Jetty Tutorial for Beginners

<https://examples.javacodegeeks.com/enterprise-java/jetty/jetty-tutorial-beginners/>

# Q:> [What is the difference between Tomcat, JBoss and Glassfish?](https://stackoverflow.com/questions/3821640/what-is-the-difference-between-tomcat-jboss-and-glassfish)

A:> Tomcat is merely an HTTP server and Java servlet container. JBoss and GlassFish are full-blown Java EE application servers, including an EJB container and all the other features of that stack. On the other hand, Tomcat has a lighter memory footprint (~60-70 MB), while those Java EE servers weigh in at hundreds of megs. Tomcat is very popular for simple web applications, or applications using frameworks such as Spring that do not require a full Java EE server. Administration of a Tomcat server is arguably easier, as there are fewer moving parts.

However, for applications that do require a full Java EE stack (or at least more pieces that could easily be bolted-on to Tomcat)... JBoss and GlassFish are two of the most popular open source offerings (the third one is Apache Geronimo, upon which the free version of IBM WebSphere is built). JBoss has a larger and deeper user community, and a more mature codebase. However, JBoss lags significantly behind GlassFish in implementing the current Java EE specs. Also, for those who prefer a GUI-based admin system... GlassFish's admin console is extremely slick, whereas most administration in JBoss is done with a command-line and text editor. GlassFish comes straight from Sun/Oracle, with all the advantages that can offer. JBoss is NOT under the control of Sun/Oracle, with all the advantages THAT can offer.

# Q:> What are the major differences between Tomcat and Jetty application servers?

A:> When it comes to choosing a Java web application server, Java developers are often faced with numerous choices such as [Tomcat](http://tomcat.apache.org/), [Jetty](https://www.eclipse.org/jetty/), [JBoss](http://www.jboss.org/), [GlassFish](https://javaee.github.io/glassfish/), [WildFly](http://wildfly.org/), [TomEE](http://tomee.apache.org/), [WebLogic](http://www.oracle.com/technetwork/middleware/weblogic/overview/index.html), [WebSphere](https://www.ibm.com/cloud/websphere-application-platform), etc. In this brief, we’re going to discuss Tomcat vs Jetty and hopefully enable you get a better understanding of both web application servers and their respective advantages and disadvantages.

Furthermore, it’s without doubt that Tomcat and Jetty are both popular web application servers among Java web developers. As a matter of fact, they are both Web servers and Servlet/JavaServer Pages containers. And as a side note may not support some (if not all) Enterprise Java Beans (EJB) or other EE features in which case you have to look at JBoss, WildFly, TomEE, etc. By the same token, Tomcat vs Jetty as open source Servlet containers both implement HTTP server, HTTP client and javax.servlet container.

## [SPECIAL OFFER]: 15 Days FREE Trial Java Web Hosting

## [BENEFITS]:

* Host any Java Web Framework listed here and much more!
* Private Tomcat Server with Dedicated JVM Heap & PermGen
* Custom Java Development Kit (JDK) version
* Super Fast Servers with 24/7/365 Technical Support

[Click here to access this [SPECIAL OFFER]](https://www.dailyrazor.com/private-tomcat-hosting/)

However, it’s worth noting that Tomcat is a supported and promoted by the Apache Project whereas Jetty on the other hand is an Eclipse Project. Hence, this is good as both application servers have a strong backing and support among the Java developer community. Consequently, both application servers implement recent Java Server Pages (JSP) and Servlet standards, deploy .WAR files with ease, and impressively lightweight and fast when serving a Java web application.

