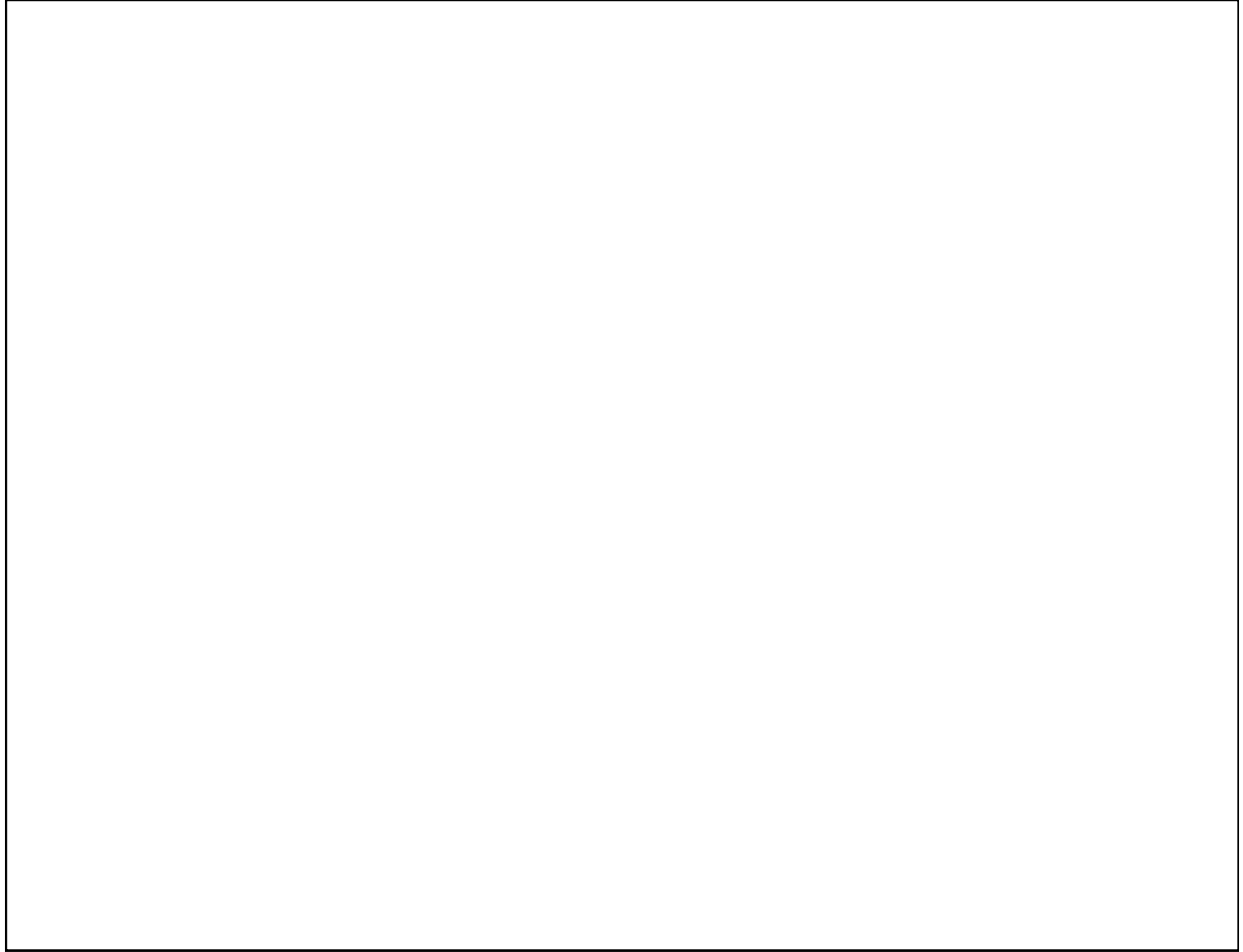


# Web Applications

- What is a Web Application?
- Packaging a Web Application
- Deploying a Web Application

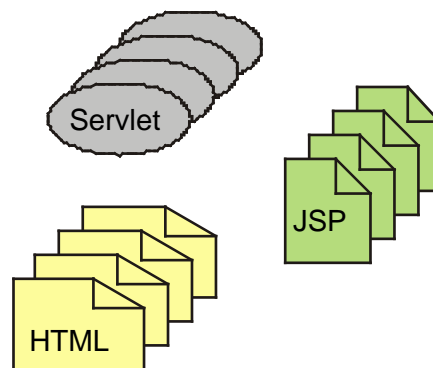
9 - 1

The lab exercises and sample programs we've examined so far have had only a few servlets and static HTML files, so deploying them was easy. It would not be so easy if you had a system with thousands of files -- deploying them one at a time would be tedious and hard to maintain. The topic of this chapter, Web Applications will make the deployment process much easier.



