Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

2) Deployment of web applications

Emmanuel Benoist

Fall Term 2016-17

Computer Science Division

Presentation of the Course

Web Application on the Server

Introduction

Deployment

- **Background**
  - Prior to the Servlet API Specification, version 2.2, there was little consistency between server platforms.
  - Servers that conform to the 2.2 (or later) specification are required to accept a Web Application Archive in a standard format.
- **Web Application = hierarchy of directories and files in a standard layout**
  - **Unpacked Form:** each directory and file exists in the filesystem separately
  - **Packed Form:** Web ARchive, or WAR file.

Deployed Directories

Document root directory

- **top-level directory of your web application hierarchy is also the document root of your application.**
  - Here, you will place the HTML files and JSP pages that comprise your application's user interface.
  - When the system administrator deploys your application into a particular server, he or she assigns a context path to your application
  - Thus, if the system administrator assigns your application to the context path `/catalog`, then a request URI referring to `/catalog/index.html` will retrieve the index.html file from your document root.

Standard Directory Layout

We sugest that you adopt the following standards for the document root directory of your application (Compatible with WAR format):

- ***.html, *.jsp, etc.** - put the files in your document root You can add a subdirectory hierarchy
- **/WEB-INF/web.xml** - The Web Application Deployment Descriptor for your application.
- **/WEB-INF/lib/** - This directory contains JAR files (for instance: third party class libraries or JDBC drivers).

Standard Directory Layout (Cont.)

- **/WEB-INF/classes/** - Java class files
- including both servlet and non-servlet classes
- package directory hierarchy root: `/WEB-INF/classes/`.
  Java class: `com.mycompany.mypackage.MyServlet`
  File: `/WEB-INF/classes/com/mycompany/mypackage/MyServlet.class`.

Web Application Deployment Descriptor

Web Application Deployment Descriptor

- **The `/WEB-INF/web.xml` file contains the Web Application Deployment Descriptor for your application.**
  - this file is an XML document
  - defines everything about your application that a server needs to know
  - except the context path, which is assigned by the system administrator when the application is deployed
- **Too complicated to be described here**
  The complete syntax and semantics for the deployment descriptor is defined in the Servlet API Specification.
- **Starting point**
  A standard `web.xml` is provided with Tomcat, this file includes comments

Deployment With Tomcat

A web application can be deployed in Tomcat by one of the following approaches:

- Copy unpacked directory hierarchy into a subdirectory in directory `webapps/`. Tomcat will assign a context path to your application based on the subdirectory name you choose.
- Copy the web application archive file into directory `webapps/`. When Tomcat is started, it will automatically expand the web application archive file into its unpacked form, and execute the application that way.
- Use the Tomcat "Manager" web application to deploy and undeploy web applications.
- Add a `<Context>` entry in the `conf/server.xml` configuration file.

An example of `web.xml` File

General Description of the Web Application

```
<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
    <display-name>My Web Application</display-name>
    <description>
      This is version X.X of an application to perform
       a wild and wonderful task, based on servlets and
       JSP pages. It was written by Dave Developer
       (dave@mycompany.com), who should be contacted for
       more information.
     </description>
```


web.xml - Servlet Definitions

- **Servlet definition**

```
    <servlet>
    <servlet-name>controller</servlet-name>
      <description> Servlet Description </description>
    <servlet-class>com.mycompany.mypackage.ControllerServlet
    </servlet-class>
    <init-param>
        <param-name>listOrders</param-name>
        <param-value>com.mycompany.myactions.ListOrdersAction
        </param-value>
      <init-param>
        <param-name>saveCustomer</param-name>
        <param-value>com.mycompany.myactions.SaveCustomerAction
        </param-value>
      </init-param>
      <!-- Load this servlet at server startup time -->
      <load-on-startup>5</load-on-startup>
</servlet>
```


web.xml - Servlet Mappings

- **Mappings translate request URI to servlets**
  The examples below correspond to the above servlet descriptions. Thus, a request URI like:

