Software Development Technologies

Information and Communication Technology

Training Institute, Pyay

[Java Programming Advanced]

Csontents at a Glance

Table of Contents

1. Web Application

1.1. Characteristics of Java Web Application

1.1.1. A static web page

A static web page is the page where only static information is presented without any change of view. To show static web pages, the following components are required.

- A browser on the client machine
- A Web server on the server machine
- Internet network

Usually users input URL address from the browser or click the Hyper Text Link to obtain the information in the Internet. The server of the *URL* (*Unified Resource Locater*) address receives the request of the clients and sends them back the requested resources (*HTML* documents, text files or graphic files) stored in the Web server machine

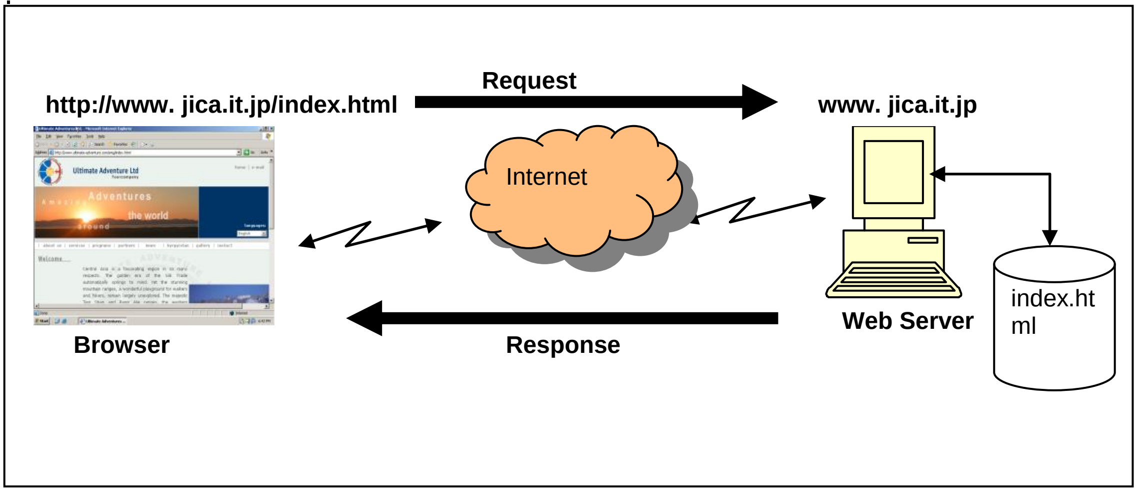
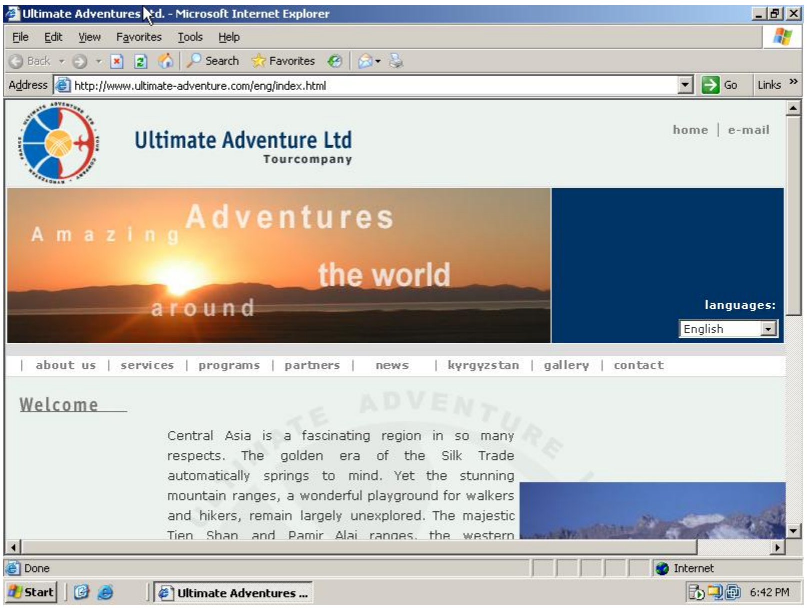


Figure 1 Web server and browser

The static pages show just information without any interaction with the user. Some tourist information, government information, news, homepages are static web pages.

Client and server communicate each other by *HTTP* (HyperText Transfer Protocol) protocol for data transmission.


Ultimate Adventures Ltd. - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://www.ultimate-adventure.com/eng/index.html
Go Links
Ultimate Adventure Ltd
Tourcompany
home | e-mail
Amazing Adventures around the world
languages:
English
| about us | services | programs | partners | news | kyrgyzstan | gallery | contact
Welcome
Done
Internet
Start
Ultimate Adventures ...
6:42 PM

Central Asia is a fascinating region in so many respects. The golden era of the Silk Trade automatically springs to mind. Yet the stunning mountain ranges, a wonderful playground for walkers and hikers, remain largely unexplored. The majestic Tien Shan and Pamir Alai ranges, the western

Figure 2 Sample of Static Web Page

- **HTML(Hyper Text Markup Language)**

  HTML is a language that Web browser can understand to show web page in a browser. You can see HTML source code by right-clicking and selecting "source code" in a web page.

[Example of HTML code: Hello.html]

```
<html>
    <head><title>Hello World!</title></head>
    <body>
        <h1>This is my Hello html </h1>
            <p>
                Hello from Hello.html!
            </p>
    </body>
</html>
```

- **HTTP (HyperText Transfer Protocol)**

  HTTP is a standard Internet protocol that consists of HTTP request and HTTP response used for client and server communication. A client sends request of HTML file and the Web sever sends the file back to the browser. A connection finishes in only one round trip.

1.1.2. What a Web server can do and can not do?

[what can do]

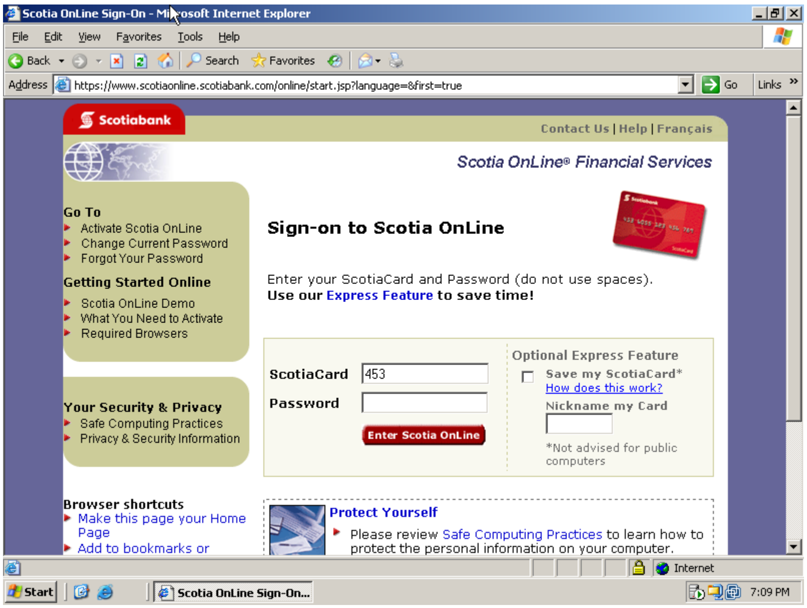
- A web server can responds the request of the client to send back information in file format (HTML documents, text files, and graphic files) stored in the server machine.
- A web server has identical URL address to be identified. It can be located from any Internet network.
- A web server can call or invoke another program or another application passing the client request data in order to process it.

[what can not do]

- A web server itself can not have enough functions to process data, e.g. to connect with database.
- A web server only supports stateless HTTP connection, which means, the connection with client is dropped in one round trip of request and response.

1.1.3. A dynamic web page

A dynamic web page is the page that changes its content according to the users interaction (data input, select some data, click a button, etc.). Usually a web server alone can not create a dynamic web page, since a web server does not have sufficient functions to process data. Some e-business sites, user registration page, shopping cart site (e.g. Amazon.com) or bank transfer sites are created with dynamic pages.



Scotia OnLine Sign-On - Microsoft Internet Explorer

Address: https://www.scotiaonline.scotiabank.com/online/start.jsp?language=&first=true

Scotia OnLine® Financial Services

Sign-on to Scotia OnLine

Enter your ScotiaCard and Password (do not use spaces).
Use our Express Feature to save time!

ScotiaCard: 453
Password:

Optional Express Feature
□ Save my ScotiaCard*
How does this work?
Nickname my Card
*Not advised for public computers

Protect Yourself
Please review Safe Computing Practices to learn how to protect the personal information on your computer.

Figure 3 Sample of dynamic web page

1.1.4. What is Web Application

- A web application can process information based on the parameters that are sent from the browser via web server, what a web server itself can not realize alone. A web application generates HTML for a new web page and sends it back to the user. Therefore a dynamic web page is created by web application.
- Web Application is a client-server application that works via Internet network. Since it is client-server connection, the server should be running to receive anytime request from a client.
- Usually users work with a web browser in a web Application.
- A client always sends request. The server sends response to the client when necessary. Usually without request of client, the server does not send response.
- The connection ends after request-response connection. A connection consists of only one round trip.
- A Web Application runs under Web Server. Apache is famous for Web Server.
- *CGI* (Common Gateway Interface) was one of the techniques applied for web application. Perl, or C++ is famous for the application program that is invoked by a web server. They work as a process in a web server memory.

1.2. Servlet

1.2.1. What is Servlet

Servlet is one of the components of Java 2 Platform, Enterprise Edition J2EE specified by Sun Microsystems for the server side Java web application. The Java web application is composed of three tiers.

- First tier

  This tier is input and output on the client machine, such as web browser.
- Second tier

  Business logic and view control belong to this tier. Servlet containers, JSP container, Java Bean or JSP are the components of this tier
- Third tier

  Database system belongs to this tier.

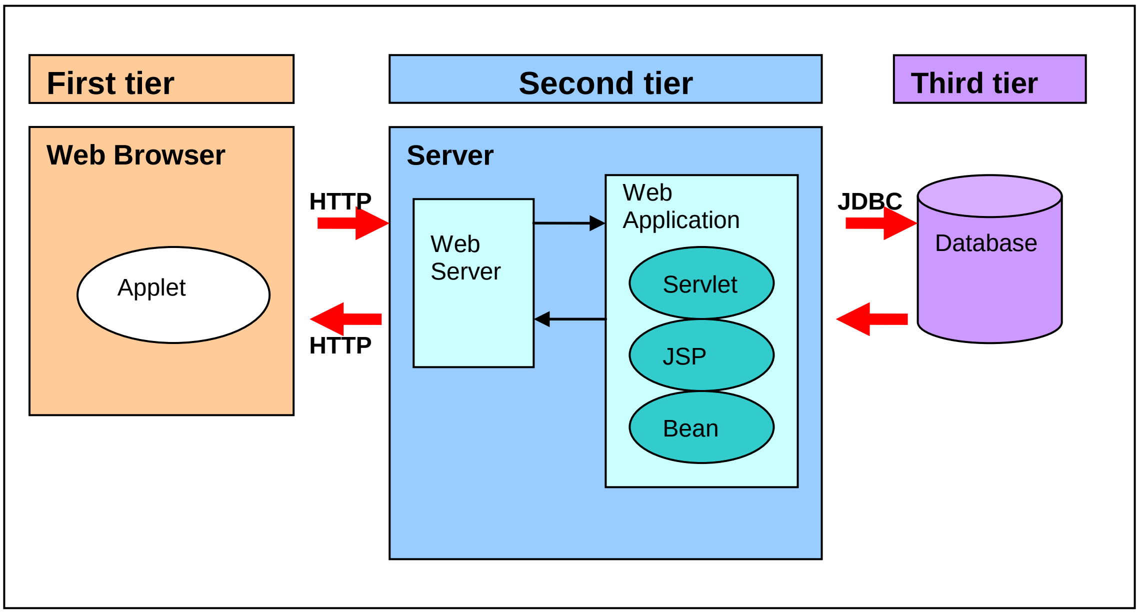


Figure 4 Java Web application components

1.2.2. What is the difference between CGI and Servlet

CGI works as a process in a web server memory, while Servlet works as a thread, that is, a small unit of execution. Suppose that there are hundreds of users accessing to the same web server. In CGI the same number of process should be running in the server memory for execution. If a process has 1MB size, hundreds of MB will be occupied in the memory. On the other hand Servlet can share some variables, resource in some memory area between threads. Although there are hundreds of users accessing to the server, there is only one process working. There is hundreds of thread inside working at the same time. This reduces the memory waste compared with CGI programs.

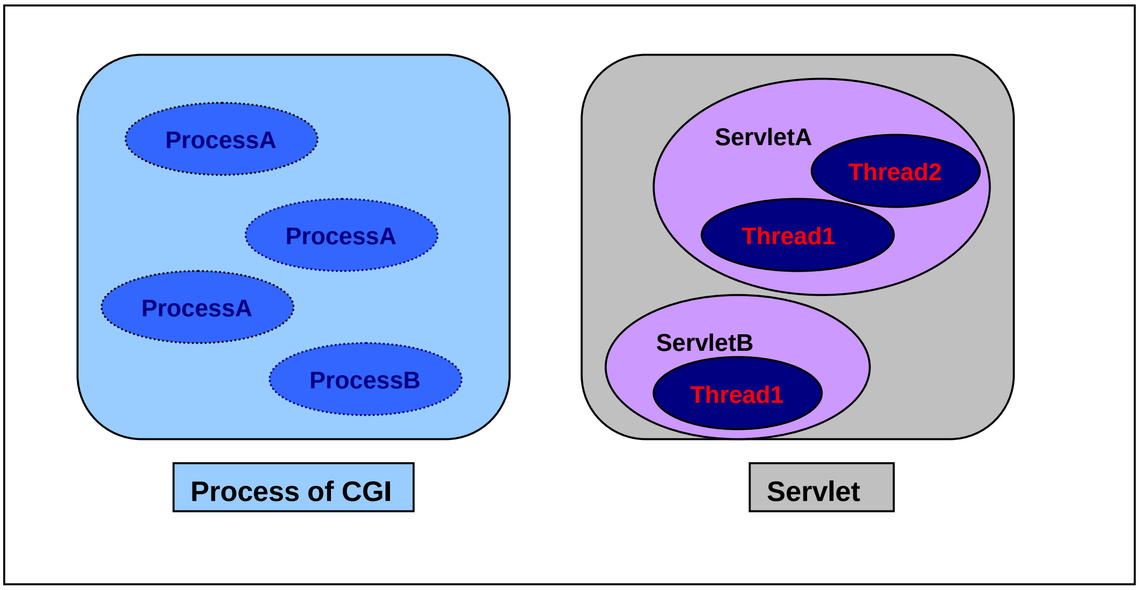


Figure 5 CGI and Servlet

1.2.3. Example code of HelloServlet

```
package ictti;

import java.io.*;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class HelloServlet extends HttpServlet {
    protected void doGet(HttpServletRequest req,
                         HttpServletResponse res)
                    throws ServletException, IOException {
        PrintWriter out = res.getWriter();
        out.println("<html>");
        out.println("<head><title>HelloServlet</title></head>");
        out.println("<body>");
        out.println("Hello Servlet!");
        out.println("</body></html>");
        out.close();
    }

}
```

- Servlet programs always need to refer classes of the following packages;

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

- Servlet always extends HttpServlet.

```
public class HelloServlet extends HttpServlet {
```

- Override the doGet or doPost method of the HttpServlet class.

```
protected void doGet(HttpServletRequest req, HttpServletResponse res)
```

HttpServletRequest is a request object sent from a client.

HttpServletResponse is a response object sent to a client.

- Exceptions should be always declared.

```
throws ServletException, IOException
```

- PrintWriter is available to send messages from server to a client.

```
PrintWriter out = res.getWriter();
```

With *getWriter*() method of *HttpServletResponse*, a *PrintWriter* is obtained. The *PrintWriter* enables to return character data to the browser.

1.2.4. What is JSP(Java Server Pages)

JSP consists of Java codes to realize dynamic web page. JSP is embedded in HTML documents and converted to HTML to Servlet in order to be executed on the Server Side by JSP engine.

[Example code]

```
<html>
    <head><title>Hello World!</title></head>
    <body>
        <h1>Test JSP </h1>
            <p>
            <%
            String hello = "Hello JSP Servlet!";
            out.println(hello);
            %>
            </p>
    </body>
</html>
```

1.2.5. What is JavaScript

JavaScrip is a script embedded in HTML documents. The difference between JavaScript and JSP is that JavaScript is executed on the client machine while JSP is translated to Servlet on the server side.

[Example JavaScript code]

```
<html>
  <head>
    <title>
    </title>
    <script language="JavaScript">
    <!--
        function event() {window.alert("alert window will be open!")}
    // -->
    </script>
  </head>
  <body>
    <form>
        <input type="button" name="ButtonCtrl" value="click here"
        onClick="event()">
    </form>
  </body>
</html>
```

1.2.6. What do you need for Servlet

1) JDK (Java Development Kit)

2) Web Server

   Apache is the most popular web server. Tomcat also comes with Web Server with light functionality.

3) Servlet Container

   There are several Servlet Containers offered by different vendors;

- Tomcat (Open source)
- WebSphere Application (IBM)
- WebLogic Server (BEA)
- Sun One Application (Sun)
- Oracle Application Server (Oracle)

1.2.7. How to call Servlet

Indicate *URL* from browser. The *URL* format is;

```
http://hostname/context-path/servlet-path/servletname
```

[Example]

```
http://localhost:8080/webhello/ictti.Hello
```

- Context-path

  The character string that represents Web Application name
- Servlet-path

  The character string represents Servlet path name.
- Servlet name

  The name of Servlet.

1.2.8. What do you need for Servlet development

- JDK
- Servlet API (Tomcat or other application servers include Servlet API)
- Development tool (such as Eclipse)
- Debugging environment (Eclipse provides debugging environment)

1.2.9. How to develop Servlet

1) Setup Servlet Container

- by installing Tomcat
- by installing other Web Application Servers.

2) Create Servlet program

- You can create a program using any text editor (like notepad), Eclipse or other IDE

3) Compile the Servlet program

- Usually development tool provides compiler. On Eclipse, a file is compiled automatically by saving.

4) Deploy the class file under "*webapp*" directory if you use Tomcat.

- "*webapp*" is a directory that Tomcat recognizes. Tomcat tries to look for web application program in the web application directory under "webapp" of Tomcat home directory. If you want to create application directory other than Tomcat directory, you have to specify it in Tomcat configuration file.

5) Test the program

1.3. Tomcat

Tomcat is a middleware that provides Web server, Servlet Container and JSP container for Web Applications. It is a web container developed at the Apache Software Foundation and implements the Servlet and JSP specifications from Sun Microsystems. There are packages both for Windows and Unix environment.

(1) Tomcat is Reference Implementation of Servlet and JSP.

(2) Open Source

(3) Tomcat5.X implements Servlet 2.4 and JSP 2.0 specifications.

1.3.1. How to install Tomcat

1) You can download various version of Tomcat from the following download site.

http://jakarta.apache.org/site/downloads/downloads_tomcat.html

2) Unzip the setup program.

3) If you are installing to Windows, you can execute the setup program. Then the configuration screen appears. You can change Connector Port if you like. The default port is "8080". Enter administrator name and password. Administrator name and password will be required when you configure Tomcat from Tomcat administration later.

☆ If you use Tomcat independently without any IDE, be sure to set the JAVA_HOME environment variable adequately.

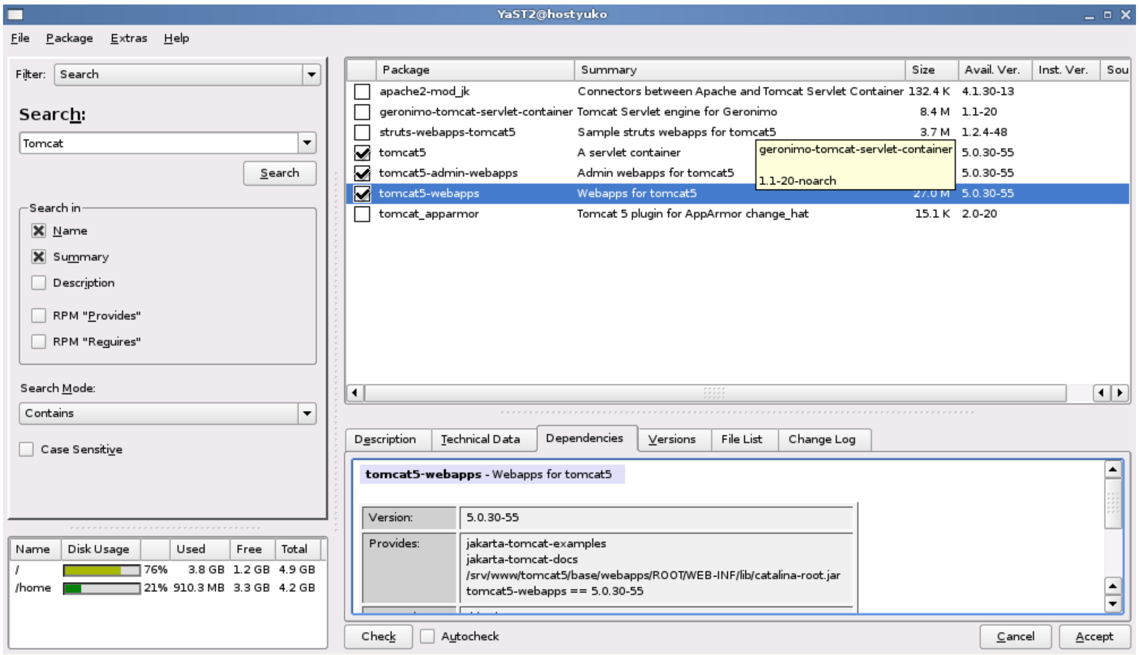
1.3.2. How to install Tomcat in Suse 10.2

You can use the YaST Control Center to install Tomcat application.

1) Start YaST Control Center and open "Software Management".

2) Input "Tomcat" in the search textbox and search the application.

3) Check the related packages and "Accept".



Filter: Search

Search: Tomcat

Search in
☒ Name
☒ Summary
☐ Description
☐ RPM "Provides"
☐ RPM "Requires"

Search Mode: Contains

☐ Case Sensitive

	Package	Summary	Size	Avail. Ver.	Inst. Ver.	Sou
☐	apache2-mod_jk	Connectors between Apache and Tomcat Servlet Container	132.4 K	4.1.30-13		
☐	geronimo-tomcat-servlet-container	Tomcat Servlet engine for Geronimo	8.4 M	1.1-20		
☐	struts-webapps-tomcat5	Sample struts webapps for tomcat5	3.7 M	1.2.4-48		
☑	tomcat5	A servlet container		5.0.30-55		
☑	tomcat5-admin-webapps	Admin webapps for tomcat5		5.0.30-55		
☑	tomcat5-webapps	Webapps for tomcat5	27.0 M	5.0.30-55		
☐	tomcat_apparmor	Tomcat 5 plugin for AppArmor change_hat	15.1 K	2.0-20		

Name	Disk Usage		Used	Free	Total
/		76%	3.8 GB	1.2 GB	4.9 GB
/home		21%	910.3 MB	3.3 GB	4.2 GB

tomcat5-webapps - Webapps for tomcat5

Version:	5.0.30-55
Provides:	jakarta-tomcat-examples jakarta-tomcat-docs /srv/www/tomcat5/base/webapps/ROOT/WEB-INF/lib/catalina-root.jar tomcat5-webapps == 5.0.30-55

☐ Autocheck

Figure 6 Download Tomcat from YaST

The YaST Control Center automatically installs the packages to;

- binary files /usr/share/tomcat5/bin/
- jar files /usr/share/tomcat5/server/lib/
- log files /var/log/tomcat5/base/
- conf files /etc/tomcat5/base/

1.3.3. How to start Tomcat server

1) To start Tomcat, execute "*%TomcatInstallDir%bin/startup.sh*" (Linux) or "*%TomcatInstallDir%bin/start.bat*" (Windows) of bin directory under Tomcat home directory.

☆ If you use Linux Suse10.2, execute;

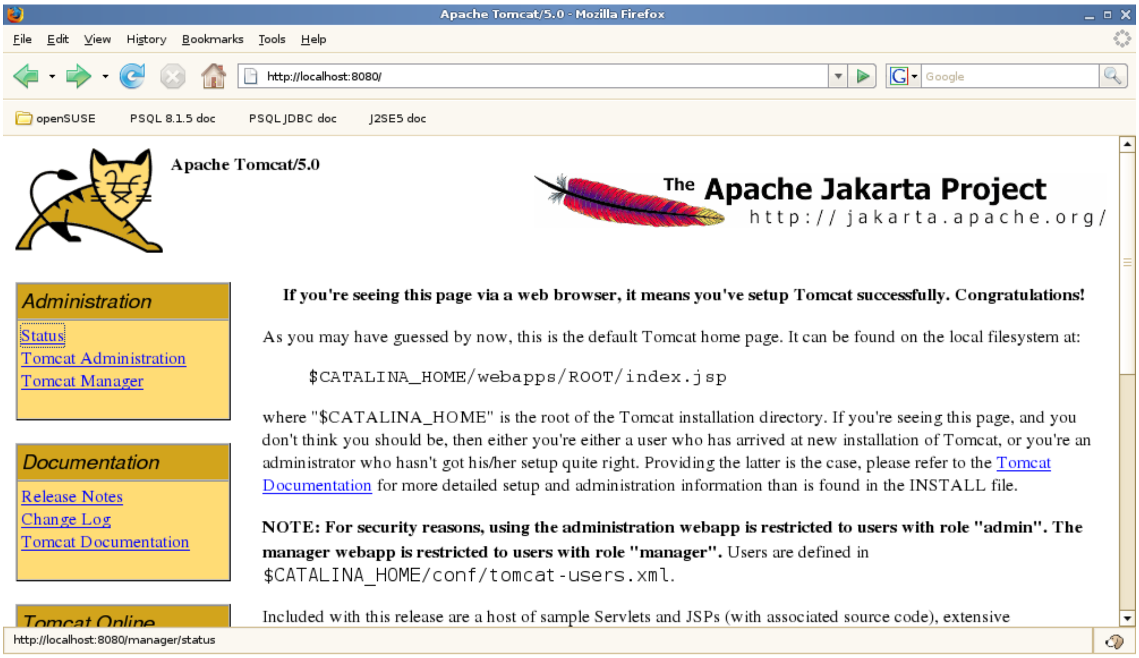
```
/usr/share/tomcat5/bin/startup.sh
```

☆ You need to have root permission to run Tomcat application (as default) in Linux environment.

2) Type the following URL from the browser;

```
http://localhost:8080
```

3) If the first page of Tomcat appears, your installation has succeeded.



If you're seeing this page via a web browser, it means you've setup Tomcat successfully. Congratulations!

As you may have guessed by now, this is the default Tomcat home page. It can be found on the local filesystem at:

```
$CATALINA_HOME/webapps/ROOT/index.jsp
```

where "$CATALINA_HOME" is the root of the Tomcat installation directory. If you're seeing this page, and you don't think you should be, then either you're either a user who has arrived at new installation of Tomcat, or you're an administrator who hasn't got his/her setup quite right. Providing the latter is the case, please refer to the Tomcat Documentation for more detailed setup and administration information than is found in the INSTALL file.

NOTE: For security reasons, using the administration webapp is restricted to users with role "admin". The manager webapp is restricted to users with role "manager". Users are defined in `$CATALINA_HOME/conf/tomcat-users.xml`.

Included with this release are a host of sample Servlets and JSPs (with associated source code), extensive

Figure 7 Tomcat First Page

4) You can check the initial log message by;

```
cat /var/log/tomcat5/base/catalina.out
```

1.3.4. How to stop Tomcat server

1) Execute "*%TomcatInstallDir%bin/shutdown.sh*" or "*%TomcatInstallDir%bin/shutdown.bat*" of Tomcat home directory.

☆ If you use Linux Suse10.2, execute;

```
/usr/share/tomcat5/bin/shutdown.sh
```

1.3.5. Tomcat directory structure

Tomcat directory structure is as follows;

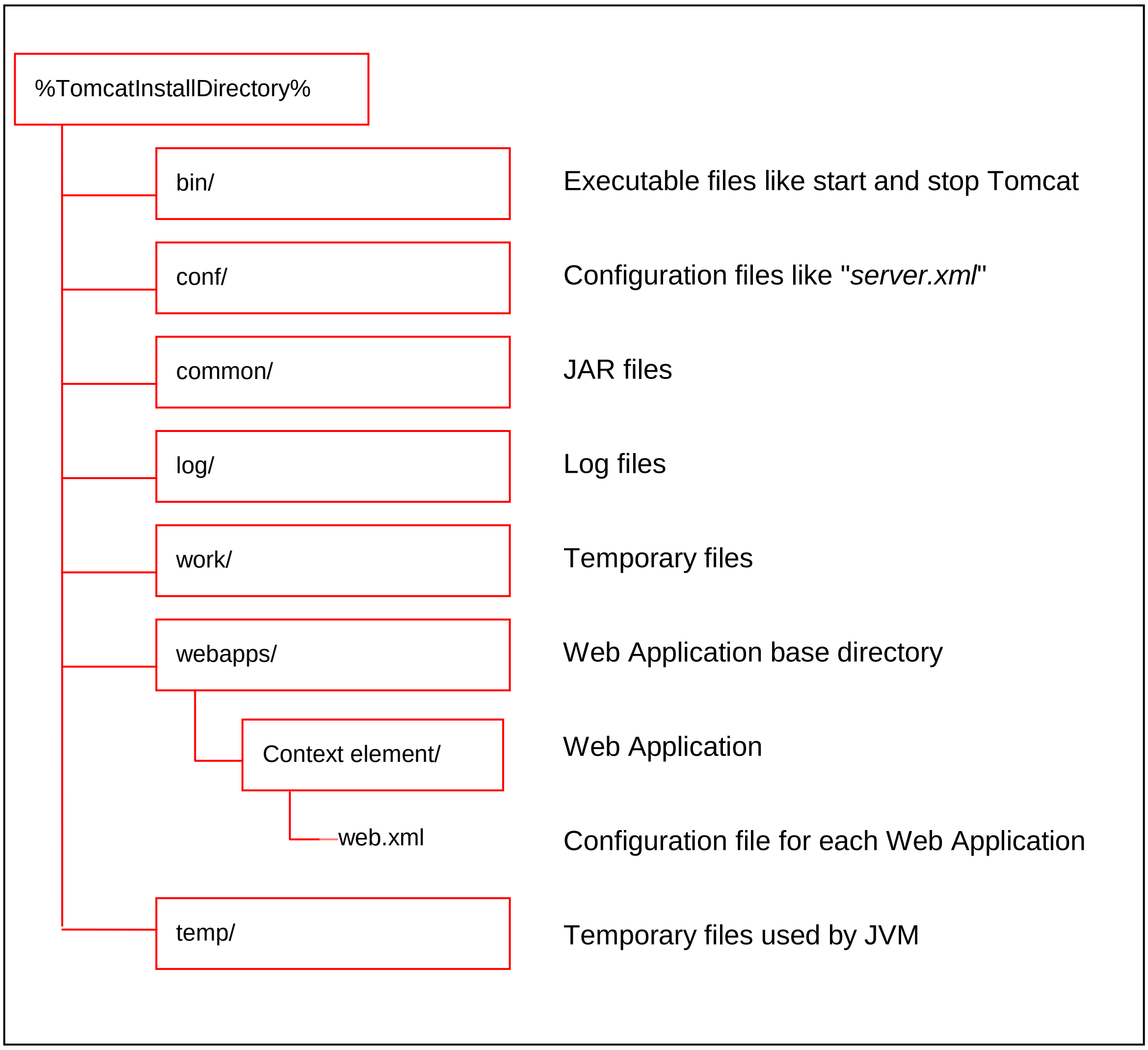

%TomcatInstallDirectory%
bin/
Executable files like start and stop Tomcat
conf/
Configuration files like "*server.xml*"
common/
JAR files
log/
Log files
work/
Temporary files
webapps/
Web Application base directory
Context element/
Web Application
web.xml
Configuration file for each Web Application
temp/
Temporary files used by JVM

Figure 8 Tomcat Directory Structure

[Web application and autoreload]

- The "*webapps/*" directory is an application base directory where Tomcat recognizes as Web Application. Any subdirectory within the *application base directory* that appears to be an unpacked web application (that is, it contains a "*/WEB-INF/web.xml*" file) will be automatically generated as Context element.

- Any XML file in the "*%TomcatInstallDir%/conf/[engine_name]/[host_name]*" directory is assumed to contain a Context element for a single web application. This application is the target of the autoreload of the Tomcat if "reloadable" is set to "true". [engine_name] is "Catalina" and [host_name] is "*localhost*" as default defined in the *"server.xml"*. In this XML file, the location of the context can be specified. IF the context directory exists other than the default context directory ("*%TomcatInstallDir%/webapps/")*, it is necessary to create XML file.

[%TomcatInstallDir%conf/Catalina/localhost/webhello.xml]

```
<Context path="/webhello" reloadable="true"
docBase="/srv/www/tomcat5/base/webapps/webhello"
workDir="/srv/www/tomcat5/base/webapps/webhello/work" />
```

1.3.6. Tomcat configuration files

Tomcat needs some configuration files for the server and for Web application.

Tomcat has the following configuration files. The content of configuration depends on Tomcat version. Please see Tomcat documentation for detail configuration.

- **server.xml**

The *server.xml* file is a configuration file for Tomcat server.

- **web.xml**

The *web.xml* file is a configuration file for each web application. It is located always under /WEB-INF directory of web application. The file always starts with <web-app> tag and ends with </web-app>. There are two ways to define Servlet from Tomcat using this file.

1) Using mapping name

The Servlet name can be registered with mapping name in the *web.xml* file so that Tomcat can locate the Servlet. The format to register in the file is as follows:

```
<web-app>
  <servlet>
       <servlet-name>HelloServlet</servlet-name>
       <servlet-class>ictti.HelloServlet</servlet-class>
   </servlet>

   <servlet-mapping>
       <servlet-name>HelloServlet</servlet-name>
       <url-pattern>/ictti.Hello</url-pattern>
   </servlet-mapping>
</web-app>
```

- <servlet-name> is the name used in this *web.xml* file.
- <servlet-class> is the class name with the package
- <url-pattern> is the pattern that is used to call this Servlet from the browser.

[how to call from the browser]

You can call the Servlet using url-pattern name instead of the real Servlet name.

```
http://localost:8080/webhello/ictti.Hello
```

2) Using the Invoker Servlet

The *Invoker Servlet* allows using anonymous *Servlet* class without registering each *Servlet* in the *web.xml* file.

```
<web-app>
    <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>
</web-app>
```

[How to call from the browser]

Any Servlet can be called from the browser by the default URL pattern "/Servlet/*".

```
http://localost:8080/webhello/servlet/ictti.HelloServlet
```

☆Each time you modify the configuration files, you have to restart Tomcat to be reflected the modifications.

1.3.7. Tomcat Common Library

To place some common libraries in the Tomcat project, there are following libraries that Tomcat automatically recognizes to look for any class libraries.

- *Tomcat_Home/*webapp/*Web_Application_Dir/WEB-INF/lib* directory

  Available only Web Application project

- *Tomcat Home/*common/lib

  Available for all Tomcat project under *Tomcat_Home*

1.3.8. Set up Eclipse Web Tools Platform

The Eclipse Web Tools Platform (WTP) Project provides Eclipse plug-in for J2EE and Web-centric application development such as Tomcat or *JBoss*. It includes both source and graphical editors with tools and APIs to support deploying, running, debugging and testing *Servlets*, JSP and XML. It also enables to run Tomcat server from Eclipse environment.

The WTP has various packages which requires EMF(Eclipse Modeling Framework), GEF(Graphical Editor Framework) and JEM(Java EMF Model) plug-in to display graphics on the Eclipse. "WTP all-in-one module" includes the complete set of software to start using WTP immediately. This package already has WTP combined with the complete set of prerequisites, eclipse sdk, EMF, GEF and JEM. Just unzip the packages and you will have Eclipse with all necessary Web tools packages installed. The download site is;

```
http://download.eclipse.org/webtools/downloads/
```

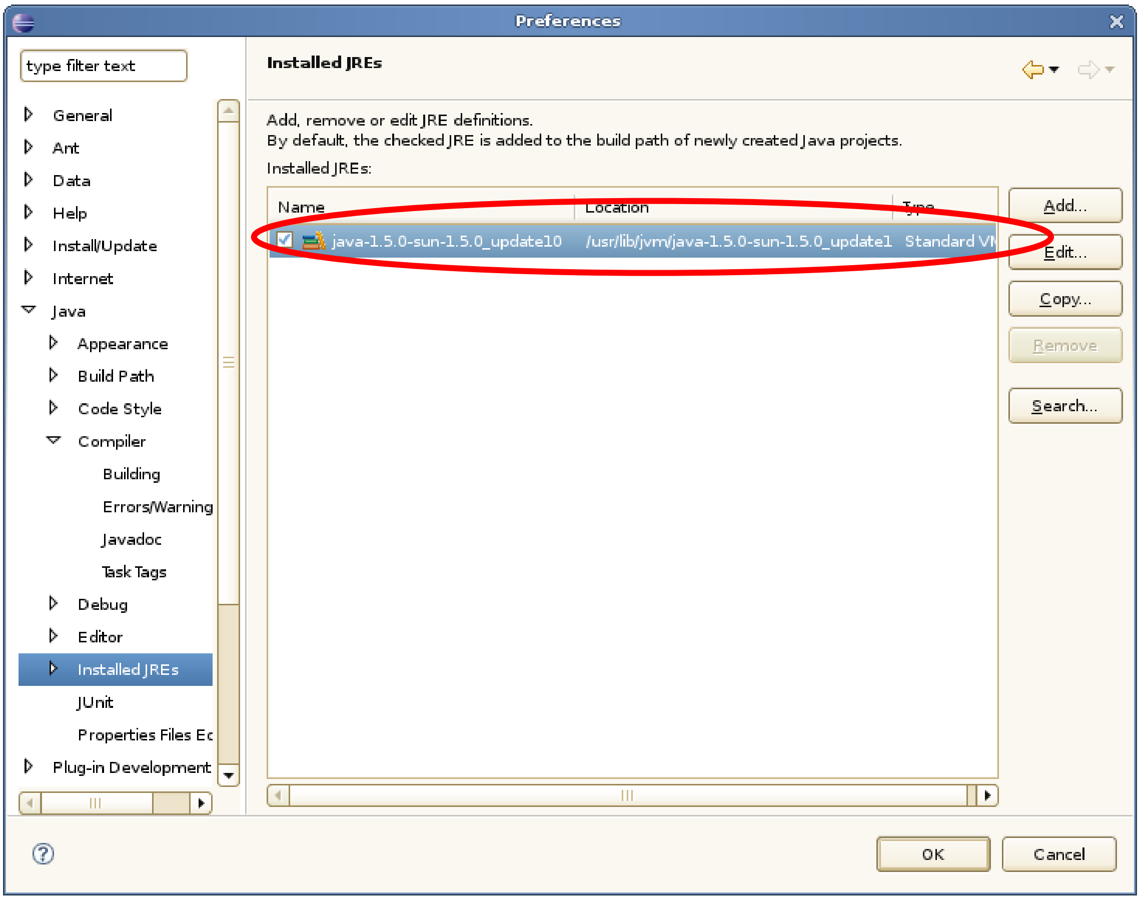
.

[How to setup WTP in Eclipse]

1) Unzip the "*wtp-all-in-one-sdk-R-1.5.2-200610261841-linux-gtk.tar.gz*" to the directory you want to have your eclipse.

2) Run Eclipse from the directory.

3) Check the Installed JRE of the eclipse environment from "Window" -> "Preferences"-> "Java" -> "Installed JREs". If the installed JDK library is not pointed out, add your installed SDK location.



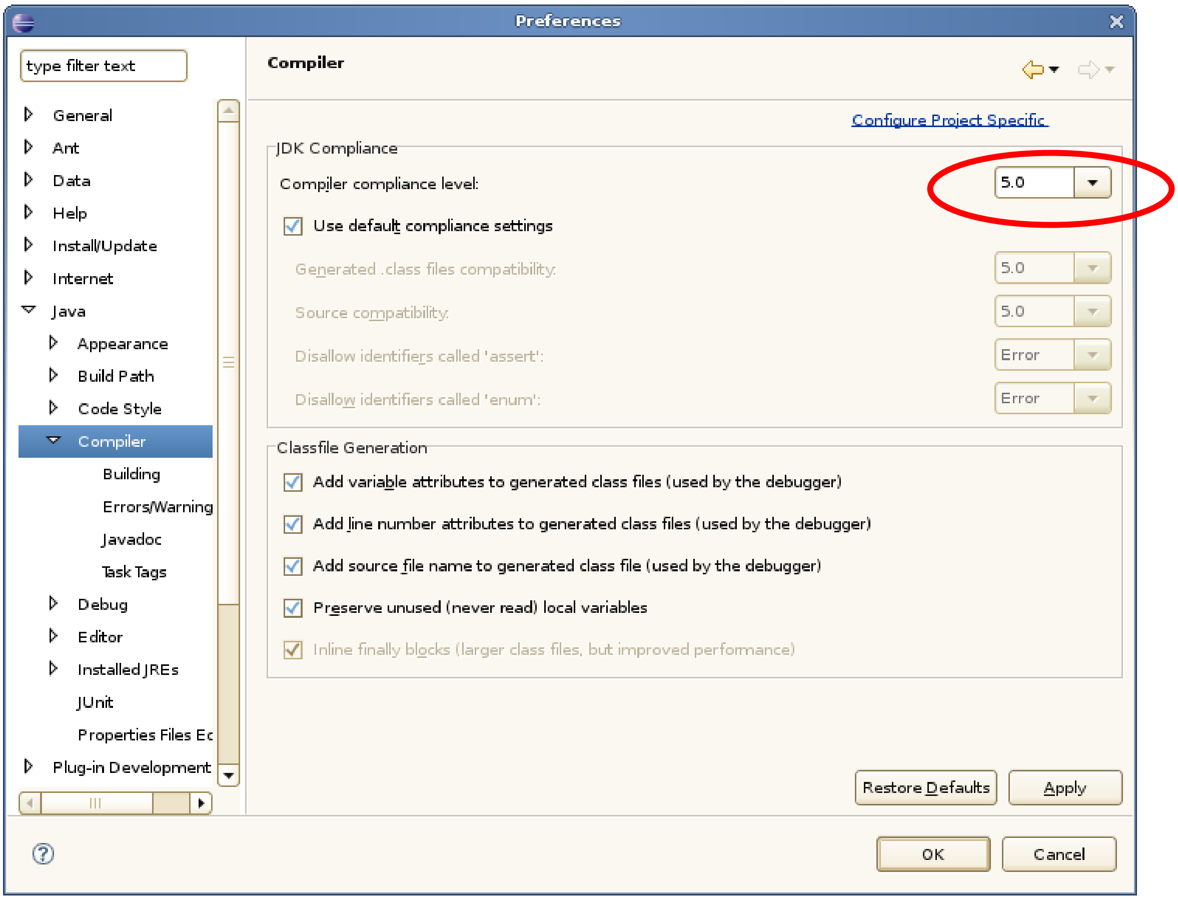
Installed JREs

Add, remove or edit JRE definitions.
By default, the checked JRE is added to the build path of newly created Java projects.
Installed JREs:

	Name	Location	Type
☑	java-1.5.0-sun-1.5.0_update10	/usr/lib/jvm/java-1.5.0-sun-1.5.0_update1	Standard VI

Figure 9 Eclipse setting for JRE

4) Check the Compiler of Eclipse environment from "Window" -> "Preferences"-> "Java" -> "Compiler". If the compiler version is not correct, change it to "5.0".



Preferences

Compiler

Configure Project Specific

JDK Compliance

Compiler compliance level: 5.0

☑ Use default compliance settings

Generated .class files compatibility: 5.0

Source compatibility: 5.0

Disallow identifiers called 'assert': Error

Disallow identifiers called 'enum': Error

Classfile Generation

☑ Add variable attributes to generated class files (used by the debugger)

☑ Add line number attributes to generated class files (used by the debugger)

☑ Add source file name to generated class file (used by the debugger)

☑ Preserve unused (never read) local variables

☑ Inline finally blocks (larger class files, but improved performance)

Figure 10 Eclipse setting for Compiler

5) Set Tomcat as Server

From menu, select "Window" -> "Preferences" -> "Server" -> "Installed Runtime" -> "Add".

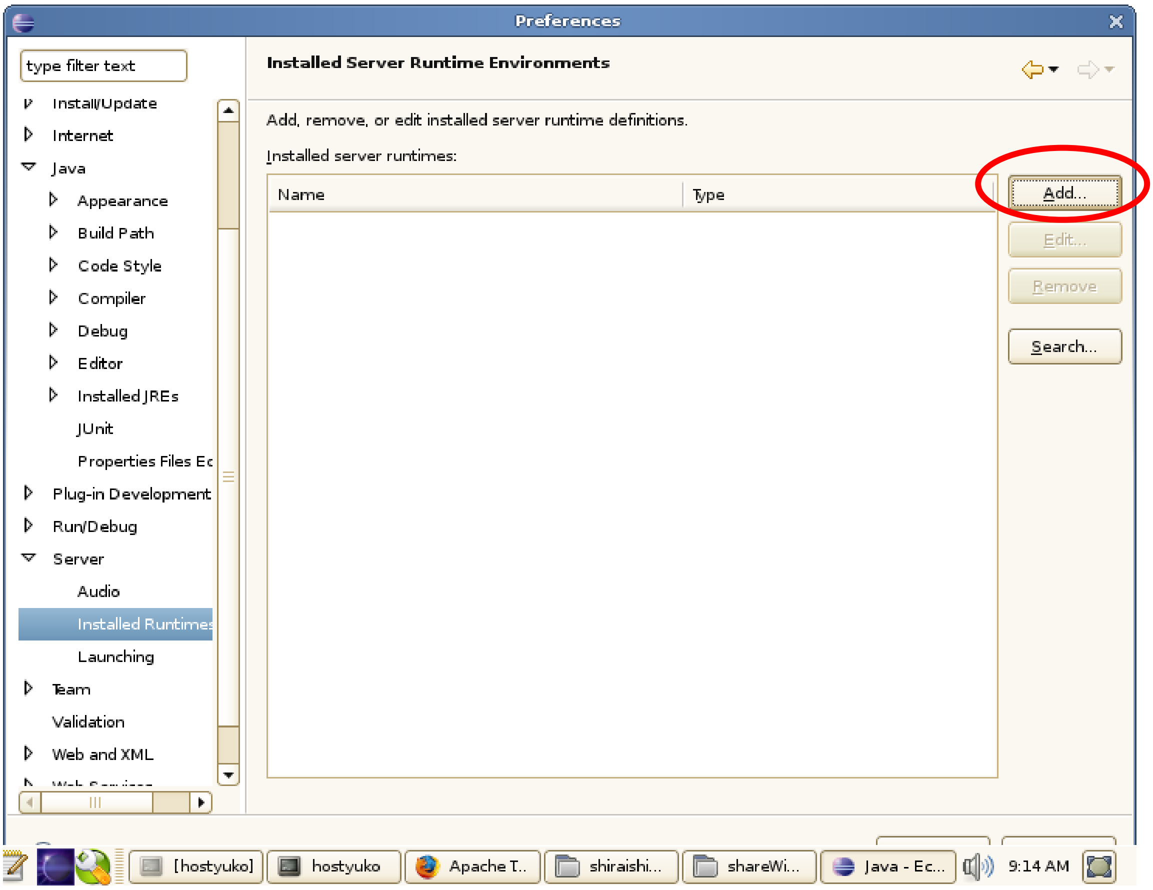
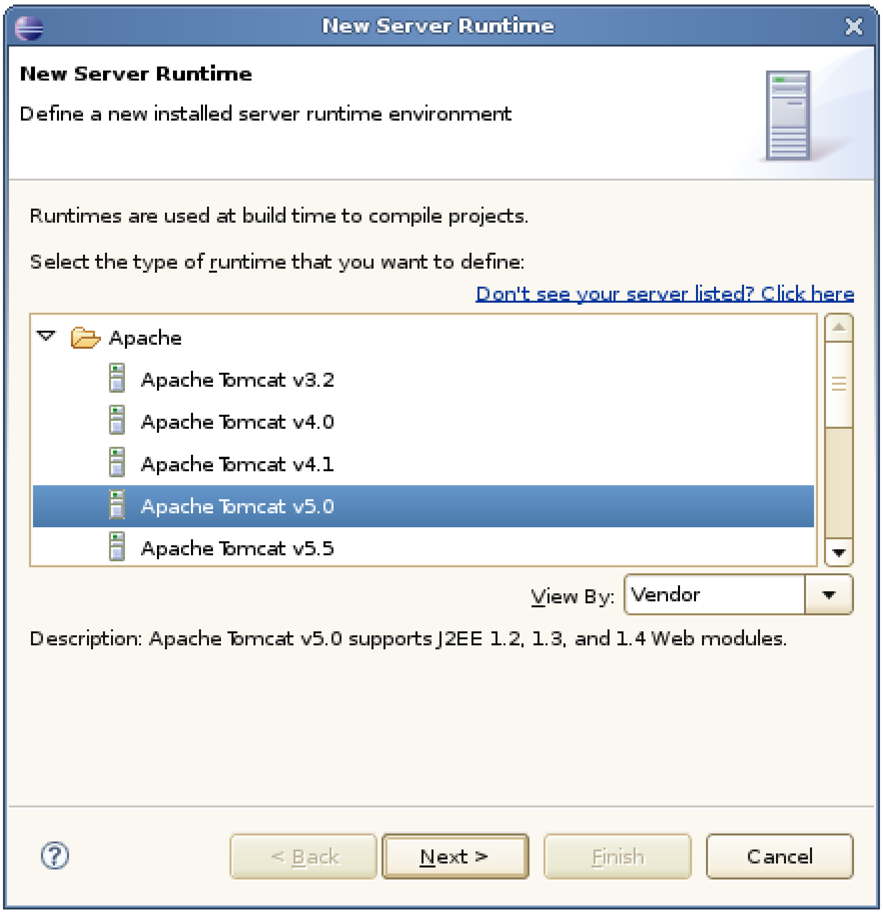


Figure 11 WTP setting Add server

6) Select the corresponding server from the New Server Runtime window.



New Server Runtime

New Server Runtime

Define a new installed server runtime environment

Runtimes are used at build time to compile projects.

