model. One thread is allocated per connection. Use when there are few very active connections.
SelectChannelConnector	Uses NIO buffers with a non-blocking threading model. Threads are only allocated to connections with requests. Use when there are many connections that have idle periods.
SslSocketConnector	SSL version of the SocketConnector
SslSelectChannelConnector	SSL version of the SelectChannelConnector
AJPConnector	Use for connections from Apache modules: mod_proxy_ajp and mod_jk.
HTTPSPDYServerConnector	Supports the SPDY protocol and performs SPDY to HTTP conversion. If SPDY is not negotiated, falls back to HTTPS.

Handler

The handler is the component that deals with requests received by a connector. There are three types of handlers in Jetty.

- Coordinating handlers route requests to other handlers.
- Filtering handlers modify a request and pass it on to other handlers.
- Generating handlers create content.
- Jetty provides the following handlers.

Name	Description
ConnectHandler	Implements a tunneling proxy that supports HTTP CONNECT.
DebugHandler	Writes details of the request and response to an outputstream.
GzipHandler	Will gzip the response
IPAccessHandler	Provides access control using white/black lists on IP addresses and URLs.
RequestLogHandler	Allows logging of requests to a request log.
ResourceHandler	Serves static content and handles If-Modified-Since headers. Doesn't handle 404 errors.
RewriteHandler	Allows you to rewrite the URL, cookies, headers, and status codes based on a set of rules.

ConstraintSecurity-Handler	Enforces security constraints based on Servlet specification 2.4.
StatisticsHandler	Collects statistics on requests and responses.
WebSocketHandler	Provides support for the websockets protocol.
HandlerCollection	Contains a collection of handlers. Handlers are called in order.
ContextHandlerCol-lection	Contains a collection of handlers. Handlers are called based on the provided context and virtual host.
HandlerList	Like HandlerCollection, but calls each handler in turn until an exception is thrown, the response is committed, or a positive status is set.
ServletHandler	Maps requests to servlets that implement the HttpServlet API.
ServletContextHan-dler	See ServletHandler. Allows you to specify the context the servlet is mapped to.
DefaultHandler	Deals with unhandeld requests from the server.
WebAppContext	Can be used to map requests to a web application (WAR).

Threadpool

The threadpool provides threads to the handlers, the work done by the connnectors and the handlers. Jetty comes with the following threadpools.

Name	Description
ExecutorThreadPool	Uses the standard Java ThreadPoolExecutor to execute jobs.
QueuedThreadPool	Uses a blocking queue to execute jobs.

CONFIGURING JETTY

When you configure Jetty, you create and configure a set of connectors and handlers. You can do this using XML or plain Java.

XML configuration

Jetty provides an XML-based configuration language based on the Java Reflection API. When starting Jetty with an XML configuration, Jetty will parse this XML and create and configure the specified objects.

The following is a basic configuration using the XML syntax with a server, connector, handler, and threadpool.

```
<Configure id="Server" class="org.eclipse.jetty.server.Server">
 <Set name="threadPool">
  <New class="org.eclipse.jetty.util.thread.QueuedThreadPool">
   <Set name="minThreads">10</Set>
   <Set name="maxThreads">10</Set>
  </New>
 </Set>
 <Call name="addConnector">
  <Arg>
   <New class="org.eclipse.jetty.server.nio.SelectChannelConnector">
    <Set name="port">8080</Set>
   </New>
  </Arg>
 </Call
 <Set name="handler">
  <New class="org.eclipse.jetty.server.handler.HandlerList">
   <Set name="handlers">
    <Array type="org.eclipse.jetty.server.Handler">
     <Item>
      <New class="org.eclipse.jetty.server.handler.ResourceHandler">
       <Set name="directoryListed">true</Set>
       <Set name="resourceBase">./files</Set>
      </New>
     </Item>
     <Item>
      <New class="org.eclipse.jetty.server.handler.DefaultHandler"/>
     </Item>
    </Array>
   </Set>
  </New>
 </Set>
```

This code creates a ResourceHandler that shows the files from the "./files" directory. The DefaultHandler takes care of the unhandled request. You can start this server using the following Java code:

```
Public class JettyExample {
  public static void main(String[] args) throws Exception {
    Resource fileCfg = Resource.newSystemResource("example.xml");
    XMLConfiguration config = new
            XMLConfiguration(fileCfg.getInputStream());
    Server.start();
    Server.join();
  }
}
```

The following elements can be used to configure Jetty.

