Department of Computing Imperial College London

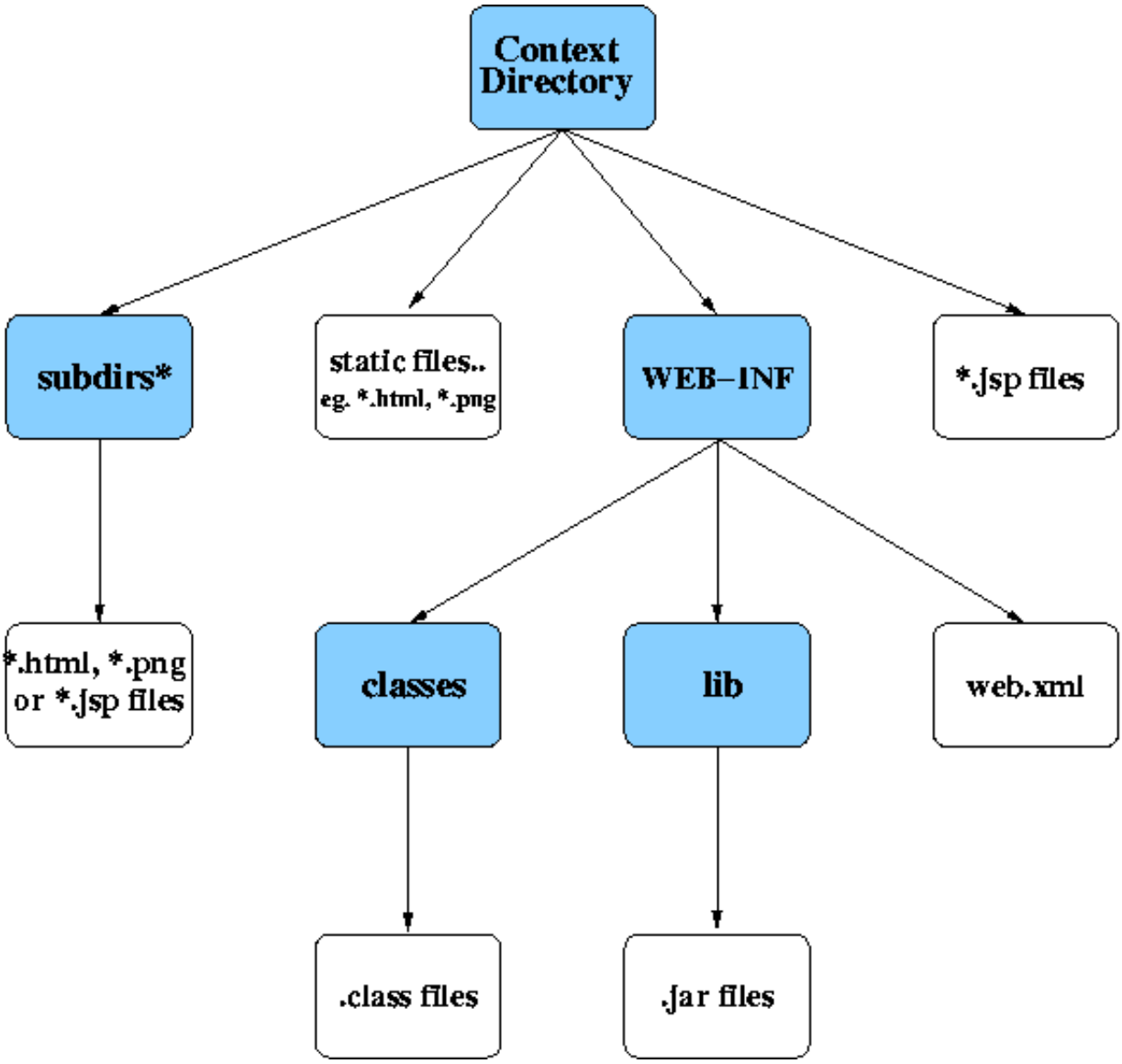
Java Servlets and JSP - Servlet Tutorial

Short Servlet Tutorial

Introduction

Ok, so what - in slightly more detail - is a Java Servlet? A Java Servlet is a Java class that subclasses from class **HttpServlet** and usually overrides the **doGet** (or **doPost**) method. These methods will be (later) invoked automatically when an appropriate web request is made, and each method produces a stream of dynamic html (or other) output which will be returned to the web browser.

To recap what we said in the main JSP and Servlets document, every context directory has a partially fixed layout as shown below:



In particular, inside each servlet context directory there should be a special directory called `WEB-INF`, often containing two subdirectories called `lib` and `classes` and an optional mapping file called `web.xml`. Any Java classes you need for your web application (whether servlet classes, support classes or java beans) can either be placed in `WEB-INF/classes` as individual precompiled .class files, or can be packaged up into .jar archive files and then be placed into `WEB-INF/lib`.