## What is a Web Application?

- A web application is a collection of servlets, HTML, and JavaServer Pages that work together as a single program
- The Servlet specification defines the structure and contents of a web application



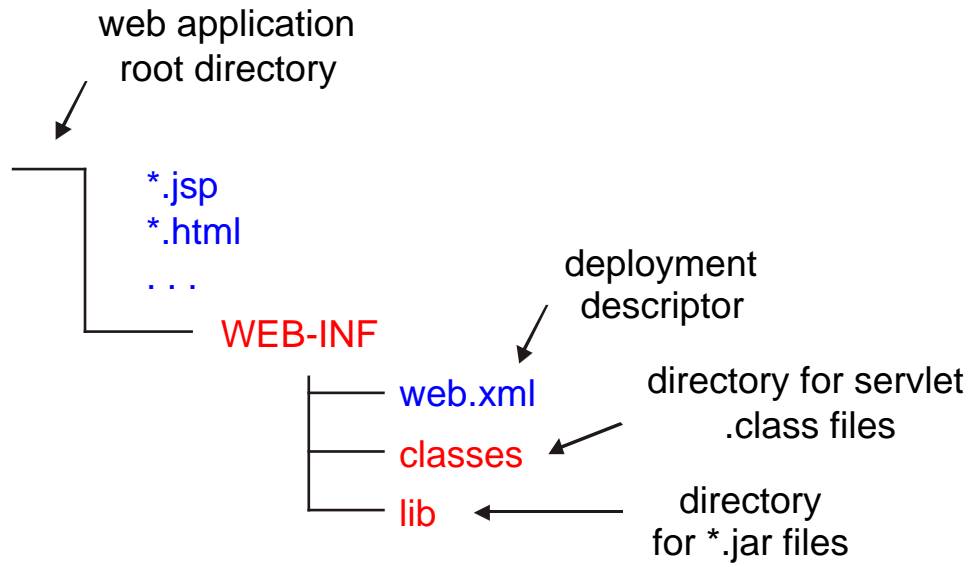
9 - 2

In enterprise systems, applications consist of hundreds, if not thousands of files. A web application lets you group them together for easier deployment.

This notion of a web application is not just a fuzzy phrase -- the contents and structure of web application is defined in the servlet specification.

## Web Application Directory Structure

- Web applications reside in a hierarchy of directories - the structure is defined by the Servlet specification



9 - 3

This figure shows the directory structure required for web applications. You create a root directory, and in it, place files and other sub-directories. The sub-directories must have the names shown here. The names are case-sensitive.

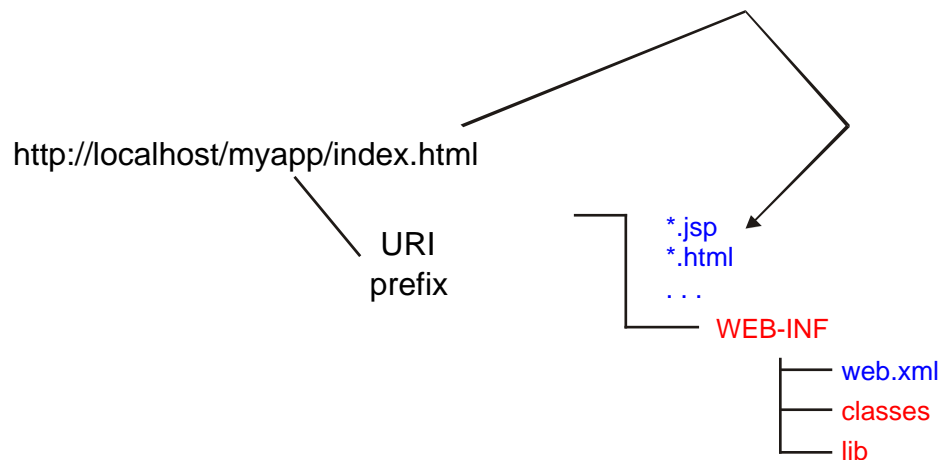
The root directory is where you deploy your JavaServer pages and static HTML files. You deploy your servlets into WEB-INF/classes directory or the WEB-INF/lib directory if you have packaged the servlets into JAR files.