XML Element	Description
Configure	The root element that specifies the class to be configured. Attribute "id": reference to created object. Attribute "class": FQN to instantiate. Can contain: <Set>, <Get>, <Put>, <Call>, <New>, <Ref>, <Array>, <Map>, <Property>
Set	Maps to a call to a setter method. Attribute "name": Setter to call Attribute "type": Type of argument Attribute "class": if present, make a static call to the specified class. Can contain: <Get>, <Call>, <New>, <Ref>, <Array>, <Map>, <Sys-temProperty>, <Property>
Get	Maps a call to a getter method. Attribute "name": getter to call Attribute "class": if present, make a static call to the specified class. Attribute "id": reference to returned object Can contain: <Set>, <Get>, <Put>, <Call>, <New>, <Ref>, <Array>, <Map>, <Property>
Put	Calls the put method on the current object that should implement Map. Attribute "name": used as put key Attribute "type": force type of value Can contain: <Get>, <Call>, <New>, <Ref>, <Array>, <Map>, <Sys-temProperty>, <Property>
Call	Makes an arbitrary call to the current object. Attribute "name": method to call Attribute "class": if present, make a static call to the specified class. Attribute "id": reference to returned object Can contain: <Arg>, <Set>, <Get>, <Put>, <Call>, <New>, <Ref>, <Array>, <Map>, <Property>
Arg	Specifies an argument for <Call> and <New> Attribute "type": force type of value Can contain: <Get>, <Call>, <New>, <Ref>, <Array>, <Map>, <Sys-temProperty>, <Property>
New	Instantiates a new object. Attribute "id": reference to created object. Attribute "class": FQN to instantiate. Can contain: <Arg>, <Set>, <Get>, <Put>, <Call>, <New>, <Ref>, <Array>, <Map>, <Property>
Ref	References a previously created Object. Attribute "id": the object to reference to. Can contain: <Set>, <Get>, <Put>, <Call>, <New>, <Ref>, <Array>, <Map>, <Property>
Array	Allows creation of new array. Attribute "type": the type of array. Attribute "id": reference to the created array Can contain: <Item>
Item	Defines an entry for an Array or Map element. Attribute "type": force the type of the item Attribute "id": reference to the created item. Can contain: <Get>, <Call>, <New>, <Ref>, <Array>, <Map>, <Sys-temProperty>, <Property>
Map	Allows creation new new HashMap. Attribute "id": reference to the created map. Can contain: <Entry>
Entry	Contains a key-value <Item> pair Can contain: <Item>
SystemProperty	Gets the value of a JVM system property. Attribute "name": the name of the property Attribute "default": default value as fallback Attribute "id": reference to the created object Can contain: nothing
Property	Allows arbitrary properties to be retrieved by name. Attribute "name": the name of the property Attribute "default": default value as fallback Attribute "id": reference to the created object Can contain: <Set>, <Get>, <Put>, <Call>, <New>, <Ref>, <Array>, <Map>, <Property>