Before a servlet's **doGet/doPost** method can be called, several things need to have been done:

- The servlet must have been compiled in the usual way (using the javac compiler), in an environment where the **HttpServlet** class and all its support classes are available on the class path. All these classes are found in the Tomcat `servlet.jar` file, which we have installed in `/usr/share/java/servlet.jar` on all DoC machines. Adding this to your classpath will enable you to compile servlets.

- For a servlet to work, it needs a Tomcat context directory to live in. Let's assume that we've already set up a personal tomcat context directory called **example-context** for all our

examples to go inside. That is, as in both the Personal Tomcat and JSP Tutorial pages, assume that we've already done the following:

- `mktomcat6 ~/example-context`
- `cd ~/example-context`

Thus, you are already sitting in `~/example-context` with nothing else on the directory stack (type `dirs`).

- The servlet class (and any associated support classes) must have been installed into the `WEB-INF/classes` subdirectory of the context directory, or jarred up into a .jar file and placed into the `WEB-INF/lib` subdirectory of the context.

- Finally, the context needs to have been configured to map a particular URL onto the particular servlet class. This mapping is contained in a special file (`WEB-INF/web.xml`) in the context directory, which this tutorial will tell you more about.

A Simple Example Servlet - Simple.java

Let's start with a very simple servlet class.

- Starting in the `~/example-context` directory, push temporarily into the `src` subdirectory: `pushd src`.

- Create `Simple.java` containing the following:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Simple extends HttpServlet {
   int accesses = 0;

   public void doGet(HttpServletRequest request, HttpServletResponse
response)
      throws IOException, ServletException
   {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      accesses++;
      out.print("Number of times this servlet has been accessed:" +
accesses);
   }
}
```

- This is almost the simplest servlet possible. It has a very simple `doGet` method, which simply sets up the output for displaying an HTML page and counts the number of times it has been accessed. It does this with a global variable, which persists as long as the servlet is loaded. This will be destroyed when the server is restarted.

- Now create a `Makefile` as follows (if you've just been following the JSP tutorial, you'll need to merge the following into the existing Java Bean Makefile, that's beyond the scope of this tutorial):

```
CONTEXT =       ../webapps/ROOT
CLSDIR  =       $(CONTEXT)/WEB-INF/classes
CLASSES =       Simple.class

all:    $(CLASSES)

clean:
        /bin/rm -f $(CLASSES) */*.class

install:        $(CLASSES)
        install -m600 $(CLASSES) $(CLSDIR)

Simple.class:   Simple.java
        javac Simple.java
```

- Now compile and install everything via `make install`, fixing any Java or Makefile errors as you go. Then `popd` to get back into the context top-level dir.
- Now, before we start Tomcat, we have to decide how to map our Simple Servlet into the URL space - there are two choices:
    1. You can set it up so that any servlet we install (called CLASSNAME) maps into URL space as:

       `http://localhost:59999/servlet/CLASSNAME`

       So, for instance, our Simple servlet will appear as:

       `http://localhost:59999/servlet/Simple`

       This is probably the best option for servlet development and experimentation, as it works for any servlet without requiring you to edit the web.xml file each time you add a servlet.
    2. Alternatively, you can choose a specific URL which should map to a chosen servlet. For example, we might want the Simple servlet to appear as URL:

       `http://localhost:59999/s`

       Note that this allows you to completely separate the servlet class name structure from the URL structure of your web application - this extra level of abstraction is very useful, especially for a more complex web application.
- Whichever option you prefer, both involve customizing the `WEB-INF/web.xml` file, so let's start by `pushd web-apps/ROOT` (where the web content and `WEB-INF` directory lives):
    1. The "map each installed servlet CLASSNAME to `http://localhost:59999/servlet/CLASSNAME`" option uses the Tomcat generic **invoker** servlet. This useful servlet (meta-servlet? proxy-servlet? whatever) needs to be configured to manage the "`/servlet/*`" part of URL space, and when invoked with a URL like `/servlet/XYZ`, searches for the XYZ.class, loads it, and then forwards the request to XYZ.

To enable the Invoker servlet, you must add the following code to the middle of `WEB-INF/web.xml` file:

```
<!-- The invoker servlet, handling /servlet/* URLs -->
<servlet>
    <servlet-name>invoker</servlet-name>
    <servlet-class>
      org.apache.catalina.servlets.InvokerServlet
    </servlet-class>
    <init-param>
        <param-name>debug</param-name>
        <param-value>0</param-value>
    </init-param>
    <load-on-startup>2</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>invoker</servlet-name>
    <url-pattern>/servlet/*</url-pattern>
</servlet-mapping>
```

2. For the second option, where Simple appears as `http://localhost:59999/s` edit `web.xml` and add the following:

```
<!-- The Simple servlet, as URL /s -->
<servlet>
  <servlet-name>simple</servlet-name>
  <servlet-class>Simple</servlet-class>
  <load-on-startup>5</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>simple</servlet-name>
  <url-pattern>/s</url-pattern>
</servlet-mapping>
```

- Having customized the `web.xml` file in whichever way you prefer, `pushd`, `tomcat6 start`, point a web browser at the appopriate URL (either `..../servlet/Simple` or `/s`) and you should see the web page generated by the Simple servlet's **doGet()** method, if we reload the page a few times you'll see the static counter increment, don't forget to `tomcat6 stop` afterwards and `pushd` back into the web app root.

- Note that you can add **both** the above sections to web.xml, and if you do so, the same servlet is accessible via both URLs, but is instantiated twice, once at each location, so the counter is not shared between both URLs. That is, every time you reload:

  `http://localhost:59999/s`

  the counter is incremented, but each time you reload:

  `http://localhost:59999/servlet/Simple`

  an entirely separate copy of the counter is incremented.

- We recommend always adding the invoker servlet stanza to the web.xml, so that you don't need to edit the web.xml every time you want to test a new servlet. Indeed, we may even add it to the default web.xml that is created by `mktomcat6` in the near future.

JDBC Servlets Examples - Servlet_Postgres.java and Servlet_MSSQL.java

See our separate JDBC with DoC Supported Databases document for a single worked example (Films and directors) shown in multiple different forms - including a pair of Servlet versions, one connecting to Postgres and the other to Microsoft SQL Server. Specifically, here's the source code of Servlet_Postgres.java and Servlet_MSSQL.java.

Let's try to get them both working in Personal Tomcat, it's simple. Proceed as follows:

- As Java source code is involved, check you're back in the `example-context` top-level directory via `dirs` and (if necessary) swap dirs with `pushd`, then, use `pushd src` to enter the source directory temporarily (just like you'd do for a java bean).

- Take a copy of both of the above files into the src directory, either by `wget http://www.doc.ic.ac.uk/csg/java/jdbc/Servlet_Postgres.java` and `wget http://www.doc.ic.ac.uk/csg/java/jdbc/Servlet_MSSQL.java`, or (more neatly) by using your DoC-specific knowledge of where the CSG web pages live in Unix filesystem terms, so we can copy both files straight in by:

  ```
  cp /vol/www/csg/java/jdbc/Servlet*.java .
  ```

- Now, modify the `Makefile` so that it reads:

  ```
  CONTEXT =       ../webapps/ROOT
  LIBDIR  =       $(CONTEXT)/WEB-INF/lib
  CLSDIR  =       $(CONTEXT)/WEB-INF/classes
  JARS    =       servlets.jar
  CLASSES =       Simple.class Servlet_Postgres.class Servlet_MSSQL.class

  all:    $(JARS) $(CLASSES)

  clean:
          /bin/rm -f $(JARS) $(CLASSES) $(CLSDIR)/* $(LIBDIR)/*

  install:        $(JARS) # $(CLASSES)
          install -m600 $(JARS) $(LIBDIR)
          #install -m600 $(CLASSES) $(CLSDIR)

  servlets.jar:      $(CLASSES)
          jar cf servlets.jar $(CLASSES)

  Simple.class:   Simple.java
          javac Simple.java

  Servlet_Postgres.class:   Servlet_Postgres.java
          javac Servlet_Postgres.java

  Servlet_MSSQL.class:   Servlet_MSSQL.java
          javac Servlet_MSSQL.java
  ```

Note that this time we're jarring all the servlet classes up, for installing into `WEB-INF/lib`.

- Now compile and install everything via `make install`, fixing any Java or Makefile errors as you go. Then `popd` to get back into the context top-level dir.
- Now, if you've followed our advice and added the invoker servlet stanza into web.xml, then - as soon as you `tomcat6 start` - your new servlets will be accessible via

  `http://localhost:59999/servlet/Servlet_Postgres`

  and

  `http://localhost:59999/servlet/Servlet_MSSQL`
- If not, then you'll need to swap dirs again via `pushd`, and modify the `WEB-INF/web.xml` file - you could choose specific URLs for the Postgres and MSSQL versions of the servlet, but why not now take our advice and add the invoker servlet stanza into web.xml! Then swap dirs again (back to the context top-level), then `tomcat6 start` as usual, and now point your web browser at the above URLs.
- Finally, `tomcat6 stop`, and `pushd` to get back into the web app root for further editing.