The required "web.xml" file is known as the deployment descriptor -- it configures the web application. We will cover its syntax later.

While we show this as an explicit directory structure, it's also possible to package the entire thing into a single archive known as a .war file.

## The URL for the Web Application

- Servlet containers normally let you define a virtual directory (URI prefix) for the web application that the container aliases to the actual deployment directory
- The virtual directory lets you insulate user URLs from the actual directory structure on the server



9 - 4

The servlet specification defines a technique to assign what's essentially a name for your web applications. Here, we've assigned the name "myapp" -- all URLs for the application's servlets, JSPs and static HTML pages have "myapp" as part of the URL. This lets you define several web applications on the same server and easily segregate them.

## The Deployment Descriptor

- The Servlet specification requires you to create a file named *web.xml* referred to as the deployment descriptor and store it in the WEB-INF directory
- The deployment descriptor describes the web application and is processed by the servlet container's deployment facility

```
1    <?xml version="1.0"?>
2    <web-app>
3    <display-name>MyApp</display-name>
4    <description>My Web Application</description>
5    <welcome-file-list>
6        <welcome-file>index.html</welcome-file>
7    </welcome-file-list>
8    </web-app>
9 - 5
```

The deployment descriptor uses XML syntax to configure the web application. Like HTML, XML is text marked up with tags -- the start and end-tags delimit elements that are processed by the container.

This page shows a deployment descriptor for a simple web application. It defines the standard XML declaration in line 1, followed by the required root element of "web-app". The "display-name" and "description" elements provide information about the web application. The "welcome-file-list" element lets you configure which files will be shown if the referencing URL does not include a file name. In this case, if the user enters a URL of <http://localhost/myapp>, the container will show the `index.html` file from the web application's root directory.

There are other elements possible in the deployment descriptor -- we will see more of them on the next few pages.

## Deployment Descriptor Elements

- ServletContextInitParameters
- Session Configuration
- Servlet/ JSP Definitions
- Servlet/ JSP Mappings
- Mime Type Mappings
- Welcome File list
- Error Pages
- Security

9 - 6

This page lists the different kinds of elements allowed in the deployment descriptor. See the Servlet Specification for a complete discussion of these.

## Using Initialization Parameters

### web.xml Fragment

```
1    <servlet>
2        <servlet-name>Login</servlet-name>
3        <servlet-class>Login</servlet-class>
4        <init-param>
5            <param-name>username</param-name>
6            <param-value>Paul Westerberg</param-value>
7        </init-param>
8        <init-param>
9            <param-name>password</param-name>
10           <param-value>Tim</param-value>
11        </init-param>
12    </servlet>
```

9 - 7

One useful technique that uses the deployment descriptor is initialization parameters. Initialization parameters let you avoid hard-coding information in your servlets -- instead, you put the information into the deployment descriptor and retrieve it from the servlet. Then if you need to change the information, you just change the deployment descriptor without needing to recompile the servlet. One possible application of this technique is for JDBC URLs.

In this fragment of a deployment descriptor, we define a servlet that has two initialization parameters. In line 2, we assign a name to the servlet, and then in lines 4 to 7 and 8 to 10 we define initialization parameters for "username" and "password". (This is not a very secure way to store passwords!)