Java configuration

Using Java for the configuration of your Jetty server is very simple and is done in the same manner as we've shown for the XML configuration. Instead of using the XML elements to instantiate the objects, you can now do this directly from Java. You can rewrite the previous XML configuration to the following piece of Java.

```
public static void main(String[] args) throws Exception {
  Server server = new Server();

  QueuedThreadPool pool = new QueuedThreadPool();
  pool.setMinThreads(10);
  pool.setMaxThreads(10);
  server.setThreadPool(pool);

  SelectChannelConnector connector = new SelectChannelConnector();
  connector.setPort(8080);
  server.addConnector(connector);

  HandlerList handlers = new HandlerList();
  ResourceHandler resourceHandler = new ResourceHandler();
  resourceHandler.setDirectoriesListed(true);
  resourceHandler.setResourceBase("./files");
  DefaultHandler defaultHandler = new DefaultHandler();
  handlers.setHandlers(new Handler[]
          {resourceHandler, defaultHandler});
  server.setHandler(handlers);

  server.start();
  server.join();
}
```

Hot Tip Since Jetty components are simple POJOs, you can also use your dependency injection framework of choice to set up the Jetty server. An example with Spring is shown here: http://wiki.eclipse.org/Jetty/Howto/Spring

BASIC USAGE

This section describes how to use Jetty for a number of basic use cases.

Serving static content

When running Jetty as an embedded web server in your own application, it can be useful to be able to serve static content (e.g., documentation). The easiest way to server static content is by using a ResourceHandler:

```
<New class="org.eclipse.jetty.server.handler.ResourceHandler">
  <Set name="directoryListed">true</Set>
  <Set name="resourceBase">./files</Set>
</New>
```

The ResourceHandler can be configured with the following properties:

Property	Description
aliases	Boolean; if true symlinks are followed.
directoryListed	Boolean; if true show directory listings.
welcomeFiles	String[]; a welcome file is shown for a directory if it matches an item from the supplied String[].
resourceBase	String: The path from where to serve content.

If you want the ResourceHandler to listen on a specific context, you can wrap this Handler in a ContextHandler:

```
<New class="org.eclipse.jetty.server.handler.ContextHandler">
  <Set name="contextPath">/documentation</Set>
  <Set name="handler">
  <New class="...ResourceHandler">...</New>
 </Set>
</New>
```

SSL configuration

To configure Jetty to use HTTPS, you can use one of the SSL-enabled connectors: SslSelectChannelConnector or SSLSocketConnector. The following listing defines two-way ssl (client authentication):

```
Call name="addConnector">
 <Arg>
  <New class="org.eclipse...SslSelectChannelConnector">
   <Arg>
    <New class="org.eclipse.jetty.http.ssl.SslContextFactory">
     <Set name="keyStore">etc/keystore</Set>
     <Set name="keyStorePassword">OBF:1vny1zlo1x8e1vnw1</Set>
     <Set name="keyManagerPassword">OBF:1u2u1wml1z7s1z/Set>
     <Set name="trustStore">/etc/keystore</Set>
     <Set name="trustStorePassword">OBF:w11x8g1zlu1vn4</Set>
     <Set name="needClientAuth">true</Set>
    </New>
   </Arg>
   <Set name="port">8443</Set>
  </New>
 </Arg>
</Call>
```

The following properties are used:

Property	Description
keystore	Keystore for the server keypair
keystorepassword	Password for the keystore
keymanagerpassword	Password for the private key
truststore	Keystore for trusted certificates
truststorepassword	Password for truststore
needClientAuth	True, if clients must use a certificate

There are more advanced properties available. For these, see the Javadocs for the SslContextFactory.

Hot Tip Jetty provides a utility that you can use to secure passwords in configuration files. By using the org.eclipse.jetty.http.security.Passwd class, you can generate obfuscated, checksummed, and encrypted passwords.

Servlets and the ServletContextHandler

Jetty allows you to easily configure servlets without having to use a web.xml. To do this, you can use a ServletContextHandler, which allows for easy construction of a context with a ServletHandler. The following properties can be set on a ServletContextHandler.

Property	Description
contextPath	The base context path used for this ServletContextHandler.
allowNullPathInfo	If "false", then /context is redirected to /context/.
compactPath	If "true", replace multiple '/'s with a single '/'.
errorHandler	An "ErrorHandler" determines how error pages are handled.
securityHandler	The "SecurityHandler" to use for this ServletContextHandler.
sessionHandler	The "SessionHandler" to use for this ServletContextHandler
welcomeFiles	List of welcomeFiles to show for this context.