With this in mind, here are some of the hightlights on Tomcat vs Jetty

## Tomcat:



* Easily the most popular open source project under Apache thus receive attention, support and updates
* Very well documented with a strong developer community
* Tested, Proven and Stable over many years with different versions to select from
* Commercially successful with many enterprises and government organizations using it
* Easy integration with other Java web application frameworks such as Spring Framework
* Looking for a servlet container to embed in your Java web application then Tomcat is a definite contender
* Java Server Pages (JSP) parsing and processing with speed
* Did we say flexible? Yes, Tomcat is very flexible and extendable – scaling to Enterprise support via TomEE, JBoss, WildFly is possible

## [SPECIAL OFFER]: 15 Days FREE Trial Java Web Hosting

## [BENEFITS]:

* Host any Java Web Framework listed here and much more!
* Private Tomcat Server with Dedicated JVM Heap & PermGen
* Custom Java Development Kit (JDK) version
* Super Fast Servers with 24/7/365 Technical Support

[Click here to access this [SPECIAL OFFER]](https://www.dailyrazor.com/private-tomcat-hosting/)

## Jetty:



* Uses less memory and is more lightweight thus offering speed and scalability
* Can be embedded with ease in your Java web application, small devices line phones and setup boxes as well as serve as asynchronous server
* It’s open source with good community backing and support
* Has small footprint and can be quickly launched and restarted with ease
* Widely used but still less know when compared to Tomcat
* Pluggable and extensive resulting in high degree of customizability
* Built into several frameworks such as GWT, JRuby, Grails, Scala/Lift, etc.
* It’s small and efficient with low maintaince and total cost of ownership

In this article, we will give brief information about Jetty and provide examples of   Java application deployment on Jetty. Our examples will consist of both standalone and Embedded modes of Jetty.

Jetty is a Servlet container and  Web Server which is known to be  portable, lightweight, robust, flexible, extensible and easy to integrate.

Jetty can be deployed as a standalone server and also can be embedded in an existing application. In addition to these, a  Maven Jetty  plugin is available in order to run applications in your development environment.

SPDY, WebSocket, OSGi, JMX, JNDI, JAAS are some of the technologies that Jetty integrates nicely.

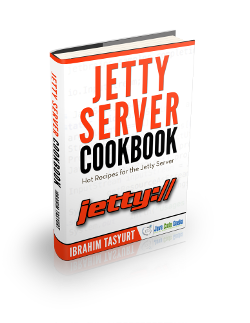
Today, Jetty is widely used in many platforms both for development and production. Small to large enterprise applications. SaaS (such as Zimbra), Cloud Applications( such as Google AppEngine), Applications Servers(such as Apache Geronimo) and tools (such as SoapUI) are powered by Jetty.

Jetty is open source, hosted by Eclipse Foundation.  Current version (as of June 2015) is 9.2.x.  You can more detailed information on [Jetty Home Page](http://www.eclipse.org/jetty/).

Want to master Jetty?

Subscribe to our newsletter and download the Jetty Server Cookbook right now!

In order to get you prepared for your Jetty development needs, we have compiled numerous recipes to help you kick-start your projects. Besides reading them online you may download the eBook in PDF format!



[**Download NOW!**](https://examples.javacodegeeks.com/enterprise-java/jetty/jetty-tutorial-beginners/)

## **1.Jetty as a Standalone server**

In the first part, we will configure Jetty as a Standalone Server.

### **1.1 Downloading  and Installing Jetty**

You can visit the [downloads](http://download.eclipse.org/jetty/) page and download the latest version (v9.2.11 currently) as an archive file in zip or tar.gz format. Size is about 13 MBs.

There is no installation procedure for Jetty. Just drop it to a folder as you wish and uncompress the downloaded archive file.

### **1.2 Prerequisites**

The only prerequisite for Jetty 9 is having installed Java 7 in your environment. You can downgrade to Jetty 8 if you have Java 6.  A complete Jetty-Java compatibility information can be viewed [here](https://wiki.eclipse.org/Jetty/Starting/Jetty_Version_Comparison_Table).