Select the type of runtime that you want to define:

Don't see your server listed? Click here

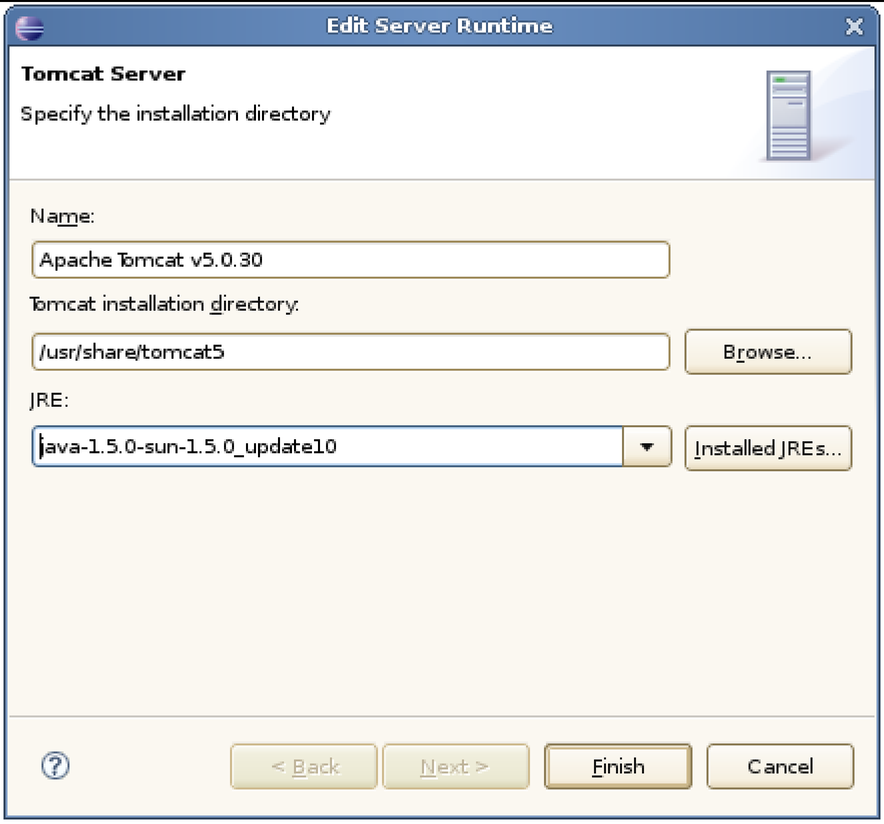
- Apache
  - Apache Tomcat v3.2
  - Apache Tomcat v4.0
  - Apache Tomcat v4.1
  - Apache Tomcat v5.0
  - Apache Tomcat v5.5

View By: Vendor

Description: Apache Tomcat v5.0 supports J2EE 1.2, 1.3, and 1.4 Web modules.

Figure 12 WTP setting for Tomcat

7) Set the Tomcat installation directory and JRE in the Edit Server Runtime Window.



Edit Server Runtime

Tomcat Server

Specify the installation directory

Name: Apache Tomcat v5.0.30

Tomcat installation directory: /usr/share/tomcat5

JRE: java-1.5.0-sun-1.5.0_update10

Figure 13 WTP setting for Server Runtime

8) From menu select "Window" -> "Show View" -> "Other" -> "Server" -> "Servers".

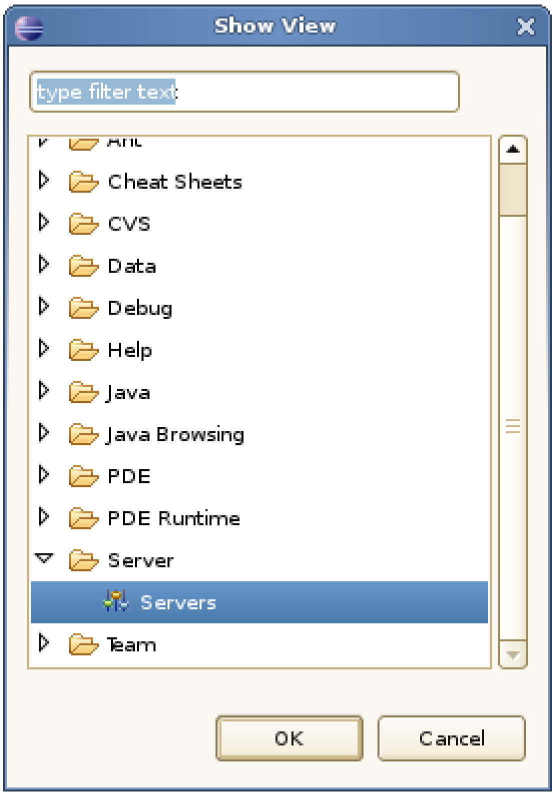
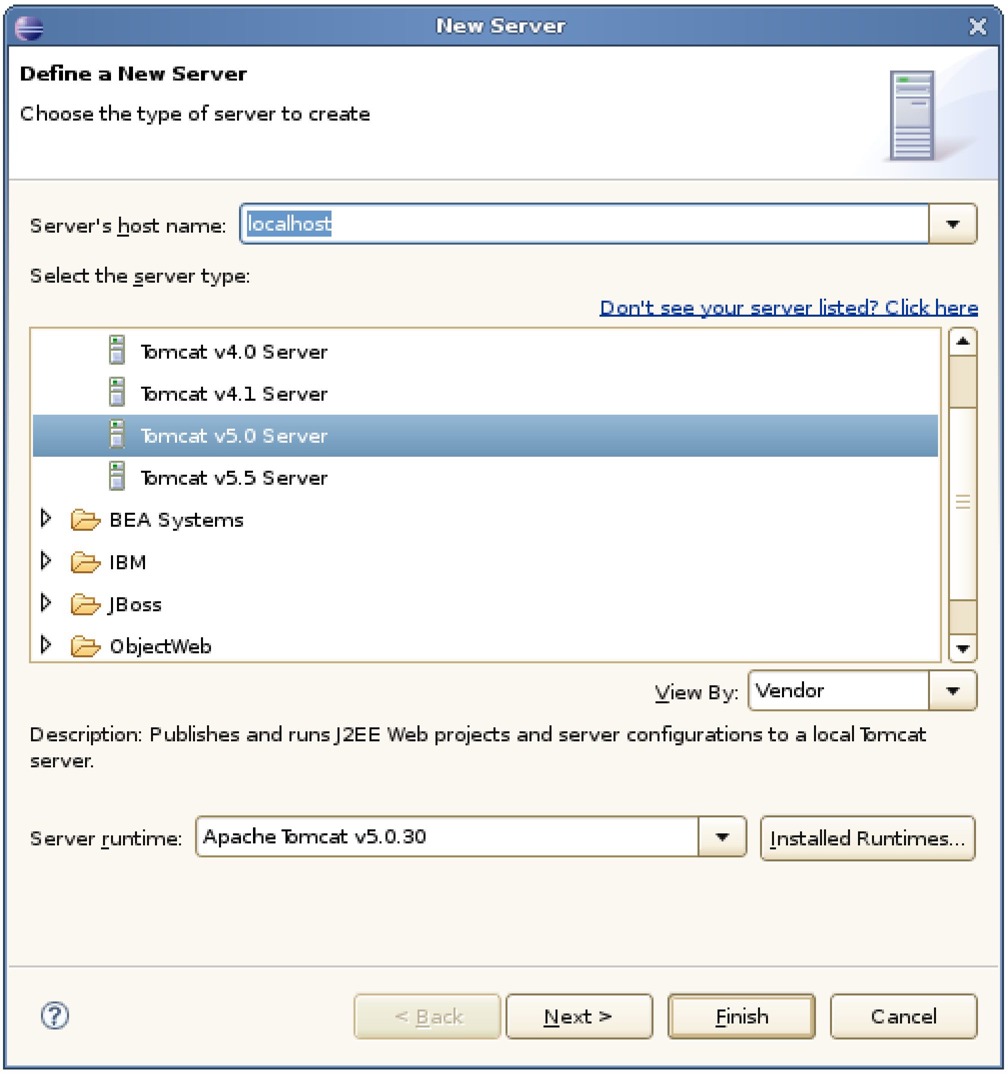


Figure 14 Eclipse show view server

9) Right click the mouse in the Server view. Select "New" -> "Server" and "Define a New Server" Window opens. Select the server you installed and click "Finish" button.



New Server

Define a New Server

Choose the type of server to create

Server's host name: localhost

Select the server type:

Don't see your server listed? Click here

- Tomcat v4.0 Server
- Tomcat v4.1 Server
- Tomcat v5.0 Server
- Tomcat v5.5 Server
- BEA Systems
- IBM
- JBoss
- ObjectWeb

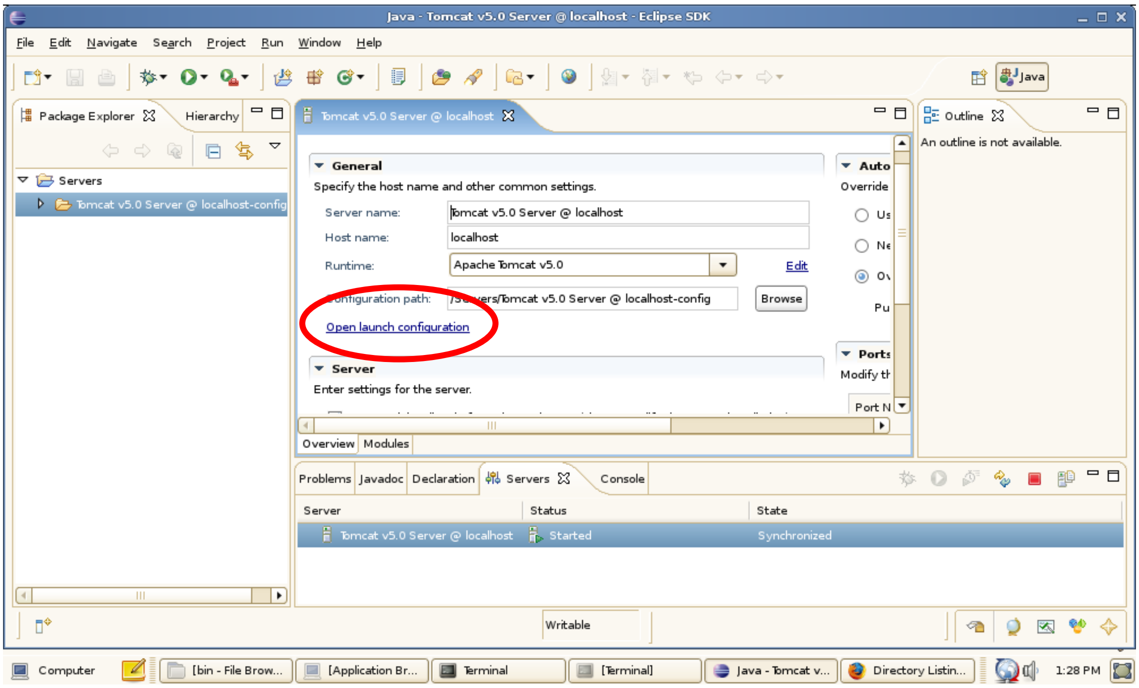
View By: Vendor

Description: Publishes and runs J2EE Web projects and server configurations to a local Tomcat server.

Server runtime: Apache Tomcat v5.0.30

Figure 15 Eclipse define new server

9) Double click the Server created above.　Click "Open launch configuration".


Java - Tomcat v5.0 Server @ localhost - Eclipse SDK
File Edit Navigate Search Project Run Window Help
Java
Package Explorer
Hierarchy
Servers
Tomcat v5.0 Server @ localhost-config
Tomcat v5.0 Server @ localhost
Outline
An outline is not available.
General
Specify the host name and other common settings.
Edit
Browse
Open launch configuration
Auto
Override
Ports
Modify th
Port N
Server
Enter settings for the server.
Overview
Modules
Problems
Javadoc
Declaration
Servers
Console
Writable
Computer
[bin - File Brow...
[Application Br...
Terminal
[Terminal]
Java - Tomcat v...
Directory Listin...
1:28 PM

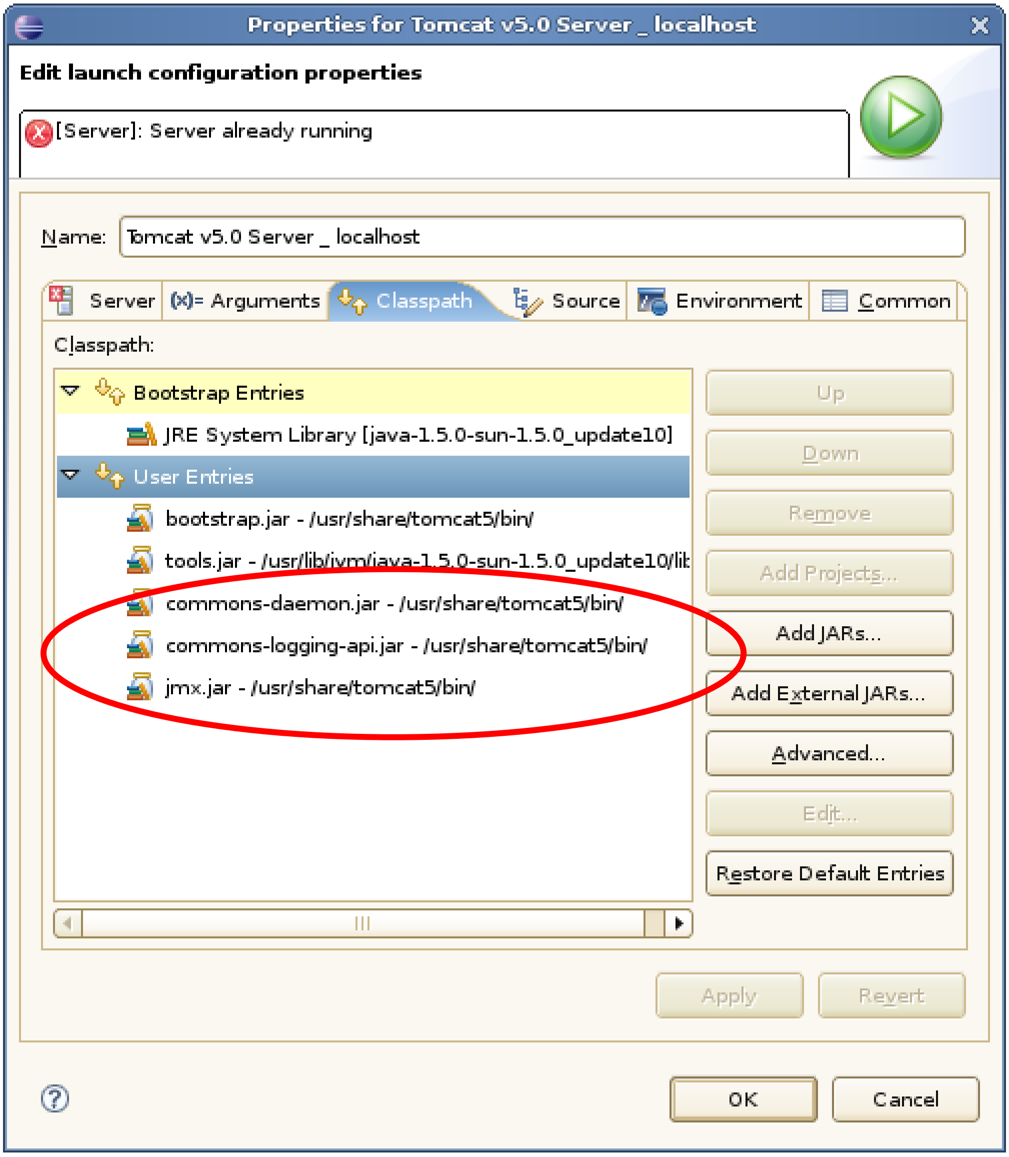
Server name: Tomcat v5.0 Server @ localhost
Host name: localhost
Runtime: Apache Tomcat v5.0
Configuration path: /Servers/Tomcat v5.0 Server @ localhost-config

Server	Status	State
Tomcat v5.0 Server @ localhost	Started	Synchronized

Figure 16 WTP Server open launch configuration

10) In the "Properties for Tomcat" window, add the following jar files clicking "Add External JARs…" in "User Entries" column of "Classpath" tab. Click "OK" button.

- %TomcatInstallDir%/bin/commons-daemon.jar
- %TomcatInstallDir%/bin/commons-logging-api.jar
- %TomcatInstallDir%/bin/jmx.jar



Properties for Tomcat v5.0 Server _ localhost

Edit launch configuration properties

[Server]: Server already running

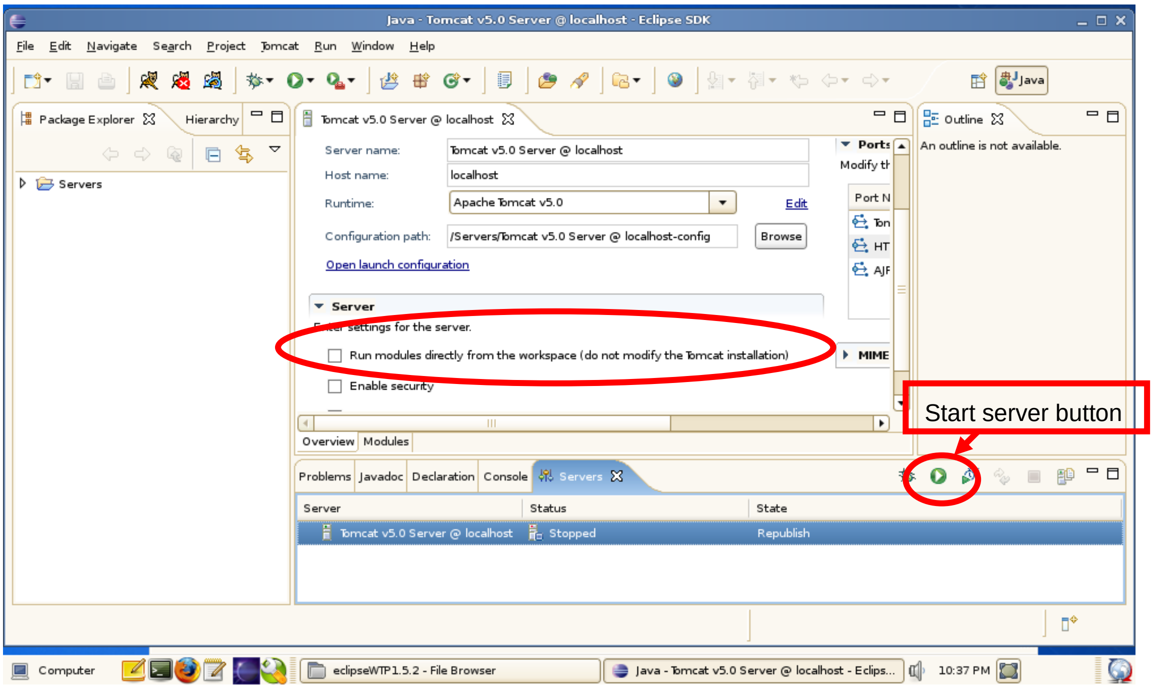
Name: Tomcat v5.0 Server _ localhost

Classpath:

- Bootstrap Entries
  - JRE System Library [java-1.5.0-sun-1.5.0_update10]
- User Entries
  - bootstrap.jar - /usr/share/tomcat5/bin/
  - tools.jar - /usr/lib/jvm/java-1.5.0-sun-1.5.0_update10/lib
  - commons-daemon.jar - /usr/share/tomcat5/bin/
  - commons-logging-api.jar - /usr/share/tomcat5/bin/
  - jmx.jar - /usr/share/tomcat5/bin/

Figure 17 WTP Tomcat classpath setting

11) Uncheck the "Run modules directly from the workspace(do not modify the Tomcat Installation)"
12) Select the server you created in the Server view, and click the start button to start the server.



Server name: Tomcat v5.0 Server @ localhost

Host name: localhost

Runtime: Apache Tomcat v5.0

Configuration path: /Servers/Tomcat v5.0 Server @ localhost-config

Server — Enter settings for the server.

□ Run modules directly from the workspace (do not modify the Tomcat installation)

□ Enable security

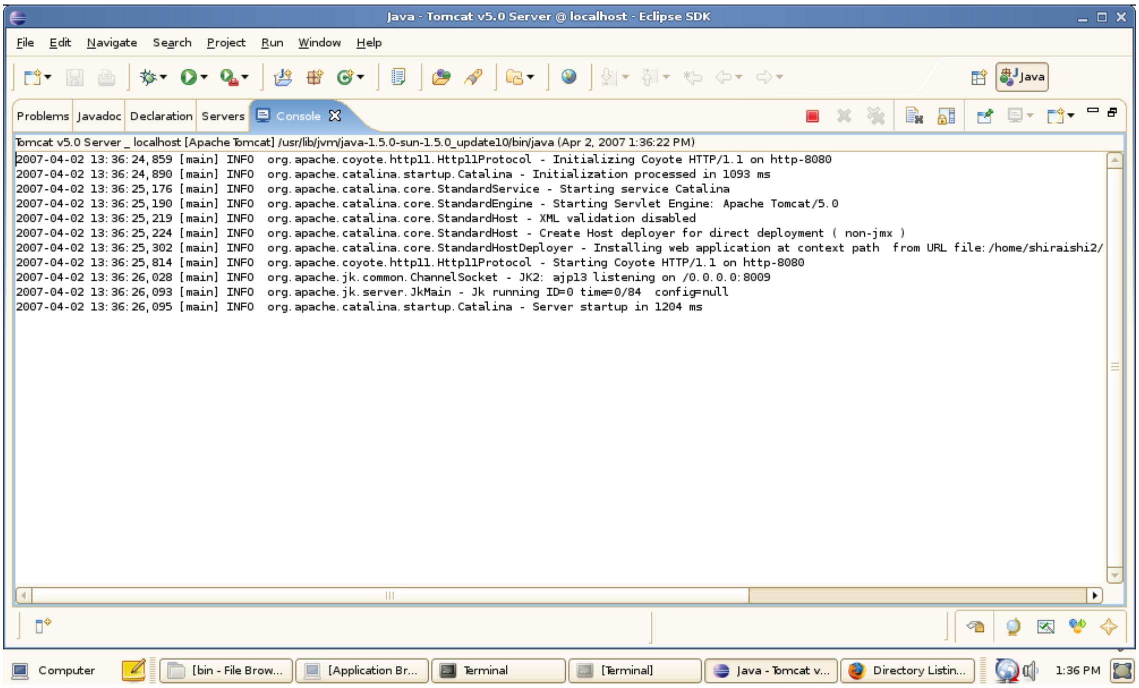
Server	Status	State
Tomcat v5.0 Server @ localhost	Stopped	Republish

Start server button

Java - Tomcat v5.0 Server @ localhost - Eclipse SDK
File Edit Navigate Search Project Tomcat Run Window Help
Package Explorer
Hierarchy
Servers
Edit
Browse
Open launch configuration
Ports
Outline
An outline is not available.
Overview
Modules
Problems Javadoc Declaration Console Servers
Computer
eclipseWTP1.5.2 - File Browser
10:37 PM

Figure 18 WTP Tomcat Server setting

13) The server starts and the message will be printed out.



Tomcat v5.0 Server _ localhost [Apache Tomcat] /usr/lib/jvm/java-1.5.0-sun-1.5.0_update10/bin/java (Apr 2, 2007 1:36:22 PM)

```
2007-04-02 13:36:24,859 [main] INFO  org.apache.coyote.http11.Http11Protocol - Initializing Coyote HTTP/1.1 on http-8080
2007-04-02 13:36:24,890 [main] INFO  org.apache.catalina.startup.Catalina - Initialization processed in 1093 ms
2007-04-02 13:36:25,176 [main] INFO  org.apache.catalina.core.StandardService - Starting service Catalina
2007-04-02 13:36:25,190 [main] INFO  org.apache.catalina.core.StandardEngine - Starting Servlet Engine: Apache Tomcat/5.0
2007-04-02 13:36:25,219 [main] INFO  org.apache.catalina.core.StandardHost - XML validation disabled
2007-04-02 13:36:25,224 [main] INFO  org.apache.catalina.core.StandardHost - Create Host deployer for direct deployment ( non-jmx )
2007-04-02 13:36:25,302 [main] INFO  org.apache.catalina.core.StandardHostDeployer - Installing web application at context path  from URL file:/home/shiraishi2/
2007-04-02 13:36:25,814 [main] INFO  org.apache.coyote.http11.Http11Protocol - Starting Coyote HTTP/1.1 on http-8080
2007-04-02 13:36:26,028 [main] INFO  org.apache.jk.common.ChannelSocket - JK2: ajp13 listening on /0.0.0.0:8009
2007-04-02 13:36:26,093 [main] INFO  org.apache.jk.server.JkMain - Jk running ID=0 time=0/84  config=null
2007-04-02 13:36:26,095 [main] INFO  org.apache.catalina.startup.Catalina - Server startup in 1204 ms
```

Java - Tomcat v5.0 Server @ localhost - Eclipse SDK
File Edit Navigate Search Project Run Window Help
Problems Javadoc Declaration Servers Console
Computer
[bin - File Brow...
[Application Br...
Terminal
[Terminal]
Java - Tomcat v...
Directory Listin...
1:36 PM

Figure 19 WTP Tomcat server start message

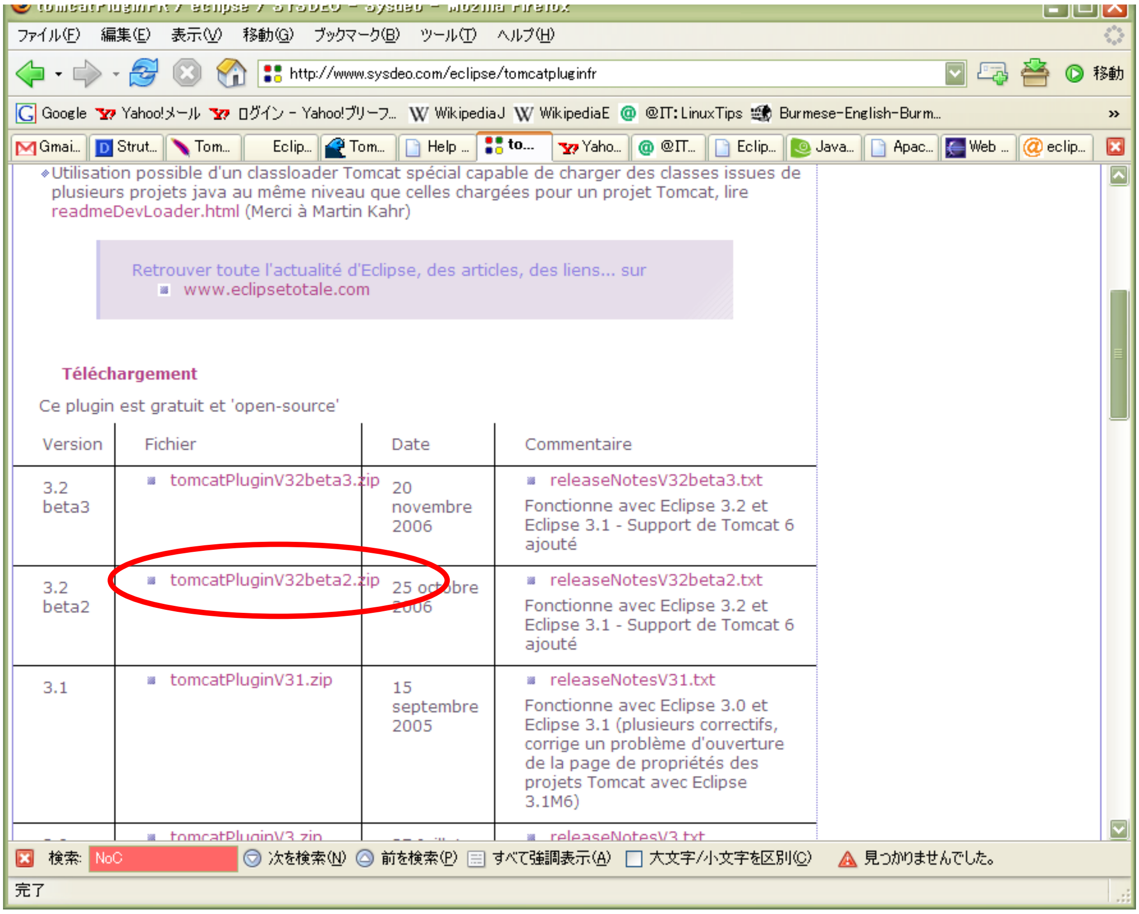
☆ If you have problem of starting Tomcat with WTP, it is possibly due to file access permission. WTP starts the Tomcat server with your username. The following permissions are required.

- Write permission in "backup" directory under Tomcat directory.
- Write permission in "conf" directory under Tomcat directory

1.3.9. Install Sysdeo Eclipse Tomcat Launcher plugin

Sysdeo Eclipse Tomcat Launcher is a Tomcat plugin which enables Tomcat server start, stop, restart and Web application development. WTP provides integrated Web application development not only Tomcat but other J2EE servers, while Sysdeo Eclipse offers the Tomcat limited development environment.

1. Download Tomcat-Eclipse plugin tomcatPluginV32beta2.zip from;
   http://sysdeo.com/eclipse/tomcatplugin


ファイル(F) 編集(E) 表示(V) 移動(G) ブックマーク(B) ツール(T) ヘルプ(H)
http://www.sysdeo.com/eclipse/tomcatpluginfr
移動
Google Yahoo!メール ログイン - Yahoo!ブリーフ... WikipediaJ WikipediaE @IT:LinuxTips Burmese-English-Burm...
Gmai... Strut... Tom... Eclip... Tom... Help ... to... Yaho... @IT... Eclip... Java... Apac... Web ... eclip...

Utilisation possible d'un classloader Tomcat spécial capable de charger des classes issues de plusieurs projets java au même niveau que celles chargées pour un projet Tomcat, lire readmeDevLoader.html (Merci à Martin Kahr)

Retrouver toute l'actualité d'Eclipse, des articles, des liens... sur
www.eclipsetotale.com

Téléchargement

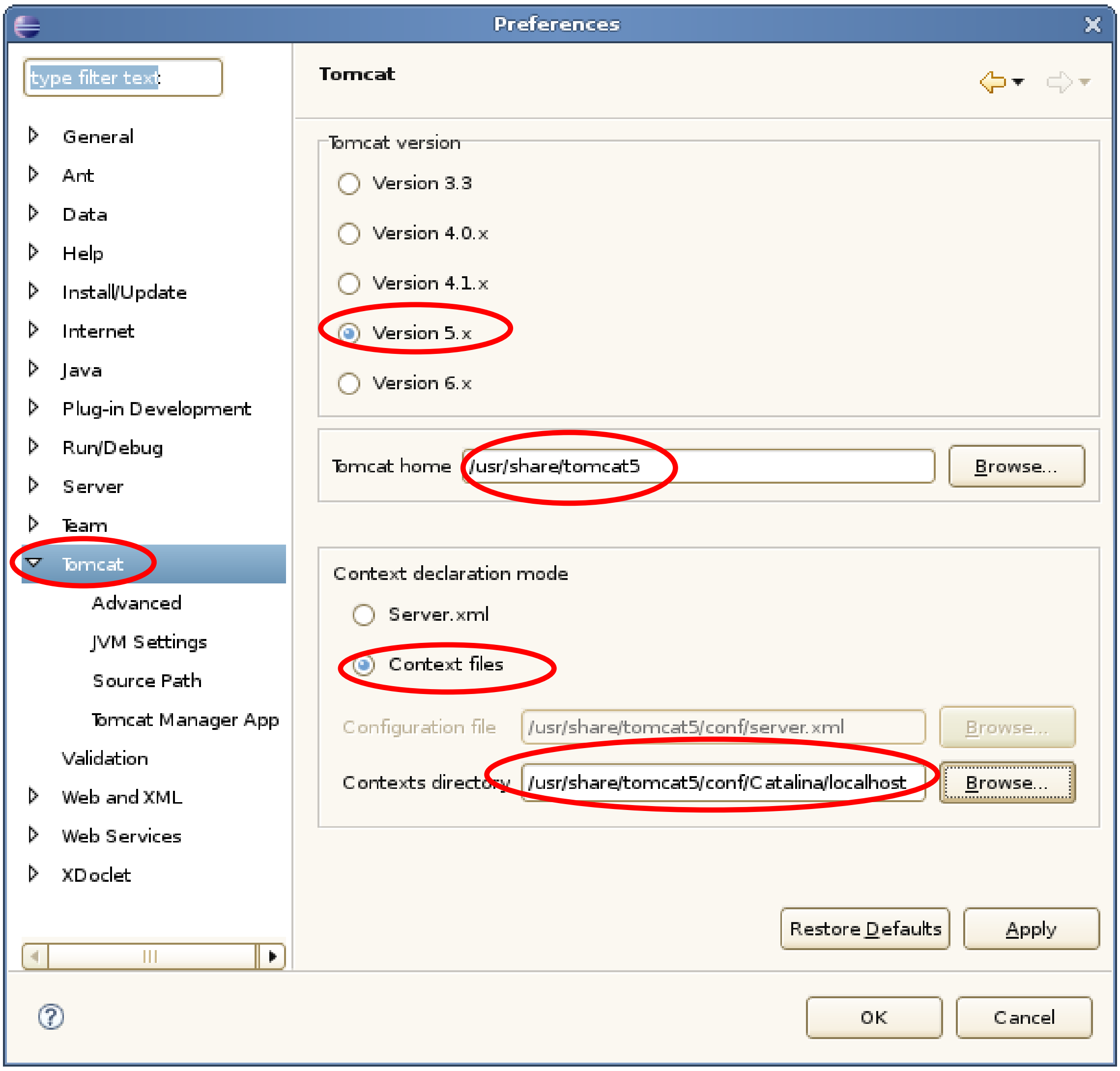
Ce plugin est gratuit et 'open-source'

Version	Fichier	Date	Commentaire
3.2 beta3	tomcatPluginV32beta3.zip	20 novembre 2006	releaseNotesV32beta3.txt Fonctionne avec Eclipse 3.2 et Eclipse 3.1 - Support de Tomcat 6 ajouté
3.2 beta2	tomcatPluginV32beta2.zip	25 octobre 2006	releaseNotesV32beta2.txt Fonctionne avec Eclipse 3.2 et Eclipse 3.1 - Support de Tomcat 6 ajouté
3.1	tomcatPluginV31.zip	15 septembre 2005	releaseNotesV31.txt Fonctionne avec Eclipse 3.0 et Eclipse 3.1 (plusieurs correctifs, corrige un problème d'ouverture de la page de propriétés des projets Tomcat avec Eclipse 3.1M6)
	tomcatPluginV3.zip		releaseNotesV3.txt

検索: NoC 次を検索(N) 前を検索(P) すべて強調表示(A) 大文字/小文字を区別(C) 見つかりませんでした。
完了

Figure 20 Sysdeo Tomcat plugin download site

2. Unzip the downloaded file and extract them to *ECLIPSE_HOME \plugins* (copy whole directory of com.sysdeo.eclipse.Tomcat_3.2.0.beta2).
3. Start Eclipse.
4. From the menu, select Window -> Preferences -> Tomcat. Set Tomcat version, Tomcat home. Choose Context files for Tomcat5.X and "choose Context" files for Tomcat5.X.



Tomcat

Tomcat version
○ Version 3.3
○ Version 4.0.x
○ Version 4.1.x
● Version 5.x
○ Version 6.x

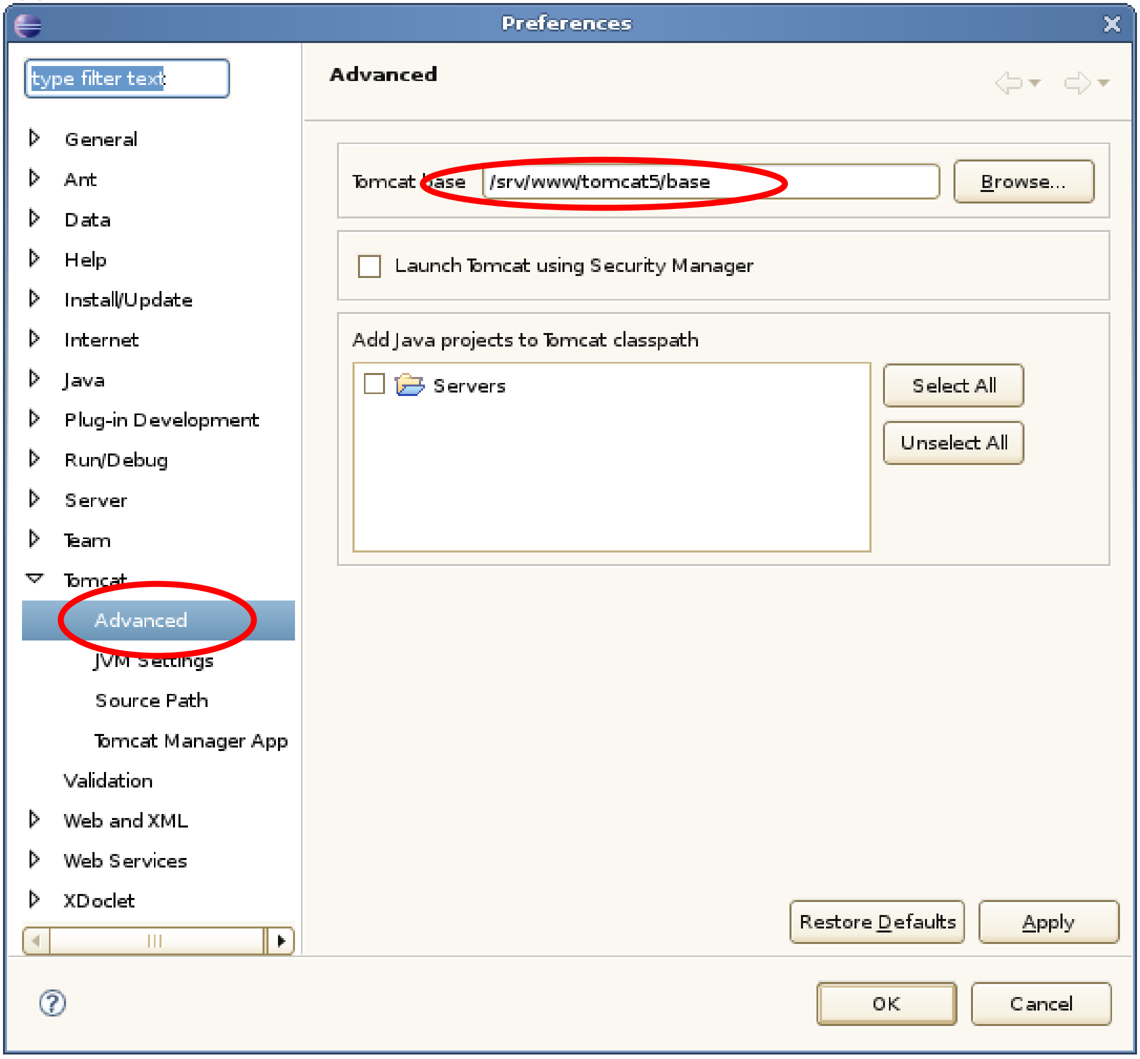
Tomcat home /usr/share/tomcat5

Context declaration mode
○ Server.xml
● Context files

Configuration file /usr/share/tomcat5/conf/server.xml
Contexts directory /usr/share/tomcat5/conf/Catalina/localhost

Figure 21 Sysdeo Tomcat Plugin setting

5. In the "Tomcat Advanced" tag set Tomcat base.



Advanced

Tomcat base /srv/www/tomcat5/base

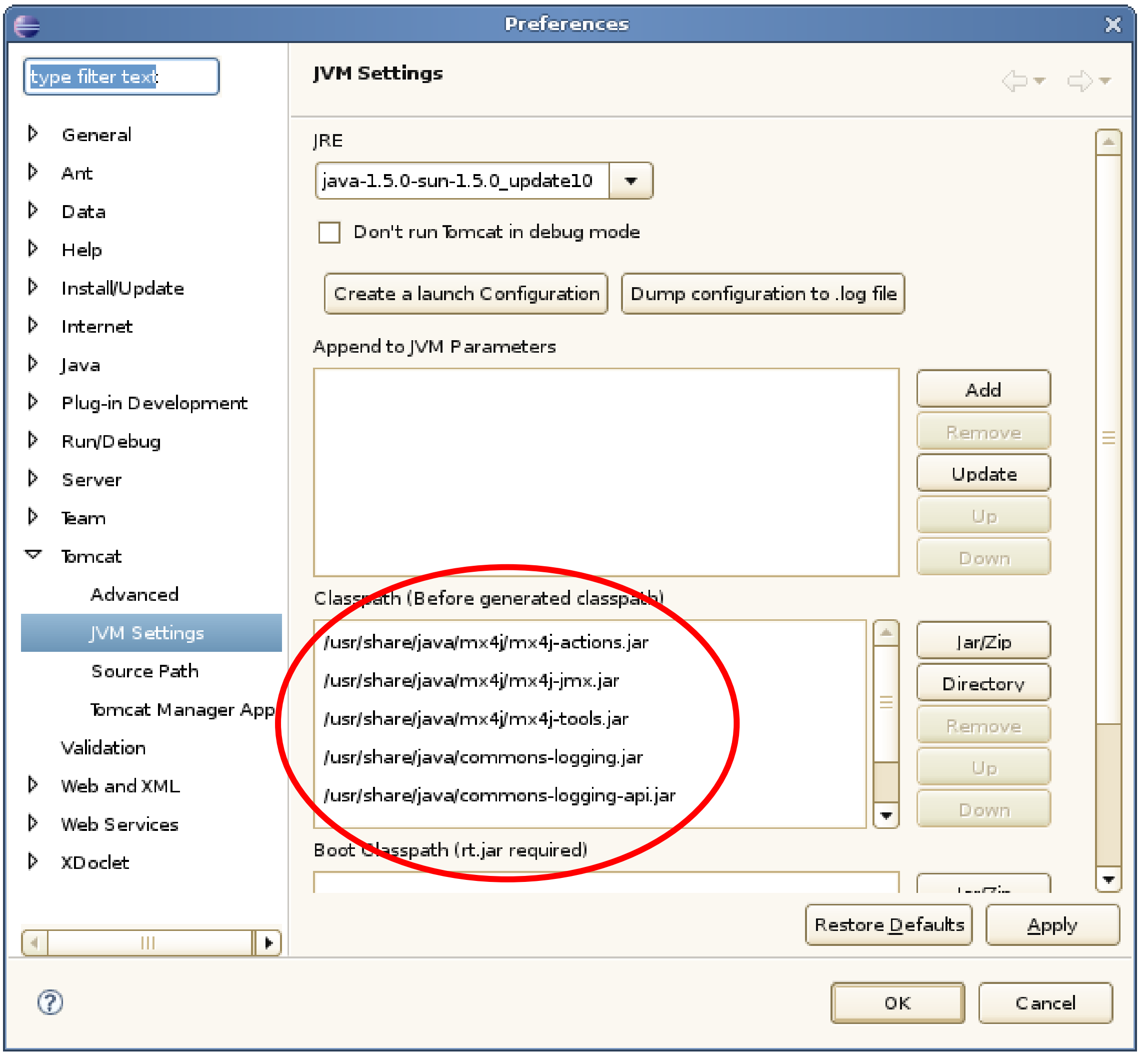
□ Launch Tomcat using Security Manager

Add Java projects to Tomcat classpath

□ Servers

Figure 22 Sysdeo Tomcat plugin advanced setting

5. In the "JVMSettings" tag, set JRE and the following Classpath.
   - /usr/share/java/mx4j/mx4j-actions.jar
   - /usr/share/java/mx4jjmx4j-jmx.jar
   - /usr/share/java/mx4j/mx4j-tools.jar
   - /usr/share/java/commons-logging.jar
   - /usr/share/java/commons-logging-api.jar
   - /usr/share/tomcat5/common/classes
   - /usr/share/java/log4j.jar



JVM Settings

JRE
java-1.5.0-sun-1.5.0_update10

☐ Don't run Tomcat in debug mode

Append to JVM Parameters

Classpath (Before generated classpath)

```
/usr/share/java/mx4j/mx4j-actions.jar
/usr/share/java/mx4j/mx4j-jmx.jar
/usr/share/java/mx4j/mx4j-tools.jar
/usr/share/java/commons-logging.jar
/usr/share/java/commons-logging-api.jar
```

Boot Classpath (rt.jar required)

Figure 23 Sysdeo Tomcat plugin Classpath setting

6. Edit /etc/sysconfig/j2ee with editor and change "TOMCAT_BASE_USER" to your user name and save it.

   For example if you login as "student";

```
TOMCAT_BASE_USER="student"
```

7. Execute the following command as "root". This changes the file owner to your user name to be able to start/stop Tomcat service from Tomcat-Eclipse plugin.

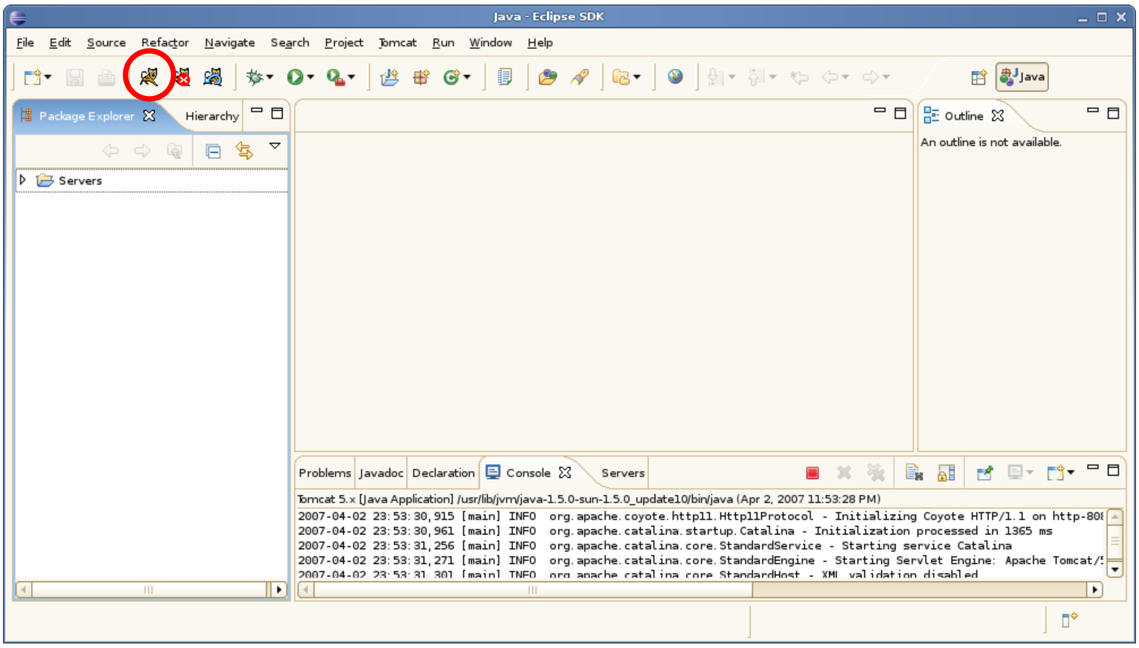
```
rctomcat5 start; sleep 30; rctomcat5 stop;
```

☆ In order to start/stop/restart Tomcat server from service, only the owner of the files is allowed to execute them. In /etc/sysconfig/j2ee, the variables *TOMCAT_BASE_USER* and

TOMCAT_BASE_GROUP are defined who has permission to start, stop and use Tomcat files.

It is required that variables *TOMCAT_BASE_USER* and *TOMCAT_BASE_GROUP in configuration file "/etc/sysconfig/j2ee"* should be the user/group names of the user who will use eclipse with the tomcat5 plugin. Executing the "rctomcat5" service command changes the Tomcat file owner (in "server" directory) according to "*/etc/sysconfig/j2ee*" variables. You are required to execute with "root". See "/usr/share/doc/packages/tomcat/README.SuSE" for more information.