A servlet on this context can be added using the addServlet operation (which can also be done through XML).

```
addServlet(String className,String pathSpec)
addServlet(Class<? extends Servlet> servlet,String pathSpec)
addServlet(ServletHolder servlet,String pathSpec)
```

Using existing WAR files and layout

Jetty allows you to directly use existing WAR files (and exploded WAR files) through the WebAppContext. This is especially useful during development.

Directly from a WAR:

```
Server server = new Server(8080);
 WebAppContext webapp = new WebAppContext();
 webapp.setContextPath("/");
 webapp.setWar(jetty_home+"/webapps/test.war");
 server.setHandler(webapp);
```

From an exploded WAR (e.g. during development):

```
Server server = new Server(8080);
WebAppContext context = new WebAppContext();
context.setDescriptor(webapp+"/WEB-INF/web.xml");
context.setResourceBase("../test-jetty-webapp/src/main/webapp");
context.setContextPath("/");
context.setParentLoaderPriority(true);
server.setHandler(context);
```

You can also configure a WebAppContext in jetty.xml using the XML-based configuration or in a context.xml file.

Security Realms

Security Realms allow you to protect your web applications. Jetty provides a number of standard LoginServices you can use to secure your web application.

Name	Description
HashLoginService	A simple implementation that stores users and roles in memory and loads them from a properties file.
JDBCLoginService	Retrieves users and roles from a database configured with JDBC.
DataSourceLogin-Service	Retrieves users and roles from a Javax.sql.DataSource.
JAASLoginService	Delegates the login to JAAS. Jetty provides the following JAAS modules: • DBCLoginModule • PropertyFileLoginModule • DataSourceLoginModule • LdapLoginModule

To use a LoginServices, you configure it in the jetty.xml file.

```
<Call name="addBean">
  <Arg>
   <New class="org.eclipse.jetty.security.HashLoginService">
    <Set name="name">RefCardRealm</Set>
    <Set name="config">etc/realm.properties</Set>
    <Set name="refreshInterval">0</Set>
   </New>
  </Arg>
 </Call>
```

The real-name defined in the jetty.xml can now be referenced from the web.xml.

```
<login-config>
 <auth-method>FORM</auth-method>
 <realm-name>RefCardRealm</realm-name>
 <form-login-config>
  <form-login-page>/login/login</form-login-page>
  <form-error-page>/login/error</form-error-page>
 </form-login-config>
</login-config>
```

JNDI usage

Jetty has support for Java:comp/env lookups in web applications.

Setup JNDI

The JNDI feature isn't enabled by default. To enable JNDI, you have to set the following configurationClasses on your WebAppContext or WebAppDeployer:

```
Configure id="wac" class="org.eclipse.jetty.webapp.WebAppContext">
 <Array id="plusConfig" 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.webapp.JettyWebXmlConfiguration</Item>
  <!-- next one not needed for Jetty 8 -->
  <Item>org.eclipse.jetty.webapp.TagLibConfiguration</Item>
 </Array>
 <Set name="war">location/of/webapp</Set>
 <Set name="configurationClasses"><Ref id="plusConfig"/></Set>
</Configure>
```

You can now use <env-entry/>, <resource-ref/>, and <resource-env-ref/> entries in your web.xml to point to resources stored in the JNDI context.