### **1.3 Running Jetty**

Running Jetty on the default configuration is as simple as following two steps:

1. Navigate to the directory where you unpacked the downloaded archive. I will call it ***JETTY\_HOME***from now on.
2. Run the following command:

java -jar start.jar

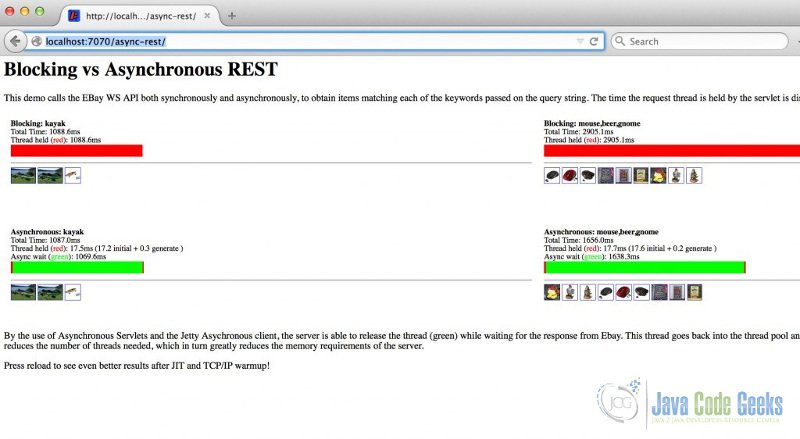
When Jetty starts running successfully; it produces the a line in the log similar to the following:

|  |  |
| --- | --- |
| 1 | 2015-06-04 14:27:27.555:INFO:oejs.Server:main: Started @11245ms |

By default, Jetty runs on port 8080, but we will see how to configure it in the next sections of this tutorial.

You can also check via the browser typing [http://localhost:8080](http://localhost:8080/) as the URL. You will see a 404 error, since no application is deployed in the root context.

The response is as below:



Server response when Jetty runs successfully

### **1.4 Changing the server port**

As mentioned above, default port jetty is 8080. If you need to change it, you can apply following steps:

1. Navigate to the ***JETTY\_HOME***.
2. Open the start.ini file with a text editor.
3. Navigate to the line where the parameter jetty.port is configured.
4. Change the parameter to the desired port number.
5. Start Jetty again.

In the following segment,  we set the Jetty port to 7070 instead of 8080

## HTTP port to listen on

jetty.port=7070

After we restart  our server will run on port 7070.

### **1.5 Deploying Web Applications on Jetty**

The procedure to deploy web applications on Jetty is  as follows:

1. Navigate to your *JETTY\_HOME* folder.
2. There is a directory named as ***webapps***under***JETTY\_HOME.***Navigate there***.***
3. Drop your WAR file in that folder.

The application is initialized immediately, you do not need to restart Jetty since the ***webapps*** directory is continuously monitored by the server.

There are a sample web applications under ***JETTY\_HOME/demo-base/webapps/****.*You can pick one of them*(for*example**async-rest.war**) and copy to the webapps directory. As you copy the WAR file, the application will be initialized.

When you type <http://localhost:7070/async-rest> , you can see the application initialized.

The application async-rest deployed on Jetty

### **1.6 Changing Webapps Directory**

***JETTY\_HOME/webapps***is the default directory to deploy your applications. But there are cases that you need to change the deployment directory. In order to do that, you should proceed as follows:

1. Open the start.ini file under***JETTY\_HOME***.
2. Remove the comment before the parameter *jetty.deploy.monitoredDirName.*
3. Change this parameter as you wish. Remember that the path should be relative to ***JETTY\_HOME*** directory.

jetty.deploy.monitoredDirName=../jcgwebapps

Now we can put our WARS in the *jcgwebapps* directory, which is at the same level as our ***JETTY\_HOME.***

## **2. Embedding Jetty in Your Application**

Until now, we have skimmed through Jetty as a standalone server. However Jetty provides another great feature. Motto of Jetty is : “Don’t deploy your application in Jetty, deploy Jetty in your application”. It means that, you can embed jetty in your existing (most probably non-web) applications easily. On this purpose a very convenient API is provided to the developers. In the following sections, we will see how we can accomplish this.

### **2.1 Environment**

In this example, following programming environment is used:

* Java 8 (Java 7 will also do fine.)
* Apache Maven 3.x.y
* Eclipse 4.4 (Luna)

### **2.2 Creating the Maven Project**

1. Go to File -> New ->Other -> Maven Project
2. Tick Create a simple project and press “Next”.
3. Enter groupId as : com.javacodegeeks.snippets.enterprise
4. Enter artifactId as : embedded-jetty-example
5. Press “Finish”.