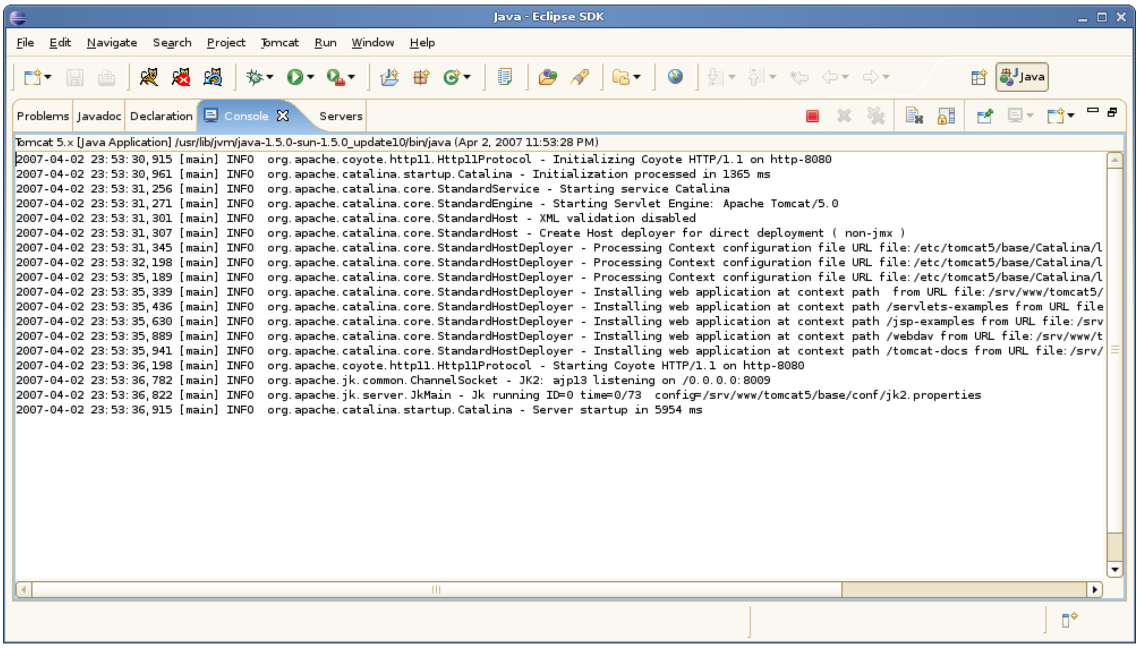
8. You have finished the configuration for Tomcat. Now you can start Tomcat clicking on Tomcat start button on the menu. You can also select from the menu Tomcat-> Start Tomcat.



```
Tomcat 5.x [Java Application] /usr/lib/jvm/java-1.5.0-sun-1.5.0_update10/bin/java (Apr 2, 2007 11:53:28 PM)
2007-04-02 23:53:30,915 [main] INFO  org.apache.coyote.http11.Http11Protocol - Initializing Coyote HTTP/1.1 on http-808
2007-04-02 23:53:30,961 [main] INFO  org.apache.catalina.startup.Catalina - Initialization processed in 1365 ms
2007-04-02 23:53:31,256 [main] INFO  org.apache.catalina.core.StandardService - Starting service Catalina
2007-04-02 23:53:31,271 [main] INFO  org.apache.catalina.core.StandardEngine - Starting Servlet Engine: Apache Tomcat/5
2007-04-02 23:53:31,301 [main] INFO  org.apache.catalina.core.StandardHost - XML validation disabled
```

Figure 24 Sysdeo Tomcat Start

9. On the Eclipse Console appears Tomcat output log.


Java - Eclipse SDK
File Edit Navigate Search Project Tomcat Run Window Help
Java
Problems Javadoc Declaration Console Servers

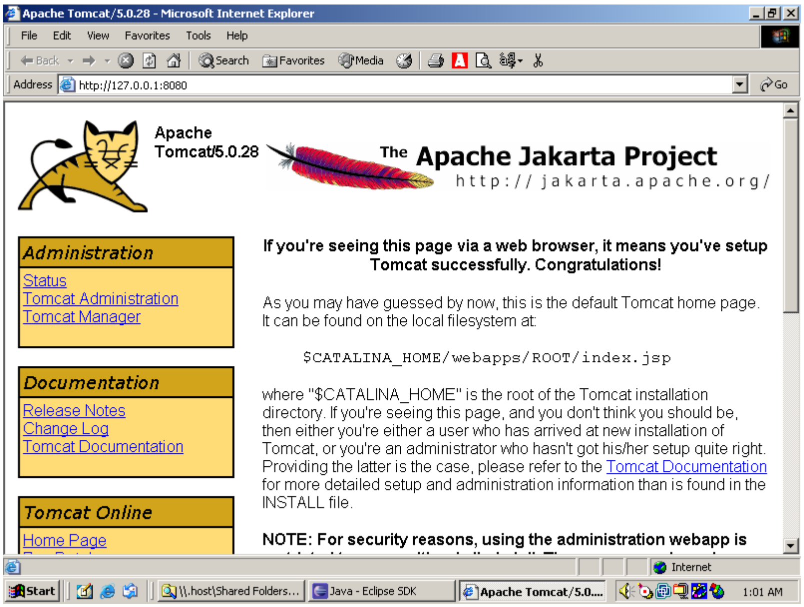
```
Tomcat 5.x [Java Application] /usr/lib/jvm/java-1.5.0-sun-1.5.0_update10/bin/java (Apr 2, 2007 11:53:28 PM)
2007-04-02 23:53:30,915 [main] INFO  org.apache.coyote.http11.Http11Protocol - Initializing Coyote HTTP/1.1 on http-8080
2007-04-02 23:53:30,961 [main] INFO  org.apache.catalina.startup.Catalina - Initialization processed in 1365 ms
2007-04-02 23:53:31,256 [main] INFO  org.apache.catalina.core.StandardService - Starting service Catalina
2007-04-02 23:53:31,271 [main] INFO  org.apache.catalina.core.StandardEngine - Starting Servlet Engine: Apache Tomcat/5.0
2007-04-02 23:53:31,301 [main] INFO  org.apache.catalina.core.StandardHost - XML validation disabled
2007-04-02 23:53:31,307 [main] INFO  org.apache.catalina.core.StandardHost - Create Host deployer for direct deployment ( non-jmx )
2007-04-02 23:53:31,345 [main] INFO  org.apache.catalina.core.StandardHostDeployer - Processing Context configuration file URL file:/etc/tomcat5/base/Catalina/l
2007-04-02 23:53:32,198 [main] INFO  org.apache.catalina.core.StandardHostDeployer - Processing Context configuration file URL file:/etc/tomcat5/base/Catalina/l
2007-04-02 23:53:35,189 [main] INFO  org.apache.catalina.core.StandardHostDeployer - Processing Context configuration file URL file:/etc/tomcat5/base/Catalina/l
2007-04-02 23:53:35,339 [main] INFO  org.apache.catalina.core.StandardHostDeployer - Installing web application at context path  from URL file:/srv/www/tomcat5/
2007-04-02 23:53:35,436 [main] INFO  org.apache.catalina.core.StandardHostDeployer - Installing web application at context path /servlets-examples from URL file
2007-04-02 23:53:35,630 [main] INFO  org.apache.catalina.core.StandardHostDeployer - Installing web application at context path /jsp-examples from URL file:/srv
2007-04-02 23:53:35,889 [main] INFO  org.apache.catalina.core.StandardHostDeployer - Installing web application at context path /webdav from URL file:/srv/www/t
2007-04-02 23:53:35,941 [main] INFO  org.apache.catalina.core.StandardHostDeployer - Installing web application at context path /tomcat-docs from URL file:/srv/
2007-04-02 23:53:36,198 [main] INFO  org.apache.coyote.http11.Http11Protocol - Starting Coyote HTTP/1.1 on http-8080
2007-04-02 23:53:36,782 [main] INFO  org.apache.jk.common.ChannelSocket - JK2: ajp13 listening on /0.0.0.0:8009
2007-04-02 23:53:36,822 [main] INFO  org.apache.jk.server.JkMain - Jk running ID=0 time=0/73  config=/srv/www/tomcat5/base/conf/jk2.properties
2007-04-02 23:53:36,915 [main] INFO  org.apache.catalina.startup.Catalina - Server startup in 5954 ms
```

Figure 25 Sysdeo Tomcat start message

10. Open the browser and access to the following URL;

```
http://127.0.0.1:8080
```

11. If the Tomcat first page appears, the Eclipse-Tomcat plugin has been set correctly.


Apache Tomcat/5.0.28 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://127.0.0.1:8080 Go
Apache Tomcat/5.0.28
The Apache Jakarta Project
http://jakarta.apache.org/
Internet
Start \\.host\Shared Folders... Java - Eclipse SDK Apache Tomcat/5.0.... 1:01 AM

Administration
Status
Tomcat Administration
Tomcat Manager

Documentation
Release Notes
Change Log
Tomcat Documentation

Tomcat Online
Home Page

If you're seeing this page via a web browser, it means you've setup Tomcat successfully. Congratulations!

As you may have guessed by now, this is the default Tomcat home page. It can be found on the local filesystem at:

$CATALINA_HOME/webapps/ROOT/index.jsp

where "$CATALINA_HOME" is the root of the Tomcat installation directory. If you're seeing this page, and you don't think you should be, then either you're either a user who has arrived at new installation of Tomcat, or you're an administrator who hasn't got his/her setup quite right. Providing the latter is the case, please refer to the Tomcat Documentation for more detailed setup and administration information than is found in the INSTALL file.

NOTE: For security reasons, using the administration webapp is

Figure 26 Tomcat first page

☆ In production phase it is strongly recommended to define some Tomcat user and set related access permission from the security point of view, but if it is used limitedly just for development or test purpose, you can change the file owner to your own user name.

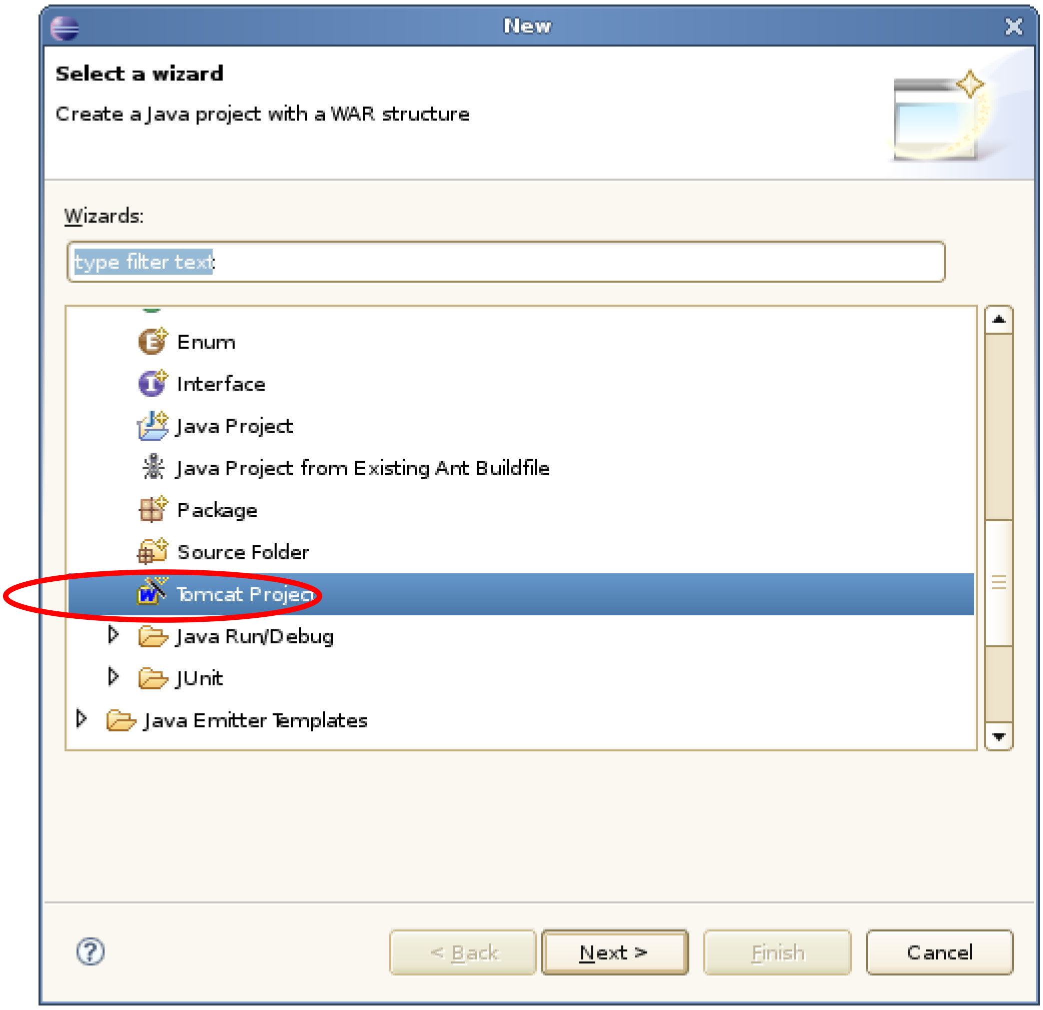
Exercise 1: Hello Servlet

1. Create a *"webhello"* project following the instruction.

[Software versions]

```
Suse Linux 10.2
Tomcat 5.0.30
Eclipse 3.2
Tomcat-Eclipse plugin V32
```

1) Run eclipse in the eclipse directory
2) Select from menu "File"-> "New"-> "Project"
3) Select Tomcat Project from Java folder and click "Next".



Select a wizard

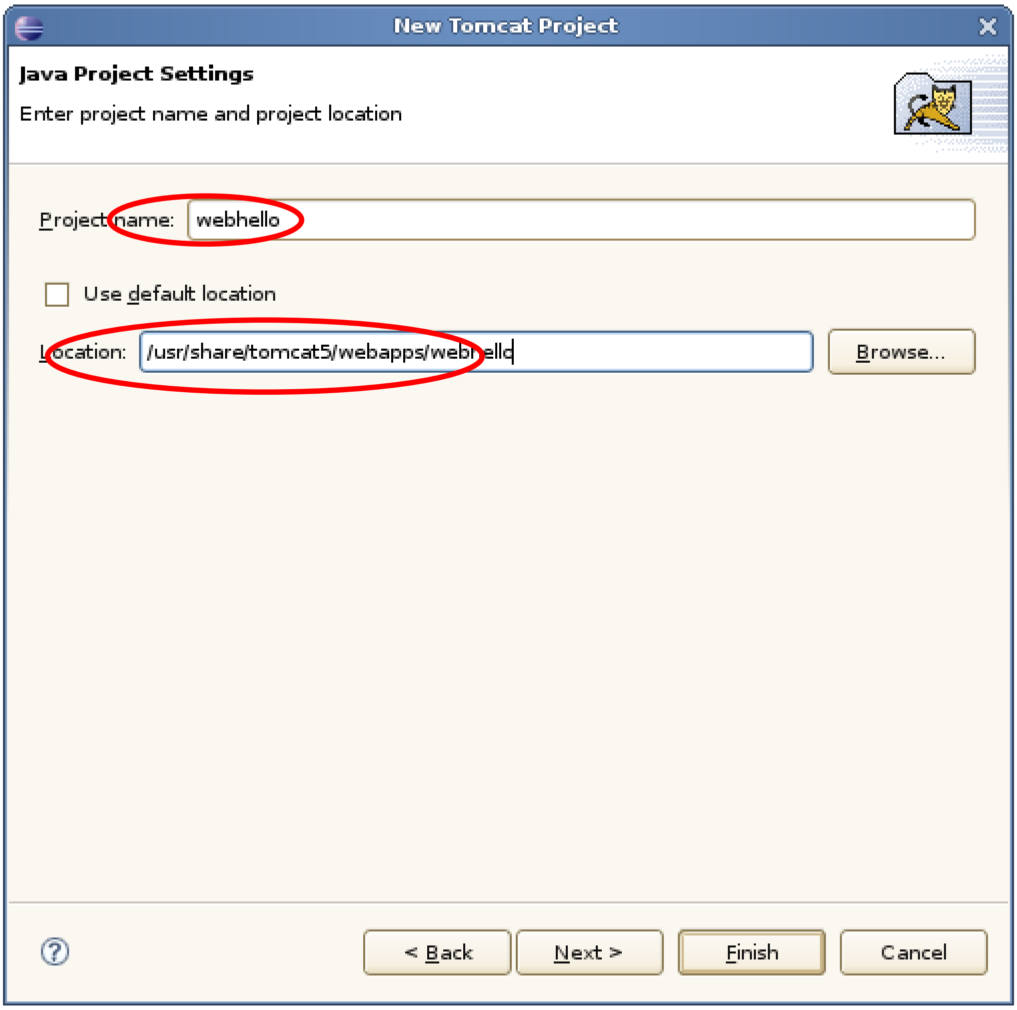
Create a Java project with a WAR structure

Wizards:

type filter text

Figure 27 Hello exercise New Project

4) Set project name as "webhello". Set also the Tomcat *webapps* directory and the project name in the "Directory" text box. Do not forget to set until the web directory name in the "Location".



New Tomcat Project

Java Project Settings

Enter project name and project location

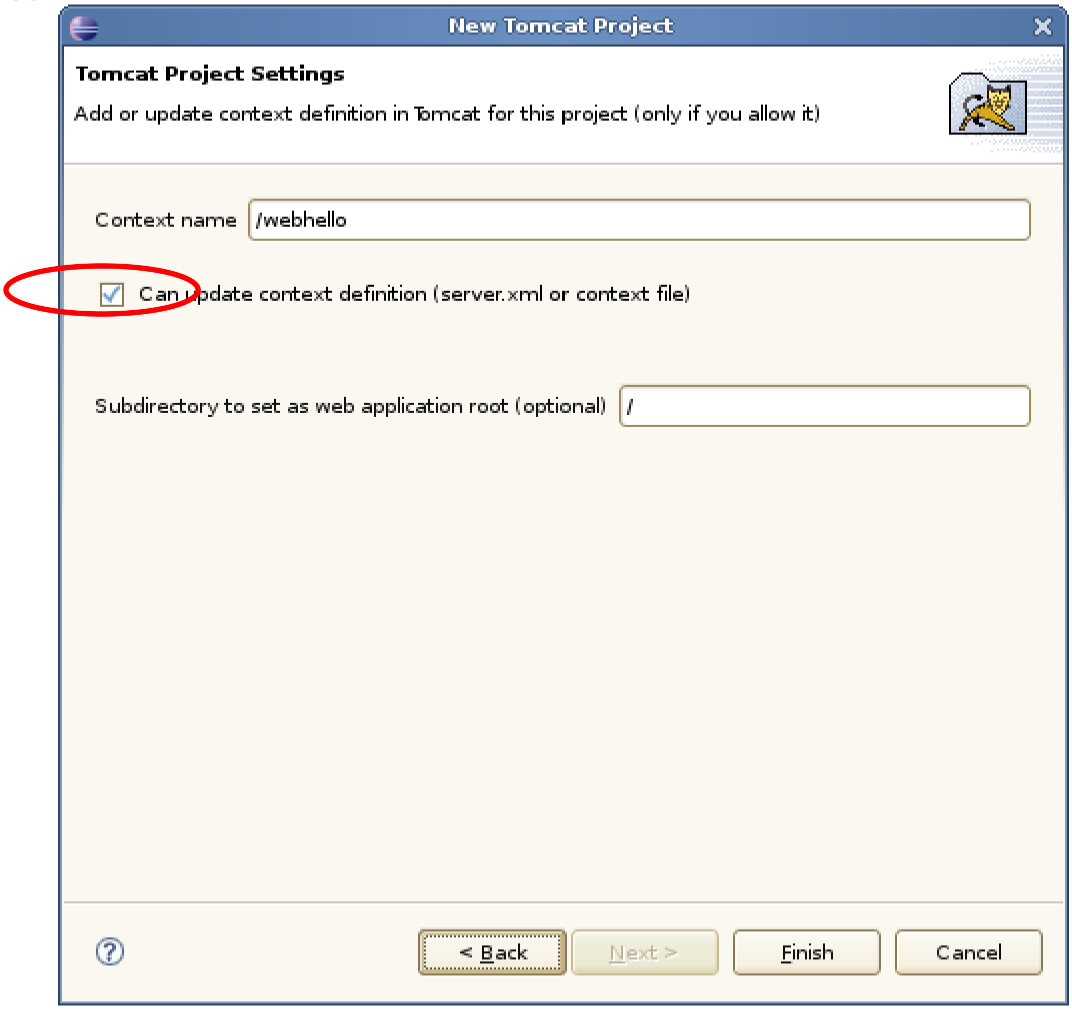
Project name: webhello

□ Use default location

Location: /usr/share/tomcat5/webapps/webhello

Figure 28 Hello exercise Java Project Settings

5) Click "Next" and check "Can update context definition".



Tomcat Project Settings

Add or update context definition in Tomcat for this project (only if you allow it)

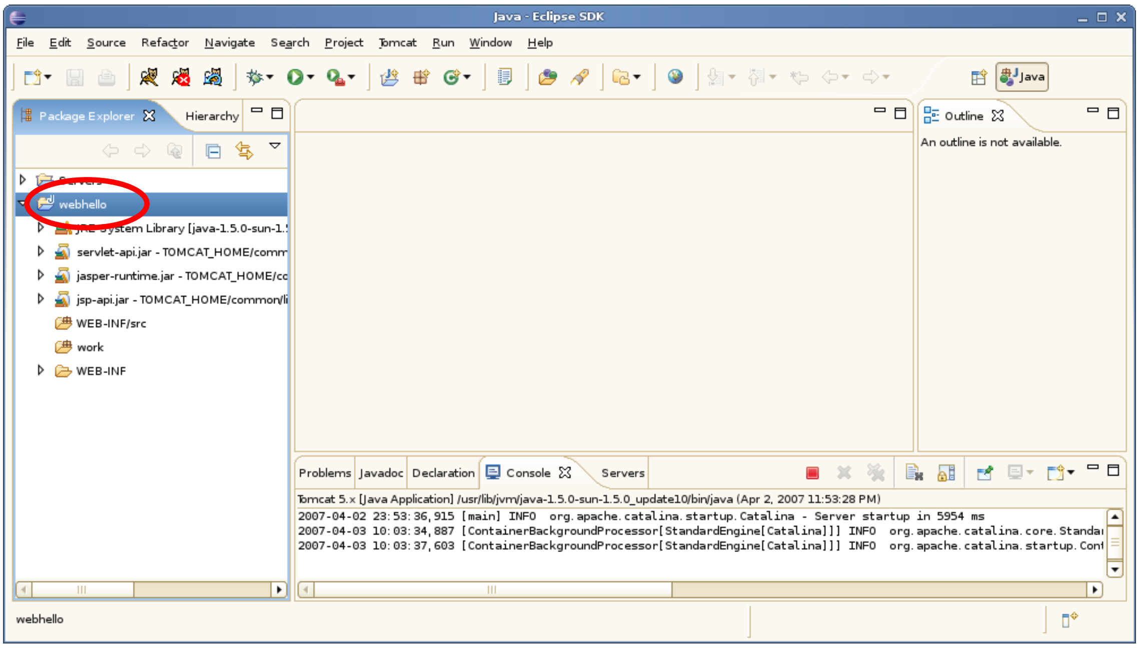
Context name: /webhello

☑ Can update context definition (server.xml or context file)

Subdirectory to set as web application root (optional): /

Figure 29 Hello exercise Tomcat Project Settings

6) Click on "Finish" and your project will be created.


Java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Tomcat Run Window Help
Java
Package Explorer
Hierarchy
Servers
webhello
JRE System Library [java-1.5.0-sun-1.5
servlet-api.jar - TOMCAT_HOME/comm
jasper-runtime.jar - TOMCAT_HOME/co
jsp-api.jar - TOMCAT_HOME/common/li
WEB-INF/src
work
WEB-INF
Outline
An outline is not available.
Problems
Javadoc
Declaration
Console
Servers
webhello

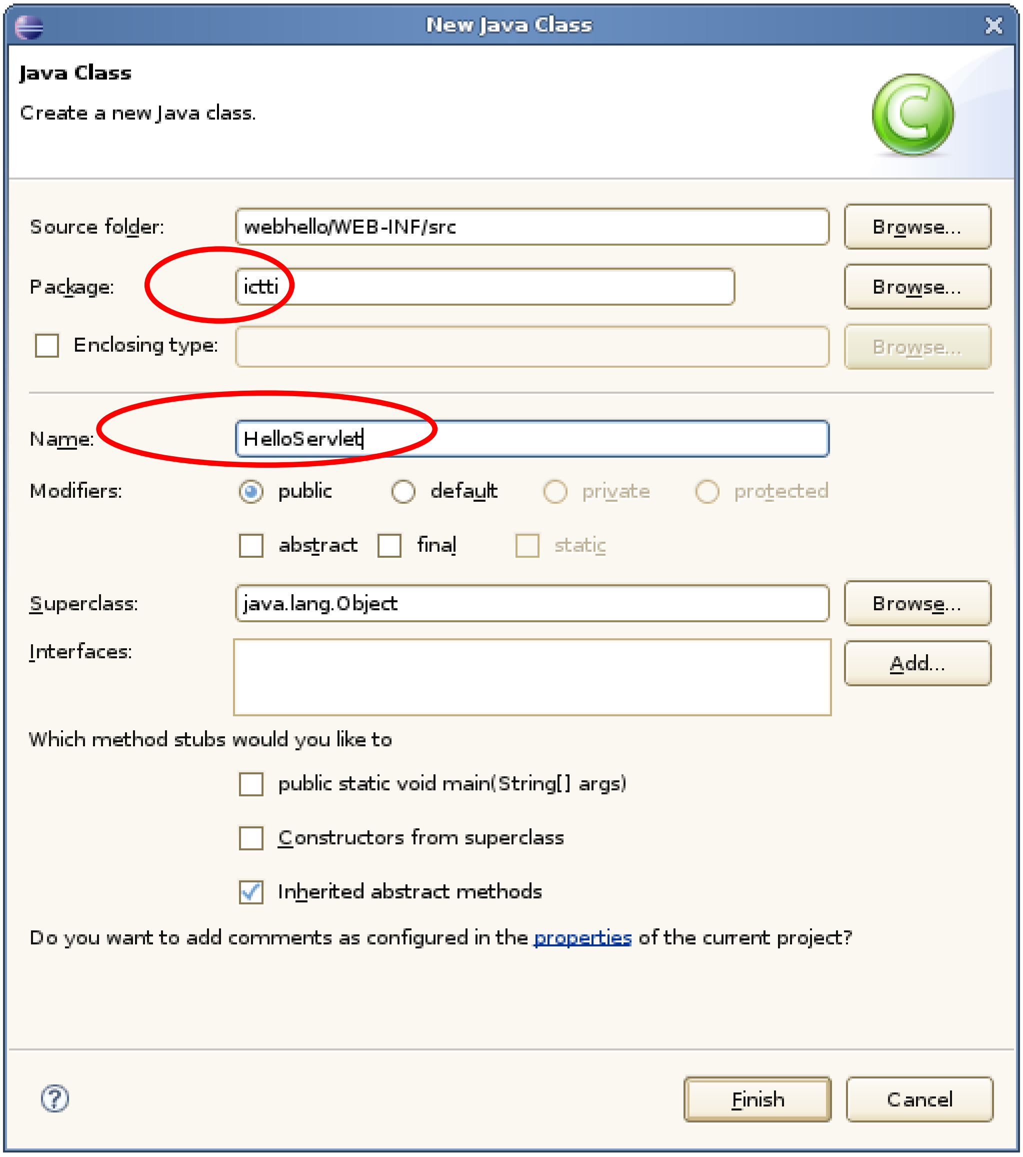
```
Tomcat 5.x [Java Application] /usr/lib/jvm/java-1.5.0-sun-1.5.0_update10/bin/java (Apr 2, 2007 11:53:28 PM)
2007-04-02 23:53:36,915 [main] INFO  org.apache.catalina.startup.Catalina - Server startup in 5954 ms
2007-04-03 10:03:34,887 [ContainerBackgroundProcessor[StandardEngine[Catalina]]] INFO  org.apache.catalina.core.Standa
2007-04-03 10:03:37,603 [ContainerBackgroundProcessor[StandardEngine[Catalina]]] INFO  org.apache.catalina.startup.Con
```

Figure 30 Hello exercise New project created

7) Check the "*server.xml*" and Tomcat directory to see how the *webhello* project is created by Eclipse.

2. Create a "*HelloServlet*" Servlet Class

1. From menu select "File"-> "new"-> "class"
2. Set "Source Folder", "Package" and class "Name". Set "ictti" for package and "HelloServlet" for class name.



New Java Class

Java Class

Create a new Java class.

Source folder: webhello/WEB-INF/src

Package: ictti

□ Enclosing type:

Name: HelloServlet

Modifiers: ● public ○ default ○ private ○ protected

□ abstract □ final □ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to

□ public static void main(String[] args)

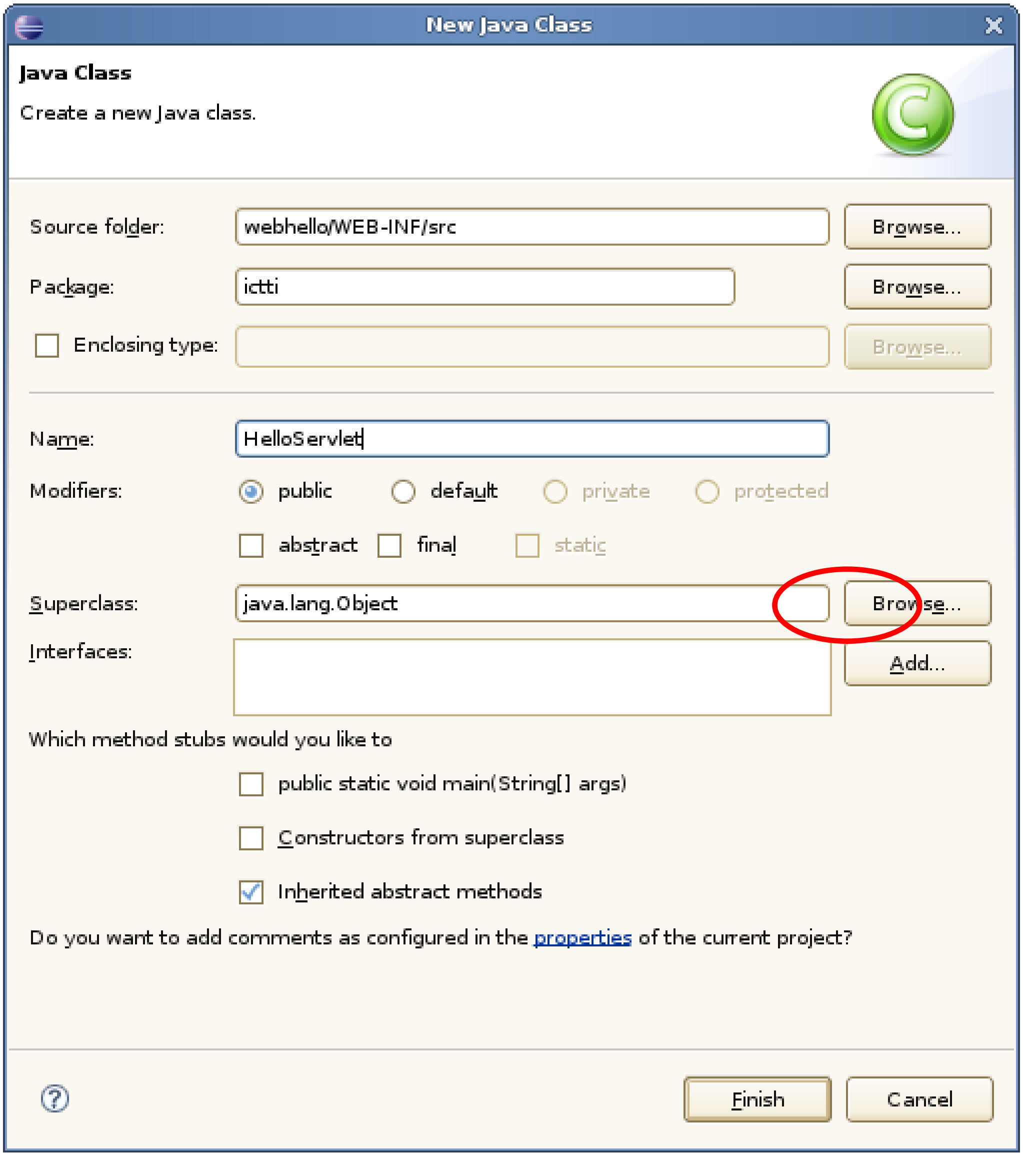
□ Constructors from superclass

☑ Inherited abstract methods

Do you want to add comments as configured in the properties of the current project?

Figure 31 Hello exercise New Java Class

3. Click on Browse button of the "Superclass".



New Java Class

Java Class

Create a new Java class.

Source folder: webhello/WEB-INF/src [Browse...]

Package: ictti [Browse...]

□ Enclosing type: [Browse...]

Name: HelloServlet

Modifiers: ● public ○ default ○ private ○ protected

□ abstract □ final □ static

Superclass: java.lang.Object [Browse...]

Interfaces: [Add...]

Which method stubs would you like to

□ public static void main(String[] args)

□ Constructors from superclass

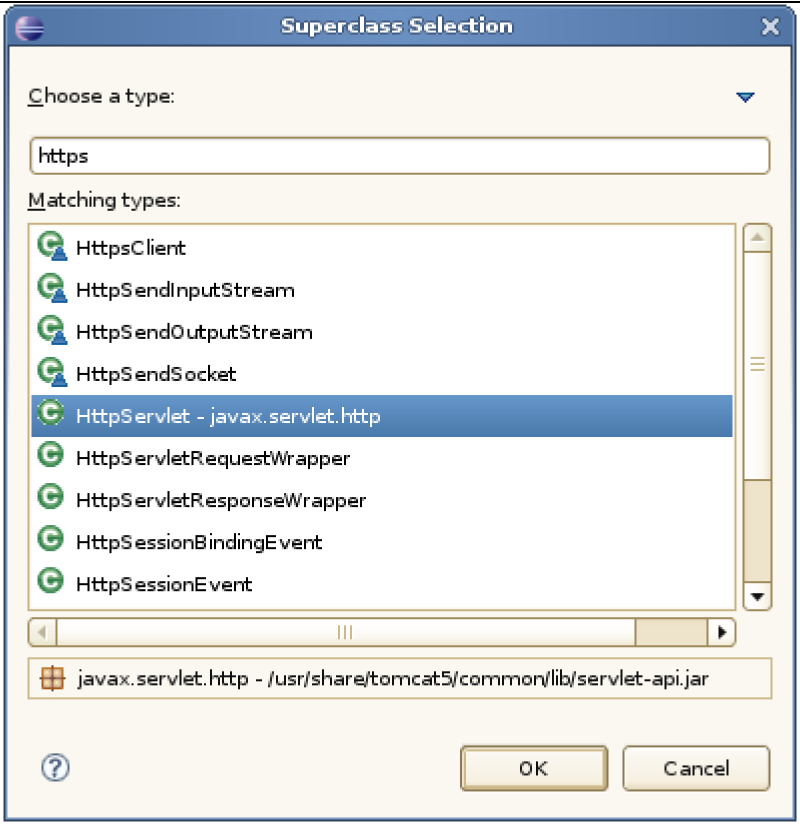
☑ Inherited abstract methods

Do you want to add comments as configured in the properties of the current project?

[Finish] [Cancel]

Figure 32 Hello exercise New Java Class Browse Superclass

4. Type "https" in the text box of "choose a type" and matching types appear below. Select "HTTPServlet" and click on "OK".



Superclass Selection

Choose a type:

https

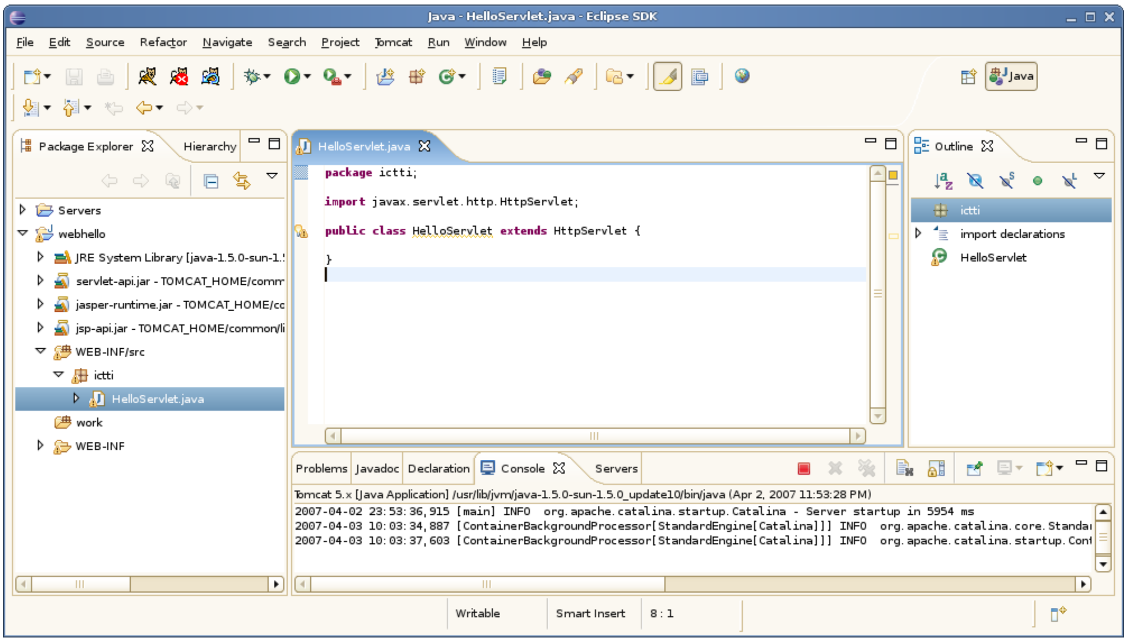
Matching types:

HttpsClient
HttpSendInputStream
HttpSendOutputStream
HttpSendSocket
HttpServlet - javax.servlet.http
HttpServletRequestWrapper
HttpServletResponseWrapper
HttpSessionBindingEvent
HttpSessionEvent

javax.servlet.http - /usr/share/tomcat5/common/lib/servlet-api.jar

Figure 33 Hello exercise Super class selection

5. Click on "Finish" and the class will be created.



Java - HelloServlet.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Tomcat Run Window Help
Java
Package Explorer
Hierarchy
Servers
webhello
JRE System Library [java-1.5.0-sun-1.!
servlet-api.jar - TOMCAT_HOME/comm
jasper-runtime.jar - TOMCAT_HOME/cc
jsp-api.jar - TOMCAT_HOME/common/li
WEB-INF/src
ictti
HelloServlet.java
work
WEB-INF
HelloServlet.java
Outline
ictti
import declarations
HelloServlet
Problems
Javadoc
Declaration
Console
Servers
Writable
Smart Insert
8 : 1

```
package ictti;

import javax.servlet.http.HttpServlet;

public class HelloServlet extends HttpServlet {

}
```

```
Tomcat 5.x [Java Application] /usr/lib/jvm/java-1.5.0-sun-1.5.0_update10/bin/java (Apr 2, 2007 11:53:28 PM)
2007-04-02 23:53:36,915 [main] INFO  org.apache.catalina.startup.Catalina - Server startup in 5954 ms
2007-04-03 10:03:34,887 [ContainerBackgroundProcessor[StandardEngine[Catalina]]] INFO  org.apache.catalina.core.Standa
2007-04-03 10:03:37,603 [ContainerBackgroundProcessor[StandardEngine[Catalina]]] INFO  org.apache.catalina.startup.Con
```

Figure 34 Hello exercise New Class created

6. Right-click the "project" -> "Tomcat project" -> "Update context definition" to update the definition of Tomcat.

☆ Your project configuration of "*webhello.xml*" will be created under context directory of "*/usr/share/tomcat5/conf/Catalina/localhost/*".

7. When you save the project, automatically the class will be compiled.

3. Let's create Servlet.

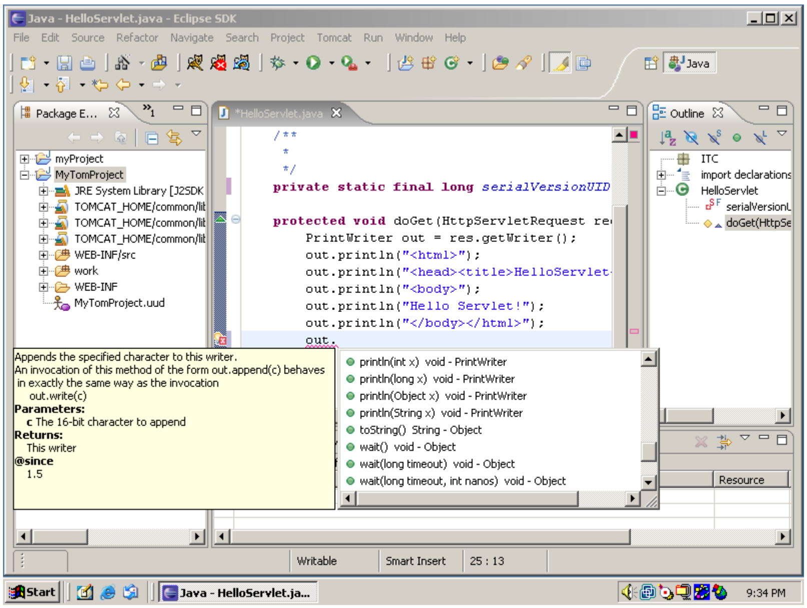
1) Import the following packages after declaration of your package name.

```
package ictti;
import javax.servlet.http.HttpServlet;
import java.io.*;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

2) Complete the *doGet* method like this.

```
protected void doGet(HttpServletRequest req, HttpServletResponse
res) throws ServletException, IOException {
        PrintWriter out = res.getWriter();
        out.println("<html>");
        out.println("<head><title>HelloServlet</title></head>");
        out.println("<body>");
        out.println("Hello Servlet!");
        out.println("</body></html>");
        out.close();
    }
```

☆ Eclipse supports Code Assist. For example type "out." in the editor. Code assist prompts you with possible completions (if automatically does not appear code assist, press Ctrl+Space).



```
    /**
     *
     */
    private static final long serialVersionUID

    protected void doGet(HttpServletRequest re
        PrintWriter out = res.getWriter();
        out.println("<html>");
        out.println("<head><title>HelloServlet
        out.println("<body>");
        out.println("Hello Servlet!");
        out.println("</body></html>");
        out.
```

Appends the specified character to this writer.
An invocation of this method of the form out.append(c) behaves in exactly the same way as the invocation
out.write(c)
Parameters:
c The 16-bit character to append
Returns:
This writer
@since
1.5

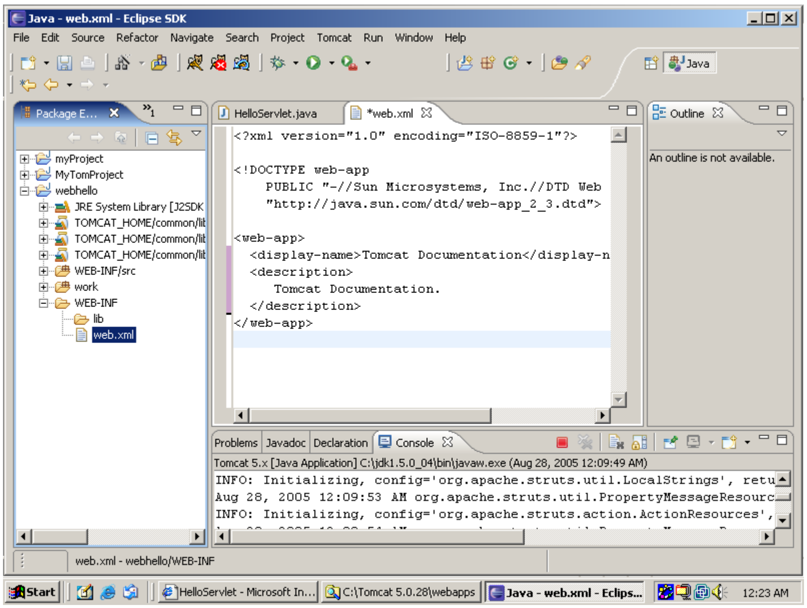
- println(int x) void - PrintWriter
- println(long x) void - PrintWriter
- println(Object x) void - PrintWriter
- println(String x) void - PrintWriter
- toString() String - Object
- wait() void - Object
- wait(long timeout) void - Object
- wait(long timeout, int nanos) void - Object

Figure 35 Hello exercise Eclipse code assist

3) Drug the following file and drop it in the project's WEB-INF folder.

%TomcatInstallDirectory%/webapp/tomcat-docs/WEB-INF/web.xml

The web.xml file will be imported. The web.xml file is a Tomcat configuration file for Web Application.



```
<?xml version="1.0" encoding="ISO-8859-1"?>

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

<web-app>
  <display-name>Tomcat Documentation</display-n
  <description>
     Tomcat Documentation.
  </description>
</web-app>
```

Tomcat 5.x [Java Application] C:\jdk1.5.0_04\bin\javaw.exe (Aug 28, 2005 12:09:49 AM)

```
INFO: Initializing, config='org.apache.struts.util.LocalStrings', retu
Aug 28, 2005 12:09:53 AM org.apache.struts.util.PropertyMessageResourc
INFO: Initializing, config='org.apache.struts.action.ActionResources',
```

Figure 36 Hello exercise web.xml

4) Open the "web.xml" file and change the contents of <web-app> tags like this.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!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>
  <servlet>
        <servlet-name>HelloServlet</servlet-name>
        <servlet-class>ictti.HelloServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>HelloServlet</servlet-name>
        <url-pattern>/ictti.Hello</url-pattern>
    </servlet-mapping>
</web-app>
```

(note1) <servlet-name> is the name used in this web.xml file.

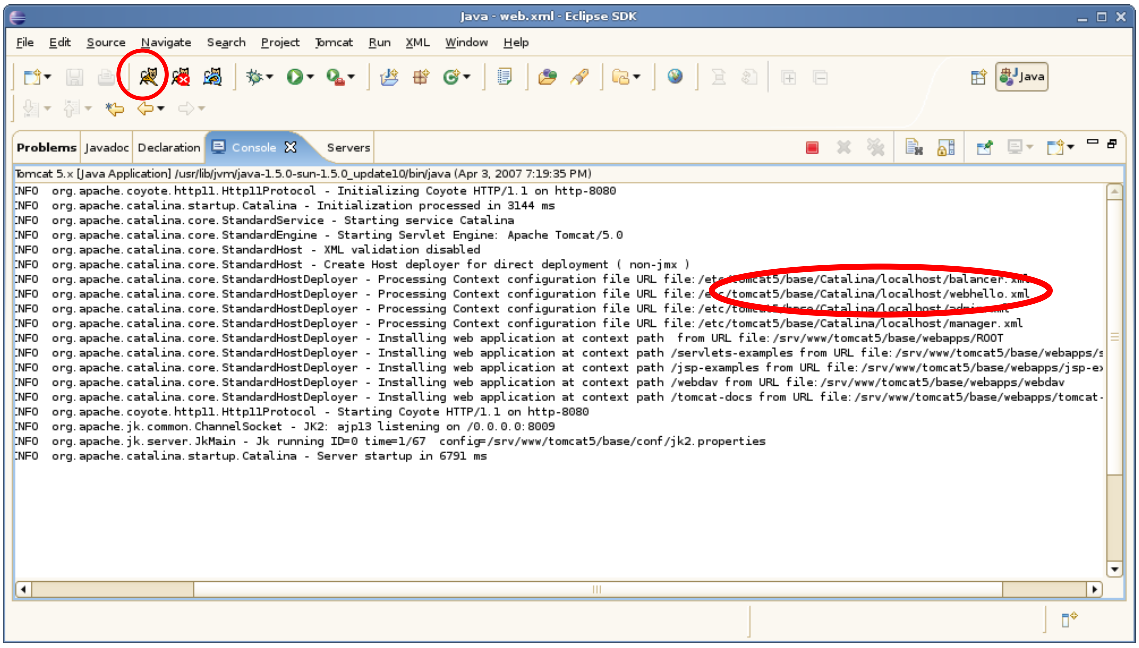
(note2) <servlet-class> is the class name with the package

(note3) <url-pattern> is the pattern that is used to call this servlet from the browser.

5) Save the project.

6) Start the tomcat clicking the Tomcat start icon, or from the menu, you can choose "Tomcat"-> "Start Tomcat".

7) On the console you can see the message informing that the context *"webhello.xml"* is processed. The *"webhello.xml"* is a Web Application Context file.


Java - web.xml - Eclipse SDK
File Edit Source Navigate Search Project Tomcat Run XML Window Help
Java
Problems Javadoc Declaration Console Servers

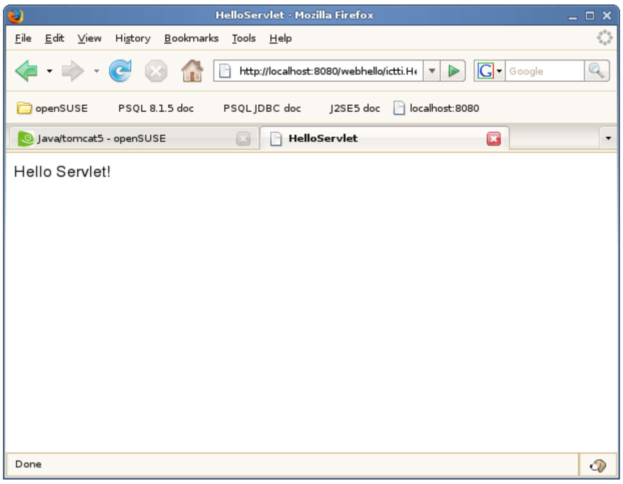
```
Tomcat 5.x [Java Application] /usr/lib/jvm/java-1.5.0-sun-1.5.0_update10/bin/java (Apr 3, 2007 7:19:35 PM)
INFO  org.apache.coyote.http11.Http11Protocol - Initializing Coyote HTTP/1.1 on http-8080
INFO  org.apache.catalina.startup.Catalina - Initialization processed in 3144 ms
INFO  org.apache.catalina.core.StandardService - Starting service Catalina
INFO  org.apache.catalina.core.StandardEngine - Starting Servlet Engine:  Apache Tomcat/5.0
INFO  org.apache.catalina.core.StandardHost - XML validation disabled
INFO  org.apache.catalina.core.StandardHost - Create Host deployer for direct deployment ( non-jmx )
INFO  org.apache.catalina.core.StandardHostDeployer - Processing Context configuration file URL file:/etc/tomcat5/base/Catalina/localhost/balancer.xml
INFO  org.apache.catalina.core.StandardHostDeployer - Processing Context configuration file URL file:/etc/tomcat5/base/Catalina/localhost/webhello.xml
INFO  org.apache.catalina.core.StandardHostDeployer - Processing Context configuration file URL file:/etc/tomcat5/base/Catalina/localhost/admin.xml
INFO  org.apache.catalina.core.StandardHostDeployer - Processing Context configuration file URL file:/etc/tomcat5/base/Catalina/localhost/manager.xml
INFO  org.apache.catalina.core.StandardHostDeployer - Installing web application at context path  from URL file:/srv/www/tomcat5/base/webapps/ROOT
INFO  org.apache.catalina.core.StandardHostDeployer - Installing web application at context path /servlets-examples from URL file:/srv/www/tomcat5/base/webapps/s
INFO  org.apache.catalina.core.StandardHostDeployer - Installing web application at context path /jsp-examples from URL file:/srv/www/tomcat5/base/webapps/jsp-ex
INFO  org.apache.catalina.core.StandardHostDeployer - Installing web application at context path /webdav from URL file:/srv/www/tomcat5/base/webapps/webdav
INFO  org.apache.catalina.core.StandardHostDeployer - Installing web application at context path /tomcat-docs from URL file:/srv/www/tomcat5/base/webapps/tomcat-
INFO  org.apache.coyote.http11.Http11Protocol - Starting Coyote HTTP/1.1 on http-8080
INFO  org.apache.jk.common.ChannelSocket - JK2: ajp13 listening on /0.0.0.0:8009
INFO  org.apache.jk.server.JkMain - Jk running ID=0 time=1/67  config=/srv/www/tomcat5/base/conf/jk2.properties
INFO  org.apache.catalina.startup.Catalina - Server startup in 6791 ms
```

Figure 37 Hello exercise Context file

8) From the browser type the following URL;

http://localhost:8080/webhello/ictti.Hello

9) "Hello Servlet!" will be appeared on the browser.


HelloServlet - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/webhello/ictti.He
Google
openSUSE
PSQL 8.1.5 doc
PSQL JDBC doc
J2SE5 doc
localhost:8080
Java/tomcat5 - openSUSE
HelloServlet
Done

Hello Servlet!

Figure 38 Hello exercise result of execution

[Tomcat configuration file using Eclipse]

(1) %TomcatInstallDirectory%/conf/server.xml

Tomcat configuration file for server.

(2) %TomcatInstallDirectory%/conf/Catalina/localhost/xxxxx.xml

Eclipse creates an application context file with the application name. For example if you create the application "webhello", then your context file is created as **%TomcatInstallDirectory%/conf/Catalina/localhost/webhello.xml**

(3) %TomcatInstallDirectory%/webapps/(context- name)/WEB-INF/web.xml

Eclipse **DOES NOT** create this file automatically. You should create it and put it under *"WEB-INF*" folder. Context-name is the name you can define in the Tomcat configuration in Eclipse. In our case we use *"webhello"* as Web Application Context name.