Binding objects to JNDI

Jetty allows you to bind POJOs, a java.naming.Reference instance, an implementation of Java.naming.Referenceable, and a link between a name in the web.xml and one of these other objects. These objects can be configured from Java or in the jetty.xml configuration files.

```
<New class=type of naming entry>
  <Arg>scope</Arg>
  <Arg>name to bind as</Arg>
  <Arg>the object to bind</Arg>
</New>
```

The scope defines where the object is visible. If left empty, the scope is set to the Configure context this entry is defined in. This scope can also be set to point to a specific server of webapplication.

```
<Arg><Ref id='wac'/></Arg>
```

The following environment types are supported:

Type	How to bind from Jetty

env-entry	<New class="org.eclipse.jetty.plus.jndi.EnvEntry">
	  <Arg></Arg>
	  <Arg>mySpecialValue</Arg>
	  <Arg type="java.lang.Integer">4000</Arg>
	  <!—set to true to override web.xml -->
	  <Arg type="boolean">true</Arg>
	</New>

resource-ref resource-env-ref	<New id="myds" class="org.eclipse.jetty.plus.jndi.Resource"> <Arg><Ref id="wac"/></Arg> <Arg>jdbc/myds</Arg> <Arg> <New class="org...EmbeddedDataSource"> <Set name="DatabaseName">test</Set> <Set name="createDatabase">create</Set> </New> </Arg> </New>

Link	With a Link you can link a resource from the web.xml to a resource in the containter context.
	<New id="map1" class="org.eclipse.jetty.plus.jndi.Link">
	  <Arg><Ref id='wac'/></Arg>
	  <Arg>jdbc/datasourceInWeb</Arg>
	  <Arg>jdbc/nameInContainer</Arg>
	</New>

Jetty-env.xml

You can store environment settings in a jetty.xml file that is stored in your WEB-INF directory.

```
<Configure class="org.mortbay.jetty.webapp.WebAppContext">
  <!-- Add entries only valid for this webapp -->
</Configure>
```

ADVANCED USAGE

This section describes a couple of advanced configurations for Jetty.

Integration with CDI

When creating applications, it's very useful to use context and dependency injection (CDI). With JEE6, this is a standard feature of an application server. You can also use CDI with Jetty. The following configuration can be used to configure Weld (the CDI reference implementation) for a specific web application on Jetty.

```
<Configure id="webAppCtx"
      class="org.eclipse.jetty.webapp.WebAppContext">
 <New id="BManager" class="org.eclipse.jetty.plus.jndi.Resource">
  <Arg><Ref id="webAppCtx"/></Arg>
  <Arg>BeanManager</Arg>
  <Arg>
   <New class="javax.naming.Reference">
    <Arg>javax.enterprise.inject.spi.BeanManager</Arg>
    <Arg>org.jboss.weld.resources.ManagerObjectFactory</Arg>
    <Arg/>
   </New>
  </Arg>
 </New>
</Configure>
```

The beanmanager is available on java:comp/env/BeanManager.

Running Jetty behind a reverse proxy

Jetty can be configured to run behind a reverse proxy, such as Apache with mod_proxy or mod_proxy_ajp. The preferred way is to use mod_proxy with a normal HTTP connector. However, it's also possible to create a connector for the AJP protocol used by the mod_proxy_ajp modules.

```
<Call name="addConnector">
 <Arg>
 <New class="org.eclipse.jetty.ajp.Ajp13SocketConnector">
  <Set name="port">8009</Set>
 </New>
 </Arg>
</Call>
```

For more information on running Jetty as a reverse proxy, see: http://wiki.eclipse.org/Jetty/Howto/Configure_mod_proxy

Websockets

Jetty has support for websockets. You can create your own websockets servlet by extending WebSocketServlet.

```
public class ExampleWSServlet extends WebSocketServlet {

 protected void doGet(HttpServletRequest request,
     HttpServletResponse response)
     throws ServletException ,IOException {
  getServletContext().getNamedDispatcher("default")
      .forward(request,response);
 }

 protected WebSocket doWebSocketConnect(
      HttpServletRequest request, String protocol) {
  return new ExampleWebSocket();
 }

 class ExampleWebSocket implements WebSocket {
  private Outbound outbound;

  public void onConnect(Outbound outbound) {
   this.outbound=outbound;
  }

  public void onMessage(byte frame
  ,byte[] data,int offset, int length) {
   // handle binary data
  }

  public void onMessage(byte frame, String data) {
   // handle String data
  }

  public void onDisconnect() {}
 }
}
```

Session clustering

Jetty uses two components for session management. First, a session ID manager ensures that the session IDs are unique across all web applications. Second, a session manager handles the session lifecycle. If you want to cluster your sessions, Jetty provides a JDBC-based SessionIDManager and SessionManager.