  `http://localhost:8080/{contextpath}/graph`

  will be mapped to the "graph" servlet, while a request like:

  `http://localhost:8080/{contextpath}/saveCustomer.do`

  will be mapped to the "controller" servlet.
  You may define any number of servlet mappings, including zero. It is also legal to define more than one mapping for the same servlet, if you wish to.

web.xml - Servlet Mappings (Cont.)

```
<servlet-mapping>
<servlet-name>controller</servlet-name>
<url-pattern>*.do</url-pattern>
</servlet-mapping>
<servlet-mapping>
<servlet-name>graph</servlet-name>
<url-pattern>/graph</url-pattern>
</servlet-mapping>
<session-config>
<session-timeout>30</session-timeout>
<!-- 30 minutes -->
</session-config>
</web-app>
```

Source Organization

Directory Structure

Source Organization

- **Directory Structure**
  Separate the directory hierarchy containing your source code from the directory hierarchy containing your deployable application
  - The contents of the source directories can be more easily administered, moved, and backed up if the "executable" version of the application is not intermixed.
  - Source code control is easier to manage on directories that contain only source files.
  - The files that make up an installable distribution of your application are much easier to select when the deployment hierarchy is separate.

Source Directory Structure (Cont.)

We recommand :
Under a top project directory

- **docs/** - Documentation for your application, in whatever format your development team is using.
- **src/** - Java source files
  Source code organized in directories representing packages
- **web/** - The static content of your web site (HTML pages, JSP pages, JavaScript files, CSS stylesheet files, and images)
- **web/WEB-INF/** - The special configuration files (`web.xml` etc.)

Source Directory Structure (Cont.)

- **During the development process, two additional directories will be created on a temporary basis:**
  - **build/** - this directory will contain an exact image of the files in the web application
  - **dist/** - When you execute the ant `dist` target, this directory will be created.
    It will create an exact image of the binary distribution for your web application, including any license information, documentation, and README files that you have prepared.

External Dependencies

External Dependencies

- **What to do if you need JAR libraries**
  (Database drivers for instance)
  - One solution : copy the JAR files into the source code control archives for every application that requires those files.
    Problem : Problem for upgrade of versions
  - Possible Solution: You do not store a copy of the package
    Instead : the external dependencies should be integrated as part of the process of deploying your application.
    It can be configured in the `build.xml` file.

Source Code Control

Source Code Control

- **Subversion (SVN)**
  - You should place your SOURCE files under SVN
  - Do not register and save generated files
- **You should not store the contents of the build/ and dist/ directories**
  - They are authomaticaly generated
  - Tell SVN to ignore them
  - Tell SVN to ignore `build.properties` (reason explained later)

Subversion (Cont.)

- **refresh** the state of your source code: `svn update`.
- **new subdirectory** in the source code hierarchy: `svn add {subdirname}`.
- **new source code file** `svn add {filename}`.
- **no longer need a particular source code file**: `svn remove {filename}`.
- **To save your changes**: `svn commit -m"Type your own comment here"`.

Ant Deployment

Build.xml Configuration File

- **The ant tool**
  - `ant` is for java what `make` is for C++
  - ant uses `build.xml` like make `Makefile`
  - `build.xml` is stored in the top-level directory of your source code hierarchy.
- **Targets**
  - "targets" support optional development activities
    - Creating the associated Javadoc documentation,
    - Erasing the deployment home directory so you can build your project from scratch,
    - Creating the web application archive file so you can distribute your application.
  - `build.xml` file will contain internal documentation describing the targets.

To ask Ant to display the project documentation,

```
cd myProject/
ant -projecthelp
```

Build.xml Configuration File (Cont.)

- **clean** - This target deletes any existing build and dist directories, so that they can be reconstructed from scratch.
- **prepare** - This target "prepares" the build directory, creating subdirectories as required.
  A common use of this target is to copy static files (documentation, HTML pages, and JSP pages) from the source directory to the build directory.
- **compile** - This target is used to compile any source code that has been changed since the last time compilation took place.
- **all** - This target is a short cut for running the clean target, followed by the compiile target.