[sample of web.xml]

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!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>
   <servlet>
            <servlet-name>HelloServlet</servlet-name>
            <servlet-class>ictti.HelloServlet</servlet-class>
      </servlet>

      <servlet-mapping>
            <servlet-name>HelloServlet</servlet-name>
            <url-pattern>/ictti.Hello</url-pattern>
      </servlet-mapping>
</web-app>
```

Servlet name for the reference

Servlet class name with package

Servlet name should be the same as defined above

Application name to call from the browser

4. Call the Servlet from html file. Follow the following instructions.

1) Create an html file "*Hello.html*" following the [Example of HTML code: Hello.html] in the textbook. Save it under "*webhello*" project.

2) Call the *"Hello.html"* from the browser and check if it is displayed correctly. Let's think how to call the html file from the browser.

3) Add the following form tag after </p> code of the "*Hello.html".*

```
<form action = "ictti.Hello" method = "GET">
  <INPUT TYPE="submit" value="Click here">
</form>
```

4) Call the above "*Hello.htm*l" from the browser. Click the button to see if the "*HelloServlet"* is called.

2. Servlet

2.1. Servlet mechanism

2.1.1. How Servlet works with web server

Servlet is managed by Servlet Container. Servlet related classes are found in "*javax.servlet*" package. All the classes are found as "servlet-api.jar" and installed in "*%tomcatdir%/common/lib/*" directory for Tomcat. All the specification is provided in JavaDoc format installed in *"%tomcatdir%/tomcat-docs"* directory.

The HTTP request is passed to Servlet Container if the request is destinated to Web application.

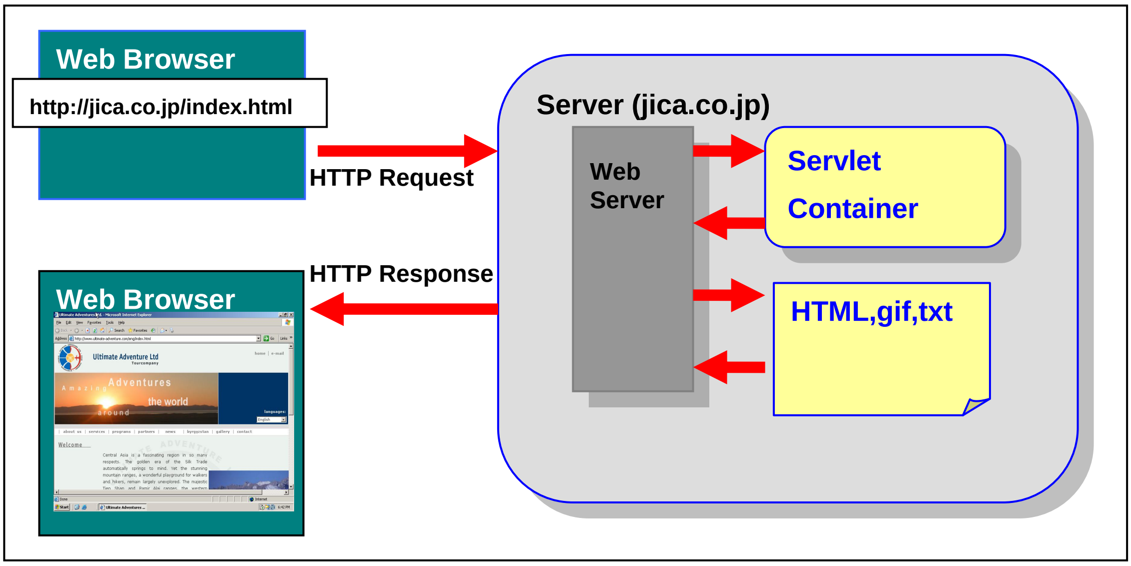


Figure 39 Servlet and Web Server

(1) A browser sends request with parameters to the web server.

(2) The web server receives the parameters and determines how to handle the request.

(3) If the request is sent to Web Server to get resources (HTML documents, text files, graphic files, etc), a web server sends back those files as response directly.

(4) If the request is designated to the Servlet, the web server sends request to the Servlet container.

(5) The Servlet is executed in the container.

(6) The Servlet sends HTML to the web server as response.

(7) The web server passes the response from the Servlet to the browser.

(8) The result is shown on the client browser.

2.1.2. Servlet instance mechanism

(1) The web server sends the client request to the Servlet Container.

(2) The Servlet container receives the client request passed by the web server.

(3) If it is the first request and Servlet is not loaded yet, load the Servlet from class libraries and instantiate it.

(4) If the Servlet is already instantiated in the container, reuse the Servlet.

(5) Execute the Servlet.

(6) The Servlet remains in the container to wait for receiving another client request.

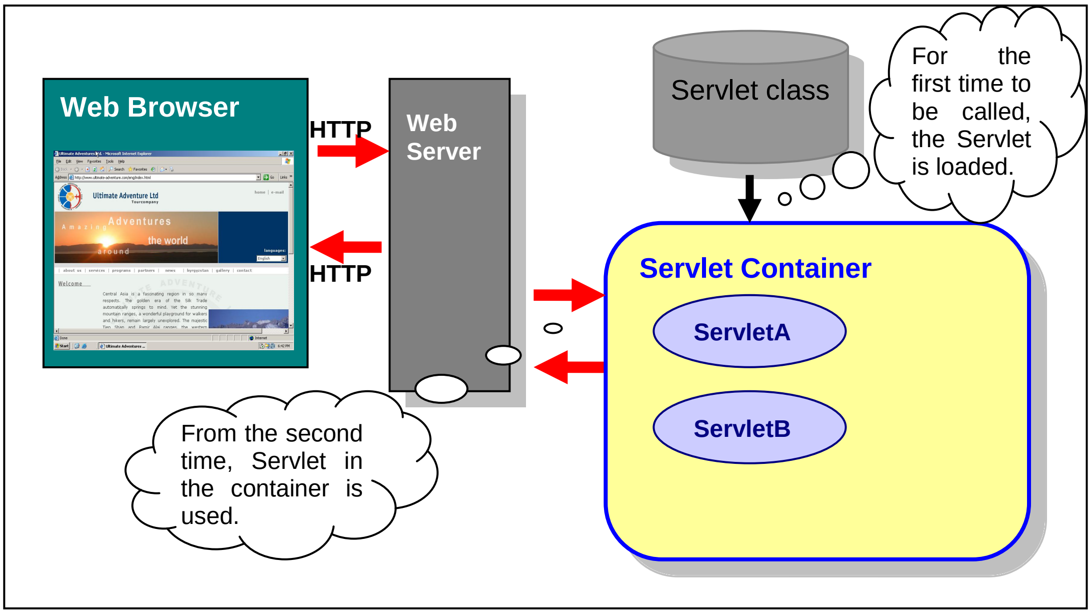

Web Browser
HTTP
Web Server
HTTP
Servlet class
For the first time to be called, the Servlet is loaded.
Servlet Container
ServletA
ServletB
From the second time, Servlet in the container is used.

Figure 40 Servlet instance mechanism

2.1.3. Servlet life cycle

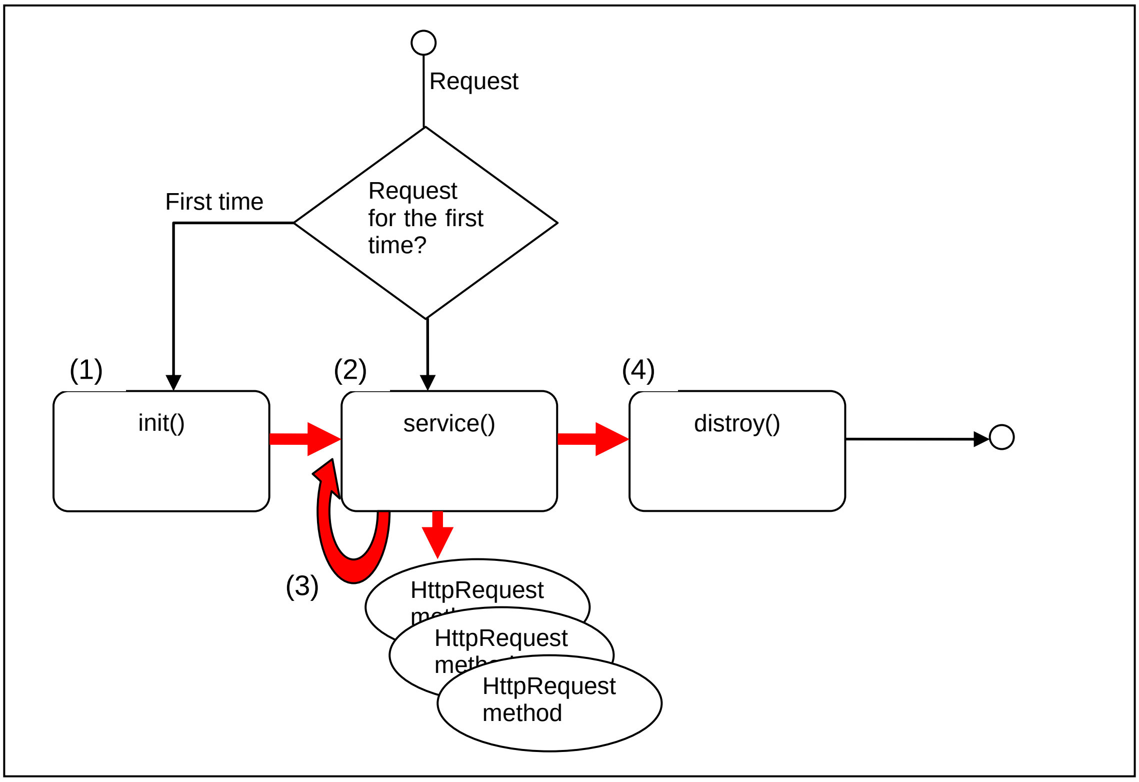


Figure 41 Servlet Life cycle

(1) The Servlet is loaded and instantiated by calling *init*() method for the first time of request.

(2) The Servlet calls *service*() method that transfers control to the HTTP request method.

(3) The Servlet is released from the request handling and waits for receiving another client request.

(4) The *destroy*() method is called to destroy Servlet. This occurs when Servlet container of application is shut down.

☆ The Servlet itself remains on the memory waiting for another client request until it is destroyed. This contributes to realize high performance of request handling.

☆ The Servlet is performed as thread. Servlet instance variables are shared among the thread.

The abstract methods *init(), service(), destroy*() are located in the "*javax.servlet.Servlet"* Interface. The "*javax.servlet.http.HttpServlet"* is a class that implements the abstract methods. You can override the methods in your own Servlets. Since the *service*() method calls *doGet*() and *doPost*() method, you can override them instead of calling *service*()

method to handle different operations.

2.1.4. Instance variables

A Servlet is instantiated only once and the instance remains until it is destroyed. This means that the Servlet is shared by multiple requests. As a result, the instance variables are also shared by all threads. Be careful to code instance variables. Do not use them to store some temporal data. Do not code in the method that can not be executed at the same time. You may synchronize the process calling synchronized statement.

```
public class HelloServlet extends HttpServlet {
    int iCount=0;
    public void doGet ( HttpServletRequest req,
                        HttpServletResponse res)
                 throws IOException, ServletException{
        synchronized (this) {
            iCount++;
        }
    }
}
```

2.2. Get/Post method

When an HTML document has a "*form*" tag, it can contain text box, radio button, etc to send parameters that the user inputs or selects. The "*method*" attribute of the form tag indicates whether the parameters are sent by ***GET*** method or ***Post*** method.

[ExampleGet.html]

```
<html>
    <head><title>Test GET method</title></head>
    <body>
        <h1>This page is to test GET method</h1>
        <form method="GET"
        action="/webhello/servlet/HelloServlet ">
            Name:  <input type="text" name="name"> <br>
            Age  :  <input type="text" name="age"> <br>
            <input type="submit" value="Click here">
        </form>
    </body>
</html>
```

2.2.1. HTTP Get method

The parameters of the browser are appended to the URL string to send them to the web server.

[example of get method parameters]

```
http://localhost/webhello/servlet/HelloServlet?name=Pablo&age=42
```

The *name* and *age* parameters are the names of parameter and after the "=" shows its value. These parameters are passed to the Web server who passes it to Servlet. Finally *doGet* () method of the Servlet is called if the method is set to *GET* in the form tag.

☆ Since the parameters are shown after the URL address, working with sensitive data such as password should not use *Get* method.

2.2.2. HTTP Post method

The parameters are sent but not displayed in the URL string. The *doPost* method of Servlet is called.

This method is useful especially for the parameters that are inconvenient to be appended in the URL string such as password, credit card number or private data.

[Example]

```
<form action="registration" method="POST">
```

2.3. Servlet parameters

A browser can send the parameters selected or input by the user to the server by GET of POST method.

2.3.1. Servlet parameter mechanism

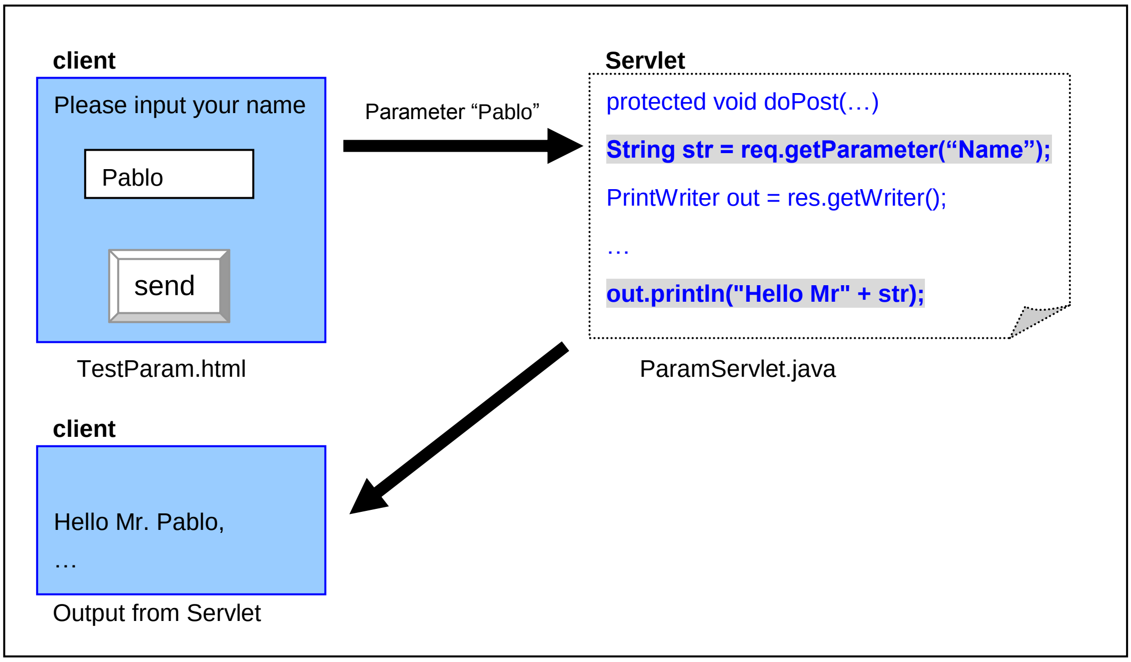


Figure 42 Servlet Parameter

[TestParam.html]

```
<html>
    <head><title>TestParam</title></head>
    <body>
        <form action="servlet/ictti.ParamServlet" method="POST">
            Please input your name.
            <input type="text" name="Name" /> <br>
            Please input your password.
            <input type="password" name="Pass" /><br>
            <input type="submit" value="send" />
        </form>
    </body>
</html>
```

- In the *action* attribute of *form* tag, the Servlet URL is specified. Here the Http Post method is applied. You can also specify absolute URL.

```
<form
action="http://localhost:8080/servletProj/Servlet/ictti.ParamServ
let" method="POST">
```

- Element type of *form* tag is text box. The name of text box is "Name". This name is referred in the Servlet to get the value.

```
<input type="text" name="Name" >
```

- Input type with "password" does not show your input content.

```
<input type="password" name="Pass" />
```

- The submit button is specified. The characters that are displayed to the button are "send".

```
<input type="submit" value="send">
```

[ParamServlet.java]

```
package ictti;
import java.io.*;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class ParamServlet extends HttpServlet {
    protected void doPost(HttpServletRequest req,
                           HttpServletResponse res)
                    throws ServletException, IOException {
        // Get input parametes from the browser
        String str=req.getParameter("Name");
        String strPass=req.getParameter("Pass");
        System.out.println("password is " + strPass);

        // Create HTML output code
        PrintWriter out = res.getWriter();
        out.println("<html>");
        out.println("<head><title>Test 
parameters</title></head>" );
        out.println("<body>");
        out.println("Hello Mr.  " + str);
        out.println("</body></html>");
        out.close();
    }
}
```

- "*HttpServletRequest.getParameter*()" method gets the parameters sent from a client. The parameter value is obtained by setting the parameter name specified in the HTML document.

```
        String strParam=req.getParameter("Name");
```

☆ you have to filter the specified message string for characters that are sensitive in HTML such as "<", ">" to "&lt;", "&gt;" to avoid potential attacks caused by including JavaScript codes in the request URL.

2.4. Servlet initialization parameters

A Servlet can obtain constant values from "*web.xml*" configuration files as initialization parameters. These parameters are used for;

(e.g.) Environmental values that depend on the specific environment of the execution.

(e.g.) The values frequently changeable and are inconvenient to write in the program.

(e.g.) The value that is preferably stored outside such as database server name, URL, etc.

2.4.1. ServletConfig and ServletContext

There are two ways to get initialization parameters.

1) Servlet initialization parameter is a parameter defined for **each** Servlet. You can get the parameters by *getInitParameter()* method of *ServletConfig* object.

```
public void init(ServletConfig config) throws ServletException{
    ServletConfig config = getServletConfig();
    String strParam1 = config.getInitParameter("PARAM1");
}
```

2) Context initialization is available to **all** servlets. First call the *getServletContext* method that returns the *ServletContext* object. Then call the *getInitParameter()* of the *ServletContext* to get Context Initialization Parameters.

```
public void init(ServletConfig config) throws ServletException{
    ServletContext context = getServletContext();
    String strDBConnect = context.getInitParameter("DBConnect");
}
```

☆ You can call *getServletConfig*() method anywhere from the Servlet if the following method is called in the beginning of the *Servlet* instantiation. The *init()* method of super class (= *GenericServlet* class) will store the ServletConfig object it receives from the servlet container for later use. What is needed is to override the method of super class.

```
public void init(ServletConfig config)
        throws ServletException{
```

```
      super.init(config);
 }
```

2.4.2. How to configure the web.xml file

The *web.xml* file is a Tomcat configuration file known as the deployment descriptor. Each application requires a *web.xml* that is stored under "*/WEB-INF*" directory. Servlet initialization parameters are described here which will be passed to the Servlet program.

[web.xml]

```
<web-app>
    <context-param>        ◀························ Context initialization parameter
       <param-name>DBConnect</param-name>
       <param-value>jdbc:oracle:thin:@127.0.0.1:1521:ORCL</param-v
alue>
    </context-param>

   <servlet>
        <servlet-name>HelloServlet</servlet-name>
        <servlet-class>ictti.HelloServlet</servlet-class>
        <init-param>       ◀························ Servlet initialization parameter
          <param-name>PARAM1</param-name>
          <param-value> Parameter test from web.xml</param-value>
        </init-param>
    </servlet>

    <servlet-mapping>
        <servlet-name>HelloServlet</servlet-name>
        <url-pattern>/ictti.Hello</url-pattern>
    </servlet-mapping>
</web-app>
```

- Servlet initialization parameters are coded within <init-param> element of <servlet> tag.
- Context initialization parameters are described within <param-name> and <param-value> of <context-param>.
- servlet-mapping is to map URL name to a Servlet. <Servlet-name> specifies the name of Servlet used in the web.xml file.
- <servlet-class> is where the Servlet class name with package name is defined.
- <url-pattern> specifies URL pattern name that is called from the browser.

2.5. How to forward the request and response

2.5.1. What is Dispatch?

A Servlet can send HTML file to the browser directly coding HTML code within the program. But there are other techniques to transfer control to another resources (Servlet, JSP or HTML) on the same server.

```
RequestDispatcher rd = request.getRequestDispatcher("/Hello.jsp");
rd.forward(request, response);
```

1) Obtain *RequesDispatcher* of *HttpServletRequest* by *getRequesDispatcher* method indicating the resource path to where it is dispatched.
2) Call *forward* method providing *request* and *response* object.

The *forward* method transfers control to a resource such as Servlet and JSP on the same server.

2.5.2. How to send HTML document from Servlet?

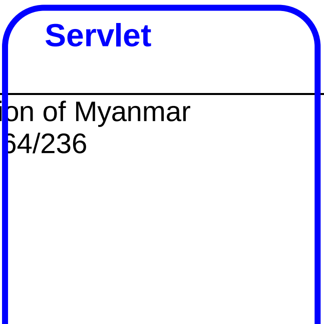
If you want to send static HTML document already stored in the server from the Servlet, *sendRedirect* method of response object sends back the indicated HTML file to the browser.

```
response.sendRedirect("/index.html");
```

2.6. Session

2.6.1. What is session?

Http protocol is a stateless protocol, that is, a round trip of request from client and response from the server. After the server returns the response the connection finishes. Although another connection starts with client another request, there is no way to associate with the data of previous connection. For example, if the server gets user name after login page, on the second page the server can not identify if the connection comes from the same user as who logged in the previous connection.



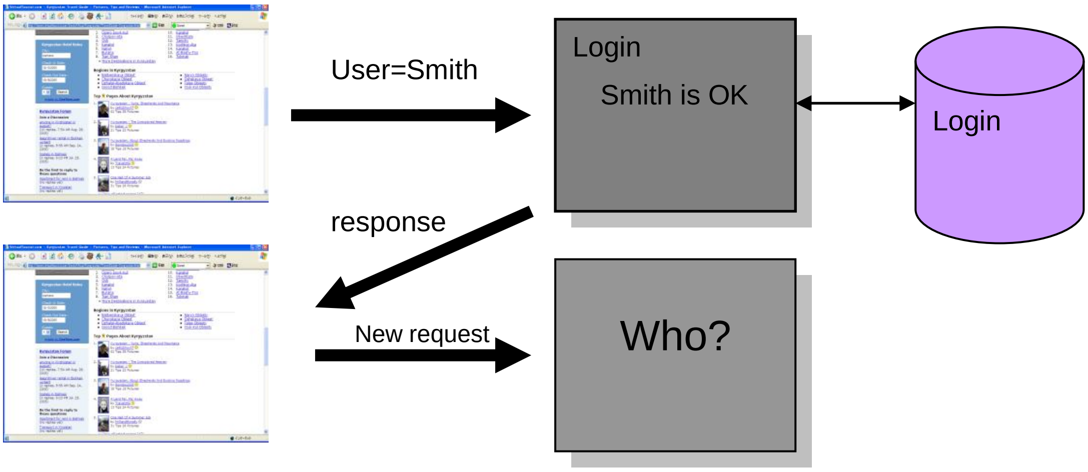


Figure 43 HTTP connection without session

A session is to keep track of user information during the user's navigation in the same Internet site. For example, in a shopping cart application, you may put in the cart the product you want to buy in the different pages. Finally you may decide to pay check by credit car. In this case your data and products you put in the cart must be kept to calculate the check until you want to finish using the service in that site.

[example of session]

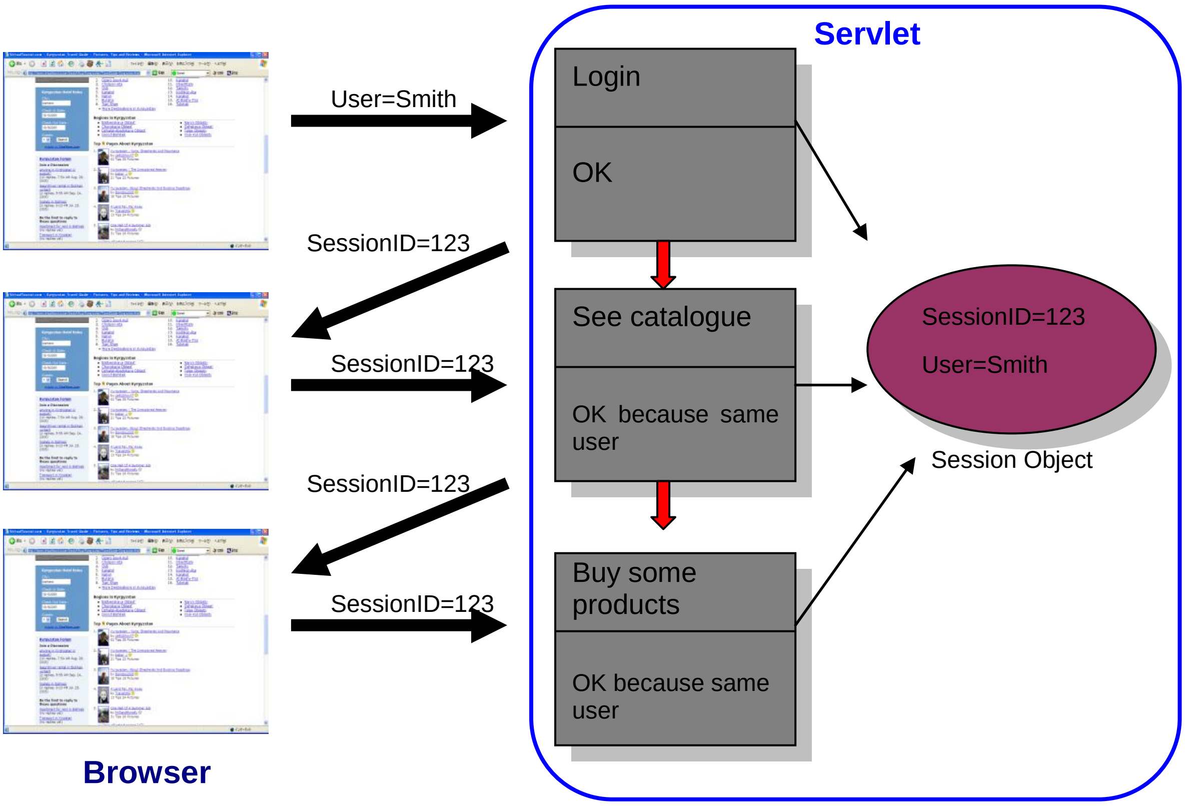


Figure 44 Servlet session

2.6.2. Mechanism of session control

- A session is controlled by SessionID
- A Server creates *SessionObject* in the memory and store users' specific information during the session.
- Each time of Http connection needs to send SessionID together with other request information.
- There are two ways to save sessionID on the client machine.

- Cookie;

SesiionID is stored in the cookie on the **client** side. A client passes the cookie information to the server each time of request.

- URLRewriting

The SessionID is appended in the URL for each **page**. The use of *URLRewriting* is limited only when the cookie is disabled.

2.6.3. How to use session data

- *getSession(true)*

Creates the session if the session does not exist. This is used for the first time of session, e.g. login page.

- *getSession(false)*

Returns null if the session does not exist. Usually it is used to check if a user has logged in correctly.

- *setAttribute(name, value)*

Store the value with the specified name in the session object.

- *getAttribute(name)*

Obtain the value of the specified parameter from the session object. The name should be the same as used to store value with *setAttribute*() method.

1) for new session (for the first time to enter the site)

```
// create session if a session object does not exist
request.getSession(true)
// set session data
SessionObject.setAttribute(name,value)
```

The parameter ***true*** creates new session if a session does not exist. If the session already exits, return it. With *setAttribute()* method, session name and value are set.

[Example]

```
protected void doPost( HttpServletRequest req,
                        HttpServletResponse res)
                        throws ServletException, IOException {
  String strName=req.getParameter("Name");
  // create session object
  HttpSession session=req.getSession(true);
  // set information to the session object
  session.setAttribute("Name", strName);
}
```

2) For the session already created (from the second page)

```
request.getSession(false);
```

```
SessionObject.setAttribute(name,value);
```

Set ***false*** to the parameter of *getSession()*. A new session is NOT created. Returns null if there's no current session.

```
protected void doPost(HttpServletRequest req,
                      HttpServletResponse res)
                        throws ServletException, IOException {
  //get information from the session
  HttpSession session=req.getSession(false);
  if (session != null){
     String strName = (String)session.getAttribute("Name");
  }
}
```

The "*getAttribute()*" method returns the value with object type. It is required to cast each object to the appropriate class.

```
String strName = (String)session.getAttribute("Name");
```

3) Finish session

```
HttpSession session = req.getSession(false);
if (session!=null){
    session.invalidate();
}
```

By *session.invalidate()* method, session will be invalidated and the object will be deleted (target of garbage collection).

☆ A session also finishes automatically after certain minutes have passed without any interaction from the user according to the configuration set in the web.xml.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!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>
   <!-- Define the default session timeout for your application,
         in minutes.  From a servlet or JSP page, you can modify
         the timeout for a particular session dynamically by using
         HttpSession.getMaxInactiveInterval(). -->

    <session-config>
      <session-timeout>30</session-timeout>    <!-- 30 minutes -->
    </session-config>

</web-app>
```

The default timeout is 30 minutes.

The following code shows the sample of the first page (usually the Login check page).

[First Page]

```
protected void doPost(HttpRequest req, HttpServetResponse res)
                throws ServletException, IOException{
    // Get user input parameters from browser
    String strName = req.getParameter("UserID");
    String strPwd  = req.getParameter("PASSWORD");

    // Check if the UserID and PASSWORD exist (usually with DB)

    // Create new session
    HttpSession session = req.getSession(true);

    // Store necessary information to the session
    session.setAttribure("UserID", strName);
}
```

The following code shows the sample code after the first page.

[Second Page]

```
protected void doPost(HttpRequest req, HttpServetResponse res)
                throws ServletException, IOException{

    // Obtain already created session (not creat new session)
    HttpSession session = req.getSession(false);

    // Check if the session exists.
    // If the session is not connected or the user get access to
    // this page without login, then error
    if (session == null) {
        // some error coding
    }

    // Get the necessary information from the session
    String strName = (String)session.getAttribure("UserID");
}
```

☆ You are storing data into session object in the memory for each session . It is important to consider how much memory will be wasted which depends on how many users will be accessed at the same time.

Exercise 2: Servlet programming

1. Request parameter programming. Follow the instructions below.

1) Create the "*TestParam.html*" of the textbook in the "*webhello"* project.

2) Create the "*ParamServlet.java*" Servlet program of the textbook with package name "*ictti*".

3) Configure the "*web.xml*" file using the "*Invoker"* Servlet to be able to call anonymous Servlet.

4) Call the "*TestParam.html"* from your browser. Input any name in the text box and click the button to see if the Servlet can obtain the parameters you input.

2. Initialization parameters programming. Follow the instructions below.

1) Open the "*web.xml"* file of the "*webhello"* project.

2) Create the following Servlet tag and Servlet-mapping tag.

```
<servlet>
     <servlet-name>TestInitParam</servlet-name>
     <servlet-class>ictti.TestInitParam</servlet-class>
  </servlet>
```

```
 <servlet-mapping>
     <servlet-name>TestInitParam</servlet-name>
     <url-pattern>/MyInitParam</url-pattern>
  </servlet-mapping>
```

3) Define both "Servlet Initialization" and "Context Initialization" following the textbook in the "*web.xml"*.

4) Create a Servlet "*TestInitParam.java*" with package name "*ictti*". Obtain the Initialization parameters you set in the "*web.xm*l" file in the "*doGet*" method. Print out the parameters to the console. Do not forget to import the correct *javax.servlet* package to be able to refer

ServletConfig and *ServletContext* classes.

3) Call the "*TestInitParam*" Servlet from the browser and check if the Servlet gets correctly the initialization parameters.

3. Create *HelloServlet.java* with the following condition. Do it by yourself **without asking to the teacher or your friends**, because this exercise is quite basic of Java and Servlet.

1) Web Context directory name is "*myhello*".

2) Package is "ictti".

3) Create HelloServlet.java that outputs the Hello message to the browser **WITHOUT USING ECLIPSE.**

4) Compile the source Servlet **WITHOUT USING ECLIPSE**!

5) Create web.xml and set necessary configuration in web.xml (You can copy from somewhere else). Place it in the adequate directory.

6) Start Tomcat **WITHOUT FROM ECLIPSE**!

7) Call HelloServlet from your browser.

☆ Servlet class is placed under "*WEB-INF/classes*" directory.

☆ If you do not remember how to compile java code manually, refer to the Java Basic text book. In the beginning we learned it.

☆ If you do not remember how to set classpath, also refer to the Java Basic text book.

Good luck!

4. From your browser input the following URL of Tomcat examples. Execute it and analyze how to program it.

```
http://localhost:8080/servlets-examples/servlet/SessionExample
```

3. JSP

3.1. What is JSP

JSP(Java Server Pages) is a code embedded to HTML page to process dynamic web pages. JSP technology is designed to simplify the process of creating pages by separating web presentation from web content.

3.1.1. What is the difference between Servlet and JSP?

Servlet and JSP are both developed to process dynamic web page. To display dynamic information in a browser, we need the following functions in a Web Application.

1) A class that handles request data sent from a browser.

2) The data is processed and business logic is performed.

3) Send back the final HTML document format as response to display in a browser.

Although a Servlet can cover all the above functions, there are the following disadvantages of Servlet to code HTML tags inside the program.

[Disadvantages of Servlet]

1) HTML code is embedded in the Servlet program. Therefore all the HTML code in the Servlet should be described in a program.

2) The Servlet program becomes less clear and difficult to maintain when the Web page design is complicated.

3) Servlet should be compiled to be executed. Each time the HTML design is modified in spite of slight change, the compilation is required.

[Advantage of JSP]

- JSP is embedded in HTML. The mayor part in JSP can be described with HTML code, which results easier than Java code. Only minimum code for dynamic information is written in JSP.
- The HTML code is separated from Servlet. Servlet program remains only with business functionality while JSP is responsible of the view functionality.
- If the HTML code is separated, a web designer can work with HTML document while programmers can concentrate on working with the implementation of business functions.

- JSP does not require compilation. Any modification can be done just modifying the file. Modification task can be faster than Servlet.

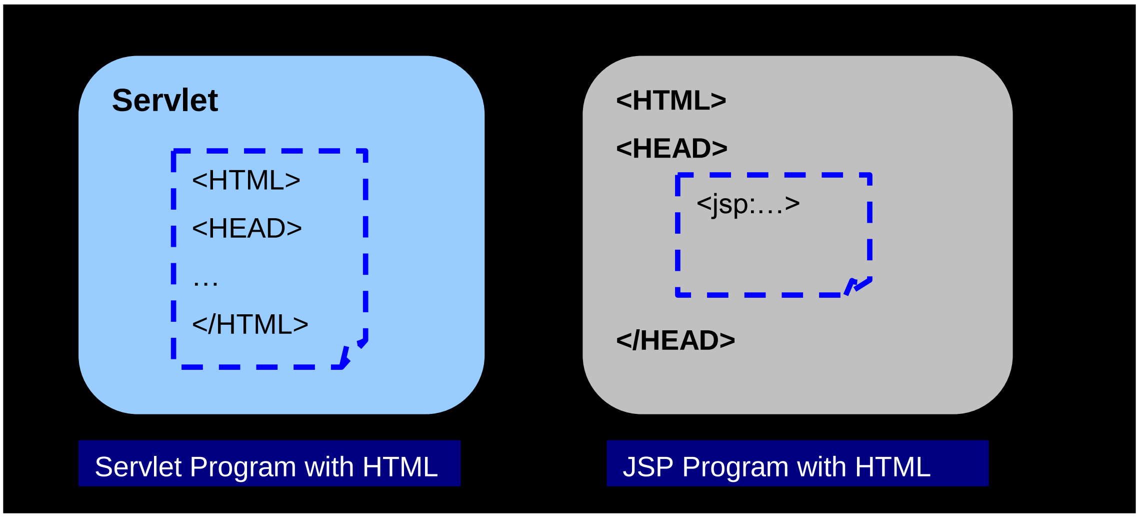


Figure 45 The difference between Servlet and JSP

3.1.2. JSP flow

1) There are two ways to call JSP

- Browser sends request directly to JSP file.
- Browser calls Servlet that transmits control to JSP.

2) JSP file is translated to Servlet.

3) Servlet is compiled.

4) Class file is executed to output HTML file.

☆Once the Servlet is compiled, it is stored as a class. From the second time to call JSP, the translation and compilation are skipped to save the time of execution unless the file is modified.

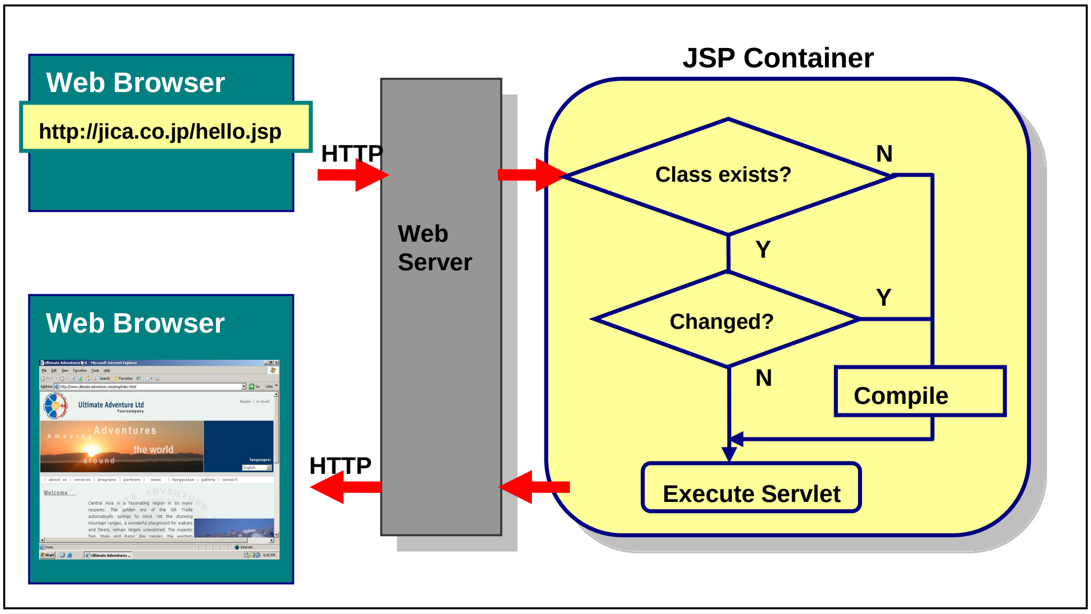


Figure 46 JSP Flow

3.1.3. What do you need for JSP

- JDK (Java Development Kit)
- Web Server (Apache, Tomcat, etc)
- JSP Container (Servlet container usually includes JSP container)
- Web browser

☆ Tomcat 5.X includes JSP2.0 container.

3.1.4. How to call JSP

```
http://hostname/applcation_name/jsp_directory_name/xxx.jsp
```

[Example]

```
http://localhost:8080/webhello/jsp/Hello.jsp
```

3.1.5. How to develop JSP

1) Setup JSP Container

Usually the web application tool that installs Servlet container includes JSP container. For example Tomcat installs both the Servlet and JSP container.

2) Create JSP file with editor

You can use any editor (like notepad.exe) to create a JSP file. The extension of the file should be ".jsp".

3) Place the jsp file

Place the JSP file to the root directory under the web application directory. The way to call JSP file is the same as html file.

4) Test

JSP is compiled automatically when the file is called for the first time. You can test JSP just calling from the browser or Servlet.

3.1.6. How to write JSP

1) Create a JSP file and write the basic HTML code.

2) Describe JSP tags where the dynamic information is required to be displayed.

3) When the JSP file is called, HTML code is shown as it is. The JSP code is converted to HTML code as a result of the execution.

[Mingalarbar.jsp]

```
 <html>
    <head>
        <%! public int i; %>
        <title>Hello JSP</title></head>
    <body>
        <h1>
            <% for (i=0; i<=2; i++){
            out.println ("Min ga lar bar from JSP <br>");
            }%>
        </h1>
    </body>
</html>
```

IF you place the above jsp file as "*%TomcatInstallDir%/webhello/MyJSPLoop.jsp*", you can call it from the browser;

```
http://localhost:8080/webhello/Mingalaba.jsp
```

3.1.7. Components of JSP

1) Comment

```
<%--    --%>
```

2) Scripting elements

- Declarations

```
<%!      %>
```

- Expression

```
<%=      %>
```

- Scriptlet

```
<%   %>
```

3) Directives

- page directives

```
<%@ page%>
```

- include

```
<%@ include %>
```

4) Implicit objects

- request
- response
- out

5) Standard actions

- useBean
- getProperty
- setProperty
- include
- forward
- param

3.2. JSP grammar

- JSP code is embedded in HTML document. Therefore the base is HTML code. JSP code is only embedded where the dynamical data is required.
- HTML code is displayed as it is coded. JSP code is converted to HTML format reflecting the **result** of code.

3.2.1. JSP comment

[Syntax of JSP comment]

```
<%-- write JSP comment here --%>
```

The comment is not sent to the HTML document. It is invisible for a user.

[Syntax of HTML comment]

```
<!-- write here HTML comment -->
```

The HTML comment is sent to the browser.

3.2.2. Scripting elements: Declarations

Declarations enable to declare instance variables and method.

[Syntax]

```
<%!declaration%>
```

[Example]

```
<%!
  public int calc(int i){
    return i++;
  }
%>
```

```
<%! public int i;%>
```

3.2.3. Scripting elements: Expression

- Expression can be primitive type, variables or return value of a method.
- The result is converted to String format automatically.

[Syntax]

```
<%= expression %>
```

[Example]

```
<%= strName %>
<%= employee.calc() %>
<%= x + y %>
```

3.2.4. Scripting elements: Scriptlet

- You can code any Java statement within a scriptlet tag.
- Since Java code is written here, do not forget to end with ";" at the end of statement.

[Syntax]

```
<% script %>
```

[Example]

```
<% String strA = "Hello";
   String strB = " JSP";
   String str=strA + strB;
%>
```

```
<%
GregorianCalendar calendar = new GregorianCalendar();
if (calendar.get(calendar.AM_PM)==calendar.AM){ %>
        <p>Good morning! </p>
<% }else{ %>
        <p>Good afternoon!</p>
<%}%>
```

The scriptlet is embedded between HTML codes in the above example.

3.2.5. Directive: page directive

Directive is used to define the conditions that the JSP engine should follow. Page Directive specifies attributes about the actual page and described at the top of the page as "%@ page".

Table 1 - Attributes of JSP page directive

Attributes	Remarks
import	Imports java classes that are available in the actual page.
session	Indicates whether the session will be used in the JSP page. The default is *true*. The session is created if it does not exist in case of *true.*
contentType	Specify MIME type and encoding type. If the contentType is omitted, "ISO-8859-1" is considered for enconding type.

[Syntax]

```
<%@ page attribute %>
```

[Example]

```
<%-- import the necessary classes --%>
<%@ page import="java.io.*" %>
<%@ page import="webhello" %>
<%@ page import="java.io.*,webhello"%>

<%-- set the session object --%>
<%@ page session="true" %>

<%-- specify the contentType for the actual JSP page --%>
<%@ page contentType="text/html;charset=Windows-31J" %>

<%-- Declare to use Expression Language --%>
<%@ page isELIgnored="false" %>
```

3.2.6. Directive: include

The "*Include"* directive includes statically the indicated file, which means, the file is included before the whole jsp is converted to Java file. After the file is included, the whole jsp file is converted to Java file and compiled to class file to be executed.

[Syntax]

```
<%@ include file="filename" %>
```

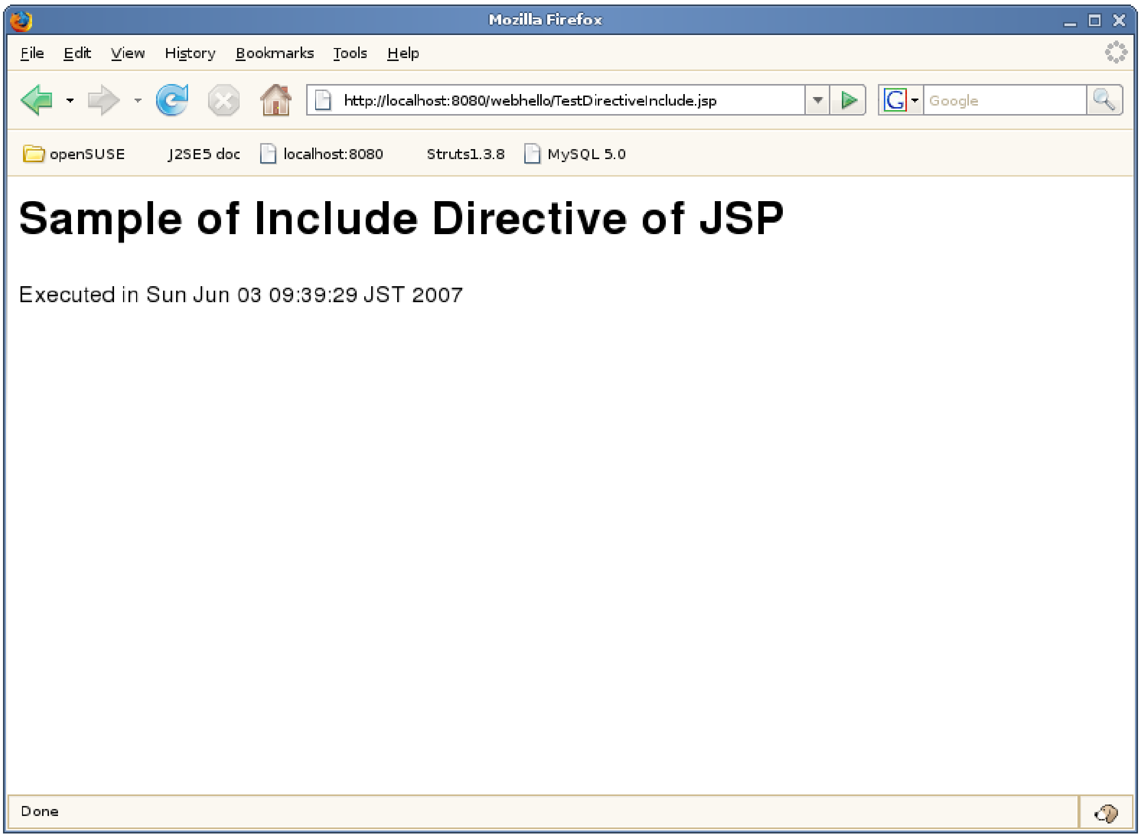
[TestDirectiveInclude.jsp]

```
<html>
  <body>
    <h1>Sample of Include Directive of JSP</h1>
  </body>
<%@ include file= "/include/footer.txt" %>
</html>
```

[/include/footer.txt]

```
<%@ page import="java.util.Date" %>
<p>Executed in  <%= new Date() %> </p>
```

[result]


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/webhello/TestDirectiveInclude.jsp
Google
openSUSE J2SE5 doc localhost:8080 Struts1.3.8 MySQL 5.0
Done

Sample of Include Directive of JSP

Executed in Sun Jun 03 09:39:29 JST 2007

Figure 47 JSP Directive Include

3.2.7. Implicit object

Implicit object is an object that is already defined in JSP engine. You can use the object without defining in the program. It should be used within JSP tags. Some of implicit objects are:

- **request**

is a variable for *javax.servlet.http.HttpServletRequest* object. The same object used in Servlet.

- **response**

is a variable for *javax.servlet.http.HttpServletResponse* object.

- **session**

is a variable for *javax.servlet.http.HttpSession* object. This object is available as long as the "session" page directive is set to *true*. The object is created for each session.

- **application**

is a variable for *javax.servlet.ServletContext* object. This object is shared among the whole application.

- **out**

javax.servlet.jsp.JspWriter object. It corresponds to output stream to HTML.

[Example]

```
<%= request.getParameter("firstName")%>
```

The input parameter "firstName" is obtained from *request* object.

```
<% for (int i=0; i <=4; i++){
    out.println("Hello JSP with Scriptlets");
} %>
```

"Hello JSP …" message is output by "*out*" implicit object of *JSPWriter* class.

```
String pass = application.getInitParameter("pass");
```

The initialization parameter for *ServletContext* object is obtained by "*application*" implicit object.

3.2.8. Standard action

Some actions that may be frequently used are defined as Standard Action to ease the JSP programming tasks. The Standard action has the following tags.

1) useBean
2) getProperty
3) setProperty
4) include
5) forward
6) param

[Syntax]

```
<prefix:tagname>
```

- Prefix for Standard action is always "**jsp:**".
- If standard action consists of one tag, the tag should be closed with "/>"

[Example of Standard Action]

```
<jsp:useBean id="testBean" class="MyBean" scope="request" />
<jsp:include page="index.html" flush="true" />
<jsp:getProperty name="testBean" property="Name" />
```

1) jsp:useBean

A Java Bean is a Java class that is developed for business objects, especially to enable access to the variable data. It contains instance variables and *get/set* method for all the variables. The *useBean* is a tag to access to a Java Bean in order to get or set variables.

The Java Bean is especially useful for data exchange between:

- database access class and Servlet
- Servlet and JSP

[example of JavaBean]

```
private String strName;
// getter/setter method for strName
public String getStrName(){return strName;}
public String setStrName(String Name){strName=Name;}
```

For example, a database access class returns to the Servlet the Java Bean class that contains the result data retrieved from a database. The Servlet sends the Java Bean to JSP so that JSP can extracts necessary data from it. Then the JSP sets the dynamic data to the HTML to show on the Browser.

Table 2 - Attributes of JSP useBean tag

Attributes	
id	object name that is referred from inside the JSP page.
class	class name of *JavaBean* with package name
scope	Scope of the *JavaBean* to which is referred. If the bean exists in the scope, it is obtained. If it does not exist, then new bean object is created.
type	class type name of the id. It this is not defined, JavaBeans type is considered. The type is required to be either the class itself, a superclass of the class, or an interface implemented by the class specified.
beanName	name of a *JavaBean.* beanName can not be used together with "*class*" attributes

Table 3 - Scope of JSP useBean tag

Scope	Remarks
page	available within a single JSP page (default).
request	The bean is stored in the *HttpServletRequest*. It is available to the request

	object that is accessed by Servlets and JSP.
session	The bean is stored in the *HttpSession* object. It is available to the session object that is accessed by Servlets and JSP. Session object is not created automatically. It is created by *getSession(true)* method of *RequestObject*.
application	The bean is stored in the *ServletContext* object. It is available to the web application object that is accessed by Servlets and JSP.

[example]

```
<jsp:useBean id="object1"
           class="Beans.ExBean"
           scope="session" />
<%-- Create object1 of the ExBean object.
    Scope is session. --%>
```

In the above case, if the object exists in the scope, already declared object is obtained. If the object does not exist, it will be newly created. It is important to remember that JSP pages and servlets in the same web application share the same sets of bean collections. For example, a bean stored as a request attribute in a servlet like this:

[The object is set to session in somewhere]

```
…
HttpSession session = req.getSession();
ExBean obj = new ExBean();
…
// Store the obj object to Session with variable name "object"
session.setAttribute("object1", obj);
```

This object stored to the session attribute is visible from the JSP by standard action tag;

```
<jsp:useBean id="object1"
           class="Beans.ExBean"
           scope="session" />
```

The following scriptlet sample is equivalent to the <jsp:useBean> tag.

```
<%-- Declare to import the class --%>
<%@ page import="Beans.ExBean" %>
…
>

<%-- Get the ExBean object stored in the session with the name "object1"
```

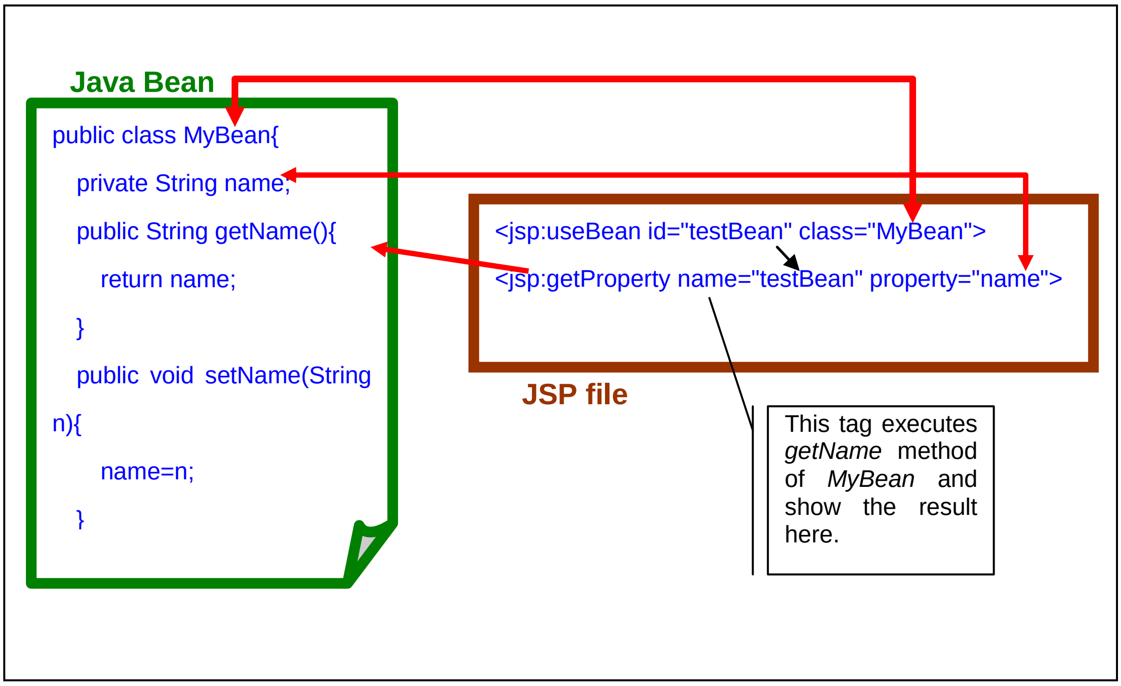
```
--%>
<% ExBean object1 = session.getAttribute("objet1");  %>
```

2) jsp:getProperty

Get the property value of the JavaBeans using Getter method of the property which is composed of "get" + "*PropertyName*". It is required that the JavaBeans is already instantiated (obtained) by <jsp:useBean> tag.