Configure the JDBCSessionIdManager:

```
JDBCSessionIdManager idMgr = new JDBCSessionIdManager(server); idMgr.setWorkerName("fred"); idMgr.
setDriverInfo("com.mysql.jdbc.Driver","jdbc:mysql://127.0.0.1:3306/sessions?user=janb"); idMgr.setScavengeInterval(60);
server.setSessionIdManager(idMgr);
```

This JDBCSessionIdManager needs to be configured on each cluster node with a unique workerName. Once you've defined this ID manager, you can can configure the JDBCSessionManager.

```
WebAppContext wac = new WebAppContext();
... //configure your webapp context
JDBCSessionManager jdbcMgr = new JDBCSessionManager(); jdbcMgr.setIdManager(server.getSessionIdManager());
wac.setSessionHandler(jdbcMgr);
```

Hot Tip Besides the JDBC-based managers, you can also use MongoDB for session clustering. For this clustering solution, you configure the MongoDBSessionIDManager and the MongoSessionManager".

SPDY support

SPDY is an experimental protocol whose goal it is to reduce the latency of web pages. Jetty can support this protocol through the HTTPSPDYServerConnector class.

```
<Configure id="Server" class="org.eclipse.jetty.server.Server">
 <New id="sslContextFactory" class="...util.ssl.SslContextFactory">
  <Set name="keyStorePath">your_keystore.jks</Set>
  <Set name="keyStorePassword">storepwd</Set>
  <Set name="protocol">TLSv1</Set>
 </New>
 <Call name="addConnector">
  <Arg>
   <New class="..http.HTTPSPDYServerConnector">
    <Arg><Ref id="sslContextFactory"/></Arg>
    <Set name="Port">8443</Set>
   </New>
  </Arg>
 </Call>
</Configure>
```

USING WITH WEB FRAMEWORKS

This section shows how to run an application based on a number of popular Java web frameworks using an embedded Jetty. This is especially useful during development for a quick compile-build-test cycle or when you want to run your application in a debugger with hot-code replace. If your favorite web application framework isn't listed here, these examples should point you in the right direction to get Jetty working with your framework.

JSF 2.0

JSF2 scans the WEB-INF/classes directories for beans. Before you can use an embedded Jetty to directly launch a JSF application, you have to make sure you configure your project to output its class files to the WEB-INF/classes directory in your project. After that, you can use the standard WebAppContext to run your JSF 2.0 web application directly from Jetty.

```
WebAppContext handler = new WebAppContext();
handler.setResourceBase("src/main/webapp");
handler.setDescriptor("src/main/webapp/WEB-INF/web.xml");
handler.setContextPath("/");
handler.setParentLoaderPriority(true);
```

Spring Web

Spring Web can be run directly from an exploded WAR file using a standard WebAppContext (see GWT). You can also configure the Spring DispatcherServlet directly.

```
ServletContextHanderl context = new ServletContextHandler
        (server,"/", Context.SESSIONS);
DispatcherServlet dispatcherServlet = new DispatcherServlet();
dispatcherServlet.setContextClass(
       AnnotationConfigWebApplicationContext.class);
ServletHolder holder = new ServletHolder(dispatcherServlet);
holder.setInitOrder(1);
context.addServlet(holder, "/example/*");
context.setInitParameter("contextConfigLocation"
        ,"classpath*:resources/spring/*.xml");
```

This context can be added as a handler to a server instance.

Grails

Grails already provides Jetty functionality out of the box. To run grails using an embedded Jetty container, you can use the grails run-app command.

