

# Classloading

## What is Jetty's classloading architecture?

Class loading in a web container is slightly more complex than a normal java application.

The normal configuration is for each web context (web application or war file) is given it's own classloader, which has the system classloader as it's parent. Such a classloader hierarchy is normal in Java, however the servlet specification complicates the hierarchy by requiring that:

- Classes contained within `WEB-INF/lib` or `WEB-INF/classes` have priority over classes on the parent class loader. This is the opposite of the normal behaviour of a java 2 class loader.
- System classes such as `java.lang.String` may not be replaced by classes in `WEB-INF/lib` or `WEB-INF/classes`. Unfortunately the specification does not clearly state what classes are "System" classes and it is unclear if all javax classes should be treated as System classes.
- Server implementation classes should be hidden from the web application and should not be available in any class loader. Unfortunately the specification does not state what is a Server class and it is unclear if common libraries like the xerces parser should be treated as Implementation classes.

## How to configure classloading

Jetty provides configuration options to control all three of these options. The method `org.mortbay.jetty.webapp.WebAppContext.setParentLoaderPriority(boolean)` allows the normal java 2 behaviour to be used and all classes will be loaded from the system classpath if possible. This is very useful if the libraries that a web application uses are having problems loading classes that are both in a web application and on the system classpath.

The methods `org.mortbay.jetty.webapp.WebAppContext.setSystemClasses(String[])` and `org.mortbay.jetty.webapp.WebAppContext.setServerClasses(String[])` may be called to allow fine control over what classes can be seen or overridden by a web application.

- **SystemClasses** cannot be *overridden* by webapp context classloaders. The defaults are; `{"java.", "javax.servlet.", "javax.xml.", "org.mortbay.", "org.xml.", "org.w3c.", "org.apache.commons.logging.", "org.apache.log4j."}`
- **ServerClasses** (on the container classpath) cannot be *seen* by webapp context classloaders but can be *overridden* by the webapp. The defaults are: `{"-org.mortbay.jetty.plus.jaas.", "org.mortbay.jetty.", "org.slf4j."}`;

Absolute classname can be passed, names ending with `.` are treated as package names and names starting with `-` are treated as negative matches and must be listed *before* any enclosing packages.

## Adding extra classpaths to Jetty

At startup, the jetty runtime will automatically load all jars from the top level `$jetty.home/lib`, along with certain subdirectories such as `$jetty.home/lib/management/`, `$jetty.home/lib/naming/` etc, which are named explicitly in the [start.config](#) file contained in the `start.jar`. In addition, it will recursively load all jars from `$jetty.home/lib/ext`. So, to add extra jars to jetty, you can simply create a file hierarchy as deep as you wish within `$jetty.home/lib/ext` to contain these jars. Of course, you can always change this default behaviour by [creating your own start.config](#) file and using that instead. Otherwise, you can use one of the methods below.

## Using `jetty.class.path` System property

If you want to add a couple of class directories or jars to jetty, but you can't put them in `$jetty.home/lib/ext/` for some reason, or you don't want to create a custom `start.config` file, you can simply use the System property `-Djetty.class.path` on the runline instead. Here's how it would look:

```
java
-Djetty.class.path=" ../my/classes: ../my/jars
/special.jar: ../my/jars/other.jar" -jar
start.jar
```

### Using the `extraClasspath()` method on `WebAppContext`

If you need to add some jars or classes that for some reason are not in `$jetty.home/lib` nor inside your webapp's `WEB-INF/lib` or `WEB-INF/classes`, you can add them directly to your webapp in a `$JETTY_HOME/contexts/mycontext.xml` file:

```
<Configure
class="org.mortbay.jetty.webapp.WebAppContext">

    ...
    <Set
name="extraClasspath"> ../my/classes: ../my/jars/special.jar: ../my/jars/other.jar</Set>
    ...
```

### Using a custom `WebAppClassLoader`

Finally, if none of the other alternatives already described meet your needs, you can always provide a custom classloader for your webapp. It is recommended, but not required, that your custom loader subclasses [org.mortbay.jetty.webapp.WebAppClassLoader](#). You configure the classloader for the webapp like so:

```
MyCleverClassLoader myCleverClassLoader =  
new MyCleverClassLoader();  
...  
  
WebApplicationContext webapp = new WebApplicationContext();  
...  
webapp.setClassLoader(myCleverClassLoader);
```