Table 4 - Attributes of JSP getProperty

Attributes of getProperty	Remarks
name	Object name same as the id defined in <jsp:useBean> tag.
property	Property name in the Object.



```
public class MyBean{
   private String name;
   public String getName(){
      return name;
   }
   public void setName(String n){
      name=n;
   }
```

```
<jsp:useBean id="testBean" class="MyBean">
<jsp:getProperty name="testBean" property="name">
```

This tag executes *getName* method of *MyBean* and show the result here.

Figure 48 example of JSP getProperty of useBean

3) jsp:setProperty

Set the property value of the JavaBeans using Setter method of the property which is composed of "set" + "*PropertyName*". It is required that the JavaBeans is already instantiated (obtained) by <jsp:useBean> tag.

Table 5 - Attributes of JSP setProperty

Attributes of setProperty	Remarks
name	Object name same as the id defined in <jsp:useBean> tag.
property	Property name in the Object.
value	Value to be set to the property.
param	Parameter that contains the value to be set to the property.

[example]

```
<jsp:setProperty name="object1"
              property="id"
              param="myParam" />

<%-- Call setID() method in the object named object1.
    The value is stored in the myParam parameter --%>

<jsp:setProperty name="object1" property="id" value="Hello" />

<%-- Call setID() method of the object named object1.
    The value of "Hello" is set. The setId("Hello") is executed.--%>
```

4) jsp:include

- The include tag includes the indicated resource to the JSP current page dynamically.
- The resources are included each time JSP receives request. Therefore, the modification of included resource is always reflected to JSP.

[Example]

```
<html>
    <body>
        <jsp:include page="Hello.jsp" flush="true" />
    </body>
</html>
```

Table 6 - Attribute of JSP standard action include tag

Attribute	Remarks
page	the resources to be included in JSP. Specify the relative path from the context root.
flush	flush of response buffer before execution of the file. default is false

It is possible to pass some parameters to the destination JSP file;

```
<jsp:include page="next.jsp">
  <jsp:param name="myParam" value="java" />
</jsp:include>
```

The destination JSP can get the parameter by *getParameter*();

[next.jsp]

```
<%! String strValue = getParameter("myParam");
%>
<%-- strValue will have "java" --%>
```

☆ The difference between <jsp:include> and "include directive" is that "include directive" includes the file at the compile time, while <jsp:include> includes it when a client sends request. As a result, <jsp:include> is convenient when an included file is decided dynamically at run time. Another difference is that include directive can refer the variables and methods defined in the original file, while <jsp:include> can not.

[TestJSPInclude.jsp]

```
<HTML>
<BODY>
<%-- declare boolean and set to true --%>
<%! boolean  bl = true; %>

<%-- if true, include true.txt and otherwise, false.txt
--%>
<%
  if (bl ) {
%>
  <jsp:include page="true.txt" />
<%
} else {
%>
  <jsp:include page="false.txt" />
<%
}
%>

</BODY>
</HTML>
```

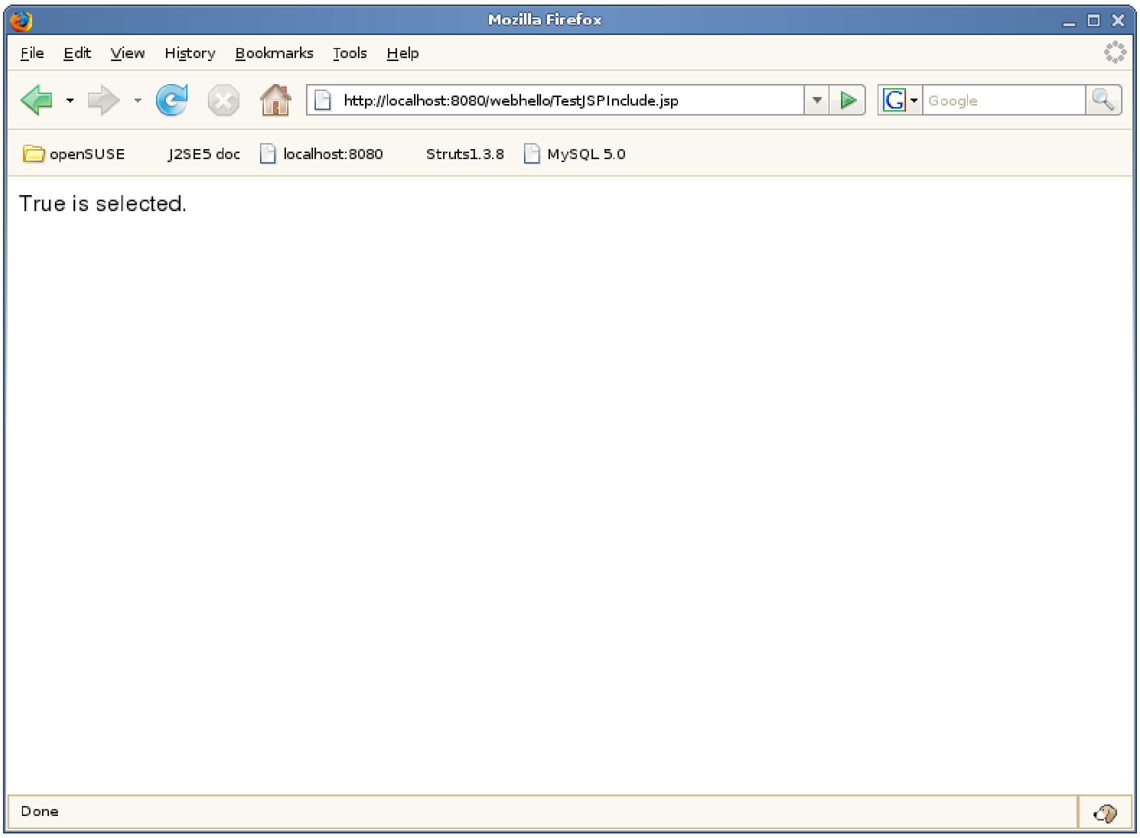
[true.txt]

```
true is selected.
```

[false.txt]

```
false is selected.
```

[result]



Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/webhello/TestJSPInclude.jsp
Google
openSUSE J2SE5 doc localhost:8080 Struts1.3.8 MySQL 5.0

True is selected.

Figure 49 Result of TestJSPInclude

5) jsp:forward

- The forward tag transfers control to the indicated resources.
- The results of the resources are displayed.

- Similar to the *RequestDispatcher* of the Servlet forward method (see 2.5.1)

[Example]

```
<jsp:forward page="error.jsp" />
```

6) jsp:param

The *param* tag provides parameters to include and forward tags.

[Example]

```
<jsp:forward page="error.jsp" >
  <jsp:param name="err1" value="I/O error has occurred" />
  <jsp:param name="err2" value="File not found" />
</jsp:forward>
```

[error.jsp]

```
<html>
<head><title>Error JSP</title></head>
<body>
<h1>Test JSP PARAMETER ; The following error has occurred;</h1>
<%= request.getParameter("err1") %>
</html>
```

3.3. JSTL (Java Server Pages Standard Tag Library)

3.3.1. What is Tag Library

There are several tags such as famous HTML tags of XML tags. Tag library is a collection of tags specialized for JSP. Tag library can be created with XML file which specifies the format of tag, and classes which is related to each tag to process some functions. When tag library is called in JSP, related class is instantiated, call the method. Therefore tag library can be described with "<" and ">" tag format, at the same time calling the method of the related class to process.

[benefits]

- Scriptlet of JSP, expression, declaration is replaced in tag format, which is more readable in the file.
- It is more reusable since there is no mixture of HTML design and Java code and easy to modify the HTML design. The library can be shared among the JSP, while scriptlet

is impossible to be shared.

[example of Tag library]

```
<%@ taglib uri="http://java.sun.com/jstl/fmt" prefix="fmt" %>
<% pageContext.setAttribute("date", new java.util.Date); %>

<fmt:formatDate value="${date}" pattern="yyyy/MM/dd hh:mm:ss" />
```

[example with scriptlet]

```
<%
    java.util.Date now
        =(java.util.Date)pageContext.getAttribute("now");
    java.text.SimpleDateFormat dateFormat =
        new java.text.SimpleDateFormat("yyyy/MM/dd hh:mm:ss");
    String nowDateString = dateFormat.format(now);
 %>
<%= nowDateString %>
```

Tag library is standardized by JCP(Java Community Process) to implements extended features of JSP tags. Declaration, outputs of variables, repetitions or some common functions are provided as tag library to increase programming task efficiency. There are two types of Tag Library:

- JSTL (Java Standard Tag Library) Standard-1.1 Taglib
- Custom Tag Library

Standard-1.1 (JSTL 1.1) requires a JSP container that supports the Java Servlet 2.4 and JavaServer Pages 2.0 specifications.

☆ Tomcat 5.0.X includes Servlet 2.4 and JSP 2.0 specifications.

3.3.2. How to install JSTL

1) Download JSTL files jakarta-taglibs-standard-1.1.2.tar.gz from the following site.

http://jakarta.apache.org/taglibs/doc/standard-doc/intro.html

2) Install JSTL jar files

Install jar files under WEB-INF/lib of Web Application directory.

3) Configure the JSP File declaring tag library in the JSP file

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

[Directory structure after installation]

```
WebApplication Directory
    ├── WEB-INF
    │       └── LIB       --- stardard.jar, jstl.jar
    │
    └── JSP       --- jsp files
```

3.3.3. How to work with JSTL

1) Declare tag library

[Example]

```
<%@ taglib uri="http://java.sun.com/jstl/fmt" prefix="fmt" %>
```

- **uri**

  Indicates the URI of TLD file.

- **prefix**

Table 7 - Prefix and uri of JSTL

prefix	URI	description
c	http://java.sun.com/jsp/jstl/core	core library. This includes store data, if condition, loop, import page, etc
fmt	http://java.sun.com/jstl/fmt	Internationalization, date format, obtain resources, etc
sql	http://java.sun.com/jstl/sql	SQL access, execute SQL query.
xml	http://java.sun.com/jstl/xml	process XML on the JSP

2) Core tag <c:set>

<c:set> is used to refer to variables.

[Syntax]

```
<c:set var="variable_name" value="value or expression" />
```

```
<c:set var="variable_name or expression "> value </c:set>
```

[Example]

```
<c:set var="a" value="0"/>
<c:set var="a">0</c:set>
```

☆You can also declare the variable with Scriptlet;

```
<% int a=0; %>
```

3) Core tag <c:out>

The <c:out> tag writes the value to the browser.

[Syntax]

```
<c:out value="value or variable" [default=default value]
[escapeXml=true/false]>
```

- The escapeXML is true when you want to display strings such as "<",">","&".
- The escapeXML is false when HTML tags such as <b>, <br> should be recognized as HTML tags.
- default value is set when the value is null.

[Example]

```
<c:out value="<h1> this is standard tag</h1>" escapeXml="false" />
```

[/jsp/Coretag.jsp]

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head></head>
<body>
  <c:set var="strMessage">
        Hello, This is JSTL core tag lib test page
  </c:set>
  <c:out value="${strMessage}" escapeXml="true" />
</body>
</html>
```

The above example shows that "*strMessage*" is a variable where the "Hello, …" message is stored with <c:set > tag. In the next line the content of the variable is output by <c:out> tag.

4) Core tag:<c:forEach>

The <c:forEach> is a tag to repeat process.

[Syntax1]

```
<c:forEach var="variable_name" begin="start_value" end="end_value"
step="incremental value">
```

[Example]

```
<c:forEach var="i" begin="1" end="5" step="1">
  <c:out value="hello" /><br>
</c:forEach>
```

[Syntax2] Iterating over a collection of objects.

```
<c:forEach var="variable_name" item="collection">
```

```
<c:forEach var="book" items="${sessionScope.bookList}" >
     <c:out value="${book.title}" />
     <c:out value="${book.author}" /><br>
</c:forEach>
```

The object is extracted from "*bookList*" collection from session scope and stored into the variable name "*book*". The values of title and author property are printed out in each iteration. This technique is quite convenient when the value is extracted in the table of HTML tag from an object repeatedly.

"sessionScope" is one of the implicit objects defined in Expression Language (see 3.3.4).

5) Core tag:<c:if>

The <*c:if*> is a tag for if conditional sentence.

[Syntax]

```
<c:if test="conditional_expression" var="variable_name" />
```

In *the variable_name* the result of the conditional sentence is set.

[Example]

```
<c:if test="${1< 5}" var="myVar">
  <c:out value=" 1< 5 is ${myVar}" /><br>
</c:if>
```

3.3.4. Expression Language

Expression Language is a description of variables and objects with expression. It can be used within Tag Library. The declaration of variables or objects is expressed with "$".

[Declaration of variable]

```
${variables or objects}
```

[Example]

```
<c:forEach var="i" begin="1" end="5" step="1">
 Hello JSTL tag ${i}
 <br>
</c:forEach>
```

In the above case the variable is output directly in the middle of HTML code.

```
<c:set var="val"> Hello Expression Language </c:set>
<c:out var="${val}">
```

This is an example of expression in side the JSTL tag.

If you express using general jsp tag:

```
<% String var=" Hello Expression Language "; %>
<%= var%>
```

The following operation is possible using EL.

- Mathematic operation (+ - * /)
- Relational operation (== != <> <= >=)
- logical operation (&& == !)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>

<html>
<head>
<title>Expression Language Sample</title>
</head>
<body>
    <c:set var="myVar5" value="5" />
    <c:set var="myVar8" value="8" />
    <c:set  var="myResult" value="${myVar5 * myVar8}" />
    <c:out value="${myResult}" /> <br/>
    <c:if test="${myResult < 50 }">
        <c:out value="${myResult} results less than 50" />
    </c:if>
```

```
</body>
</html>
```

Table 8 - EL and Standard JSP

EL example	Standard JSP	process
${value}	<%=value %>	Get value
${myBean.name}	<%= myBean.getName() %>	Get the value of name property in myBean object.
${requestScope.myBean}	% String value = (String)request. getAttribute("myBean"); %>	Get value of name property in myBean object in the request scope.
${param.name}	<% String value = request. getParameter("name"); %>	Get the parameter of "name" sent from the previous page
${paramValues.myCheck}	<% String[ ] values = request. getParameterValues("family"); %>	Get the value of checkbox named "myCheck" sent from the previous page.

Table 9 - Scopes in EL

Scope	Description	example
pageScope	Valid only inside the actual JSP page.	${pabeScope.user} Refer to the user object that resides in page scope.
requestScope	Scope for pages of request scope. This is valid between the original page and the destination page. This is used to share between those pages	${requestScope.name} Refer to the user object that resides in request scope.
sessionScope	Scope for session that is hold for certain period until session is closed, browser is closed or fixed time has passed. This is used to keep information such as ID or password.	${sessionScope. .id} Refer to the id object that resides in session scope.
applicationScope	Application scope is hold until JSP container is shutdown. This is created only one object for each application and shared by all.	${applicationScope.configA} Refer to the configA object that is held in application scope.

☆ You can get JSP Syntax reference and Standard Tag Library 1.1. implementation from:

- http://java.sun.com/products/jsp/syntax/2.0/syntaxref20.html
- http://jakarta.apache.org/taglibs/doc/standard-doc/intro.html
- http://java.sun.com/developer/technicalArticles/javaserverpages/faster/

Exercise 3: JSP programming

1. JSP loop programming. Follow the instructions.

1) Create a "jsp" directory under "webhello" project.

2) Create a JSP program "*TestJSPLoop.jsp*" under "jsp" directory.

3) Declare *int* type "i" variable in a Declaration tag. This variable is used for a counter.

3) Write a *for* loop that repeats five times to write out the string "Hello JSP with Scriptlets" and variable counter using Scriptlets and HTML tags. Use the variable declared with Declaration tag.

[output image]

```
Hello JSP with Scriptlets 1
Hello JSP with Scriptlets 2
…
```

4) Write another loop program using "*out*" of implicit object. Print out the string "Hello JSP with implicit object" and the loop counter.

[output image]

```
Hello JSP with implicit object 1
Hello JSP with implicit object 2
…
```

5) Call the "*TestJSPLoop.jsp*" from the Browser.

2. JSP Include tag programming. Follow the instructions.

1) Create a "*include*" directory under the "*webhello*" project.

2) Create a text file *"header.txt*" in the "include" directory. Write the following html tags.

```
<title> ICTTI Software Development: Java Programming Advanced
</title>
<head> JSP programming: practice of Include Directive and Standard
Action Include
</head>
```

3) Create another text file *"footer.txt"* and save it in the "*include*" directory. Write the following *jsp* codes.

```
<p>Executed in <%= new Date() %> </p>
```

4) Create a jsp file "*TestInclude.jsp*" and save it in the "jsp" directory. Write just <html> </html>tags and <body></body> tags correctly.

5) Include "*header.txt"* to the "*TestInclude.jsp*" using <jsp:include > tag of "Standard Action" between <html> and <body>.

6) Include *"footer.txt"* to the *"TestInclude.jsp"* using <%@ include > of Directive next to the <body> tag.

6) Call the *"TestInclude.jsp"* from the Browser and check if the both text files are correctly included.

☆ You have to import into the JSP page the necessary classes to refer.

3. JSTL programming.

1) Check if the "*standard.jar"* and *"jstl.jar"* are installed in the "*LIB*" directory of "*WEB-INF*" of your project.

2) Create a JSP file "*TestJSTLLoop.jsp*" under "*jsp*" directory.

3) Write JSTL core taglib <%@ taglib …> with prefix and uri for core taglib in the *"TestJSTLLoop.jsp*"

4) Write a "*for"* loop that repeats five times to write out the string "Hello JSTL Tag " and variable counter using JSTL <c:forEach> and <c:out> tags.

6) Call the "*TestJSTLLoop.jsp*" from the Browser.

4. Create a Servlet that dispatches the JSP page. Use the same input parameters of "*TestParam.html*" (name and password).

1) Copy *"TestParam.html"* to "*TestParamJSP.html*".

2) Copy "*ParamServlet.java"* to "*ParamServletWithJSP.java*".

3) Copy "*HelloJSP.jsp"* to "*HelloJSPwithServlet.jsp*".

4) Change "*ParamServletWithJSP.java"* to dispatch "*HelloJSPwithServlet.jsp*". Set the input parameter of "name" from the browser to request object, so that JSP can obtain the name.

5) Change "*HelloJSP.jsp"* to obtain the name from request object and output it.

6) Change the "*TestParam.html"* to call "*ParamServletWithJSP*".

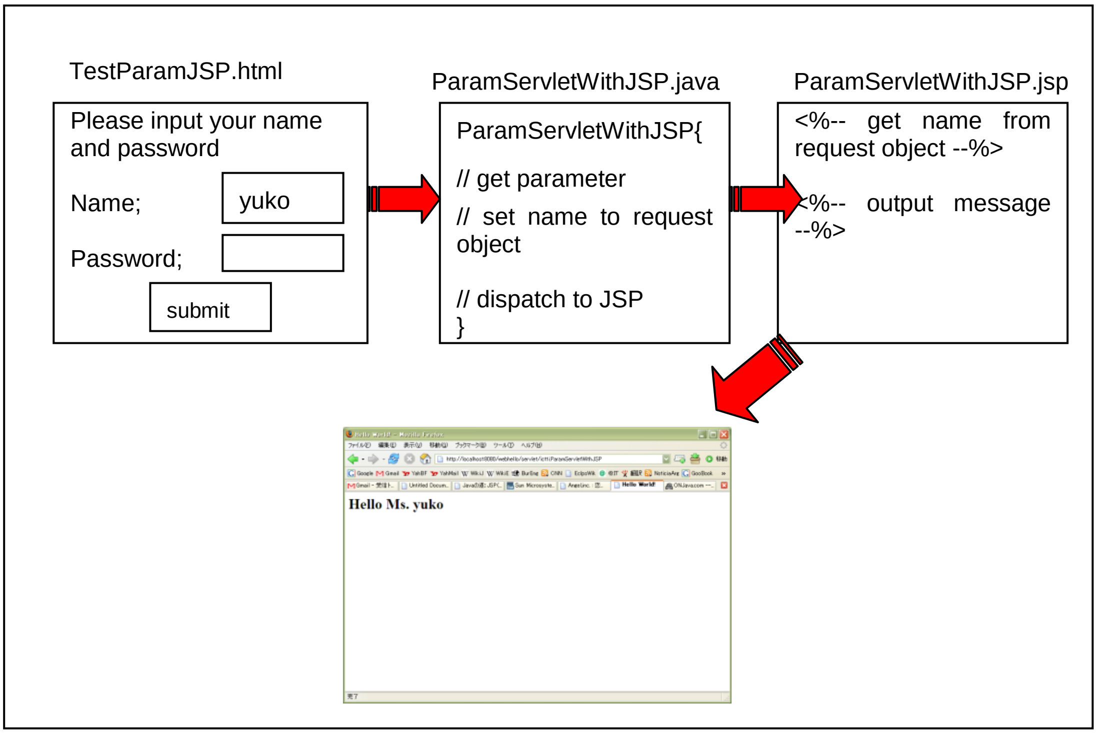


Figure 50 Exercise JSP 4

4. JDBC

4.1. JDBC

4.1.1. What is JDBC

- JDBC (Java Data Base Connectivity) is a standard Java interface specified by Sun Microsystems in order to access to DBMS (DataBase Management System) from Java.
- JDBC enables data access independent on DBMS. This means that a program is independent on the specific data access coding for specific database. Therefore it is possible to migrate to another database with minimum modification such as database connection or database driver.
- Similar to ODBC(Open DataBase Connectivity) provided by Microsoft.
- JDBC is oriented towards relational databases.
- The JDBC classes are contained in the Java core package java.sql of J2SE.

4.1.2. JDBC version and functionalities

Table 10 - JDBC Version

Version	Functionalities	Supported J2SE
JDBC1.0	- Basic DB access (connection to DBMS, create and execute SQL)	J2SE1.0
JDBC2.0	- Scroll forward and backward in a result set or move to a specific row - Make updates to database tables using methods in the Java programming language (instead of using SQL commands) - Send multiple SQL update statements to the database as a unit, or batch - Use the new SQL3 data types as column values - Create new SQL user-defined types (UDTs) - Map an SQL UDT to a class in the Java programming language - Make a connection that participates in connection pooling - Make a connection that can be used for a distributed transaction	J2SE1.2
JDBC3.0	- Savepoint support - Reuse of prepared statements by connection pools - Connection pool configuration - Retrieval of parameter metadata - Retrieval of auto-generated keys - Ability to have multiple open ResultSet objects - Passing parameters to CallableStatement objects by name - Holdable cursor support	J2SE1.4

	- BOOLEAN data type - Making internal updates to the data in Blob and Clob objects - Retrieving and updating the object referenced by a Ref object - Updating of columns containing BLOB, CLOB, ARRAY and REF types - DATALINK/URL data type - Transform groups and type mapping - Relationship between the JDBC SPI (Service Provider Interface) and the Connector architecture - DatabaseMetadata APIs	
JDBC4.0	- Auto-loading of JDBC driver class - Connection management enhancements - Support for RowId SQL type - DataSet implementation of SQL using Annotations - SQL exception handling enhancements - SQL XML support	J2SE6.0

☆ MySQL5.0 supports the JDBC3.0 specification.

4.1.3. How to install JDBC

(1) Obtain JDBC driver for each DBMS.

(2) Place the jar file that contains the JDBC driver in the directory where the classpath is defined. For Tomcat, place jar files under common/lib or WEB-INF/lib directory.

☆ The JDBC file for MySQL5.0 is found in the following site.

http://dev.mysql.com/downloads/connector/j/5.0.html

4.2. JDBC types

4.2.1. Type1: JDBC-ODBC bridge

JDBC driver gets access to database by way of ODBC of Microsoft. The driver converts JDBC method calls into ODBC function calls. The driver is implemented in the sun.jdbc.odbc.JdbcOdbcDriver class and comes with the Java 2 SDK, Standard Edition.

[Example of database]

- Microsoft Access

[Advantage]

- It is useful if the ODBC driver for the specific database is provided
- It can be applied for the database which JDBC driver is not provided but ODBC is

offered by the vendor, e.g. Microsoft Access.

[Disadvantages]

- Both JDBC for ODBC and ODBC driver for Database are required to be installed in a client machine.
- To connect with Database, it is necessary to pass two drivers. It takes more time and the functions are limited to what both drivers have.

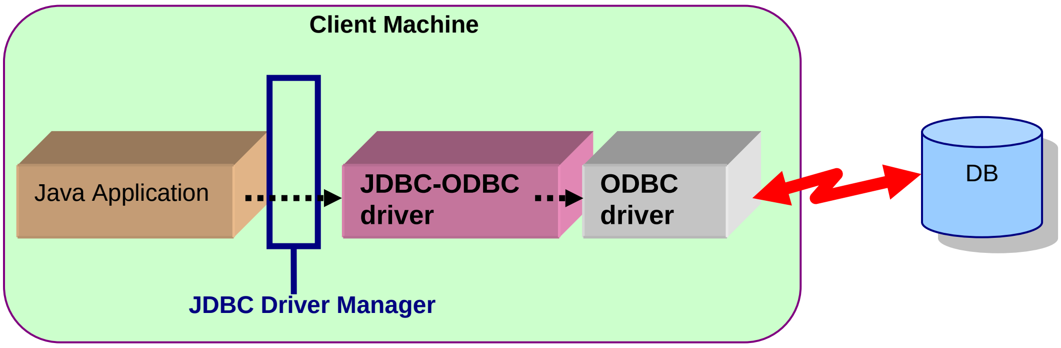


Figure 51 JDBC Type1

4.2.2. Type2: Native Bridge

- This type converts JDBC call to native specific DBMS protocol. which means that the type 2 driver is not written entirely in Java and the driver is compiled for use with the particular operating system.
- Platform dependent.
- The driver converts JDBC method calls into native calls of the database API.

[Advantage]

- ODBC driver is not required.
- Since the type2 does not have the overhead of the additional ODBC function calls, it provides more functionality and performance..

[Disadvantage]

- DBMS binary code should be installed and running on the client machine. For Oracle, Net8 client service should be running.
- Can not be used in the Internet from the browser because the client software is needed.

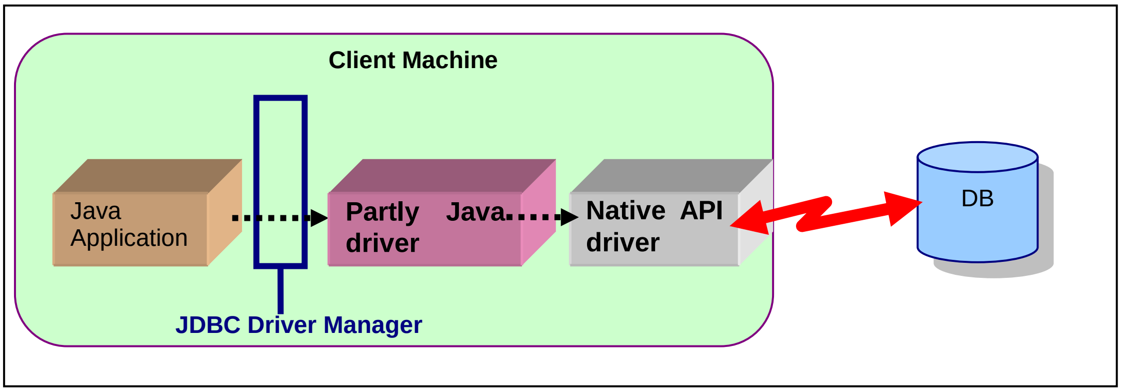


Figure 52 JDBC Type2

[Example of database]

- Oracle OCI (Oracle Call Interface) driver is Type 2. The Oracle net client library should be installed on the client machine. Therefore the OCI driver is Oracle platform-specific.

4.2.3. Type3: Net protocol driver

- A net protocol driver is described all with Java code. This type does not include native code.
- A net protocol communicates with middleware software running on a Server. This middleware service program converts JDBC call to vendor-specific native DBMS protocol. The protocol conversion logic resides not at the client, but in the middle-tier.

[Advantages]

- Net protocol driver is quite light.
- The Middleware Server can provide typical middleware services like connections, query results, load balancing, logging, auditing etc.
- No need of vendor specific JDBC driver.
- Client machine can be light while the middle service program has high functionality and performance

[Disadvantage]

- The middle service program should be running on the server machine.
- Another additional tier for Middleware Server.

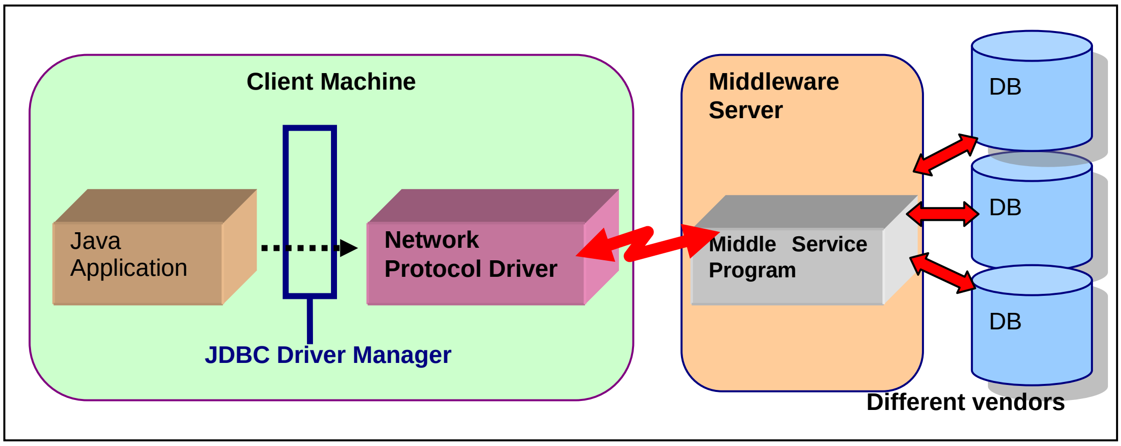


Figure 53 JDBC Type3

☆This is used when the client machine does not have enough hardware specification to execute high function of database. Usually this type is not applied for Web Application.

4.2.4. Type4: Native protocol driver

- The driver is written all in Pure Java code and does not include native code.
- The driver accesses to the database directly by way of network protocol used by DBMS. Because of this, the driver is platform independent; once compiled, the driver can be used anywhere.
- The driver communicates directly with a vendor's database through TCP/IP socket connections.

[Advantages]

- The native protocol driver is placed on the server.
- No installation is required on the client machine.

☆ MySQL provides a *type 4* JDBC driver.

[Disadvantages]

- Since all the functionality is written in Java, the driver size may be usually bigger than type3.

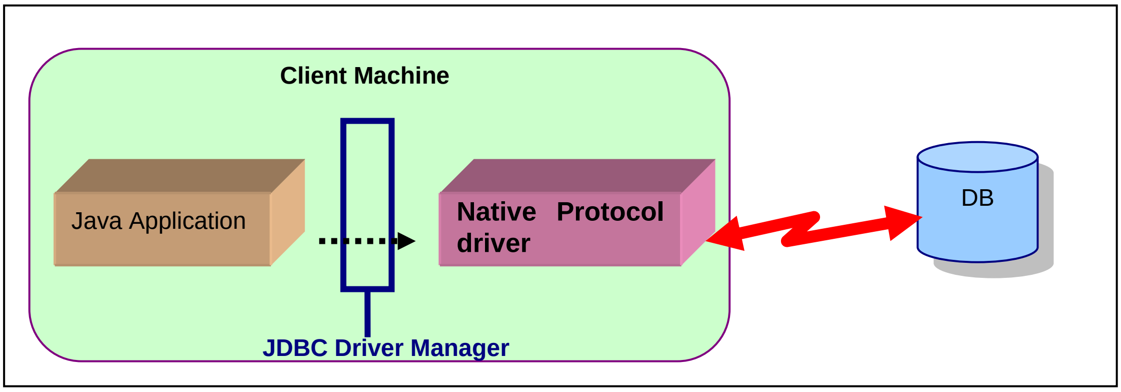


Figure 55 JDBC type4

4.3. How to connect to a database

The connection to a database is required before any data access. To connect with Database, first you need to load database driver.

4.3.1. Load the JDBC driver

To load the JDBC driver, *Class.forName()* method is used providing driver name as parameter.

[Driver loading example]

```
Class.forName("com.mysql.jdbc.Driver");
Class.forName("org.postgresql.Driver");
```

- **Connection**

The *getConnection()* method of the *DriverManager* class connects to the database and returns a *Connection* object. URL string, userID, and password are supplied for the connection.

[Example]

```
try {
  Class.forName("com.mysql.jdbc.Driver");
  String strURL = "jdbc:mysql://localhost/ictti";
  Connection con =
               DriverManager.getConnection(strURL,"root", "root");
  System.out.println("connection succeeded.");
}catch (Exception e){
  // error code
}
```

4.3.2. URL String

URL is a string to specify the host name for the connection.

- **Syntax**

```
jdbc:subprotocol:subname
```

Table 11 - URLString for JDBC connection

URLstring	description	example
jdbc	protocol name. Always *jdbc* is set.	
subprotocol	identifier of database.	(MySQL) "mysql" (Oracle) "oracle:thin" for type 4. (Postgres)" postgresql"

subname	depends on each database. Usually URLaddress, parameters,etc are set	(MySQL) //192.168.0.1/webapp
		(Oracle) @192.168.0.1:1521:ORCL
		(Postgres)//192.168.0.1/postgres

[Example of URL String]

- jdbc:postgresql://192.168.0.1/webapp
- jdbc:mysql://192.168.0.1/webapp
- jdbc:oracle:thin:scott/tiger@192.168.118.130:1521:ORCL

☆ URLstring depends on each database and sometimes it depends on the database version. Please see the database documentation to obtain correct URL String for JDBC connection.

4.3.3. DataSource

A DataSource is an interface of *javax.sql.DataSource* which is implemented by a dviver vendor. This is an alternative connection to the *DriverManager* facility providing connection pooling. Usually an object that implements the *DataSource* is registered as *JNDI* (Java Naming and Directory Interface) using *InitialContext* specification. A *DataSource* object is the preferred means of getting a connection since the connection or driver information is obtained from external configuration file. This enables the program to refer to only the data source name and allows system to change server or DBMS without any change in the program. This is quite important from the portability point of view.   .

[Example]

```
InitialContext ic = new InitialContext();
DataSource ds =
(DataSource)ctx.lookup("java:comp/env/jdbc/MySQLDB");
Connection con = ds.getConnection();
```

[Connection Pooling]

Connection Pooling is a technique to create and handle a pool of connections that are shared by threads who get access to JDBC. This is based on the facts that applications usually process a transaction that takes only few milliseconds to complete. When the transaction has finished, the connection is returned to the connection pooling. Then the connection pooling remains idle to be used by some other thread. The benefits of connection pooling are;

- Reduce the connection time
- Simplified the programming code
- Reduce the resource usage,and overhead (memory, CPU, context switches, etc)

4.3.4. DataSource setting with Tomcat 5.X

The J2EE Platform Specification requires J2EE Application Servers to make available a *DataSource* implementation (that is, a connection pool for JDBC connections). Tomcat 5 provides a JNDI InitialContext implementation instance for each web application according to J2EE specifications.

1) Place JDBC jar file in "*%TomcatInstallDir%/common/lib*". This JDBC driver will be used both from Tomcat and Web application. The restart is required to reflect the package.
2) Place the following code inside the **<Context> tag** of the "*conf/server.xml*" or web application context file found under "*%TomcatInstallDir%/conf/Catalina/localhost/*" (e.g. "*webhello.xml*")

```
<Resource name="jdbc/MySQLDB" scope="Shareable"
type="javax.sql.DataSource" />
<ResourceParams name="jdbc/MySQLDB">
    <parameter>
        <name>validationQuery</name>
        <value>select version();</value>
    </parameter>
    <parameter>
        <name>url</name>
```

Resource name is reffered from the program

```
        <value>jdbc:mysql://localhost/ictti</value>
    </parameter>
    <parameter>
        <name>password</name>
        <value>root</value>
    </parameter>
    <parameter>
        <name>maxActive</name>
        <value>4</value>
    </parameter>
    <parameter>
        <name>maxWait</name>
        <value>5000</value>
    </parameter>
    <parameter>
        <name>driverClassName</name>
        <value>com.mysql.jdbc.Driver</value>
    </parameter>
    <parameter>
        <name>username</name>
        <value>root</value>
    </parameter>
    <parameter>
        <name>maxIdle</name>
        <value>2</value>
    </parameter>
</ResourceParams>
```

Hostname DBName

- maxActive - The maximum number of active instances that can be allocated from this pool at the same time.
- maxIdle - The maximum number of connections that can sit idle in this pool at the same time.
- maxWait - The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception.

☆ Be careful of <Resource> tag which closes in one line. <ResourceParams> surrounds the datasource information inside.

3) Write source code to get connection to Database server by way of Datasource.

[webhello2/MysqlConnectDataSourceTest.java]

```
import java.io.IOException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.sql.DataSource;

public class MysqlConnectDatasourceTest extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req,
                        HttpServletResponse res)
            throws ServletException, IOException {
        String name = "Not Connected";
        int id = -1;
        try {
            Context ctx = new InitialContext();
            if (ctx == null)
                throw new Exception("Initial Context not
instantiated");

            // /jdbc/postgres is the name of the resource
            DataSource ds = (DataSource) ctx
                    .lookup("java:comp/env/jdbc/MySQLDB");

            if (ds != null) {
                Connection conn = ds.getConnection();
                if (conn != null) {
                    name = "Got Connection " + conn.toString();
                    Statement stmt = conn.createStatement();
                    ResultSet rst = stmt
                              .executeQuery("select id, name from
book");
                    while (rst.next()) {
                        name = rst.getString(2);
                        id = rst.getInt(1);
                        System.out.println("name = " + name);
                        System.out.println("id = " + id);
                    }
                    conn.close();
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

3) Configure the web.xml to be able to call above Servlet.

```
  <servlet>
       <servlet-name>MysqlConnectDatasourceTest</servlet-name>
       <servlet-class>MysqlConnectDatasourceTest</servlet-class>
   </servlet>

   <servlet-mapping>
       <servlet-name>MysqlConnectDatasourceTest</servlet-name>
       <url-pattern>/mysql</url-pattern>
   </servlet-mapping>
```

4) Execute the Servlet from browser.

```
http://localhost:8080/webhello2/mysql
```

4.4. How to retrieve and modify data

4.4.1. SQL

SQL (Structured Query Language) allows creating or accessing to the relational database where data is stored in a table. Many database vendors support SQL language for data manipulation. In JDBC connection, SQL is specified in the SQL statement to retrieve data.

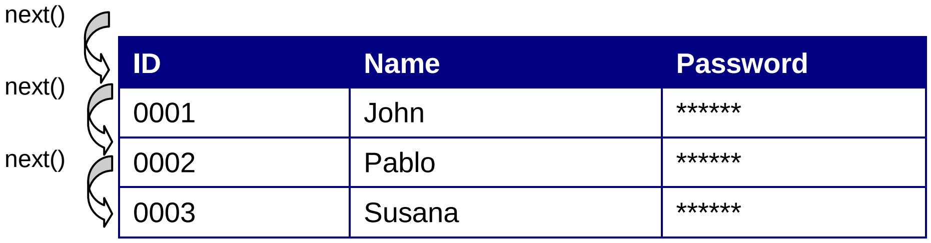
4.4.2. executeQuery method

To retrieve data, *Statement* object is created to specify SQL statement. Then the data specified in SQL is retrieved by way of *executeQuery* method of *Statement* class in JDBC.

[Example]

```
Statement stmt=con.creatStatement();
ResultSet resultSet = stmt.executeQuery("SELECT * FROM emp");
```

All the rows and columns of the result of retrieving data are set in *ResultSet* Object. The data of one row is obtained by calling *next*() method. Calling another *next*() method moves cursor (pointer of the data) to the next row, so that the data of the next row can be referred.



ID	Name	Password
0001	John	******
0002	Pablo	******
0003	Susana	******

Figure 56 JDBC next method of ResultSet

When the *ResulSet* is created, the first row is pointed by calling first *next*() method. After that it points to the data of next row. While the row is valid, the *next*() method returns *true*. It returns *false* when it arrives at the end of data.

The getter method is to retrieve the data in the columns of the "*ResultSet*". The data in the actual row is obtained either by column number or by column name. The getter method has ***getXXX*** format for all eight primitive types. For example *getInt()* method returns *int* type and *getLong()* returns *long* type. The *getDate()* and *getTime()* returns *Date* and *Time* of "*java.sql*" package respectively.

[Example of ResultSet]

```
String strQuery = "SELECT * FROM emp";
ResultSet rs = stmt.executeQuery(strQuery);
int i_ID;
String strName;
String strPwd;
while(rs.next()){
  i_ID = rs.getInt("ID");
  strName=rs.getString("NAME");
  strPwd=rs.getString(3);              // Get the third column
}
```

☆ You can get the data from "*ResultSet*" by column name or by column number.

☆ *SQLException* is thrown by the *createStatement()*, *executeQuery()* and *next()* methods.

4.4.3. How to modify data: executeUpdate

The *executeUpdate* method of *Statement* Interface modifies data for update, insert and delete in a database by setting SQL code as a parameter.

[Example of executeUpdate]

```
String strQuery = "UPDATE customer SET name = 'Pablo' WHERE id=0002";
int iRowCount = stmt.executeUpdate(strQuery);
```

```
String strQuery = "INSERT INTO customer (ID, Name, Password)" +
"VALUES (0010, 'Ana', 'pass0010')";
int iRowCount = stmt.executeUpdate(strQuery);
```

```
String strQuery = "DELETE FROM customer" +
"WHERE ID = '" + strID + "'";
int iRowCount = stmt.executeUpdate(strQuery);
```

☆ Inside the SQL string, single quotation (') is used to indicate the value.

☆ When *executeUpdate()* method is called, it returns an *int* value that identifies the number count of modified rows by SQL statement.

4.4.4. Close the connection

After the data is accessed, it is better close the connection.

- *close()* method

```
Statement.close()
Connection.close()
```

- The *ResultSet* is closed automatically when another SQL statement is executed.
- The *Connection* and *Statement* are collected by Garbage Collection when they are not used any more according to Sun specification. But it depends on how they are implemented by database vendors who implement the specification. It is recommended to close the connection explicitly.
- The best way to close the connection is in the final close that is executed both usual case and exceptional case.

```
    try{
    // write the usual case here
    } catch (SQLException e) {
        e.printStackTrace();
    //Close the objects although the exception happens
    }finally{
        try {
            rs.close();
            stmt.close();
            conn.close();
        }catch (Exception ex){
            ex.printStackTrace();
        }
    }
```

☆ In this manual this connection close is not always described due to saving of coding space, but if you implement some database access, it is better follow this way of close connection.

4.4.5. PreparedStatement

PreparedStatement is a statement prepared before the execution of SQL statements. The database server checks the statement for syntax error, prepare a plan and execute it. The SQL is precompiled in *PreparedStatement*. This improves the performance at the time of execution.

[How to work with *PreparedStatement*]

- A question mark (?) can be included in the SQL statement for values that are replaced later at the execution.
- Before the execution, real values are set to replace a question mark by setter method, such as *setString()* for *String* value, or *setInt()* for int value according to the data type.

[Example]

```
PreparedStatement ps = con.prepareStatement(
                        "UPDATE emp SET name=? WHERE id=?");
ps.setString(1,"Smith");// replace the first question mark
ps.setInt(2,12345);     // replace the second question mark
ps.executeUpdate();
```

4.4.6. JDBC transaction

Transaction is used to control to take an effect of data modification in accordance with other modifications. For example, if you need to write two SQL statements of Insert data to the different tables. You may not want one statement to take effect unless another one also succeeds for data consistency. A transaction can group the statements to take effect together at the same time.

1) Auto-commit mode

JDBC is set to auto-commit mode by default. The statement is automatically committed right after it is executed individually.

2) Committing a transaction

- Start of transaction

  - ***setAutoCommit(false)***

- End of transaction

  - ***commit*()**

    The statements are committed together as a unit

  - ***rollback*()**

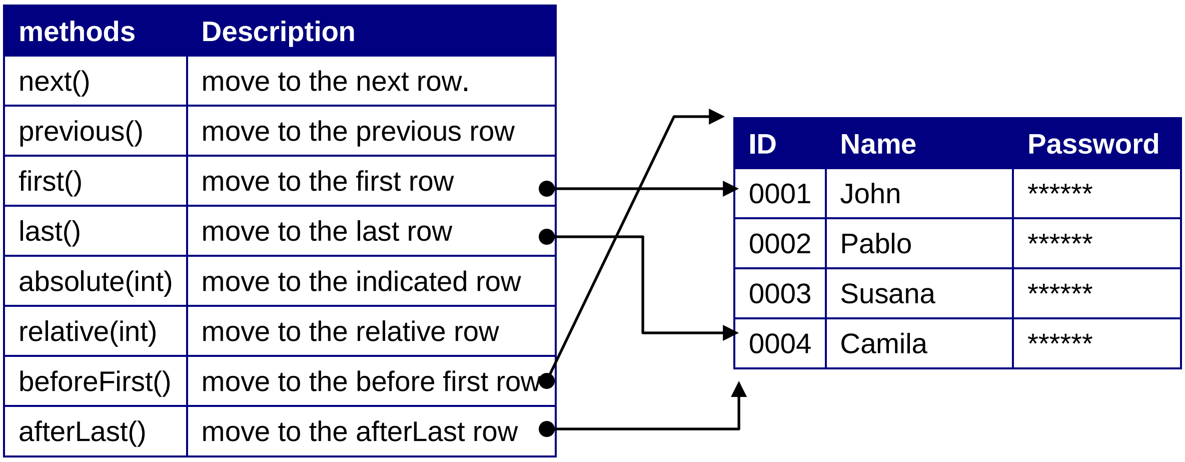
    The transaction is aborted and the values modified by SQL statements are returned to the previous values

[MysqlTransactionTest.java]

```
import java.sql.*;

public class MysqlTransactionTest {
    public static void main(String args[]){

        String url =
    "jdbc:mysql://localhost/ictti?user=root&password=root";
        Connection conn = null;
        Statement stmt=null;
        try{
            Class.forName("com.mysql.jdbc.Driver");
            conn = DriverManager.getConnection(url);

            // Set autocommit mode to false
            conn.setAutoCommit(false);

            //Insert row to the database
            stmt = conn.createStatement();
            String strSQL = "INSERT INTO emp(empno, ename, sal)" +
                              "values (101, 'Commit1', 0101)";
            int iRowCount= stmt.executeUpdate(strSQL);
            System.out.println("The row affected by insert is "
                              + iRowCount);

            stmt = conn.createStatement();
            strSQL = "INSERT INTO emp(empno, ename, sal)" +
                              "values (102, 'Commit2', 0102)";
            iRowCount= stmt.executeUpdate(strSQL);
            System.out.println("The row affected by insert is "
                               + iRowCount);

            conn.commit();
        }catch (SQLException e){
            e.printStackTrace();
            try {
                conn.rollback();
                System.out.println("The data is roll backed.");
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
        }catch (ClassNotFoundException e){
            System.out.println("The JDBC driver not found" +
             e.getMessage());
        }
    }
}
```

4.4.7. Scrollable cursor

Until JDBC2.0, there was only *next*() method for cursor movement. From JDBC2.0 the new cursor functions are included.



methods	Description
next()	move to the next row.
previous()	move to the previous row
first()	move to the first row
last()	move to the last row
absolute(int)	move to the indicated row
relative(int)	move to the relative row
beforeFirst()	move to the before first row
afterLast()	move to the afterLast row

ID	Name	Password
0001	John	******
0002	Pablo	******
0003	Susana	******
0004	Camila	******

Figure 57 New cursor features of JDBC

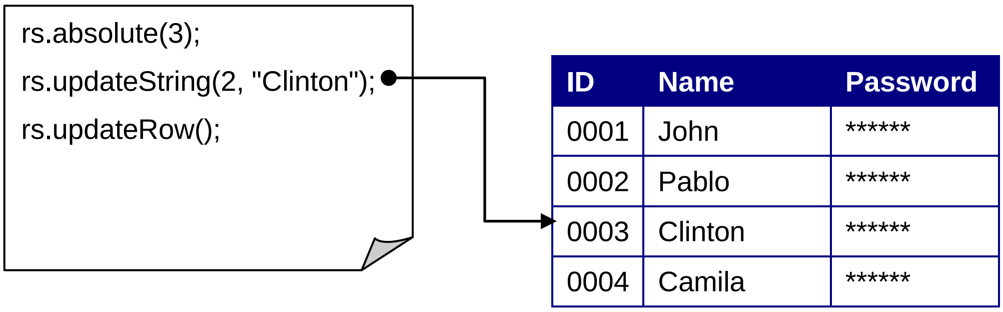
4.4.8. Programmable update

Instead of calling SQL statement for data modification, you can work with *ResultSet*.

1) Update by ResultSet

- updateString()
- updateInt()

- updateRow()



```
rs.absolute(3);
rs.updateString(2, "Clinton");
rs.updateRow();
```

ID	Name	Password
0001	John	******
0002	Pablo	******
0003	Clinton	******
0004	Camila	******

Figure 58 update by ResultSet

[MysqlResultsetUpdateTest.java]

```
import java.sql.*;

public class MysqlResultsetUpdateTest {
    public static void main(String args[]) {
        String url =
        "jdbc:mysql://localhost/ictti?user=root&password=root";
        Statement stmt = null;
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection conn = DriverManager.getConnection(url);
            // Create statement with scrorable and updatable
            stmt =
        conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                      ResultSet.CONCUR_UPDATABLE);
            String strSQL;

            // Select data from emp table
            strSQL = "SELECT empno,ename,sal,deptno from emp";
            System.out.println("The connection succeeded");
            ResultSet rs = stmt.executeQuery(strSQL);

            // update data from Resultset
            rs.absolute(3);
            rs.updateString(2, "JDBC2");
            rs.updateRow();
            rs.close();

            strSQL = "SELECT * from emp";
            rs = stmt.executeQuery(strSQL);
            int iID;
            String strName;

            // Close the objects
            conn.close();
            stmt.close();
            rs.close();
        } catch (ClassNotFoundException e) {
            System.out.println("ClassNotFound in
Class.forName()");
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

```
stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
```

Creates a Statement object that will generate *ResultSet* objects with the given type and concurrency.