Now our maven project is created.

Creating simple Maven project in Eclipse

### **2.3 Adding dependencies for Embedded Jetty**

Following Maven dependencies have to be added in the project:

* jetty-server : Core Jetty Utilities
* jetty-servlet: Jetty Servlet Utilities

You have to  add these dependencies to your pom.xml. After the dependencies are added,  your pom.xml seems as follows:

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | <project xmlns="<http://maven.apache.org/POM/4.0.0>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"      xsi:schemaLocation="<http://maven.apache.org/POM/4.0.0> <http://maven.apache.org/xsd/maven-4.0.0.xsd>">      <modelVersion>4.0.0</modelVersion>      <groupId>com.javacodegeeks.snippets.enterprise</groupId>      <artifactId>embedded-jetty-example</artifactId>      <version>0.0.1-SNAPSHOT</version>        <dependencies>            <!--Jetty  dependencies start here-->          <dependency>              <groupId>org.eclipse.jetty</groupId>              <artifactId>jetty-server</artifactId>              <version>9.2.11.v20150529</version>          </dependency>            <dependency>              <groupId>org.eclipse.jetty</groupId>              <artifactId>jetty-servlet</artifactId>              <version>9.2.11.v20150529</version>          </dependency>          <!--Jetty  dependencies end here-->        </dependencies>  </project> |

Now our project configuration is complete and we are ready to go.

### **2.4 Creating Embedded Jetty Server Programmatically**

Now we are going to create an Embedded Jetty Server programmatically. In order to keep things simple, we will create the Server in the main() method of our application.

In order to this, you can proceed as follows:

1. Create package com.javacodegeeks.snippets.enterprise.embeddedjetty.
2. Create a class named EmbeddedJettyMain.
3. Add a main method to this class.

The code that creates and starts and Embedded Jetty is as follows:

EmbeddedJettyMain.java

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19 | package com.javacodegeeks.snippets.enterprise.embeddedjetty;    import org.eclipse.jetty.server.Server;  import org.eclipse.jetty.servlet.ServletContextHandler;    import com.javacodegeeks.snippets.enterprise.embeddedjetty.servlet.ExampleServlet;    public class EmbeddedJettyMain {        public static void main(String[] args) throws Exception {            Server server = new Server(7070);          ServletContextHandler handler = new ServletContextHandler(server, "/example");          handler.addServlet(ExampleServlet.class, "/");          server.start();        }    } |

1. In the first line(Line 12), we create a Server on port **7070**.
2. In the next line(Line 13), we create a ServletContextHandler with the context path***/example***
3. In Line 14, we bind the servlet class ExampleServlet (which is described below) to this servlet context handler created in the previous line.
4. On the last line, we start the server.

|  |  |
| --- | --- |
| 1 | <em>ServletContextHandler</em> is a powerful facility enabling creation and configuration of Servlets and Servlet Filters programatically. |

ExampleServlet is a simple HttpServlet, does nothing but returning a constant output “EmbeddedJetty” as the response.

ExampleServlet.java

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21 | package com.javacodegeeks.snippets.enterprise.embeddedjetty.servlet;    import java.io.IOException;    import javax.servlet.ServletException;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;    import org.eclipse.jetty.http.HttpStatus;    public class ExampleServlet extends HttpServlet {        @Override      protected void doGet(HttpServletRequest req, HttpServletResponse resp)              throws ServletException, IOException {            resp.setStatus(HttpStatus.OK\_200);          resp.getWriter().println("EmbeddedJetty");      }  } |

### **2.5 Running Embedded Jetty**

Run the EmbeddedJettyMain class through the Eclipse Run, Embedded Jetty starts to run on the defined port(7070).

You can access the application through your browser on the following URL:  
<http://localhost:7070/example>

Here you can see the response below:

ExampleServlet response

## **3. Conclusion**

In this article, we have provided brief information on Jetty and discussed the steps to create standalone and Embedded Jetty servers.