Build.xml Configuration File (Cont.)

- **deploy** - This target is used to install your application into a servlet container so that it can be tested. If your application requires external JAR files, they will be copied to the /WEB-INF/lib directory at deployment time.
- **javadoc** - This target creates Javadoc API documentation for the Java classes in this web application.
- **dist** - This target creates a distribution directory for your application, including
  - any required documentation,
  - the Javadocs for your Java classes,
  - a web application archive (WAR) file

Example of build.xml

Properties of Files and Directory Names
These properties generally define file and directory names (or paths) that affect where the build process stores its outputs.

- **app.name** Base name of this application, used to construct filenames and directories.
- **app.version** Version identifier for this application.
- **build.home** The directory into which the `prepare` and `compile` targets will generate their output. Defaults to `build`.
- **catalina.home** The directory in which you have installed a binary distribution of Tomcat.
- **deploy.home** where the deployment hierarchy will be created, usually: `$catalina.home/webapps/$app.name`.
- **dist.home** The name of the base directory in which distribution files are created. Defaults to `dist`.

Build.xml (Cont.)

```
<property name="app.name"      value="myapp"/>
<property name="app.version"   value="1.0"/>
<property name="build.home"    value="build"/>
<property name="catalina.home" value="../../../.."/>
<property name="deploy.home"
              value="${catalina.home}/webapps/${app.name}"/
<property name="dist.home"     value="dist"/>
```

Build.xml (Cont.)

- **Compilation Control Options**
  These properties control option settings on the Javac compiler when it is invoked using the <javac> task.
  - **compile.debug** Should compilation include the debug option?
  - **compile.deprecation** Should compilation include the deprecation option?
  - **compile.optimize** Should compilation include the optimize option?

```
<property name="compile.debug"       value="true"/>
<property name="compile.deprecation" value="false"/>
<property name="compile.optimize"    value="true"/>
```


Build.xml (Cont.)

- **External Dependencies**
  Use property values to define the locations of external JAR files on which your application will depend. In general, these values will be used for two purposes:
  - Inclusion on the classpath that is passed to the Javac compiler
  - Being copied into the `/WEB-INF/lib` directory during execution of the "deploy" target.

  Because we will automatically include all of the Java classes that Tomcat exposes to web applications, we will not need to explicitly list any of those dependencies. You only need to worry about external dependencies for JAR files that you are going to include inside your `/WEB-INF/lib` directory.

```
<property name="foo.jar"
          value="/path/to/foo.jar"/>
```


Build.xml (Cont.)

Compilation Classpath

- Rather than relying on the CLASSPATH environment variable,
- Ant includes features that makes it easy to dynamically construct the classpath you need for each compilation.
- The example below constructs the compile classpath to include the `servlet.jar` file, as well as the other components that Tomcat makes available to web applications automatically, plus anything that you explicitly added.
- Include all JAR files that will be included in /WEB-INF/lib

Build.xml (Cont.)

Compilation Classpath

```
<path id="compile.classpath">
<pathelement location="${foo.jar}"/>

<!-- Include all elements that Tomcat exposes to ↘
 →applications -->
<pathelement location="${catalina.home}/common/↘
 →classes"/>
<fileset dir="${catalina.home}/common/lib">
<include name="*.jar"/>
</fileset>

<pathelement location="${catalina.home}/classes"/>
<fileset dir="${catalina.home}/lib">
<include name="*.jar"/>
</fileset>
</path>
```

Conclusion

- **There are two different arborescences**
  - A source one on which you work, that is "subversioned"
  - A deployment one that is on the Servlet Engine (Tomcat)
- **Most of you won't see it**
  - IDE (Eclipse for instance) hide the details
  - But deployment is often a big issue (source of problems)
  - Understanding it helps to solve a lot of problems.
- **Examples of this course are platform-independent**
  - They are developped with text editeur (emacs)
  - They are deployed with `ant`.
  - They can be integrated in any IDE (just need to adapt)