The first parameter represents *resultSetType* and the second parameter represents *resultSetConcurrency*:

Table 12 - resultSetType

resultSetType	Description
ResultSet.TYPE_FORWARD_ONLY	Cursor movement forward only
ResultSet.TYPE_SCROLL_INSENSITIVE	scrollable but generally not sensitive to changes made by others
ResultSet.TYPE_SCROLL_SENSITIVE	scrollable and generally sensitive to changes made by others

Table 13 - resultSetConcurrency

resultSetConcurrency	Description
ResultSet.CONCUR_READ_ONLY	may NOT be updated
ResultSet.CONCUR_UPDATABLE	may be updated

2) Add by ResultSet

-moveToInsertRow

-insertRow

[Example]

```
rs.moveToInsertRow();          //move to the InsertRow
rs.updateInt(1,0005);          //set the column
rs.updateString(2,"Bush");
rs.updateString(3,"pwd001");
rs.insertRow();
```

ID	Name	Password
0001	John	******
0002	Pablo	******
0003	Clinton	******
0004	Camila	******
0005	**Bush**	**pwd001**

Figure 59 insert by Resultset

3) Delete by ResultSet

-deleteRow

[Example]

```
rs.absolute(5);
rs.deleteRow();
```

ID	Name	Password
0001	John	******
0002	Pablo	******
0003	Clinton	******
0004	Camila	******

Figure 60 delete by ResultSet

4) Other methods by ResultSet

Table 14 - Other methods　by ResultSet

Method	Description
moveToCurrentRow()	Return to the original cursor location after insert operation.
cancelRowUpdates()	Discard the update of the ResultSet before commit.
refreshRow()	Reread the record of the current row from the database.

4.4.9. Batch updates

Batch updates enable to send multiple updates to the database to be executed as a batch rather than sending each update separately. This will improve the performance because of the traffic cut between the server and client. The transaction should be set false to auto-commit mode so that the multiple statements can be sent together as the same transaction.

```
// Set transaction to manual mode
con.setAutoCommit(false);
Statement stmt = con.createStatement();
stmt.addBatch("INSERT INTO emp VALUES (1001, 'Pablo')");
stmt.addBatch("INSERT INTO departments VALUES (10, 'Product')");
stmt.addBatch("INSERT INTO emp_dept VALUES (1001,10)");
int [] updateCounts = stmt.executeBatch();
// Commit all the insert statement together
con.commit();
```

4.4.10.　Large Object

Large object (also known as BLOB or CLOB) can be created in JDBC2.0. BLOB (Binary Large OBject) is a collection of binary data stored as a column in a database management system. Typically images, audio, multimedia objects, or binary executable code is stored as BLOB. CLOB(Character Large Object) is a collection of text data stored in a database management system, generally applied for text over than 4000 characters length. CLOB max length is 4GB.

There are following limitations to use Large objects;

- Index can not be defined for BLOB or CLOB column.
- It can not be referred inside the where close.
- The row that contains BLOB or CLOB should be updated or inserted one row by one. Multiple rows can not be operated.

The following code shows how to store binary data to the database.

[Insert BLOB]

```
File file = new File("docu001.jpg");
FileInputStream fis = new FileInputStream(file);
PreparedStatement ps = conn.prepareStatement("INSERT INTO
myImageTable VALUES (?, ?)");
ps.setString(1, file.getName());
ps.setBinaryStream(2, fis, (int)file.length());
ps.executeUpdate();
ps.close();
fis.close();
```

For MySQL, the table should have been created using *BLOB(*65535 characters*)* or MEDIUMBLOB(16777215 characters) type to be able to store long data;

```
CREATE TABLE myImageTable (imgname text, img MEDIUMBLOB);
```

In case of PostgreSQL, the table should have been created using "*bytea"* type to be able to store binary data;

```
CREATE TABLE myImageTable (imgname text, img bytea);
```

The following code shows how to get binary data from the *ResultSet*.

```
PreparedStatement ps = conn.prepareStatement("SELECT img FROM
myImageTable WHERE imgname = ?");
ps.setString(1, "docu001.jpg");
ResultSet rs = ps.executeQuery();
while (rs.next()) {
    byte[] imgBytes = rs.getBytes(1);

}
rs.close();
ps.close();
```

PostgreSQL has another better suited way of Large Object feature to store huge binary data, since using *bytea* type for a large amount of data would require a huge amount of memory, although it is allowed to store until 1 GB of binary data. The Large Object feature stores the binary data in a separate table and refers to that table by type "*oid*".

First create table with oid data type;

```
CREATE TABLE imageslo (imgname text, imgoid oid);
```

The following code shows how to insert binary data using PostgreSQL

LargeObjectManager.

[PostgreLargeObjectTest.java]

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

import org.postgresql.largeobject.LargeObject;
import org.postgresql.largeobject.LargeObjectManager;

public class PostgreLargeObjectTest {
    public static void main(String args[]) {

        String url =
"jdbc:postgresql://localhost/postgres?user=postgres&password=post
gres";
        Connection conn = null;
        try {
            Class.forName("org.postgresql.Driver");
            conn = DriverManager.getConnection(url);

            // All LargeObject API calls must be within a transaction
block
            conn.setAutoCommit(false);
            // Get the Large Object Manager to perform operations with
            LargeObjectManager lobj =
        ((org.postgresql.PGConnection)conn).getLargeObjectAPI();

            // Create a new large object
            int oid = lobj.create(LargeObjectManager.READ |
LargeObjectManager.WRITE);

            // Open the large object for writing
            LargeObject obj = lobj.open(oid,
                        LargeObjectManager.WRITE);

            // Now open the file
            File file = new File("docu0001.JPG");
            FileInputStream fis = new FileInputStream(file);
            // Copy the data from the file to the large object
            byte buf[] = new byte[2048];
            int s, tl = 0;
            while ((s = fis.read(buf, 0, 2048)) > 0) {
                obj.write(buf, 0, s);
                tl += s;
            }

            // Close the large object
            obj.close();

            // Now insert the row into imageslo
            PreparedStatement ps = conn.prepareStatement("INSERT
INTO imageslo VALUES (?, ?)");
            ps.setString(1, file.getName());
            ps.setInt(2, oid);
            ps.executeUpdate();
            ps.close();
            fis.close();

            // Finally, commit the transaction.
            conn.commit();

        } catch (SQLException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            System.out.println("The JDBC driver not found" +
e.getMessage());
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

The following code shows how to retrieve binary data from PostgreSQL using LargeObjectManager.

[PostgreLargeObjectReadTest.java]

```
import java.io.BufferedOutputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import org.postgresql.largeobject.LargeObject;
import org.postgresql.largeobject.LargeObjectManager;

public class PostgreLargeObjectReadTest {
    public static void main(String args[]) {

        String url =
"jdbc:postgresql://localhost/postgres?user=postgres&password=post
gres";
        Connection conn = null;
        try {
            Class.forName("org.postgresql.Driver");
            conn = DriverManager.getConnection(url);

            // All LargeObject API calls must be within a transaction
block
            conn.setAutoCommit(false);

            // Get the Large Object Manager to perform operations with
            LargeObjectManager lobj =
((org.postgresql.PGConnection)conn).getLargeObjectAPI();

            PreparedStatement ps = conn.prepareStatement("SELECT
imgoid FROM imageslo WHERE imgname = ?");
            ps.setString(1, "docu0001.JPG");
            ResultSet rs = ps.executeQuery();
            BufferedOutputStream  buffer = new
        BufferedOutputStream(new FileOutputStream("myNew.jpg"));
            while (rs.next()) {
                // Open the large object for reading
                int oid = rs.getInt(1);
                LargeObject obj = lobj.open(oid,
                                LargeObjectManager.READ);

                // Read the data
                byte buf[] = new byte[obj.size()];
                obj.read(buf, 0, obj.size());
                buffer.write(buf);
                // Close the object
                obj.close();
            }
            buffer.close();
            rs.close();
            ps.close();

            // Finally, commit the transaction.
            conn.commit();

        } catch (SQLException e) {
            e.printStackTrace();

        } catch (ClassNotFoundException e) {
            System.out.println("The JDBC driver not found" +
e.getMessage());
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

☆ The above sample refers the *Postgre* JDBC 81 manual that comes with Database package. Please refer to the Database Vendor manual to get the correct information for your database version.

4.5. Oracle10g

4.5.1. JDBC drivers

Oracle10g comes with the following drivers that can be located in *Oracle_home*/jdbc/lib.

- **ojdbc14.jar**;	Contains the JDBC drivers for JDK1.4.
- **classes12.jar**	Contains the JDBC drivers for JDK1.2 and 1.3.
- **orai18n.jar**	Contains the classes that support full globalization.

You must set the CLASSPATH for your JDBC OCI or Thin driver. For example, for Tomcat,

- *Web_Application_Dir/WEB-INF/lib* directory (available only Web Application project).
- *Tomcat Home*/common/lib (available for all Tomcat project under Tomcat_Home) *are* used to place common libraries so that Tomcat can recognize the files.
- If you use Eclipse IDE, you have to include to the build path of the project. Click your project on the Package Explorer and right-click the mouse -> Build Path -> Configure

Build Path -> libraries of Java Build Path -> Add External JARs. Select your Oracle jar file.

For OCI drivers you must set the path to *ORACLE_Home*/bin for Windows. The "ocijdbc10.dll" is dynamic link library that the Oracle uses for the connection with database. For Thin Driver, you do not have to set any other environment variables.

4.5.2. JDBC Connection for Oracle10g

The JDBC connection has changed in Oracle10g. The following explication is "*Some Useful Hints In Using the JDBC Drivers*" extracted from "*readme.txt*".

```
1. Import the necessary JDBC classes in your programs that use JDBC.
    For example:

     import java.sql.*;
     import java.math.*;

    To use OracleDataSource, you need to do:
     import oracle.jdbc.pool.OracleDataSource;

 2. Create an OracleDataSource instance.

     OracleDataSource ods = new OracleDataSource();

 3. set the desired properties if you don't want to use the
    default properties. Different connection URLs should be
    used for different JDBC drivers.

     ods.setUser("my_user");
     ods.setPassword("my_password");

    For the JDBC OCI Driver:
     To make a bequeath connection, set URL as:
     ods.setURL("jdbc:oracle:oci:@");

     To make a remote connection, set URL as:
     ods.setURL("jdbc:oracle:oci:@<database>");

     where <database> is either a TNSEntryName
     or a SQL*net name-value pair defined in tnsnames.ora.
    For the JDBC Thin Driver, or Server-side Thin Driver:
     ods.setURL("jdbc:oracle:thin:@<database>");

     where <database> is either a string of the form
     //<host>:<port>/<service_name>, or a SQL*net name-value pair.

Oracle JDBC Drivers release 10.1.0.2.0 (10g) README.txt.
```

[example of OCI connection string]

```
ods.setURL("jdbc:oracle:oci:scott/ictti@ORCL");
```

Exercise 4: JDBC programming

The "emp" table and web application structure that are used in this exercise are as follows:

Table 15 - EMP table

Colomn name	Data Type	Size	
empno	INTEGER		primary key
ename	TEXT	10	
job	TEXT	9	
mgr	INTEGER		
hiredate	DATE		
sal	INTEGER		
com	INTEGER		
deptno	INTEGER		

Table 16 - Application information

Application information	value
package name	icttidb
user name for data access	root
password for data access	root
URL String	jdbc:mysql://localhost/ictti

☆ Please replace "*localhost*" to the database name if the database server is not located in your own machine.

1. JDBC Programming with SQL Insert statement. Follow the instructions.

(1) Create a Tomcat project *"Ex4JDBCPrj*".

(2) Include web.xml file under I directory.

(3) Create a servlet class "*DBEmpAccessInsert.java*" with the package name "*icttidb*" that connects to the MySQL database using Type4 connection. Do not forget to set JDBC jar file to the "build path" of Eclipse.

(4) Insert the following data using *INSERT* statement to the *GET* method.

- EMPNO; 10
- ENAME; "Pablo"
- SAL; 9999

(5) Retrieve the above data using *SELECT* statement.

(6) Print out the data retrieved from the above data.

(7) Call from the browser with the following URL:

```
http://localhost:8080/Ex4JDBCPrj/servlet/icttidb.DBEmpAccessInsert
```

In the template *"web.xml"*, the servlet is configured to be able to be called with "servlet" mapping name.

☆You can not insert twice the same data to the database since the "*empno*" column is a key of the table. You have to delete the data before execution if the data already exists.

2. JDBC Programming with SQL Update statement.

(1) Create a java class "*DBEmpAccess.java*" with the package name "icttidb" that connects to the database using Type4 thin connection in the same project *"Ex4JDBCPrj"*.

(2) Update the following data using UPDATE statement:

- EMPNO; 10
- DEPTNO; 40

(3) Retrieve the above data using SELECT statement.

(4) Print out all data retrieved from the above data.

3. JDBC programming with PreparedStatement.

1) Create a java class "*DBAccessEmpPre.java*" in the same project *"Ex4JDBCPrj".* Update the following data using PreparedStatement.

- EMPNO; 10

- ENAME; Smith

[PreparedStatement]

```
UPDATE emp SET ename=? WHERE empno=?
```

2) Retrieve the data and print them out the same way as the previous exercise.

5. J2EE architecture

J2EE Java 2 Enterprise Edition defines the standard for developing server side component based of enterprise application. J2EE is specified originally by Sun Microsystems. Actually the specification was developed under the Java Community Process. It includes technology of Servlet, JSP, JDBC, EJB(Enterprise Java Beans), Web service and XML to develop and run multi tiered architecture on an application server of distributed J2EE platform.
The current version is JEE5.0.

☆ The name was changed to Java Platform, Enterprise Edition Java EE in version 1.5 which the final release was made on May 11, 2006.

5.1. Application architecture

5.1.1. What is application architecture

Application architecture is an entire structure of application and common mechanism that contributes to the efficiency of development, scalability, portability or maintainability. For example, some user interfaces mechanism which all application in the system can commonly use, or mechanism to get access to the database that can be reused in different systems. According to the architecture It is convenient to design how to distribute the software components, and how to design the software packages.

[The purpose of application architecture]

- **Efficiency of development**

  Efficiency of development is one of the elements that are always required in the system development. Application architecture should be easily understandable by developers to be able to contribute to the ease of development. At the same time it is required that it helps to create test environment of the program.

- **Reusability**

  Application architecture designs the reusability of the component, such as the business logic that can be shared and reused in the enterprise.
  [example]
  - Session logic of the User Interface
  - User control of the User Interface

- Validation of the input parameter of the User Interface
- Tax calculation
-

- **Scalability and maintainability**

  Generally a user requirement tends to change or expand according to the market demand during the system life cycle. If all logics such as user interface, database access, and business logics are included in one component together, it is quite complicated to expand and modify the system. In case some exception happens, it takes a lot of time to locate where the problem resides. It is also possible that another web server or web application server will be installed additionally due to the increase of user access. The purpose of application architecture is to be expandable considering the scalability and maintainability of the system.

  [example]

  - Load balancing of Web server
  - Integration to other external system (ERP, Legacy system, Other platform, etc)
  - Clustering of Web Application Server
  - Mirroring system of DBMS (Always having mirror of the Database)

- **Portability**

  According to the requirements of scalability and maintainability, sometimes the immigration or replacement from one physical component to another will occur. In that case, if the software component resists to those changes of platform, dramatically the time and cost are saved. The application architecture also discusses the possibility of the portability, for example in case of changing DBMS, if the data access component is made independent to the DBMS, the high portability is expected. If the business logic components are limitedly placed in the business tier, changing application server will not affect to the business logic, The idea of multi tiers is one of the examples to be able to realize the portability.

  [example]

  - Replace application server of Tomcat to JBoss
  - Replace PostgreSQL to Oracle
  - Replace Mycrosoft to Linux

5.1.2. Multi Tiers design pattern of J2EE

J2EE Web Application Architecture consists of multi tiers: **Client**, **Presentation**, **Business,**

Integration and **Resource** tier.

Table 17 - Multi Tiers design pattern

Tier	Description	Examples
Client tier	Represents User Interface, User interaction, presentation of the screen.	- Web browser - Java applet - Flash - Ajax - JavaScript
Presentation tier	Represents the interface with users. This tier realizes four basic functions in a specific order: 1) interprets client requests, 2) dispatches those requests to business logic, 3) selects the next view for display, and 4) generates and delivers the next view.	- Struts Framework - JSF (Java Server Faces) - Servlet - JSP
Business tier	This is a tier where the specific application logic is performed, such as data access control, saving data received from the presentation tier or retrieving data to submit to the presentation tier.	- EJB (Enterprise Java Beans) - Control of Transactions between distributes database -Business logic such as Tax calculation - Spring Framework - Hibernate
Integration tier	Responsible for communication with external resources and systems such as data stores and legacy systems	- JDBC - Connectors to legacy systems
Resource tier	Contains business data, external resources	- DBMS - mainframes - legacy systems

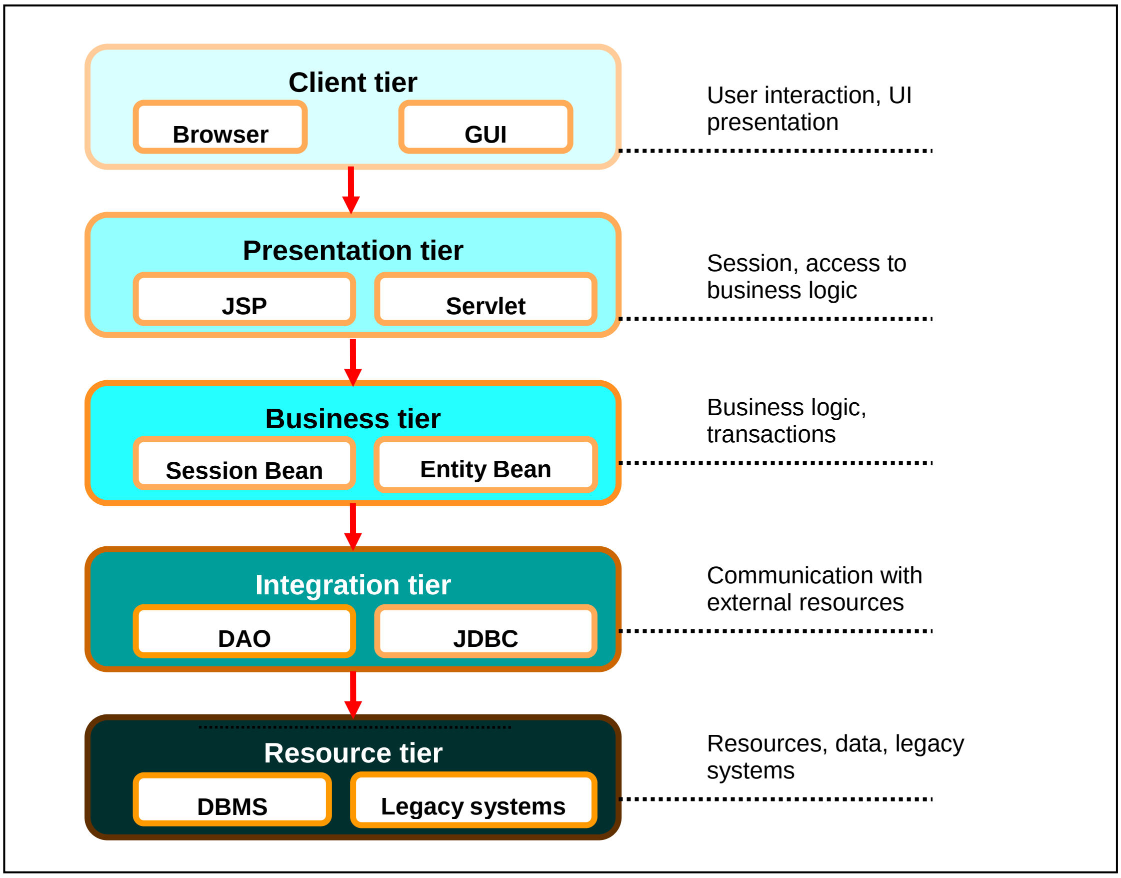

Client tier
Browser
GUI
User interaction, UI presentation
Presentation tier
JSP
Servlet
Session, access to business logic
Business tier
Session Bean
Entity Bean
Business logic, transactions
Integration tier
DAO
JDBC
Communication with external resources
Resource tier
DBMS
Legacy systems
Resources, data, legacy systems

Figure 61 J2EE multitiered architecture pattern

The multi tiered architecture represents logical tiers for the distributed Web application. The software components are placed in the tier according to its character of the service or process. The tier is also convenient to apply to physical tiers in order to discuss how to distribute the physical components.

The tier represents the independency of software and hardware to any other tier. For example the software component that implements the business logic is confined just inside the Business tier so that in case of any modification of this tier does not affect to other tiers. The developer can quickly find where to modify, which source to change in those cases. This contributes to the maintainability and scalability.

☆ Sharing some details with other tier increases coupling between tiers. For example, *HttpServletRequest* is a Presentation-tier data structure, that should not be passed to the next tier of Business tier same as the object *RelsultSet,* the data structure of JDBC should be confined inside the Business Logic, and should not be passed to the JSP or Servlet as a raw data.

☆ The tier is also applicable from the portability point of view, since it is independent physically. For example, the Presentation tier corresponds to the Web tier where Web Application Server is located. The Business tier corresponds to the Enterprise Java Bean tier where EJB container can be located. In case of some replacement from one application server to another server, or from POJO(Plain Old Java Object) to EJB or Vice Versa, this logical independency contributes to high portability and scalability.

5.2. MVC architecture

5.2.1. What is MVC architecture

MVC architecture is a Web application model that divides the application structure into three elements: ***Model***, ***View*** and ***Controller*** in order to increase efficiency and maintainability of the system development.

5.2.2. Structure of MVC

- ***model***

  Model consists of Java Beans (POJO or Enterprise Java bean) that defines data access or business logics.

- ***View***

  It consists of JSP or HTML file that presents the view to the browser.

- ***controller***

  It consists of Servlet that manages flow of the application, and determines how to handle a request and chooses the next view to display.

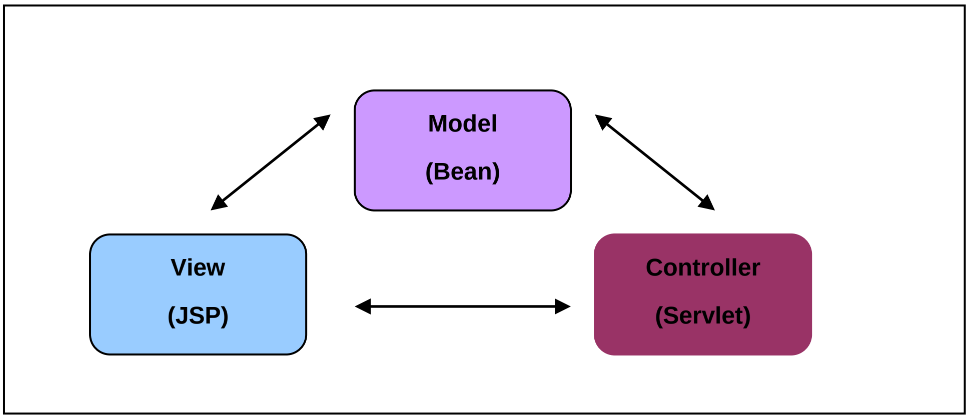


Figure 62 MVC architecture

5.2.3. Why MVC architecture

The most important purpose of MVC is to separate presentation and business logic. Inside the presentation tier, as we have seen before, the Presentation tier focuses on four basic functions in a specific order:

1) interprets client requests
2) dispatches those requests to business logic
3) selects the next view for display
4) generates and delivers the next view.

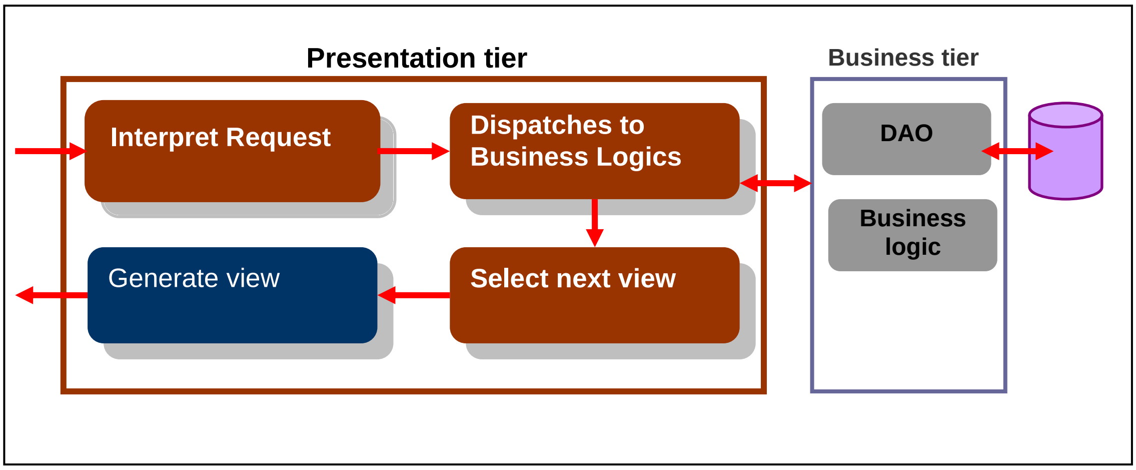


Figure 63 The rolls of Presentation tier

There are several models to design Web application;

- Before JSP Model

Before JSP technology appeared, Servlet covered all the presentation functions. This model is what we have learned in the beginning of the manual to see how a Servlet works. JSP appeared in the end of 1990s, until when the output of HTML code was embedded inside the Servlet.

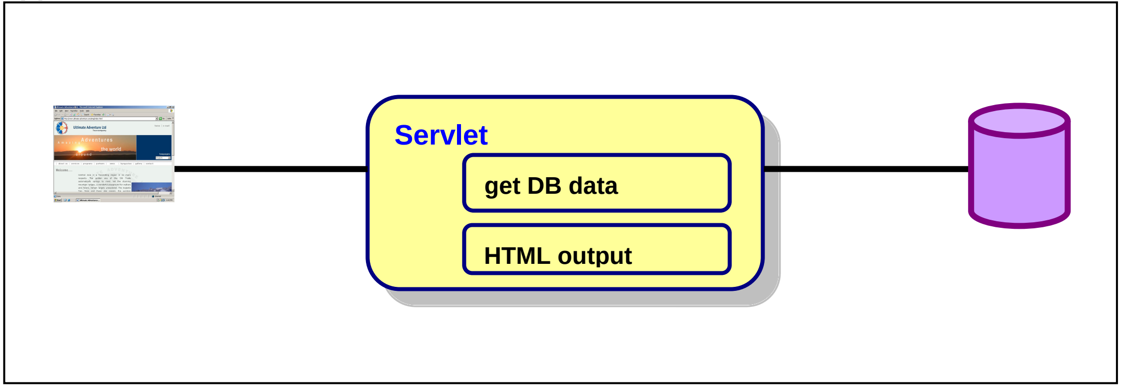


Figure 64 Servlet Model of Web Application

- JSP Model1

JSP first appeared in 1998. The official versions, 1.0 and 1.1, both appeared in 1999. Since Java code can be embedded into static HTML content without need to compile, the JSP technique was widely accepted. It is possible to include all the business logic, or database access logic in JSP tags. A Model 1 architecture consists of a Web browser directly accessing Web-tier JSP pages

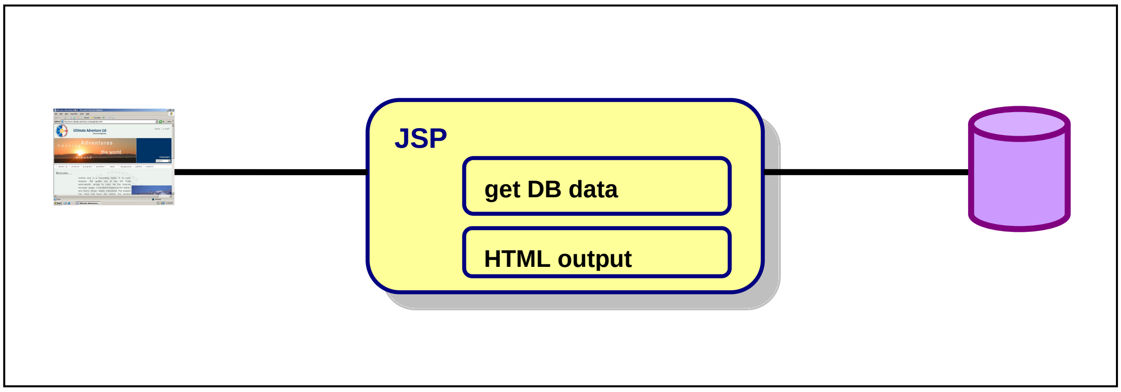


Figure 65 JSP Model1 of Web Application

- JSPModel1.5

Another model of JSP is that POJO(Plain Old Java Object) is used to separate the business logic and database access logic. The JSP pages access to Java Objects that represent the application model,

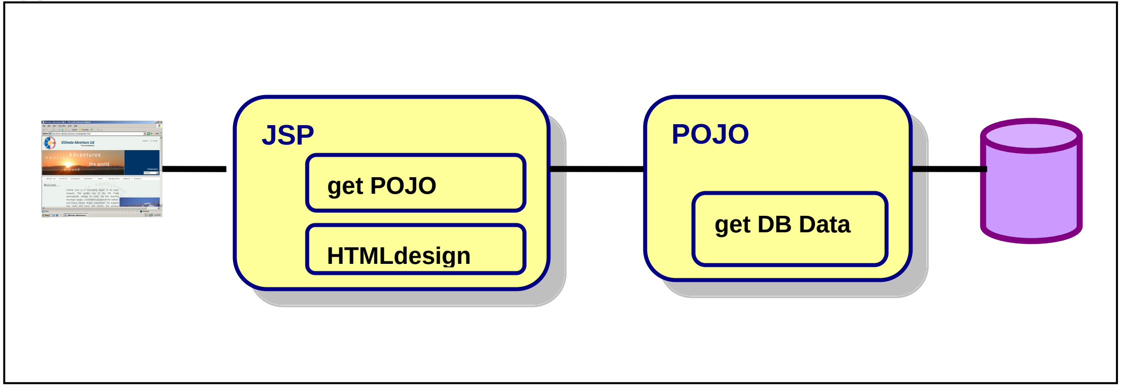


Figure 66 JSPModel1.5 of Web application

- JSPModel2(MVC2)

Model 2 architecture introduces a controller servlet between the browser and the JSP pages. As a business layer, Plain Old Java Object (not EJB) is introduced.

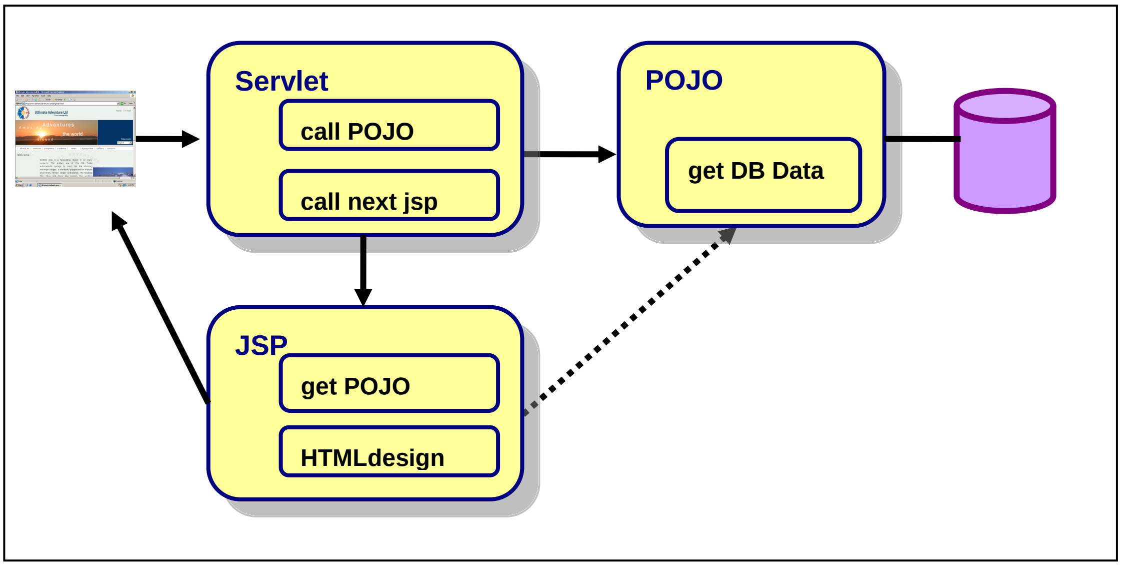


Figure 67 MVC2 of Web application

- JSPModel2(MVC2 with EJB)

A Model 2 with EJB architecture introduces EJB as business logic and data access functionality.

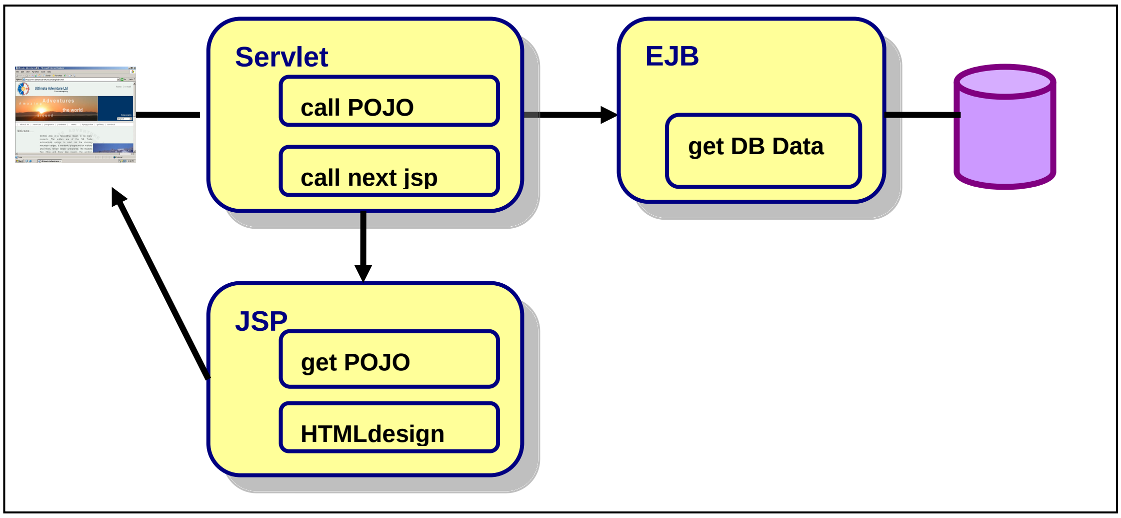


Figure 68 MVC2 with EJB of Web application

[The merits of MVC pattern]

- **Clarity**

MVC pattern provides clarity of the programs and roll of the components since the code is separated according to the pattern. For example, you can code in *Servlet* everything such as database access and HTML output which causes complexity of the program and makes the program difficult to maintain.

- **Independency on the objects**

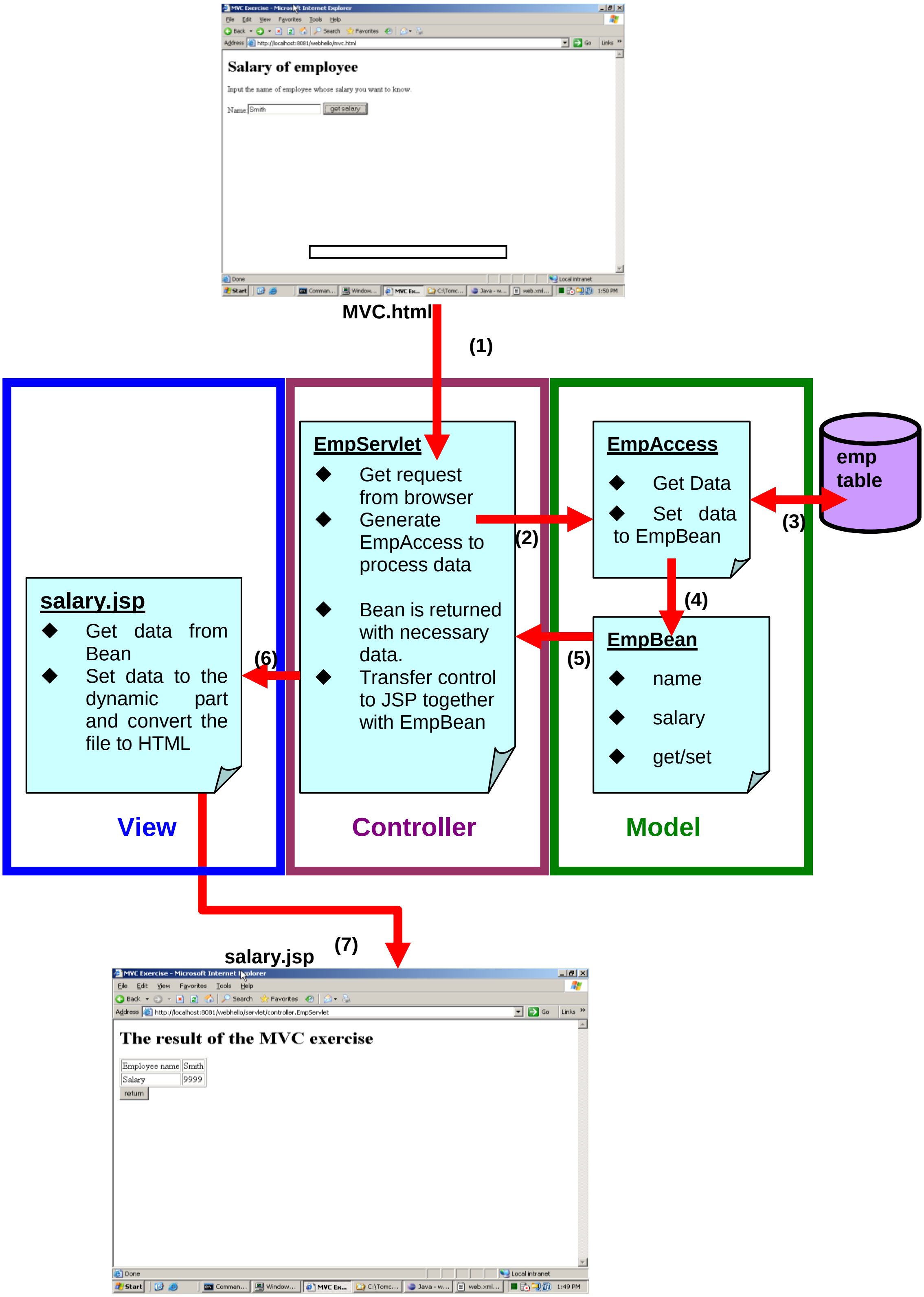
Since the three patterns have the well defined functionality, a developer can easily locate which program to modify and maintain.

- **Reusability**

Since the each element is concentrated in each role, it can be reusable in other systems or other applications. Especially if the business logic is described independently, the reusability is quite high.

5.2.4. Example of MVC application

This is a quite simple Web Application that a user input name of an employee and submit the page. The next page shows salary of the employee retrieved from the "*emp*" table of database.



Salary of employee

Input the name of employee whose salary you want to know.

Name Smith get salary

The result of the MVC exercise

Employee name	Smith
Salary	9999

return

MVC.html
(1)
EmpServlet
◆ Get request from browser
◆ Generate EmpAccess to process data
◆ Bean is returned with necessary data.
◆ Transfer control to JSP together with EmpBean
(2)
EmpAccess
◆ Get Data
◆ Set data to EmpBean
(3)
emp table
(4)
EmpBean
◆ name
◆ salary
◆ get/set
(5)
(6)
salary.jsp
◆ Get data from Bean
◆ Set data to the dynamic part and convert the file to HTML
View
Controller
Model
salary.jsp
(7)

Figure 69 Flow of the MVC Application

[Flow of the sample application]

(1) The user calls the MVC.html to input employee name.

(2) The *EmpServlet* (= Controller) receives the request from the browser. It obtains the Name parameter. The Servlet then instantiates the *EmpAccess* class (= Model) passing the Name parameter to process data.

(3) The *EmpAccess* class gets the salary data from the *emp* table.

(4) The *EmpAccess* class sets the salary data to the *EmpBean* by *setSalary()* method.

(5) The *EmpServlet* gets return of *EmpBean* that contains salary data.

(6) The *EmpServlet* sets the *EmpBean* to the request object and transfers control to sajary.jsp (=View).

(7) The *salary.jsp* sets salary data to the jsp tags and converts it to HTML. The jsp sends the response to the browser.

(8) The result page of *salary.jsp* is shown displaying the salary of the employee.

[mvc.html]

```
<html>
<head>
    <title>MVC Exercise</title>
</head>
<body>
<h1>Salary of employee</h1>
Input the name of employee whose salary you want to know.<br>
<form action="servlet/controller.EmpServlet" method="POST">
    Name:<input type="text" name="Name">
         <input type="submit" value="get salary" >
</form>
</body>
</html>
```

[EmpServlet.java]

```
package controller;

import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import model.EmpAccess;
import model.EmpBean;

public class EmpServlet extends HttpServlet {

    protected void doPost(HttpServletRequest req,
                         HttpServletResponse res)
                throws ServletException, IOException {
        // Get parameter of the employee name
        String strName = req.getParameter("Name");

        // Generate the EmpAccess class
        EmpAccess empAccess = new EmpAccess();

        // Call getSalary method of EmpAccess that returns the EmpBean
with retrieved data
        EmpBean empBean = empAccess.getSalary(strName);

        // Set the javabean to the request object
        req.setAttribute("myBean", empBean);

        // Dispatch the JSP
        RequestDispatcher rd =
                req.getRequestDispatcher("/jsp/salary.jsp");
        rd.forward(req, res);
    }
}
```

```
req.setAttribute("myBean", empBean);
```

The *setAttribute*() method stores an attribute in this request. This method is most often used in conjunction with *RequestDispatcher* class. The first parameter specifies the name of the attribute and is referred with this name. The second parameter specifies the object to be stored.

[EmpAccess.java]

```
package model;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class EmpAccess {
    public EmpBean getSalary(String a_strName) {

        // Generate EmpBean to set data retrieved from Database
        EmpBean empBean = new EmpBean();
        String url =
        "jdbc:mysql://localhost/ictti?user=root&password=root";

        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;
        try {
            Class.forName("com.mysql.jdbc.Driver");
            conn = DriverManager.getConnection(url);
            stmt = conn.createStatement();
            System.out.println("The connection succeeded");

            String strSQL;
            // select row
            strSQL = "SELECT sal from emp where ename= '" + a_strName
+ "'";
            System.out.println("SQL Statement; " + strSQL);
            rs = stmt.executeQuery(strSQL);
            int iSal = 0;

            // retrieve data from ResultSet
            while (rs.next()) {
                iSal = rs.getInt("sal");
                System.out.println("ename; " + a_strName);
                System.out.println("sal: " + iSal);
            }
            // set data to the EmpBean
            empBean.setName(a_strName);
            empBean.setSalary(iSal);

        } catch (SQLException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } finally {
            try {
                // Close the objects
                rs.close();
                stmt.close();
                conn.close();
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }
        return empBean;
    }
}
```

[EmpBean.java]

```
package model;

public class EmpBean {
    private String name;
    private int salary;
    public String getName() {
    return name;
    }
    public void setName(String name) {
    this.name = name;
    }
    public int getSalary() {
    return salary;
    }
    public void setSalary(int salary) {
    this.salary = salary;
    }
}
```

[salary.jsp]

```
<jsp:useBean id="myBean" class="model.EmpBean" scope="request"/>
<html>
```

```
    <head>
        <title>MVC Exercise</title>
    <head>
  <body>
    <h1>The result of the MVC exercise </h1>
    <table border="1">
        <tr>
            <td>Employee name </td>
    <td><jsp:getProperty name="myBean" property="name"/></td>
        </tr>
        <tr>
            <td>Salary</td>
    <td><jsp:getProperty name="myBean" property="salary"/></td>
        </tr>
    </table>
    <input type="button" value="return"
        onclick="javascript:history.back()">
  </body>
</html>
```

```
<jsp:useBean id="myBean" class="model.EmpBean" scope="request"/>
```

The jsp gets data from *model.EmpBean* class with "*myBean*" attribute name that was stored to Request scope of *HttpServletRequest*.