## Using Initialization Parameters, cont'd

### Login.java (servlet)

```
1   private String user;
2   private String pass;
3
4   public void init ()
5   {
6       ServletConfig config = getServletConfig();
7       user =
8           config.getInitParameter ( "username" );
9       pass =
10          config.getInitParameter ( "password" );
11  }
```

9 - 8

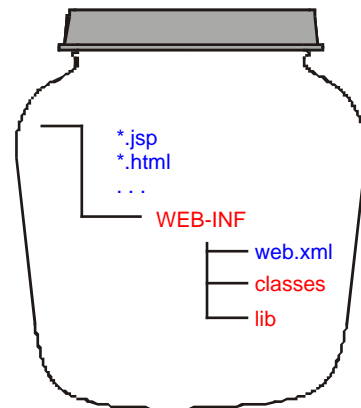
Continuing the example from the last page, here's the servlet that uses the initialization parameters. In the `init()` method, in line 6, we retrieve a reference to a `ServletConfig` object that gives access to the initialization parameters. The `getServletConfig` method is inherited from the servlet's superclass.

Then, in lines 7 to 10, the servlet retrieves the "username" and "password" initialization parameters.

## Packaging the Web App into a .war File

- Optionally, you can use the standard JAR utility from the JDK to package your web application into an archive
- The servlet container's deployment facility will accept the .war file as input during deployment

```
jar cvf MyApp.war *.*
```



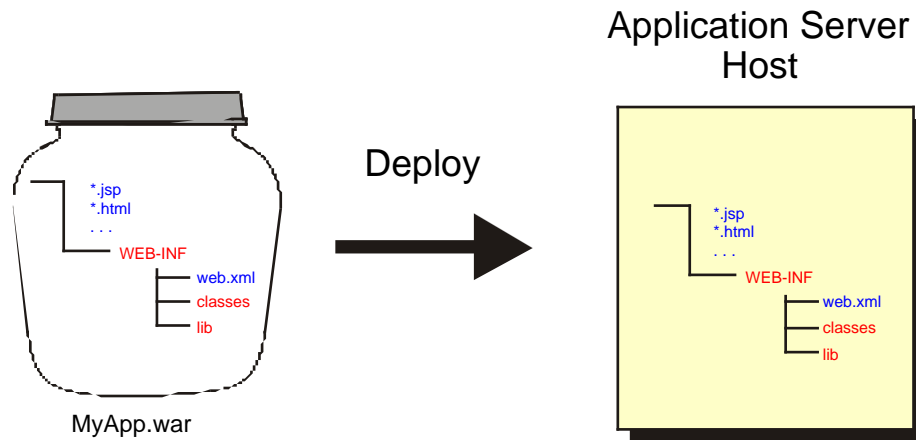
MyApp.war

9 - 9

Since deploying hundreds or thousands of files is tedious and error-prone, it's a good idea to package your web application's directory structure and files into a .war file. You can then deploy the web application using just the single .war file.

## Deploying the Web Application

- The Servlet specification allows servlet container vendors to provide their own deployment facilities that read the deployment descriptor and .war file



9 - 10

The actual process of deploying web applications is container specific -- see your container's documentation for more details. In some cases, though, all you'll need to do is copy the .war file to a particular directory on the web server.

## The Servlet Context

- The Servlet API provides a programmatic access from a servlet or JSP to the web application via the ServletContext interface
- According to the spec, the ServletContext is a "view" onto the web application that lets you maintain state across the web application
- This is similar to a session, but more global

ServletContext
getAttribute ( name: String ) : Object setAttribute ( name: String, value: Object ) : void ...

9 - 11

The servlet context is similar to a session, but more global. Sessions apply to a single user, while the servlet context refers to all of the principals using the web application. You can thus use the servlet context to store information, such as a user count, that's global to any single user.

Another example is a chat room application that could use the ServletContext to maintain the record of everything that's been said in the chat room. This "transcript" would be global to any user.

To use the servlet context, first call `getServletContext()`, which is inherited from the `GenericServlet` superclass. You can then call `setAttribute` to store data and `getAttribute` to later retrieve it. This works in a similar fashion to the session API covered earlier.

## Lab Exercise

- **Title:** Lab 5: Web Applications
- **Lab doc:** /servclass/doc/lab05.pdf
- **Lab directory:** /servclass/lab05
- **Lab solution:** /servclass/lab05/solution
- **Approximate time:** 30 minutes



## Chapter Summary

In this chapter, you learned:

- What a Web Application is
- How to create and deploy a Web Application