Wicket

To run Wicket from Jetty, you can configure the WicketServlet with the applicationClassName you want to start.

```
ServletContextHandler context = new ServletContextHandler
        (server, "/", Context.SESSIONS);ServletHolder ServletHolder holder = new ServletHolder(new WicketServlet());
holder.setInitParameter(
        "applicationClassName","dzone.refcard.Application");
// set init order to initialize when handler starts
holder.setInitOrder(1);
context.addServlet(servletHolder, "/*");
```

This context can be added as a handler to a server instance.

GWT, Vaadin and Tapestry

You can directly run a GWT or a Vaadin application from an exploded WAR file using a standard WebAppContext.

```
WebAppContext handler = new WebAppContext();
handler.setResourceBase("./apps/TheGWTApplication");
handler.setDescriptor("./apps/TheGwtApplication/WEB-INF/web.xml");
handler.setContextPath("/");
handler.setParentLoaderPriority(true);
```

This context can be added as a handler to a server instance.

RUNNING STANDALONE

If you want to configure the standard Jetty distribution and not run Jetty embedded from your own application, you can configure Jetty using the following XML files.

File	Description
jetty.xml	Main configuration file. Configures a Server class. Normally located in $JETTY_HOME/etc/jetty.xml
jetty-web.xml	Configure a web application context. Located in the WEB-INF directory of a specific web application.
jetty-env.xml	Allows you to configure JNDI resources for a web application. Located in the WEB-INF directory of a web application.
webdefault.xml	Set default values on a web application context that will be applied to all web applications. This is loaded before the web.xml is processed. Located in ${jetty.home}/etc/webdefault
override-web.xml	Jetty applies the configuration in this file after the web.xml from a web application is parsed. This file is located in the WEB-INF directory of a web application

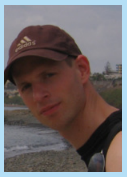
More information on running Jetty standalone can be found at the Jetty Wiki: http://wiki.eclipse.org/Jetty/Feature/Start.jar

INTEGRATION WITH MAVEN

Jetty is also often used in a maven build file to easily run and test a webapp project. When running from Maven, you can configure Jetty using the configuration elements provided by the plugin.

XML Element	Description
connectors	List of connectors objects. If not specified, a SelectChannelConnector will be configured on port 8080.
jettyXML	Location of a jetty.xml configuration file. Use this for objects that can't be configured from the plugin.
scanIntervalSeconds	Interval to check for changes to the webapp. If a change is detected, the webapp is redeployed.
systemProperties	Sets system properties that are used during the execution of the plugin.
systemPropertiesFile	Loads system properties from a file. These properties are available during the execution of the plugin.
loginServices	A list of LoginService implementations that are available to the webapp.
requestLog	Configures an implementation of a RequestLog.

ABOUT THE AUTHOR



Jos Dirksen works as Architect for JPoint. In the last couple of years Jos has worked on large projects in the public and private sector, ranging from very technology-focussed integration projects to SOA/BPM projects using WS-* and REST based architectures. Jos has given many presentations on conferences such as Javaone, NL-JUG, Devoxx etc., and has written two books for Manning: Open Source ESBs in Action and SOA Governance in Action. He also has his own blog where he writes about interesting technologies and shares his ideas about REST, API Design, Scala, Play and more.

RECOMMENDED RESOURCES

Jetty provides a Web server and javax.servlet container, plus support for Web Sockets, OSGi, JMX, JNDI, JASPI, AJP and many other integrations. These components are open source and available for commercial use and distribution. Jetty is used in a wide variety of projects and products. Jetty can be embedded in devices, tools, frameworks, application servers, and clusters. See the Jetty Powered page for more uses of Jetty.

http://www.eclipse.org/jetty/

DZone, Inc.
150 Preston Executive Dr.
Suite 200
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com

Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-936502-47-9
ISBN-10: 1-936502-47-X
50795
9 781936 502479
$7.95

Version 1.0