```
<td><jsp:getProperty name="myBean" property="name"/></td>
```

You can obtain the data in the bean class by <jsp:getProperty> tag to show on the browser.

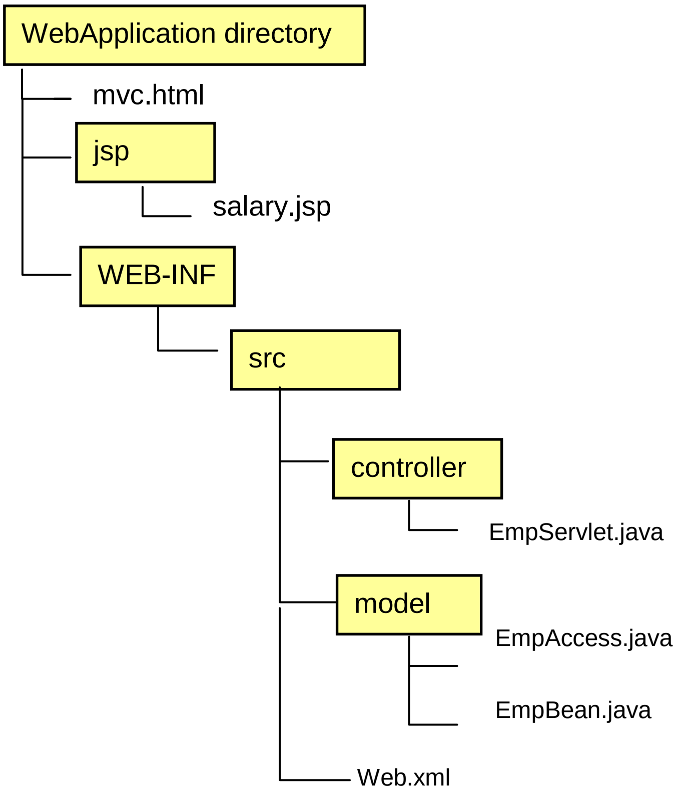


Figure 70 Directory structure of sample of MVC Web Application

[How to call from the Browser]

```
http://localhost:8080/web_application_name/mvc.html
```

☆Do not forget to set CLASSPATH for JDBC drivers.

5.3. EJB

5.3.1. What is EJB

- EJB(Enterprise Java Bean) is a Java server side component that enables to implement business logic.
- It facilitates to manage transaction or data access.
- EJB is specified in J2EE.
- Usually EJB is called from Servlet.
- The principal difference between Java Bean and Enterprise Java Bean is that EJB is designed to be able to execute in the remote and distributed machine under EJB server (or container) while Java Bean is a simple Java class working in the same JVM as the

Presentation tier.

- EJB are deployed in an EJB container within the application server. You need to set up some application server to execute EJB application.

5.3.2. EJB merits

- EJB can manage distributed objects that are located separately in the network.
- EJB can manage transactions.
- EJB has object caching that keeps objects in the memory.
- By EJB objects can map to relational database (O/R mapping). The objects managed and modified in the program are automatically reflected to the database.

5.3.3. EJB demerits

- Implementation of EJB is quite complicated. Some interfaces and deployment descriptor file are required beforeJ2EE1.4.

5.3.4. What kind of system is better to apply EJB compared with Servlet

- If the system only retrieves data to show on the browser, it is better make a program using Servlet. EJB has little merit for only data retrieve.
- If the system needs complicated transaction management for data consistency, it is better apply EJB than Servlet, since EJB can implement RollBack of database easier.

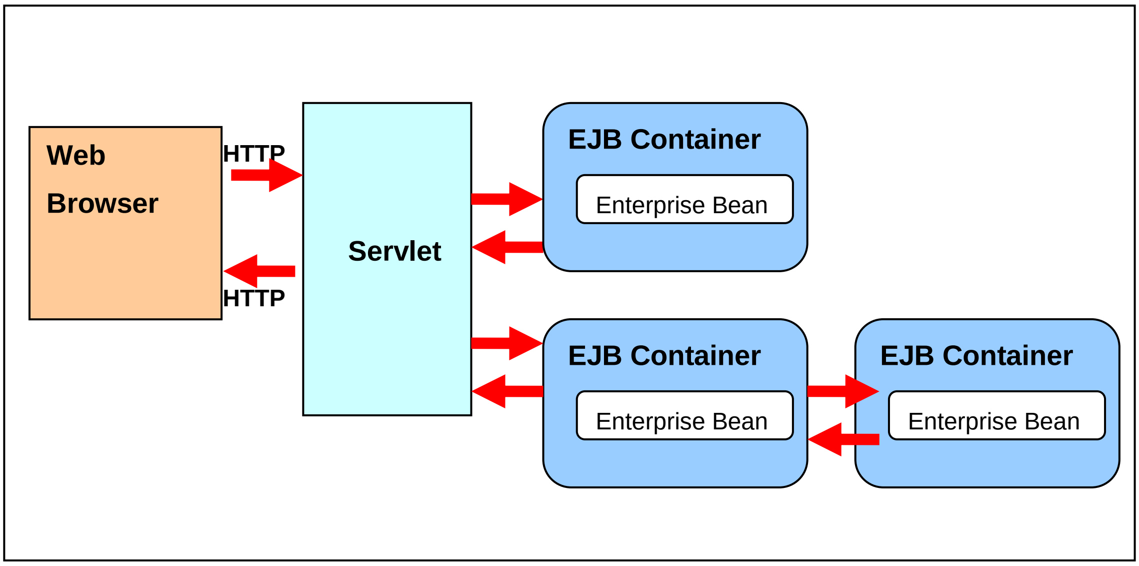


Figure 71 Example of Remote EJB communication

5.3.5. Type of EJB

2) Session Bean

In Session Bean, EJB is created for each client to realize business logic.

- **Stateless Session Bean**

  Session data of the client in Stateless Session Bean is not kept. This is like a function call that returns the same result.

- **Statefull Session Bean**

  Session data of the client is kept to associate EJB with a client. This is similar to use session object with Servlet. Shopping cart is an example of Statefull Session Bean.

3) Entity Bean

The business logic is shared by clients. Usually entity bean corresponds to database access operation.

- **Bean Managed Persistence (BMP)**

  SQL is required to be described in a program for data operation.

- **Container Managed Persistence (CMP)**

  EJB takes care of data operation. SQL description is not required.

5.3.6. EJB3.0

EJB3.0 was released in May 2006. The new EJB has introduced the annotation technology of J2SE5 to make development easier. This new feature uses annotation that is embedded in the source code as @tag so that the compiler can interpret the configuration information from the source code and not from complicated external configuration xml file.

The following program demonstrates the stateless session bean example in EJB3.0. From the annotation, it is clear that this is stateless EJB using stub for local.

[Calculator.java] **Remote Interface**

```
package trail.slsb;

public interface Calculator {
    public double calculate (int start, int end, double growthrate,
double saving);
}
```

[StatelessCalculator.java] **Implemented class**

```
package trail.slsb;
import org.jboss.annotation.ejb.LocalBinding;
import javax.ejb.*;

@Stateless
// @LocalBinding (jndiBinding="EJB3Trail/slsb/Calculator")
public class StatelessCalculator implements Calculator {
  public double calculate (int start, int end, double growthrate,
double saving) {
    double tmp = Math.pow(1. + growthrate / 12., 12. * (end - start)
+ 1);
    return saving * 12. * (tmp - 1) / growthrate;
  }
 }
```

(sample code from http://trailblazer.demo.jboss.com/EJB3Trail/)

☆ You need JDK5 to compile EJB3 program files.

5.3.7. JBoss

To work EJB, EJB container (or EJB server) is required. JBoss is open source J2EE server that enables implement EJB container. The official site is

http://www.jboss.org/

You can download the setup file from the above site.

JBoss has plug-in for Eclipse which you can download from the following site.

http://www.jboss.com/products/jbosside/downloads

Be sure of the version of Eclipse. You have to choose correct version of plug-in for JBoss. Since JBoss comes with Tomcat, the EJB can be called from Servlet as client.

The sample program of EJB3.0 using JBoss4.x Application Server is well explained in;

http://trailblazer.demo.jboss.com/EJB3Trail/

The simple example of session bean is demonstrated in;

http://trailblazer.demo.jboss.com/EJB3Trail/serviceobjects/slsb/index.html

5.4. Web Service

5.4.1. What is Web Service

Web Service is a distributed application that provides data services across the Internet. Real time information about bonds, validation check for credit card, actual weather forecasts, hotel reservation, and ticket reservation will be some examples of Web Service. It is true that the actual Web Application has already distributed application technology such as DCOM (Distributed Component Object Model), CORBA(Common Object Request Broker Architecture), and JAVA RMI(Remote Method Invocation) locating business components in the remote servers. But their technique is quite original without allowing transparent data exchange between different technologies, which causes the limitation of platform, data exchange format and protocols. Web Service is a technology standardized officially using open protocols such as HTTP or SOAP(Simple Access Protocol). Actually Microsoft, Sun Microsystems, IBM and other enterprises collaborate for Web Service standardization.

The following figure shows the traditional way to get services;

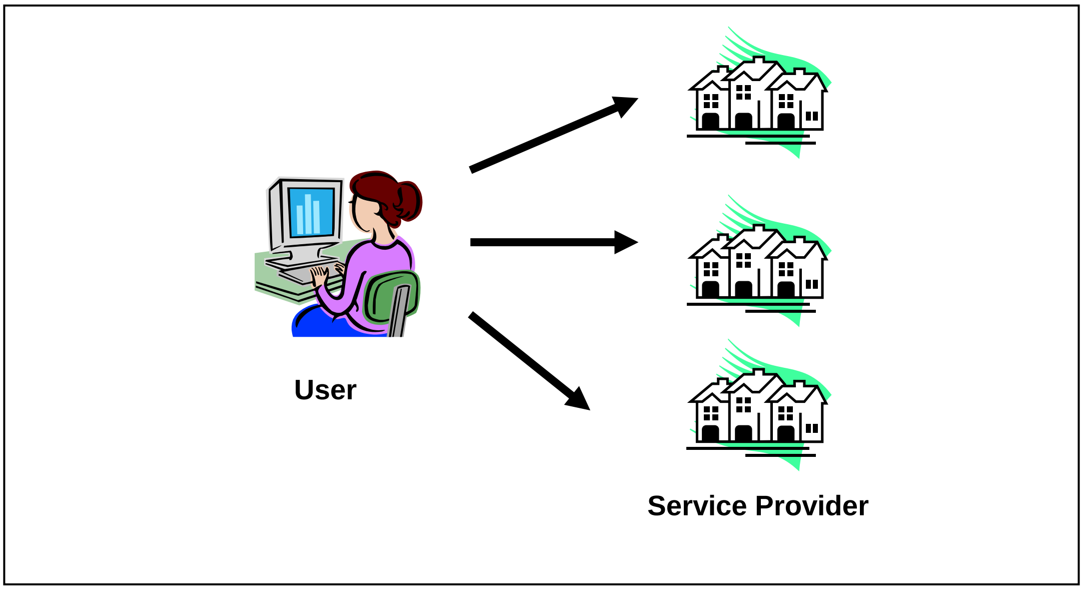


Figure 72 Traditional Service

If a user wants to obtain information about some service, it is necessary to visit each site of the Service provider. For example, if he wants to compare the price of a product, he has to go to one site, look for the page of price list of the product, go to another site of provider, again search the page and get the price, so on.

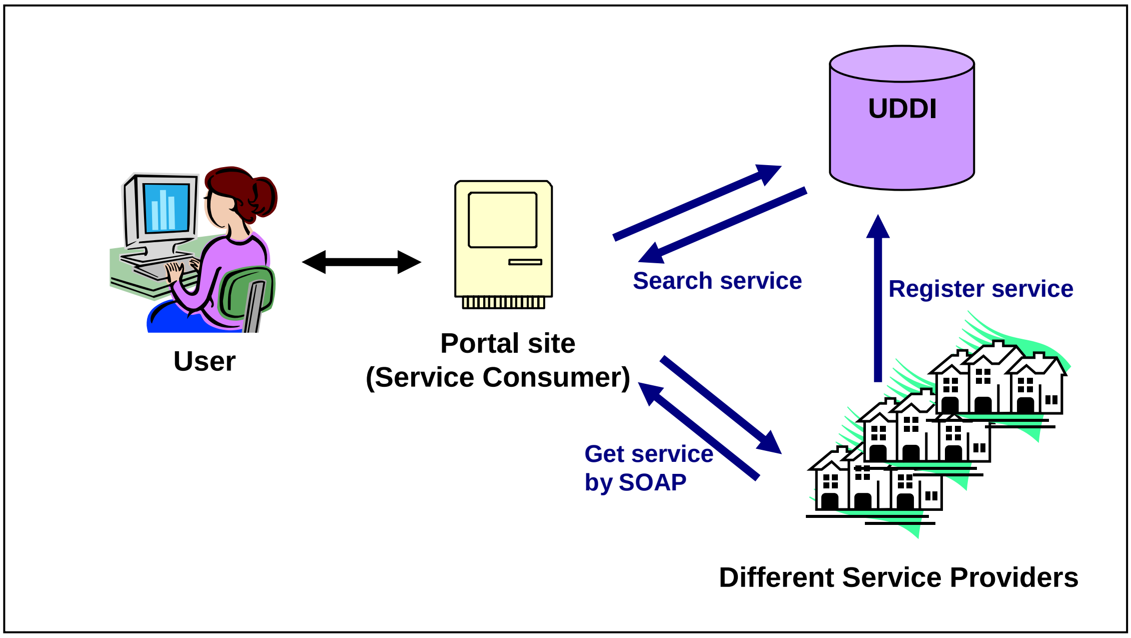


Figure 73 Web Service

The idea of Web service is that different service providers disclose the specification of their service to WSDL and register it to certain information directory. A Service Consumer searches the service from the information directory and collects different services to be able to provide users the collection of the information about different services. As a result, a user can obtain that information of different service providers just visiting the portal site of the service consumer.

[Characteristics of Web Service]

- It is basically Remote Procedure Calls (RPC) over HTTP.
- User, Service Consumer and Service provider are main actors in the Web Service.
  - **User:** who requests a service.
  - **Service Consumer:** a kind of agent who collects the services from different Service providers to respond to users.
  - **Service provider**: who publishes their business service as method.
- Web Service is a collection of Application logics formed by methods that provide data service to the other applications.
- Platform independent
- The access to the XML Web Service is established as international standardization by XML specification, SOAP or HTTP.

- Providers publish Web Method that can be called in the Internet

[Technical characteristics]

- Web Service is executed under the Web Server.
- Listens HTTP requires of method call
- Returns the result of requirement to the client.

5.4.2. Standardization

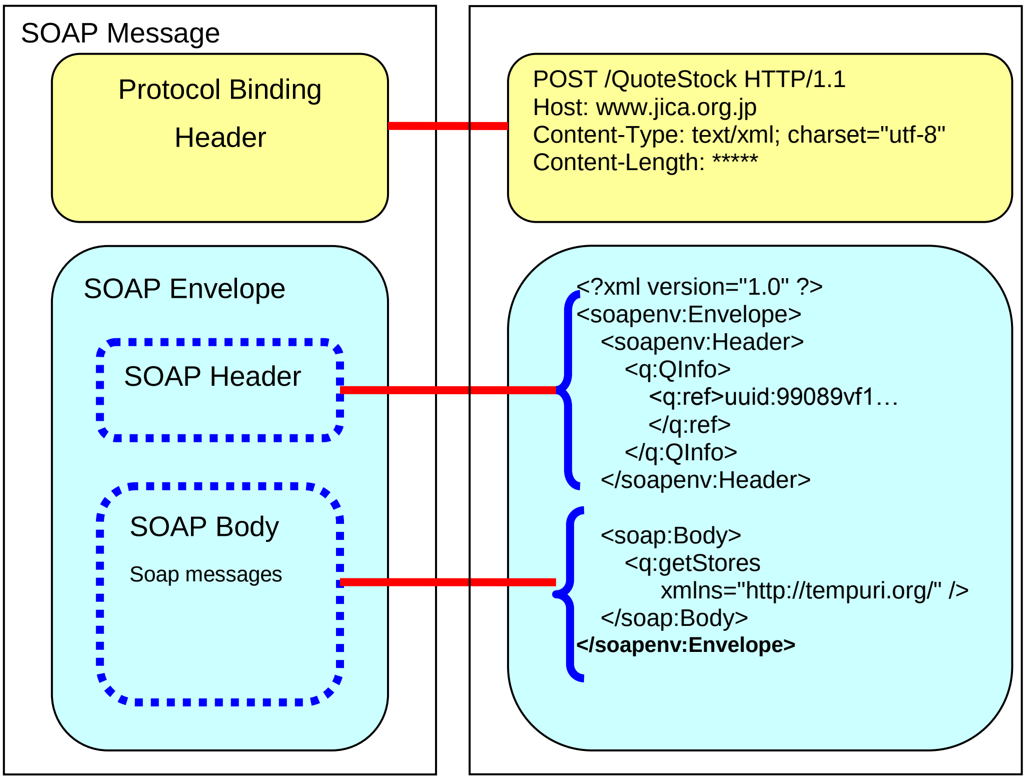
Web Service is not an original technology of Microsoft but is based of standardized solution.

.

- **SOAP**

SOAP (Simple Object Access Protocol) is a protocol used for Web Service to be able to exchange data over HTTP. It is presented in May, 2000 to W3C (World Wide Web Consortium) by Microsoft, IBM and Lotus.

Soap message consists of header and SOAP Envelope. The message is included in the Body tag and sent to the server covered by envelope.



```
POST /QuoteStock HTTP/1.1
Host: www.jica.org.jp
Content-Type: text/xml; charset="utf-8"
Content-Length: *****
```

```
<?xml version="1.0" ?>
<soapenv:Envelope>
  <soapenv:Header>
    <q:QInfo>
      <q:ref>uuid:99089vf1…
      </q:ref>
    </q:QInfo>
  </soapenv:Header>

  <soap:Body>
    <q:getStores
       xmlns="http://tempuri.org/" />
  </soap:Body>
</soapenv:Envelope>
```

Figure 74 SOAP message

In this example *getStores* of the *Body* tag is the name of the method.

- **WSDL**

WSDL (Web Service Description Language) is a language that defines the procedure to request the Web Service, e.g.;

- What kind of Web method is published
- Which protocol is supported
- What is method signature
- What is URL

WSDL1.1 was presented in March 2001 by Microsoft and IBM.

You can get the specification from

http://www.w3.org/TR/wsdl.html

- **UDDI**

- UDDI (Universal Description Discovery and Integration) is a specification for information registries of Web services specifying how to construct distributed database for registries.

- A UDDI registry service is a Web service that manages information about service providers, service implementations, and service metadata in order to be discovered by consumers who seek some services that suite to their requirements. The consumer can obtain the service metadata to consume those services. For example the following information can be obtained:

- Information about service classification
- Information to get access to the service
- access point

SOAP is used as protocol with XML format.

you can get the specification from:

http://www.uddi.org

- **XML Schema**

XML document concentrates in document structure but ignores the data type. It is important to distinguish the data type for actual data exchange, especially in XML Web Service. XML Schema provides a mechanism to assign data type to XMLInfoSet. XSD(XML Schema Definition Language) is defined to describe the data type.

5.4.3. Procedure of use of the Web Service

1) Obtain the information about the business partner from UDDI
2) Obtain the WSDL from the Business partner.
3) Develop the client logic
4) Execute the client logic to send the request of SOAP message by way of HTTP, SMTP or other protocols. Since SOAP message is defined by XML schema, service side can validate the message.

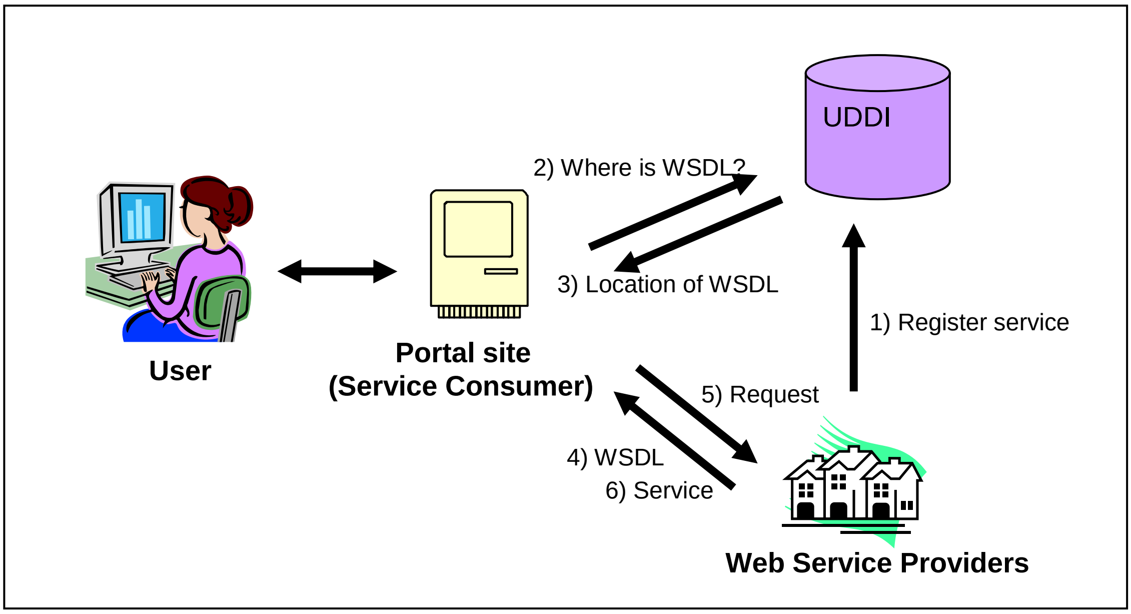


Figure 75 Flow of Web Service

The following site explains how to set up Web Service in Eclipse using WTP plugin and Tomcat server.

http://www.eclipse.org/webtools/jst/components/ws/M4/tutorials/BottomUpWebService.html

Exercise 5: J2EE

1. Practice the Web application "Salary of Employee" of MVC pattern referring to the textbook.

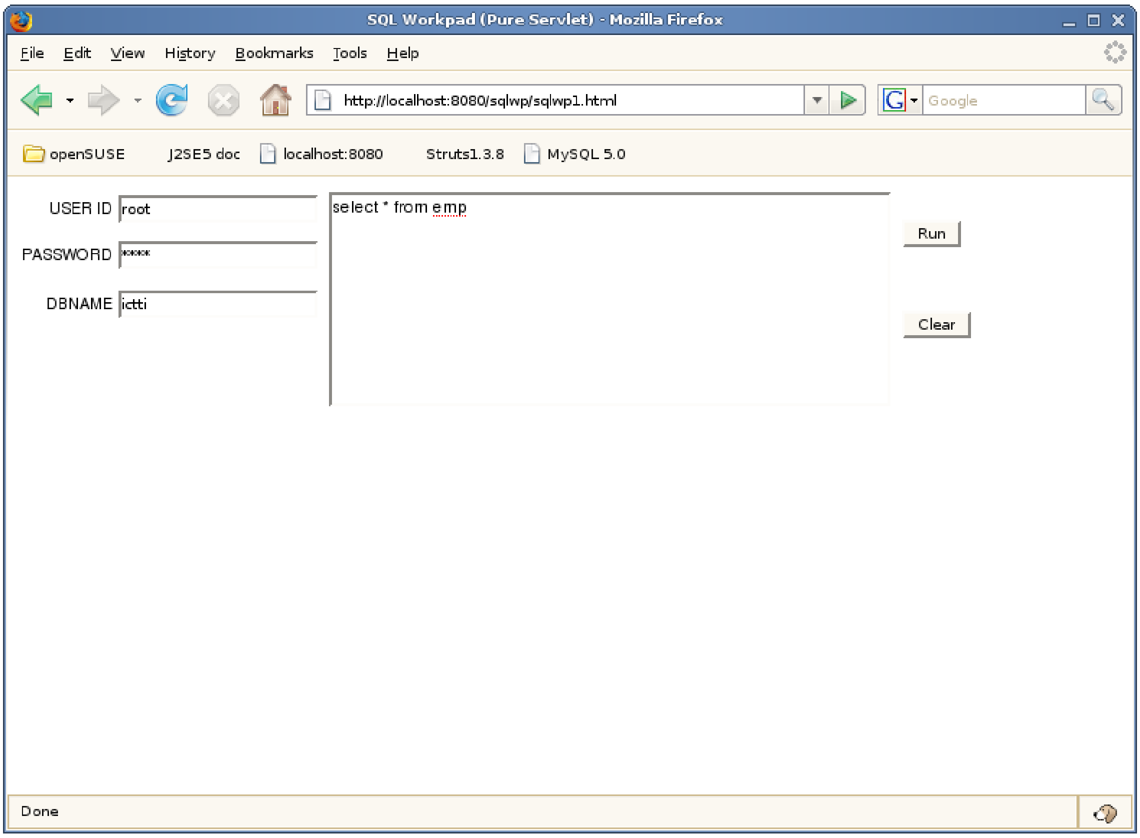
2. SQLWorkpad exercise. Make SQLWorkpad application that shows the result of the execution of Query in the specified the table of database. MySQL is used for database.

   1) The first page is static page (*"sqlwp1.html"*) where the following parameters can be input.

Table 18 - Input Parameters for SQLWorkPad

Parameters	Type	length	Description
USER_ID	Textbox	20	User id for DB connection
PASSWORD	Textbox	20	Password for DB connection
DBNAME	Textbox	20	Database name for DB connection
QUERY	Textarea		SQL Query to be executed.

[sqlwp1.html]


SQL Workpad (Pure Servlet) - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/sqlwp/sqlwp1.html
Google
openSUSE
J2SE5 doc
localhost:8080
Struts1.3.8
MySQL 5.0
Run
Clear
Done

USER ID root

PASSWORD *****

DBNAME ictti

select * from emp

Figure 76 First page of SQLWorkPad exercise

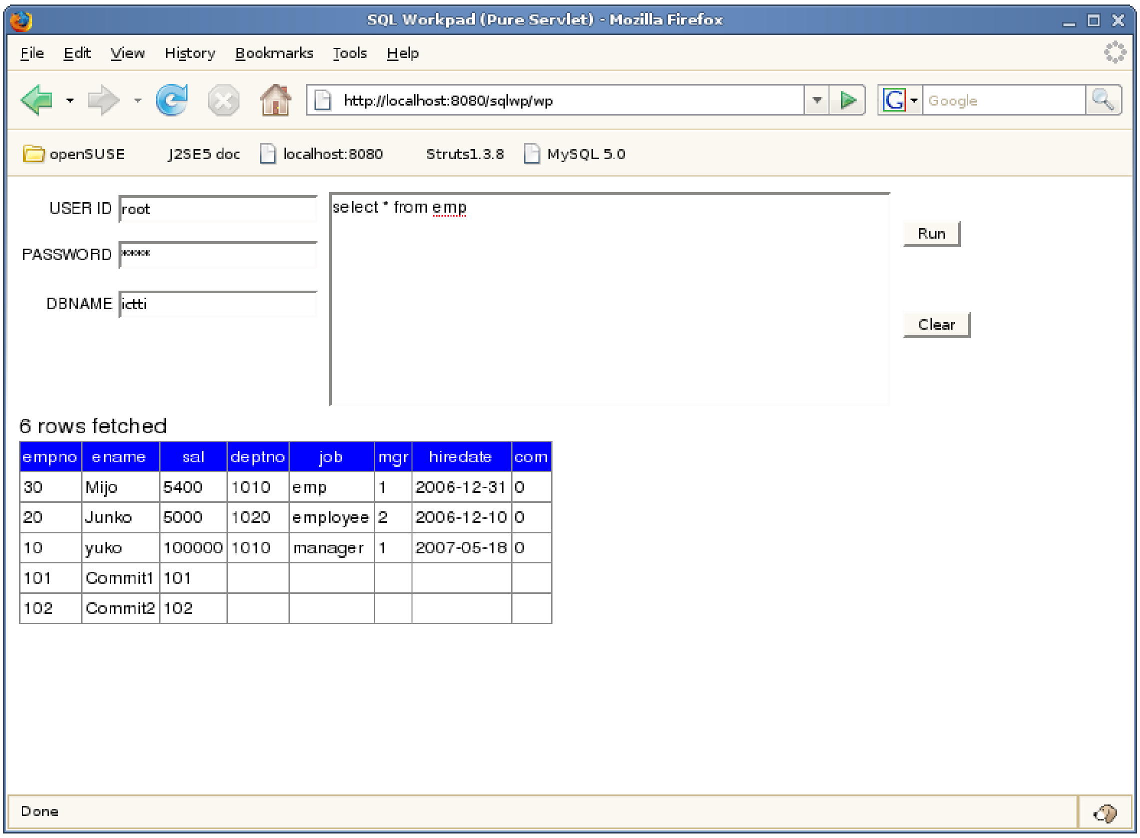
2) Set necessary environment configuration.

3) Create the following resources for development.

Table 19 - Development resources

Resources	Package	Name	Roll
Eclipse Project name		sqlwp	
Servlet file	controller	SqlwpServlet.java	Servlet as a controller
Bean file	model	SqlwpBean.java	Bean file to store data
DAO file	model	SqlwpDAO.java	DB access file
jsp file		sqlwp1.jsp	Display the result of the execution of Query.
web configuration file		web.xml	web configuration file

4) Make a program that receives the parameters from the browser and execute the specified Query connecting the Database. Show all the result of the Query in the next page.



USER ID root

PASSWORD ****

DBNAME ictti

```
select * from emp
```

6 rows fetched

empno	ename	sal	deptno	job	mgr	hiredate	com
30	Mijo	5400	1010	emp	1	2006-12-31	0
20	Junko	5000	1020	employee	2	2006-12-10	0
10	yuko	100000	1010	manager	1	2007-05-18	0
101	Commit1	101					
102	Commit2	102					

6. Framework; Apache Struts

6.1. What is Struts Framework

6.1.1. What is framework

Framework is a software base structure where the most useful and common function classes or packages are collected to be reused especially in the large system for efficiency, maintainability and high quality of the system saving cost and time. Since the framework provides base of infrastructure implementing the common functionality, the developer can embed just the system business specific logic. On the other words, framework forces the developers to keep a kind of architecture rules in order to get the integrated and structured application.

In framework, polymorphism plays quite significant roll. The basic tasks are supplied by framework while application specific process will be introduced by way of inheritance. In other words, the principal procedure of the system is constructed in framework and application is not who determines the main procedure.

☆ The mechanism of framework is sometime said to be similar as the Hollywood principal "Don't call us, we will call you", which means, actor is not who calls the director to ask if there is any job for him, but it is director who will call the actor when necessary.

6.1.2. What is Struts

Apache Struts is a free open-source framework for creating Java web applications developed by Jakarta project of Apache Software Foundation. Struts provides a J2EE framework for building Web applications based on a Model 2 approach of MVC(Model-View-Controller) primarily based on Servlet and JSP technology.

Struts is constructed implementing the MVC architecture and provides framework mainly for Presentation and Control tier.

[What are common functions in Web application]

- Session Control
- **Getting parameters from the browser and mapping to Java Bean class**
- **Dispatch to the next page**
- **validation of the input data from the browser**

- Transaction control
- Data mapping with Database

Struts can cover the functions colored in red above and the necessary information for framework will be described in "*struts-config.xml*", the external configuration file.

Struts also provides quite rich Tag library for JSP to avoid using scriptlet.

6.2. How Struts works

6.2.1. How to set the Struts framework to your Web application

1) *Struts* package can be downloaded from the following site;

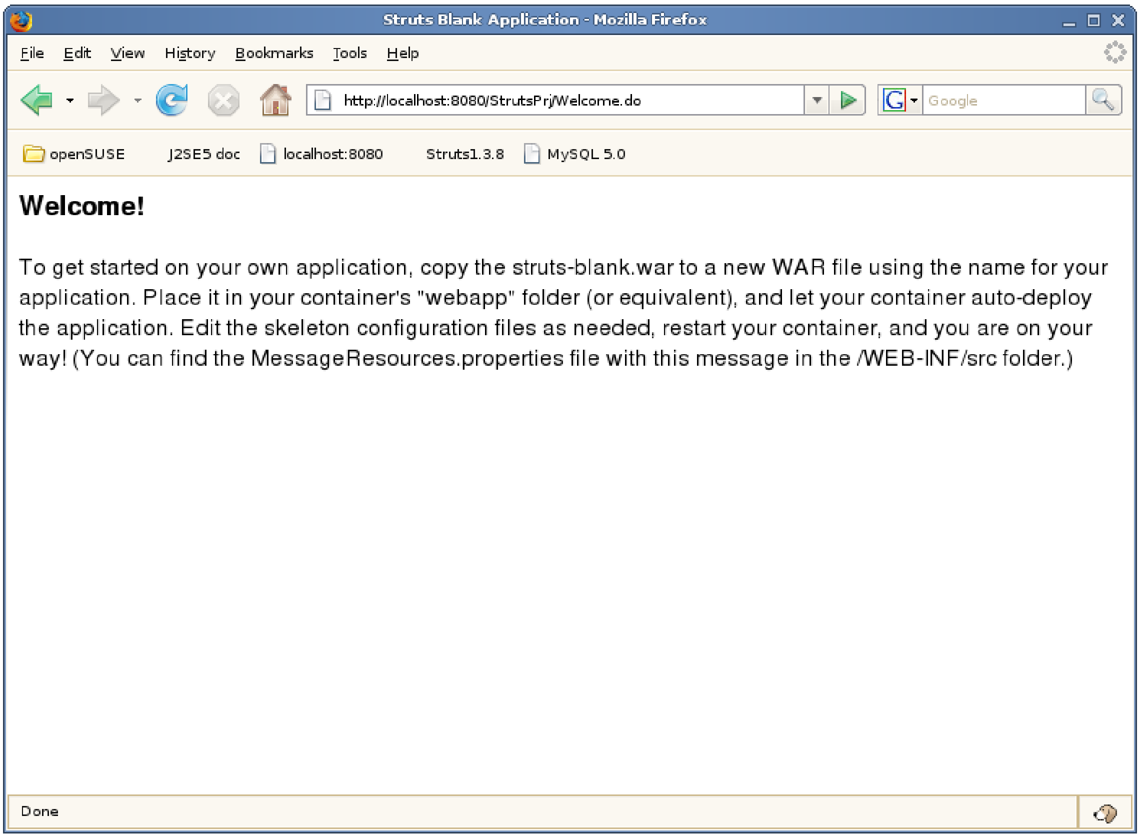
   http://struts.apache.org/download.cgi

   Select the production release of "*Struts 1.3.8-all.zip*" in a full distribution.

2) Unzip the above package. Also unpack the "*struts-blank-1.3.8.war*". Installed unpacked "*struts-blank-1.3.8.war*" under "*apps/*" directory of *Struts*.
3) Run *Eclipse* and create a new Web application as *Tomcat* project.
4) Select all the unpacked files of "*struts-blank-1.3.8.war*" and drug them under the above project name in *Eclipse*.
5) Start *Tomcat.*
6) From the browser, call your Web application without any file name. For example if you created a Web application with the project name "*StrutsPrj"*, then you can call it from browser as follows;

```
localhost:8080/StrutsPrj
```

7) If the following page appears, the necessary packages are set correctly into your project.


Struts Blank Application - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/StrutsPrj/Welcome.do
Google
openSUSE J2SE5 doc localhost:8080 Struts1.3.8 MySQL 5.0
Done

Welcome!

To get started on your own application, copy the struts-blank.war to a new WAR file using the name for your application. Place it in your container's "webapp" folder (or equivalent), and let your container auto-deploy the application. Edit the skeleton configuration files as needed, restart your container, and you are on your way! (You can find the MessageResources.properties file with this message in the /WEB-INF/src folder.)

☆ Since *Struts* is not a software but a framework, it is required only to deploy (copy) the libraries to the *Servlet* container. The necessary packages to use *Struts* exist in "*WEB-INF/lib*" directory in the "*struts-blank-xxx.war*" file.

6.2.2. Mechanism of Struts framework

Struts is composed of the following components;

Table 20 - Basic components of Struts

Components	Description	Roll of MVC	Remarks
struts-config.xml	Configuration file for flow control.		
Action Servlet	The main Servlet of Struts which dispatches the request to each action.	Controller	No need of configuration
Action Form Bean	Bean file where the data corresponds to the browser are stored.	Model (Data Object)	
Action class	A class which describes what process is executed and to where the next page is dispatched.	Bridge between Controller and Model	
JSP	Returns the result of the request to the browser.	View	

Tag library	Tag library used in JSP		

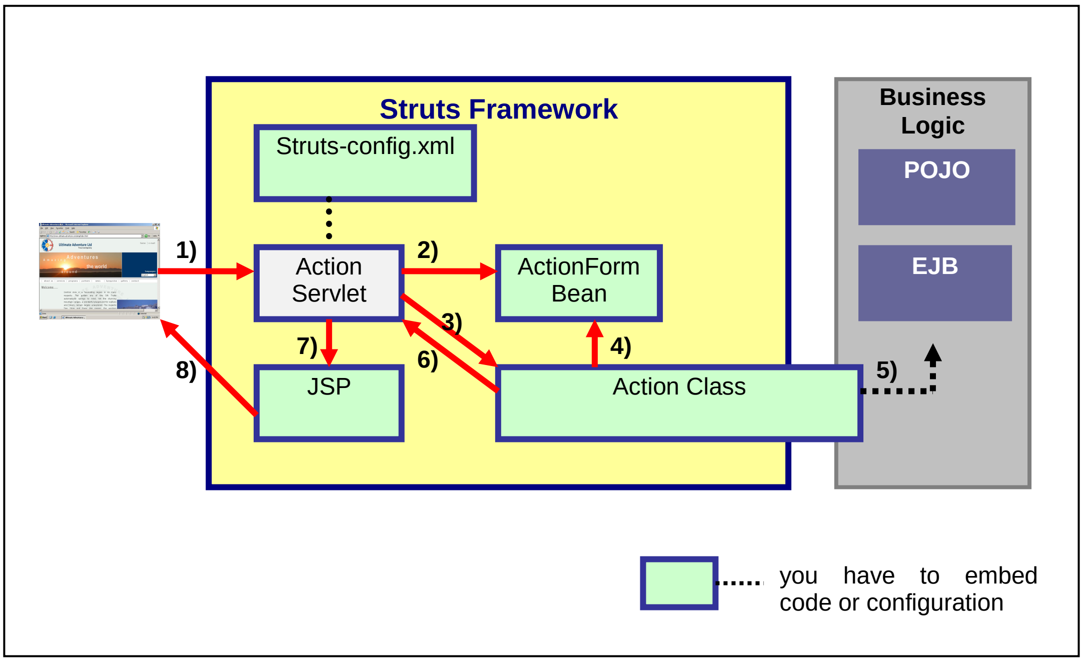


Figure 77 Flow of Struts

1) "*Action Servlet*" gets the request from the browser.

2) "*Action Servlet*" sets the parameter data sent from the browser to the "*ActionForm"* bean class.

3) "*Action Servlet*" calls corresponding "*Action"* class.

4) "*Action"* Class refers to the parameter data if necessary in ActionForm Bean.

5) "*Action"* Class executes the corresponding business logic.

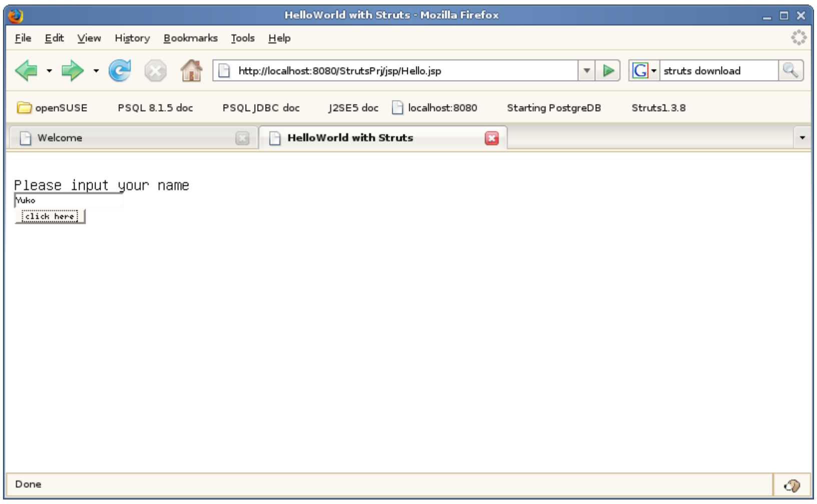
6) "*Action"* Class returns the result to Action Servlet specifying the next page.

7) "*Action Servlet"* dispatches the next page of JSP.

8) The result is shown in the browser.

6.2.3. HelloWorld with Struts

Let's create a Sample program of HelloWorld with Struts. The first page asks to input your name. After submit the page, the Hello page appears with your name input in the previous page.

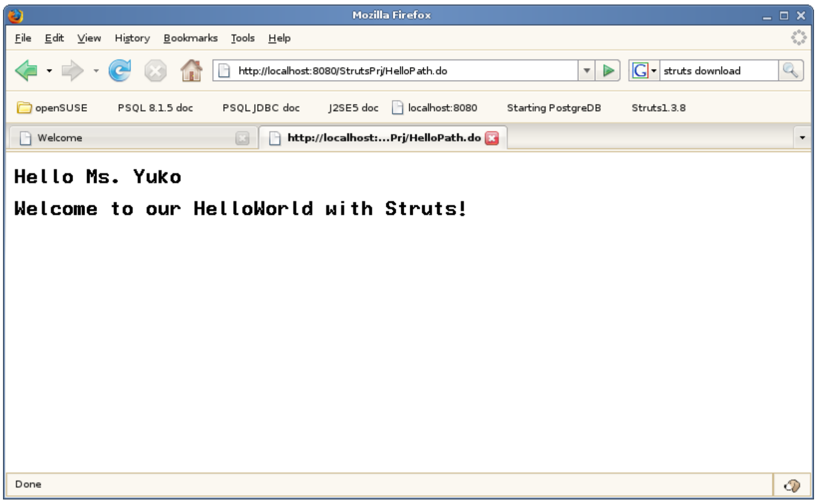

HelloWorld with Struts - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/StrutsPrj/jsp/Hello.jsp
struts download
openSUSE PSQL 8.1.5 doc PSQL JDBC doc J2SE5 doc localhost:8080 Starting PostgreDB Struts1.3.8
Welcome
HelloWorld with Struts
Done

Please input your name

Yuko

click here

Figure 78 First page of HelloWorld with Struts


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/StrutsPrj/HelloPath.do
struts download
openSUSE PSQL 8.1.5 doc PSQL JDBC doc J2SE5 doc localhost:8080 Starting PostgreDB Struts1.3.8
Welcome
http://localhost:...Prj/HelloPath.do
Done

Hello Ms. Yuko

Welcome to our HelloWorld with Struts!

Figure 79 The HelloWorld response with Struts

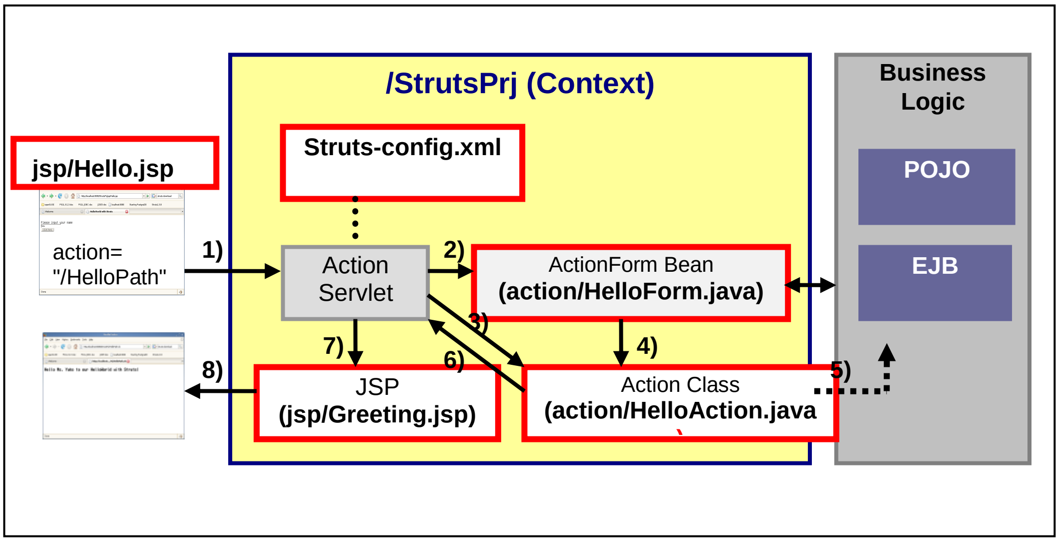


Figure 80 Files that compose the HelloWorld

2) Configuration file; struts-config.xml

struts-config.xml is a configuration file for Struts application. It describes;

- what is the Action Class
- what is the Action Form Bean Class
- what is the next JSP file
- what is message file

[sturts-config.xml]

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE struts-config PUBLIC
 "-//Apache Software Foundation//DTD Struts Configuration 1.3//EN"
        "http://struts.apache.org/dtds/struts-config_1_3.dtd">
<struts-config>
<!-- ========= Form Bean Definitions -->
   <form-beans>
      <form-bean
         name="BeanNameHelloForm"
         type="action.HelloForm"/>
   </form-beans>
<!-- ========= Action Mapping Definitions -->
   <action-mappings>
      <action
        name="BeanNameHelloForm"
        type="action.HelloAction"
```

Name used in this file to indicate ActionForm Bean. The input data is stored in this class. This name is also used in JSP as bean name.

ActionForm Bean class name

Action class name that is called from JSP with path name "/HelloPath"

```
        path="/HelloPath"
        scope="request">

        <forward    name="forwardHello"
                    path="/jsp/Greeting.jsp" />
       </action>
    </action-mappings>

<!-- ================ Message Resources Definitions -->
    <message-resources parameter="MessageResources" />
</struts-config>
```

The destination path described in <form action="/HelloPath"> of JSP.

The name of next page that is dispatched from Action class

The real path of the next page

The name of Resource file for messages. The file extension is "properties."

Once the JSP describes <Form action="/HelloPath">, the Struts can understand;

- "*action.HelloForm*" is the action bean class where the input parameter will be stored.
- "*action.HelloAction*" is a class that will be called on the server to execute necessary process.
- if "*action.HelloAction*" class wants to forward the next page to "*forwardHello*", *Struts* understands that its real file path is "*/jsp/Greeting.jsp*".
- If the <bean:message> tag is described in JSP, *Struts* looks for "*MessageResources*" class.

☆ "*struts-config.xml*" is not reloaded by *Tomcat* on modification. If you modify the file, you need to **reload** the application. If you modify the xml file, at the same time adding some change in java source file will cause the application to reload.

3) ActionForm

ActionForm is a kind of data object class where the input parameters data are defined with getter/setter method, so that *Struts* automatically stores the data obtained from the browser.

[HelloForm.java]

```
package action;
import org.apache.struts.action.ActionForm;

public class HelloForm extends ActionForm{
    private String name;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

The form bean class should always extend *ActionForm* class.

```
public class HelloForm extends ActionForm{
```

In this form bean class, "*name*" instance variable and its get/set methods are defined. The input parameter value of textbox from "*Hello.jsp*" will be set in the "name" variable by *Struts*.

☆ It is not any more necessary to use *getParameter()* of *Request* object to get input parameter. Instead you need to create this *ActionForm* class with all the necessary input data.

4) Action class

Action class is the main class in the framework. Some data access logic or other business logic will be called from action class passing the *Bean* class that contains input data as a parameter. In this case, since there is no special process to do, the next page is called and forwarded.

[HelloAction.java]

```
package action;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

public class HelloAction extends Action {

    public ActionForward execute(
                ActionMapping mapping,
                ActionForm form,
                HttpServletRequest request,
                HttpServletResponse response) throws Exception {
    // forward to the next page named "forwardHello"
        return mapping.findForward("forwardHello");
    }
}
```

Your action class always needs to extend *Action* class and override the *execute()* method.

```
public class HelloAction extends Action {
    public ActionForward execute(
                ActionMapping mapping,
                ActionForm form,
                HttpServletRequest request,
                HttpServletResponse response) throws Exception {
```

ActionForm object is passed as one of the parameter of *execute*() method with input parameter data inside. You can get the subclass by casting;

```
HelloForm myFormBean = (HelloForm)ActionForm;
String strName=myFormBean.getName();
```

The *findForward*() method forwards to the specified page;

```
return mapping.findForward("forwardHello");
```

The forwarded page name is "*forwardHello*" and its real path is defined in the "*struts-config.xml*".

5) View; Hello.jsp

As a view, usually JSP is used although it is not the only solution. *Struts* provided quite rich tag library.

[Hello.jsp]

```
<%@ taglib uri="http://struts.apache.org/tags-html"
prefix="html" %>
<%@ taglib uri="http://struts.apache.org/tags-bean"
prefix="bean" %>
<html:html>
<Title>HelloWorld with Struts
 <BODY>
```

```
  <!-- Converted to Form tag -->
  <html:form action="/HelloPath" >
  <BR>
   Please input your name<BR><html:text property="name" /><BR>
    <html:submit>
      <bean:message key="mysubmit" />
    </html:submit>
  </html:form>
```

```
 </BODY>
</html:html>
```

To use Struts tags, it is necessary to declare taglib directory specifying its URI and prefix.

```
<%@ taglib uri="http://struts.apache.org/tags-html"
prefix="html" %>
<%@ taglib uri="http://struts.apache.org/tags-bean"
prefix="bean" %>
```

The above directive declares that in this JSP page *Struts* html tag and bean tag will be referred.

The following code demonstrates that the struts html form tag is used.

```
<html:form action="/HelloPath" >
```

The above *Struts* html form tag is converted to HTML Form tag as follows.

```
<form name="HelloForm" method="post"
      action="/StrutsPrj/HelloPath.do">
```

☆ The action path is created with Web context name and the extension ".do" which is mapped to the *Struts ActionServlet* class.

```
<html:text property="name" />
```

This <html:text> tag will be converted to;

```
<input type="text" name="name" value="">
```

The following code shows the <bean;message> tag. This tag is used to retrieve an internationalized message for the specified locale, using the specified message key, and write it to the output stream. The message file name is obtained from the message source. The value will be extracted with the key name "mysubmit".

```
     <bean:message key="mysubmit" />
```

The result of the conversion of *Struts* tag will be;

```
   <input type="submit" value="click here">
```

You will prepare the message resources file with the name defined in the "*struts-config.xml*".

```
<!-- ================ Message Resources Definitions -->
   <message-resources parameter="MessageResources" />
</struts-config>
```

The *MessageResources* file has the extension "properties". This file is located where CLASSPATH is passed, such as "*WEB-INF/classes*" directory.

[MessageResources.properties]

```
greeting=Hello
welcome=Welcome
error=The error happens.
mysubmit=click here
```

☆ If the message-resources file has directory structure, parameter name is separated by ".". For example, if the message file is located in "*WEB-INF/classes/java/MessageResources.properties*", then its definition will be;

```
<message-resources parameter="java.MessageResources" />
```

☆ In *Eclipse* environment, properties file can be created under "*/src*" directory, so that each time you save it, *Eclipse* automatically copies it under "*/classes*" directory.

6) View;Greeting.jsp

"*Greeting.jsp*" is a page that is shown as a result of request of the previous page. The "*name"* input parameter in the first page is obtained from the java bean class which is set automatically by *Struts*.

[Greeting.jsp]

```
<%@ taglib uri="http://struts.apache.org/tags-html"
prefix="html" %>
<%@ taglib uri="http://struts.apache.org/tags-bean"
prefix="bean" %>

<html:html>
   <BODY>
     <H1>
       <bean:message key="greeting" />
       <!-- Show the name sent from previous page -->
      Ms. <bean:write name="BeanNameHelloForm" property="name" />
      <br>Welcome to our HelloWorld with Struts!
     </H1>
   </BODY>
</html:html>
```

```
       <bean:message key="greeting" />
```

This tag uses the message resources with the key name "greeting". This tag will be converted to;

```
Hello
```

according to the message resources file.

The following code uses <bean:write> tag.

```
<bean:write name="BeanNameHelloForm" property="name" />
```

This extracts the java bean named "*BeanNameHelloForm*" in the scope of Page, Request, Session or Application object. The value of the property "name" will be shown in the page executing *getName()* method of "*BeanNameHelloForm"* bean.

☆ The bean name should be the same as the bean name specified in the "*struts-config.xml*".

7) web.xml

In the web.xml the following struts Servlet should be configured.

```
  <!-- Standard Action Servlet Configuration -->
  <servlet>
   <servlet-name>action</servlet-name>
<servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
   <init-param>
     <param-name>config</param-name>
     <param-value>/WEB-INF/struts-config.xml</param-value>
   </init-param>
   <load-on-startup>2</load-on-startup>
 </servlet>


  <!-- Standard Action Servlet Mapping -->
  <servlet-mapping>
   <servlet-name>action</servlet-name>
   <url-pattern>*.do</url-pattern>
  </servlet-mapping>
```

The Servlet with ".do" extension is mapped to *ActionServlet* of *Struts*.

8) Directory structure

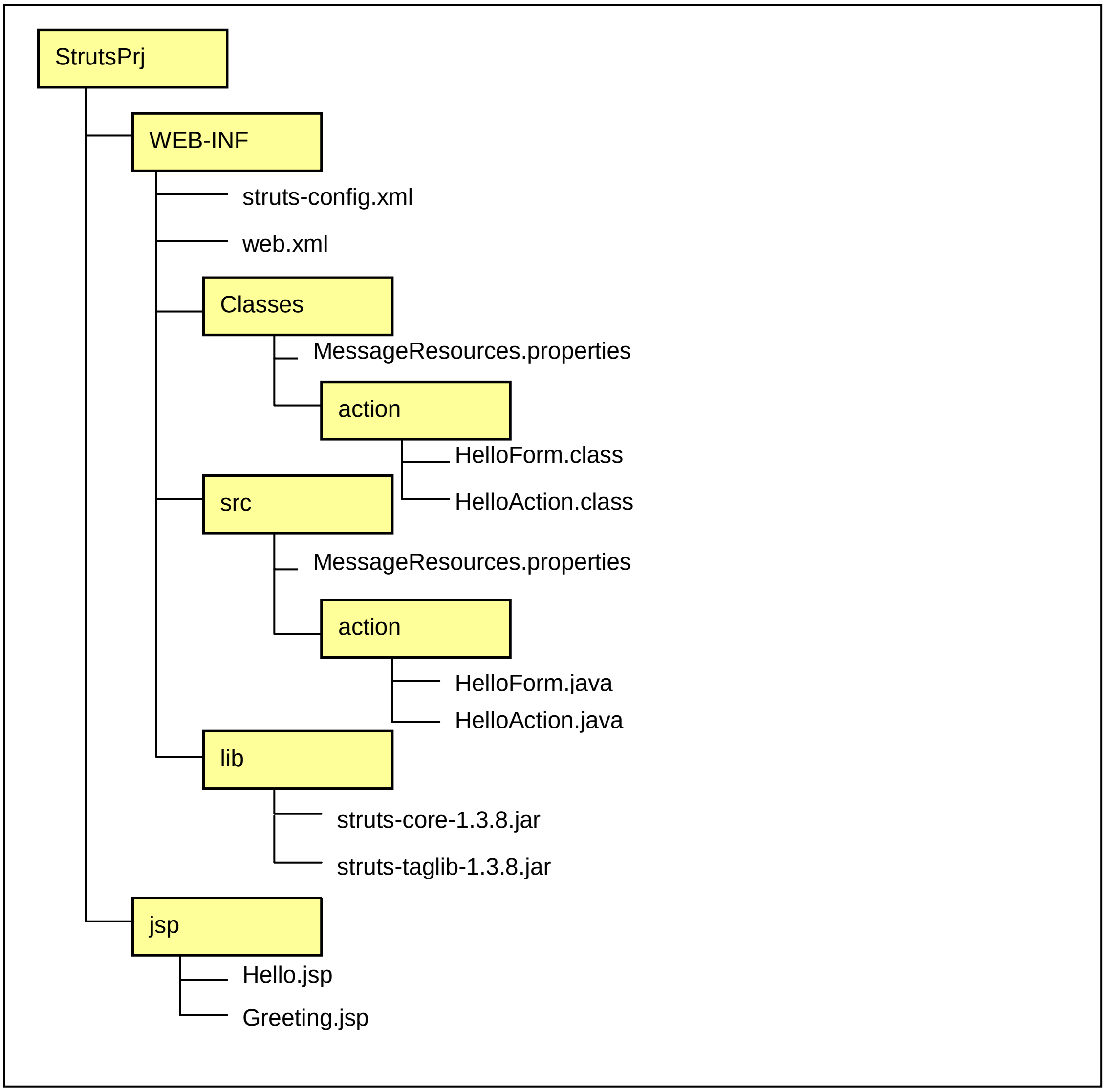


Figure 81 Directory structure of HelloWorld with Struts

☆ You need to include the following Struts library into ClassPath if you use Eclipse.

- struts-core-1.3.8.jar
- struts-taglib-1.3.8.jar

6.3. Struts Custom Tag

Custom Tug is a tag library for JSP which contains commonly used and helpful form-based applications. Sturts has the following four tag libraries.

Table 21 - Struts Custom Tag

Tag library	URI	Description
Bean	http://struts.apache.org/tags-bean	Create HTML based tags associating with Java bean class
html	http://struts.apache.org/tags-html	Provides to get property in Java Bean class as well as variables, cookie and header.
logic	http://struts.apache.org/tags-logic	Provides the logic tags for conditional and iteration operation.
nested	http://struts.apache.org/tags-nested	Provides the tags that can refer directly to nested properties with <nested;nest> tag.

☆ This textbook does not cover all the *Struts* Custom Tag and its attributes. Please refer to the *Struts* document for detail specifications under "*docs"* directory of *Struts* install directory.

6.3.1. HTML Tag Library

Struts HTML Tag Library generates HTML tags associating Java Beans class. The value input or selected is stored to Java Bean class, and also the values in the class are reflected to the values in JSP tags.

Table 22 - Struts HTML Tag Library

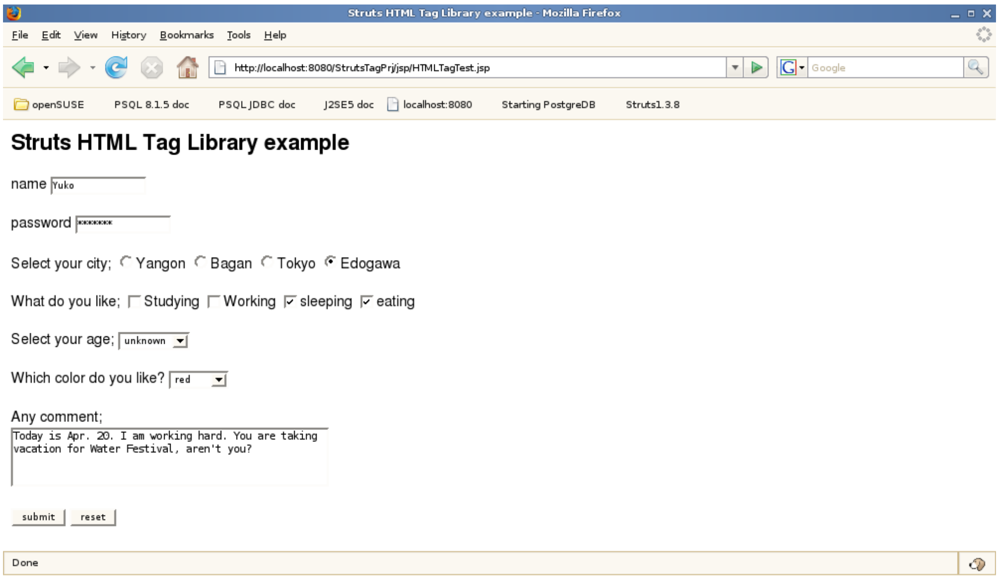
HTML Tag	Equivalent HTML Tag	Remarks
<html:base>	<base>	Describe base URL
<html:button>	<input type="button">	
<html:cancel >	<input type="cancel">	
<html:checkbox>	<input type="checkbox">	property value associated with this field should be of type boolean. Any value you specify should correspond to one of the Strings that indicate a true value ("true", "yes", or "on").
<html:errors>		Show the content of ActionErrors object, String object, or String array.
<html:file>	<input type="file">	
<html:form>	<form>	
<html:frame>	<frame>	
<html:html>	<html>	
<html:image>	<input type="image">	
<html:img>	<img>	
<html:javascript>		It is defined when javascript is used for Validator validation.
<html:link>	<a>	
<html:messages>		Show the content of ActionErrors object,

		String object, or String array.
<html:multibox>	<input type="checkbox">	useful when you have large numbers of checkboxes,
<html:option>	<option>	Available to select only one element
<html:options>	<option>	Both value and text are obtained from collection.
<html: optionsCollection>	<option>	Extract from collection that contains Java Beans. Java Bean should have value and text property.
<html:param>		Adds a parameter to <html:frame>, <html:link>, <html:rewrite>
<html:password>	<input type="password">	
<html:radio>	<input type="radio">	
<html:reset>	<input type="reset">	
<html:rewrite>	<a>	Described inside the javascript
<html:select>	<select>	
<html:submit>	<input type="submit">	
<html:text>	<input type="text">	
<html:textarea>	<textarea>	
<html:xhtml>		Used to generate XHTML

[example]

Let's create a sample program using Struts HTML Tag Library. The first page is as follows;

[first page]


Struts HTML Tag Library example - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/StrutsTagPrj/jsp/HTMLTagTest.jsp
Google
openSUSE PSQL 8.1.5 doc PSQL JDBC doc J2SE5 doc localhost:8080 Starting PostgreDB Struts1.3.8
submit
reset
Done

Struts HTML Tag Library example

name Yuko

password *******

Select your city; ○ Yangon ○ Bagan ○ Tokyo ● Edogawa

What do you like; □ Studying □ Working ☑ sleeping ☑ eating

Select your age; unknown

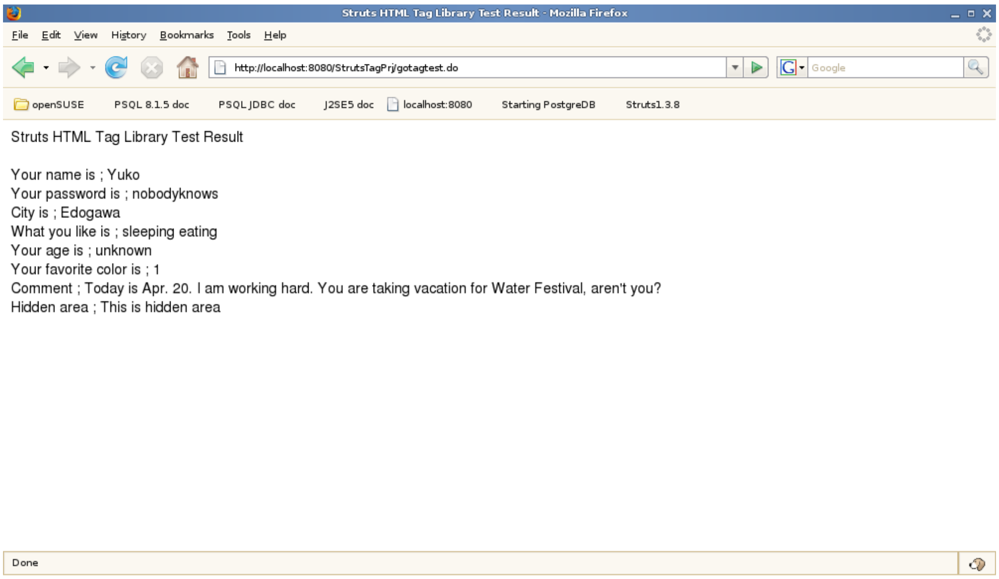
Which color do you like? red

Any comment;
Today is Apr. 20. I am working hard. You are taking vacation for Water Festival, aren't you?

Figure 82 Struts HTML Tag Test example

After submit the page, the values of input parameter will be shown;

[result]


Struts HTML Tag Library Test Result - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/StrutsTagPrj/gotagtest.do
Google
openSUSE PSQL 8.1.5 doc PSQL JDBC doc J2SE5 doc localhost:8080 Starting PostgreDB Struts1.3.8
Done

Struts HTML Tag Library Test Result

Your name is ; Yuko
Your password is ; nobodyknows
City is ; Edogawa
What you like is ; sleeping eating
Your age is ; unknown
Your favorite color is ; 1
Comment ; Today is Apr. 20. I am working hard. You are taking vacation for Water Festival, aren't you?
Hidden area ; This is hidden area

Figure 83 Struts HTML Tag Library Example result

[jsp/HTMLTagTest.jsp]

```
<%@ taglib uri="http://struts.apache.org/tags-html" prefix="html"%>
<html:html>
<head><title>Struts HTML Tag Library example</title></head>
<body>
<H2>
Struts HTML Tag Library example
</H2>

<%-- (2)<html:form> --%>
<html:form action="/gotagtest" focus="name">
  <%-- (3)<html:text> --%>
  name     <html:text property="name" size="16"/><br>
  <br>
  password  <html:password property="mypassword"
                 size="16"  redisplay="false"/>
  <br><br>

  <%-- (4)<html:radio> --%>
  Select your city;
  <html:radio property="city" value="Yangon"/>Yangon
  <html:radio property="city" value="Bagan"/>Bagan
  <html:radio property="city" value="Tokyo"/>Tokyo
  <html:radio property="city" value="Edogawa"/>Edogawa
  <br><br>

  <%-- (5)<html:multibox> --%>
  What do you like;
  <html:multibox property="hobby" value="Studying"/>Studying
  <html:multibox property="hobby" value="Working"/>Working
  <html:multibox property="hobby" value="sleeping"/>sleeping
  <html:multibox property="hobby" value="eating"/>eating
  <br><br>

  <%-- (6)<html:select>, <html:option> --%>
  Select your age;
  <html:select property="age">
    <html:option value="0-10">0-10</html:option>
    <html:option value="11-20">11-20</html:option>
    <html:option value="21-30">21-30</html:option>
    <html:option value="31-40">31-40</html:option>
    <html:option value="unknown">unknown</html:option>
  </html:select>
  <br><br>

   <%-- (7)<html:optionsCollection> --%>
Which color do you like?
 <html:select property="color" size="1">
  <html:optionsCollection property="choices" value="value"
label="label" />
</html:select>
 <br><br>
  <%-- (7)<html:textarea> --%>
  Any comment;<br>
  <html:textarea property="other" rows="3" cols="50"/>

  <%-- (8)<html:hidden> --%>
  <html:hidden property="id" value="This is hidden area" />
  <br><br>

  <%-- (9)<html:submit> --%>
  <html:submit property="submit" value="submit"/>

  <%-- (10)<html:reset>--%>
  <html:reset value="reset"/>
</html:form>

</body>
```

```
</html:html>
```

[jsp/result.jsp]

```
<%@ taglib uri="http://struts.apache.org/tags-html" prefix="html"%>
<%@ taglib uri="http://struts.apache.org/tags-bean"
prefix="bean" %>
<%@ taglib uri="http://struts.apache.org/tags-logic"
prefix="logic"%>

<html:html>
<head><title>Struts HTML Tag Library Test Result</title></head>
<body>

Struts HTML Tag Library Test Result
<br><br>
Your name is ;
<bean:write name="MyTagBean" property="name"
          scope="request" ignore="true" />

<br> Your password is ;
<bean:write name="MyTagBean" property="mypassword"
          scope="request" ignore="true" />
<br> City is ;
<bean:write name="MyTagBean" property="city"
          scope="request" ignore="true" />

<br> What you like is ;
<logic:iterate id="element" name="MyTagBean"
            property="hobby" scope="request">
  <bean:write name="element"/>
</logic:iterate>

<br> Your age is ;
<bean:write name="MyTagBean" property="age"
          scope="request" ignore="true" />

<br> Your favorite color is ;
<bean:write name="MyTagBean" property="color"
          scope="request" ignore="true" />
<logic:equal name="MyTagBean" property="color" value="1">
 Red.
</logic:equal>
<logic:equal name="MyTagBean" property="color" value="2">
 green.
```

```
</logic:equal>
 <logic:equal name="MyTagBean" property="color" value="3">
 yellow.
</logic:equal>

<br> Comment ;
<bean:write name="MyTagBean" property="other"
          scope="request" ignore="true" />

<br> Hidden area ;
<bean:write name="MyTagBean" property="id"
          scope="request" ignore="true" />

</body>
</html:html>
```

"MyTagBean" is the name of FormBean. All the input parameters using HTML Tag Library in the first page are set to the properties of FormBean. Those values can be retrieved from the bean in the next JSP page. <bean:write> tag is a bean tag that output the value of the property in the specified bean.

[struts-config.xml]

```
<?xml version="1.0" ?>
<!DOCTYPE struts-config PUBLIC
 "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
 "http://struts.apache.org/dtds/struts-config_1_2.dtd">

<struts-config>

  <!-- (1)Action Form Bean -->
  <form-beans>
   <form-bean
    name="MyTagBean"
    type="action.TagFormBean"/>
  </form-beans>

  <!-- (2)Specify behaviors  -->
  <action-mappings>
   <action path="/gotagtest"
    type="action.TagInputAction"
    name="MyTagBean"
    scope="request">
```

```
     <!-- (3)Where to dispatch -->
     <forward name="result" path="/jsp/result.jsp"/>
   </action>
  </action-mappings>

</struts-config>
```

[TagFormBean.java]

```
package action;

import java.util.LinkedList;
import java.util.List;

import org.apache.struts.action.ActionForm;
import org.apache.struts.util.LabelValueBean;
//(1)Declare Bean
public class TagFormBean extends ActionForm {

      // Constructor. Initialize the array for hobby
      public TagFormBean() {
          hobby = new String[0];
      }
      //(2)Variable to store the values of property
      private String name;
      private String city;
      private String[] hobby;
      private String age;
      private String other;
      private String id;
      private String mypassword;
      private String color;

      public String getColor() {
        return color;
    }
    public void setColor(String occupation) {
        this.color = occupation;
    }
    //(3)Access method
      public void setName(String name) {this.name = name;}
      public String getName() {return name;}
      public void setCity(String address) {this.city = address;}
      public String getCity() {return city;}
      public void setHobby(String[] hobby) {this.hobby = hobby;}
```

```
 public String[] getHobby() {return hobby;}
 public void setAge(String age) {this.age = age;}
 public String getAge() {return age;}
 public void setOther(String other) {this.other = other;}
 public String getOther() {return other;}
 public void setId(String id) {this.id = id;}
 public String getId() {return id;}
public String getMypassword() {
   return mypassword;
}
public void setMypassword(String mypassword) {
   this.mypassword = mypassword;
}
public List getChoices(){
  List list=new LinkedList();
  list.add(new LabelValueBean("red","1"));
  list.add(new LabelValueBean("green","2"));
  list.add(new LabelValueBean("yellow","3"));
  return(list);
 }
}
```

All the input parameters are specified as instance values in the *FormBean*. Since the multibox for hobby returns multiple values, the *String* array is declared.

<html:optionsCollection> can define the values and labels dynamically from the program. In this sample, the values are created in the *FormBean* and provides the *getChoices*() method to return collection as *List* type　The "*org.apache.struts.util.LabelValueBean"* is a class that has label and value property provided by *Struts*.

```
public List getChoices(){
     List list=new LinkedList();
     list.add(new LabelValueBean("red","1"));
     list.add(new LabelValueBean("green","2"));
     list.add(new LabelValueBean("yellow","3"));
     return(list);
    }
```

In the "*HTMLTagTest.jsp*", <html:optionsCollection> is defined specifying the "choices" as its property.

```
<html:select property="color" size="1">
 <html:optionsCollection property="choices" value="value"
```

```
label="label" />
</html:select>
```

This tag will be converted to;

```
<select name="color" size="1">
    <option value="1">red</option>
    <option value="2">green</option>
    <option value="3">yellow</option>
</select>
```

☆ The array used for collection should be initialized in the constructor. If the array is not created, the exception "Collection not found" occurs when none of the option is selected.

```
// Constructor. Initialize the array for hobby
      public TagFormBean() {
          hobby = new String[0];
      }
```

6.3.2. Bean Tag Library

Bean Tag Library is used to obtain Java Bean and store it in the scope or manage the bean stored in the scope.

Table 23 - Struts Bean Tag Library

Tag	Description	example
<bean; cookie >	Defines a variable based on the value of the specified request cookie.	<bean:cookie id="c" name="myCookie" /> <bean:write name="c" property="value" />
<bean;define>	Define a variable based on the value of the specified bean property.	see example below
<bean:header>	Define a variable based on the value of the specified request header.	<bean:header id="c" name="User-Agent" /> <bean:write name="c" />
<bean:include>	Dynamically load the resources and store it in a page scope. Similar to <jsp:include>.	<bean:include id="top" page="/header.jsp" /> <bean:write name="top"/>
<bean:message>	Output the message that corresponds to the key from Message Resource file.	<bean:message key="greeting" />
<bean:page>	Define a variable of implicit object of response, request, session, application and config.	<bean:page id="mySession" property="session" />
<bean:parameter>	Define a variable of the value of request parameter.	<bean:parameter id="myAge" name="age" value="no Parmeter"/>
<bean:resource>	Load a web application resource. The resource is specified with "/" from the	<bean:resource id="myResource" name="/web-INF/web.xml" />

	root of the Context directory.	
<bean:size>	Define a bean containing the number of elements in a Collection or Map.	see example below
<bean:struts>	Retrieve the value of Struts internal object, and define it as a page scope. FormBean, ActionForward, or ActionMapping can be obtained.	<bean:struts **id**="myFormbean**"** formbean="validateForm" />
<bean:write>	Retrieve the value of the specified bean property, and render it to the current JspWriter as a String	see example below

1) <bean:define> tag

Defines new bean with specified name.

Table 24 - Struts <bean:define> tag

Attribute	Description	Type
id	Specifies the name of the variable used in this page. mandatory.	String
name	Bean name to be referred.	String
property	The property declared in the Bean.	String
scope	The scope of the Bean to be accessed.	String
toScope	The scope where the newly defined Bean will be created	String
type	Type of the Bean. If not specified and if value attribute exists then String type, if no value attribute, then Object type is set.	String
value	The value that is set to newly created variable. The value is set when the name attribute is not specified.	String

[example 1]

```
<%@ taglib uri="http://struts.apache.org/tags-bean" %>
<bean:define id="mySalary" name="employee" property="salary"
type="java.lang.Integer" />

<bean:write name="mySalary" />
```

[result 1]

If the salary property of employee bean contains "100,000" then ;

```
100000
```

[example 2]

```
<%@ taglib uri="http://struts.apache.org/tags-bean" %>
<bean:define id="myVar toScope="session" value="Hello">

<bean:write name="myVar" />
```

[result1]

```
Hello
```

The new variable named "myVar" has value "Hello" and available within session scope.

2) <bean:size> tag

Obtain the size of collection and creates a new bean, of type "*java.lang.Integer*", whose value is the number of elements in that collection.

Table 25 - Struts <bean:size> tag

Attribute	Description	Type
collection	Collection or Array specified in a runtime expression, such as scriptlet expression.	String
id	The name used in this page that contains the collection size with Integer type. Mandatory.	String
name	Bean name that contains the collection.	String
property	Property name declared in the Bean.	String
scope	The scope of the Bean to be accessed. If not specified, the available scopes are searched in ascending sequence.	String

[example 1]

```
<%@ taglib uri="http://struts.apache.org/tags-bean" %>
<bean:size id="mySize" name="list" scope="session"/>
```

Search the Bean named "list" from session scope and store the list.size to "mySize" variable.

[example 2]

```
<%@ taglib uri="http://struts.apache.org/tags-bean" %>
<bean:size id="mySize" name="mybean" property="list" />
```

Get the bean named "mybean" from where the property named "list" is obtained. In the variable "mySize" the list size is counted.

[example 3]

```
<%@ taglib uri="http://struts.apache.org/tags-bean" %>
<bean:size id="mySize"
   collection="<%= request.getParameterValues("myCheckBox") %>" />
```

Get the parameter values named "myCheckBox" and its size is stored to "mySize" variable.

3) <bean:write> tag

Output the content of Bean within the scope.

Table 26 - Struts <bean:write> tag

Attribute	Description	Type
filter	If this attribute is set to true, the value will be filtered for characters that are sensitive in HTML and escaped ("<"-> "&lt;"). The default is true.	boolean
ignore	If this attribute is set to true, although the bean does not exist, returns without any exception. If false, exception happens.	boolean
name	Bean name that contains the collection. Mandatory	String
property	Property name declared in the Bean.	String
scope	The scope of the Bean to be accessed. If not specified, the available scopes are searched in ascending sequence.	String

[example 1]

```
<%@ taglib uri="http://struts.apache.org/tags-bean" %>
<bean:write name="name" scope="session" />
```

Output the bean named "name" in the session scope.

[example 2]

```
<%@ taglib uri="http://struts.apache.org/tags-bean" %>
<bean:write name="user" property="name" scope="session" />
```

Output the value of property named "name" from the bean named "user" in the session scope.

6.3.3. Logic Tag Library

Logic Tag Library enables to process logical functions. It contains iterate tag to extract element from the collection and conditional operation.

Table 27 - Struts Logic Tag Library

Tag	Remarks
<logic:empty>	Evaluate if the requested variable is either null or an empty string
<logic: equal>	Evaluate if the requested variable is equal to the specified value
<logic: forward>	Forward control to the page specified in <global-forwards> element of struts-config.xml.
<logic: greaterEqual>	Evaluate if the requested variable is greater than or equal to the specified value
<logic: greaterThan>	Evaluate if the requested variable is greater than the specified value
<logic: iterate>	Repeat the nested body content of this tag over a specified collection

<logic: lessEqual>	Evaluate if the requested variable is less than or equal to the specified value
<logic: lessThan>	Evaluate if the requested variable is less than the specified value
<logic: match>	Evaluate if the specified value is an appropriate substring of the requested variable
<logic: messagesNotPresent>	Generate the specified object if the specified message is not present in any scope
<logic: messagesPresent>	Generate the specified object if the specified message is present in any scope
<logic: notEmpty>	Evaluate if the requested variable is neither null, nor an empty string, nor an empty java
<logic: notEqual>	Evaluate if the requested variable is not equal to the specified value
<logic: notMatch>	Evaluate f the specified value is not an appropriate substring of the requested variable
<logic: notPresent>	Generate the specified object if the specified value is not present in this request
<logic: present>	Generate the specified object if the specified value is present in this request
<logic: redirect>	Render an HTTP Redirect Performs an HttpServletResponse

1) **<logic:iterate> tag**

This tag repeats the nested tag content until the end of specified collection.

Table 28 - Struts <logic:iterate> tag attribute

Attribute	Remarks
collection	A collection that is received from scriptlet.
id	The name of a page scope JSP bean that will contain the current element of the collection on each iteration, if it is not null.
indexId	The name of a page scope JSP bean that will contain the current index of the collection on each iteration.
length	The maximum number of entries (from the underlying collection) to be iterated through on this page.
name	The name of the JSP bean containing the collection.
offset	The zero-relative index of the starting point at which entries from the underlying collection will be iterated through.
property	Name of the property, of the JSP bean specified by name, whose getter returns the collection to be iterated.
scope	The bean scope within which to search for the bean named by the name property, or "any scope" if not specified.
type	Java class name of the element to be exposed through the JSP bean named from the id.

[example 1]

```
<%@ taglib uri="http://struts.apache.org/tags-logic"
prefix="logic" %>
```

```
<%@ taglib uri="http://struts.apache.org/tags-bean"
prefix="bean" %>

<% String[] strsFroots={"orange","apple","banana"};
   request.setAttribute("froots", strsFroots); %>
<ul>
<logic:iterate id="myFroots" name="froots">
    <li><bean:write name="myFroots" />
</logic:iterate>
</ul>
```

<logic:iterate> tag get the *String* array collection of "froots". Each element is stored in "myFroots". <bean:write> tag output the value of the variable "myFroots".

[result]

```
    * orange
    * apple
    * banana
```

[example 2]

```
<logic:iterate id="text" collection="<%=strsFroots %>" offset="1"
length="2">
    <bean:write name="text" /><br>
</logic:iterate>
```

This is an example of using collection attribute. The offset is 1, and the number of iteration is 2.

[result]

```
apple
banana
```

2) **Conditional logic**

<logic:equal>, <logic:notEqual>, <logic:greaterEqual>, <logic:greaterThanEqual>, <lessEqual>, <lessThan> tag compares the specified value with the value of Java bean, cookie, HttpRequest, or Http header. If the expression is true, the body tag will be executed.

Table 29 - Struts Attributes of Conditional logic

Attribute	Remarks
name	The name of the JSP bean to be evaluated.
property	Name of the property of the JSP bean specified by name

cookie	The variable to be compared is the value of the cookie whose name is specified by this attribute.
parameter	The variable to be compared is the first, or only, value of the request parameter specified by this attribute.
header	The variable to be compared is the value of the header whose name is specified by this attribute.
value	The constant value to which the variable, specified by other attribute(s) of this tag, will be compared.

[example]

```
<logic:equal name="MyTagBean" property="color" value="1">
    Red.
</logic:equal>
```

If the value of color property of bean named "MyTagBean" equals to "1" then, "Red" will be output.

6.4. Struts Validation

One of the common functions in the Web application is validate the input values before they are passed to business logic to avoid that error data are processed unexpectedly. *Struts* offers an additional facility to validate the input fields it has received.

6.4.1. Validate method

The *validate*() method is a method of "*ActionForm"* class. This method is called by the controller servlet after the bean properties have been populated, but before the corresponding action class's *execute()* method is invoked.

[How to set]

1) In your form bean class override the *validate()* method.

2) Write the validation in the *validate()* method.

3) In case of error, set error content to *ActionErrors* class that is the return value of the *validate*() method.

4) Add "*validate*" and "*input*" attribute in the *"struts-config.xml*"

5) Define the error message to be displayed in the message resources file.

[How it works]

1) In case No error;

AcctionErrors class always contains *ActionMessage* class. If no error happens, Struts returns either null or a zero-length *ActionErrors* instance, and the controller servlet will proceed to call the execute method of the appropriate *Action* class.

2) In case Error exists;

An *ActionErrors* instance is returned containing *ActionMessage*, which are classes that contain the error message keys (into the application's MessageResources bundle) that should be displayed. The controller servlet will store this array as a request attribute suitable for use by the <html:errors> tag, and will forward control back to the input form (identified by the input property for this *ActionMapping*).

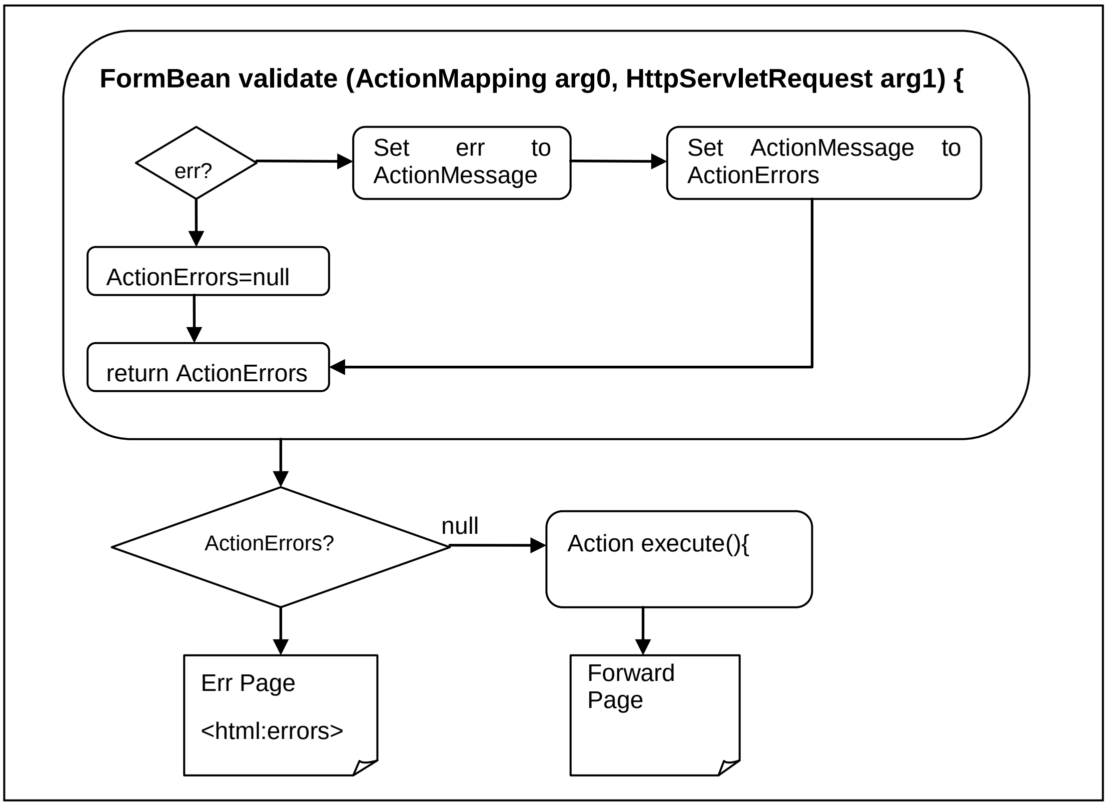


Figure 84 Struts Validate Method flow

3) Form Bean file

The bean class is created as usual way extending the *ActionForm* class.

[ValidateFormBean.java]

```
public class ValidateFormBean extends ActionForm {
…
    @Override
    public ActionErrors validate(ActionMapping arg0,
                               HttpServletRequest arg1) {
```

```
        ActionErrors errors = new ActionErrors();
        try{
        // Required check
            if ((getName()==null)||("".equals(getName()))){
                errors.add("name",
                new ActionMessage("errors.namerequired"));
                return errors;
            }
        }catch (Exception e){
            e.printStackTrace();
        }
    return errors;
    }
}
```

This class creates *validate()* method to define error case.

```
        ActionErrors errors = new ActionErrors();
```

The ActionErrors object is created.

```
            if ((getName()==null)||("".equals(getName()))){
                errors.add("name",
                        new ActionMessage("errors.namerequired"));
                return errors;
```

Check if the "*name*" input has value. If it is null or space, then it is considered as error. By *add*() method of "*ActionErrors"* class, set "*ActionMessage"* object to "*ActionErrors*". "*ActionMessage"* is created indicating what error message will be shown extracting from "*MessageResources"* file. The "*errors.namerequired*" is a key in the "*MessageResources"* file to get error message.

4) MessageResources file

```
errors.prefix=<FONT color="#FF0000">
errors.suffix=</FONT>
# -- validator --
errors.namerequired= Please input your name.
```

In the MessageResources file (with extension of "*properties*"), the constant messages are defined with key. "errors.namerequired" key has "Please input your name" value, which is reffered by "*ActionMessage"*.

"errors.prefix" and "erros.suffix" are keys that are shown as prefix and suffix respectively in case of error. For example, if the "*ActionMessage"* is set as follows;

```
errors.add("name",
            new ActionMessage("errors.namerequired"));
```

then

```
<FONT color="#FF0000">
Please input your name.
</FONT>
```

will be described in HTML and the error message is shown in red.

4) struts-config.xml

```
<action path="/govaltest"
        validate="true"
         input="/jsp/ValidateTest.jsp"
    type="action.ValidateAction"
    name="MyTagBean"
    scope="request">
    <!-- (3)Where to dispatch -->
    <forward name="result" path="/jsp/result.jsp"/>
</action>
```

In the "*struts-config.xml*", "*validate*" attribute is set to "*true"* to be valid the validation. The "*input*" attribute indicates the page that is forwarded in case of error exists.

5) <html:errors> tag in JSP

To display the error content, <html:errors> tag is described.

```
  <html:errors property="name" />
```

This tag extracts String, String array or ActionErrors object from scope. In this case, it looks for the "name" property from the ActionErros and shows the error message that is stored in ResourceMessages file.

6.4.2. Struts Validator

Struts validator is a mechanism to validate input values according to the error content defined in the "*validation.xml*" in stead of inside the program.

1) struts-config.xml

The configuration for "*struts-config.xml"* is the same as the previous "*Validate()"* method. The "*validate"* and "*input"* attributes are defined to indicate that validation is effective and where to forward in case of error.

```
<action path="/govalidator"
```

```
        validate="true"
        input="/jsp/ValidatorTest.jsp"
        type="action.ValidateAction"
        name="MyValidatorBean"
        scope="request">
        <!--Where to dispatch -->
        <forward name="result"
        path="/jsp/result.jsp"/>
</action>
```

Additionally, the following plug-in tag will be required to load Validator Resources.

```
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
   <set-property
       property="pathnames"
       value="/org/apache/struts/validator/validator-rules.xml,
             /WEB-INF/validation.xml"/>
</plug-in>
```

2) Form Bean class

To use *Struts* Validator the form bean class now extends *ValidatorForm* class which takes charge of Validation.

[action/MyValidatorForm.java]

```
package action;

import javax.servlet.http.HttpServletRequest;

import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionMessage;
import org.apache.struts.util.LabelValueBean;
import org.apache.struts.validator.ValidatorForm;
//Declare Bean
public class MyValidatorForm extends ValidatorForm{
    private String name;
    private String age;

    // Access method
    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }

    public String getAge() {
        return age;
    }
    public void setAge(String age) {
        this.age = age;
    }
}
```

2) validation.xml

This error configuration file describes validator rule to be applied and the message to be shown.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE form-validation PUBLIC
    "-//Apache Software Foundation//DTD Commons Validator Rules
Configuration 1.3.0//EN"
    "http://jakarta.apache.org/commons/dtds/validator_1_3_0.dtd">

<form-validation>
    <formset>
        <!-- Validator form -->
        <form name="MyValidatorBean">
            <field
                property="name"
                depends="required">
                    <arg key="name" resource="false"/>
            </field>

            <field
                property="age"
                depends="required,intRange">
                    <arg position="0" key="age" resource="false"/>
                    <arg position="1"
                           name="intRange"
                           key="${var:min}"
                           resource="false"/>
                       <arg position="2"
                           name="intRange"
                           key="${var:max}"
                           resource="false"/>
                       <var>
                       <var-name>min</var-name>
                       <var-value>10</var-value>
                    </var>
                    <var>
                       <var-name>max</var-name>
                       <var-value>80</var-value>
                    </var>
          </field>
       </form>
   </formset>

</form-validation>
```

```
       <form name="MyValidatorBean">
```

The error configuration is specified for each bean form. The name of form should match the action element's name attribute in "*struts-config.xml*".

```
             property="name"
             depends="required">
```

This specifies the property to be validated. "depends" attributes indicates the content of validations. *Struts* has standard built in validations defined in "*validator-rules.xml*". The built-in validations are associated with the default error message key which values are defined in the message resources properties file. The following table shows some of the rules used frequently.

Table 30 - Struts standard built in validations

rule	remarks	Default error message key
required	mandatory field validation.	errors.required
minlength	validate input data isn't less than a specified minimum length.	errors.minlength
maxlength	validate input data doesn't exceed a specified maximum length.	errors.maxlength
mask	validate format according to a regular expression.	errors.invalid
integer	validates that a field can be converted to an Integer.	errors.integer
intRange	validates that an integer field is within a specified range.	errors.range

- With <arg> tag, some parameters can be specified to replace the message resources file. For example the following <arg> tag specifies the parameter with "name" as character.

```
<arg key="name" resource="false"/>
```

- If the message resource file has a key "errors.required" as follows;

```
errors.required={0} is required.
```

the paramer {0} will be replaced with "name", and "name is required" will be generated as the error message. If resource="false" is omitted, then the system searches the key named "name" in the message file.

For the validation of "age", "required" and "intRange" validations are applied to the field.

```
property="age"
depends="required,intRange">
```

The same as the case of "name", if "age" field does not have any input, then "age is required" message will be shown according to the next <arg> tag..

```
<arg position="0" key="age" resource="false"/>
```

The following <arg> tag shows how to set the error message for intRange rule.

```
<arg position="1"
       name="intRange"
       key="${var:min}"
       resource="false"/>

    <var>
       <var-name>min</var-name>
       <var-value>10</var-value>
    </var>
```

position="1" means the second parameter for "intRange" error that corresponds to the key "errors.range" in message file. "${var:min}" means that a variable that is specified in <var> tag with <var-name>, that is value 10, will be replaced with the second parameter. The error message defined in the message resource file by default is;

```
errors.range={0} is not in the range {1} through {2}.
```

As a result of the definition for "intRange", if the value in the age field has range error, then the following message will be shown;

```
age is not in the range 10 through 80.
```

3) Action file

There is nothing special to define in action file for validation. It is created as usual and the business logic is called.

[action/ValidateAction.java]

```
package action;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

public class ValidateAction extends Action {

       public ActionForward execute(
                 ActionMapping mapping,
                 ActionForm form,
                 HttpServletRequest request,
                 HttpServletResponse response)
                              throws Exception {

        return mapping.findForward("result");
    }
}
```

4) jsp file

The same as the *validate()* case, <html:errors> shows the error message to HTML in case of field error.

```
name     <html:text property="name" size="16"/><br>
  <html:errors property="name" />
  <br>
age  <html:text property="age" size="2"/>
  <br>
  <html:errors property="age" />
```

Exercise 6: Apache Struts

1. Create "*StrutsPrj*" project in Eclipse with Tomcat. Locate necessary files to create "HelloWorld" described in the textbook. Execute it from the browser if the programs and configuration files are set correctly.

2. Create "*StrutsValidationPrj*" project in Eclipse with Tomcat. Create the jsp named "*ValidatorTest.jsp*" file which validates "required" validation for "name" and "age" field, "Integer" and "intRange" validation for "age" field.

[first page of ValidatorTest.jsp]

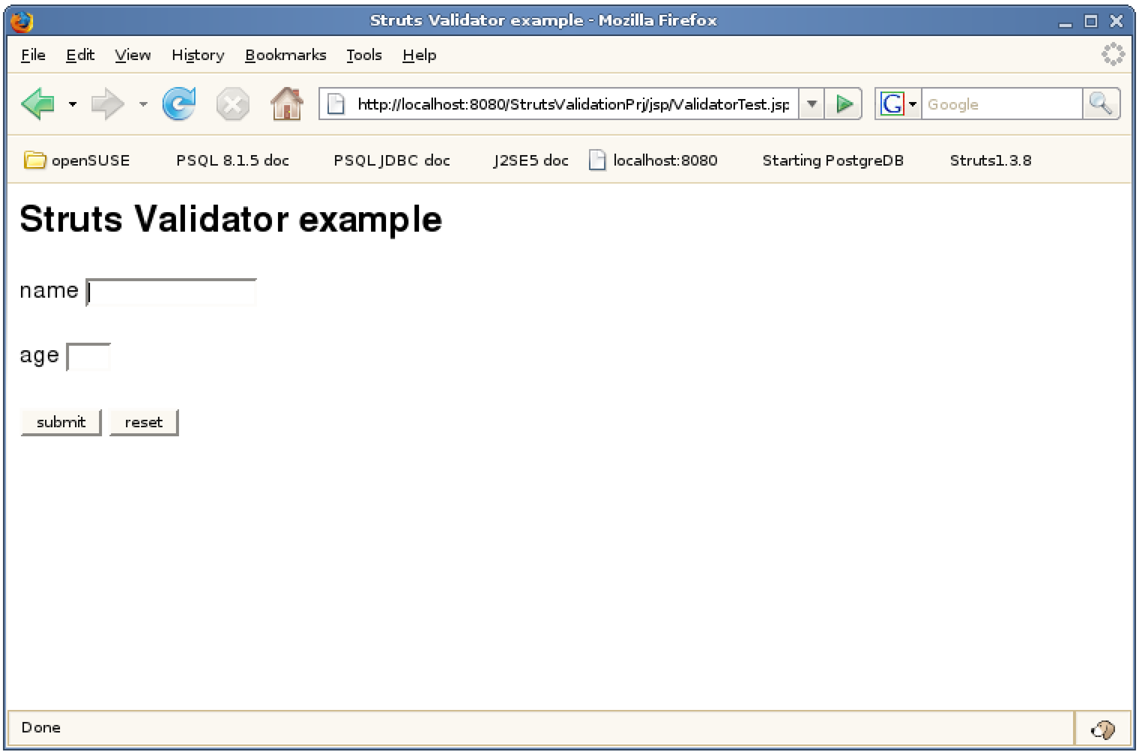
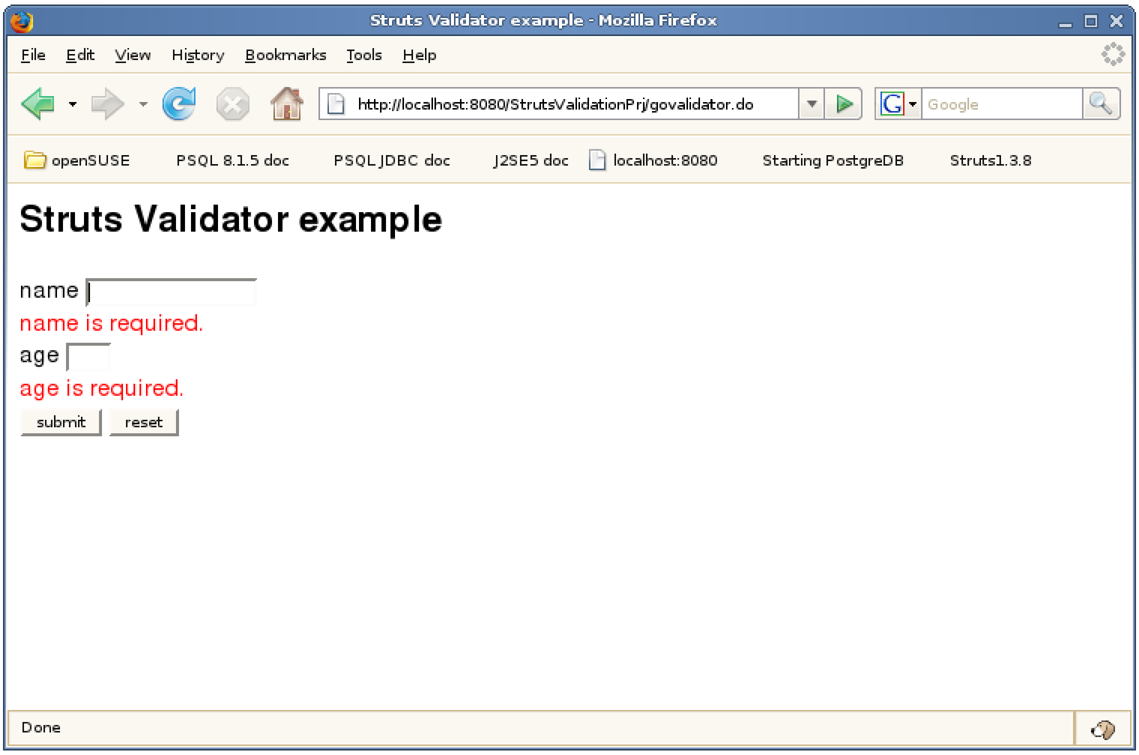

Struts Validator example - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/StrutsValidationPrj/jsp/ValidatorTest.jsp
Google
openSUSE
PSQL 8.1.5 doc
PSQL JDBC doc
J2SE5 doc
localhost:8080
Starting PostgreDB
Struts1.3.8
Struts Validator example
name
age
submit
reset
Done

Figure 85 Struts Validation Exercise first page

[When the fields are empty]



http://localhost:8080/StrutsValidationPrj/govalidator.do

Struts Validator example

name

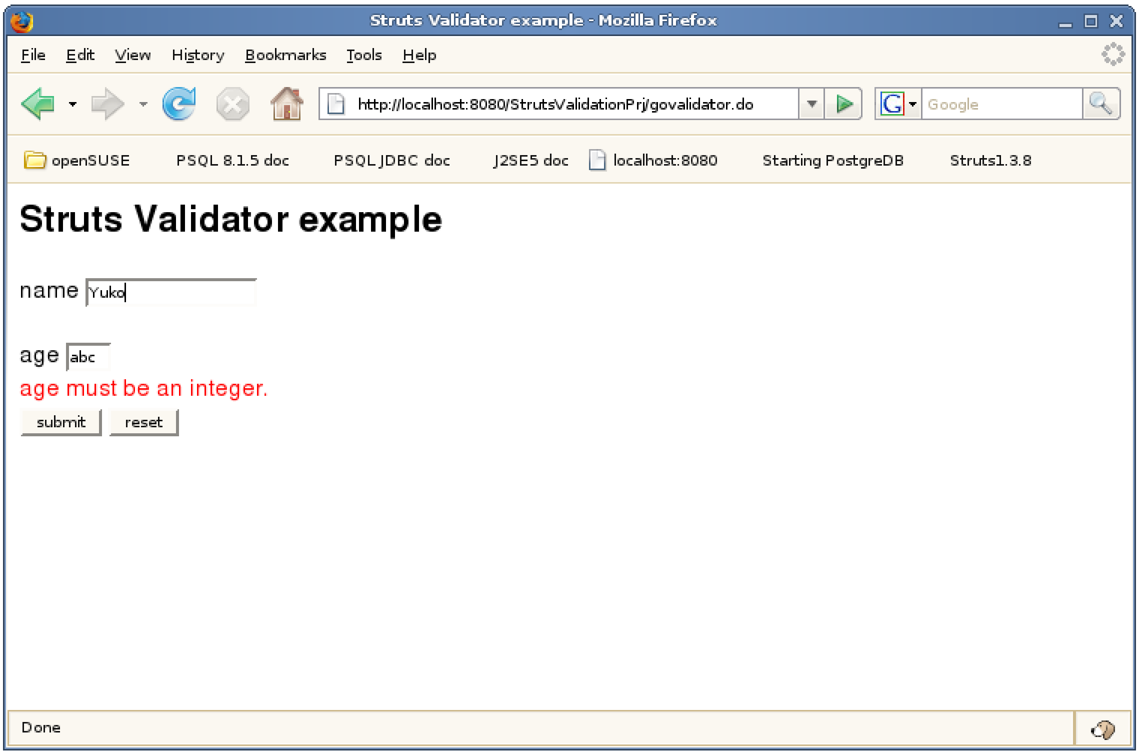
name is required.

age

age is required.

Figure 86 Struts validation exercise required error

[When age is other than Integer]



http://localhost:8080/StrutsValidationPrj/govalidator.do

Struts Validator example

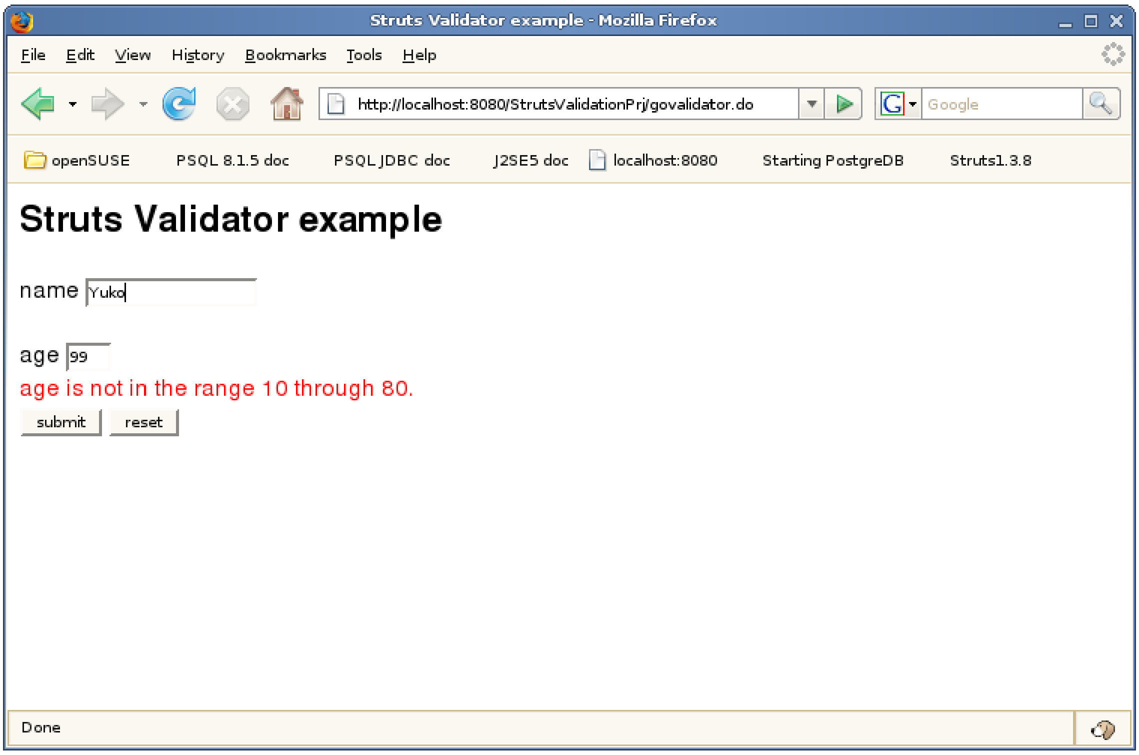
name Yuko

age abc

age must be an integer.

Figure 87 Struts validation exercise Integer error

[When age is not between 10 and 80]


Struts Validator example - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Google
openSUSE PSQL 8.1.5 doc PSQL JDBC doc J2SE5 doc localhost:8080 Starting PostgreDB Struts1.3.8
submit
reset
Done

http://localhost:8080/StrutsValidationPrj/govalidator.do

Struts Validator example

name Yuko

age 99

age is not in the range 10 through 80.

Figure 88 Struts validation exercise range error

You need the following files;

- struts-config.xml
- action/MyValidatorForm.java
- action/ValidateAction.java
- jsp/ValidatorTest.jsp (You can use HTMLTagTest.jsp)
- jsp/result.jsp (You can use result.jsp for HTMLTagTest)
- validation.xml
- java/MessageResource.properties (if it is created under "java/" directory)

7. Framework; Spring

7.1. What is Spring Framework

Spring Framework is an open source application framework based on the Java/J2EE platform. It is a light weight container to create J2EE application architecture that provides functionalities such as

- DI (Dependency Injection)
- AOP (Aspect Oriented Programming)
- MVC framework
- Abstract JDBC framework
- ORM(Object Relational Mapping) integreation

Struts Framework covers mainly the presentation tier, and *Hibernate* covers data access tier, while *Spring* covers whole tiers.

7.1.1. How to install Spring

1) Spring Framework can be downloaded from;

http://www.springframework.org/download

The current version is 2.0.4 and *spring-framework-2.0.4-with-dependencies.zip* contains all the modules on which Spring depends.

2) Unzip the package.

3) Under "dist/modules" directory, the necessary packages can be found. According to the application needs, pass the classpath to the jar files. The "*spring.jar*" contains all the modules. If it is used in Web Application, the jar file will be placed under *WEB-INF/lib* directory.

7.2. Spring DI Container

7.2.1. What is DI Container

Dependency Injection (DI) is a software design pattern which removes the dependency between components from the program source code and enables to inject the dependent object from external configuration file. The relation between components is established by way of interfaces to achieve loose coupling without directly instantiating classes from the component. The information of which component is dependent is described in the configuration file.

Let's see some example of isolating the interface and implementation of the program.

1) The first Hello program that we create when we start to learn Java Language will be like;

[**MyFirstHello.java**]

```
public class MyFirstHello {
    public static void main(String[] args) {
        System.out.println("Hello Pablo");
    }
}
```

As you understand, in this example the *main()* method directly outputs the message "Hello Pablo". In case the people name changes, or the way to output the message changes, the program should be recompiled.

2) Let's separate the compoment so that the program will be more generalized one.

[HelloServiceWithoutInterface.java]

```
public class HelloServiceWithoutInterface {
    String i_strPersonName;
    public void printHello() {
        System.out.println("Hello," + i_strPersonName);
    }
    public void setPersonName(String strName) {
        i_strPersonName = strName;
    }
}
```

In this class the name of person is still unknown until it is set from outside. The *printHello()* method decides how the message is printout.

The main program that calls the above program will be;

[HelloWithoutInterface.java]

```
public class HelloWithoutInterface {

    /**
     * Hello program without using Interface
     */
    public static void main(String[] args) {
        HelloServiceWithoutInterface helloService =
            new HelloServiceWithoutInterface();
        helloService.setPersonName("Pablo");
        helloService.printHello();
    }
}
```

Now the person name other than "Pablo" can be set from main program. The way to output the message is declared in the *HelloServiceWithoutInterface* class which is instantiated by "*new*" keyword from main program. As a result this program is more flexible than the previous "*MyFirstHello*" class due to no need of recompilation in case of the change of person name. But What is the problem?

- The main program can not be compiled until the called program is created, since the main program needs to know the exact class name (*HelloWithoutInterface*) and its method names.
- When the main program needs to change to call another better program to output the message, the modification of the class name should occur. All the class type with the class name also should be modified.

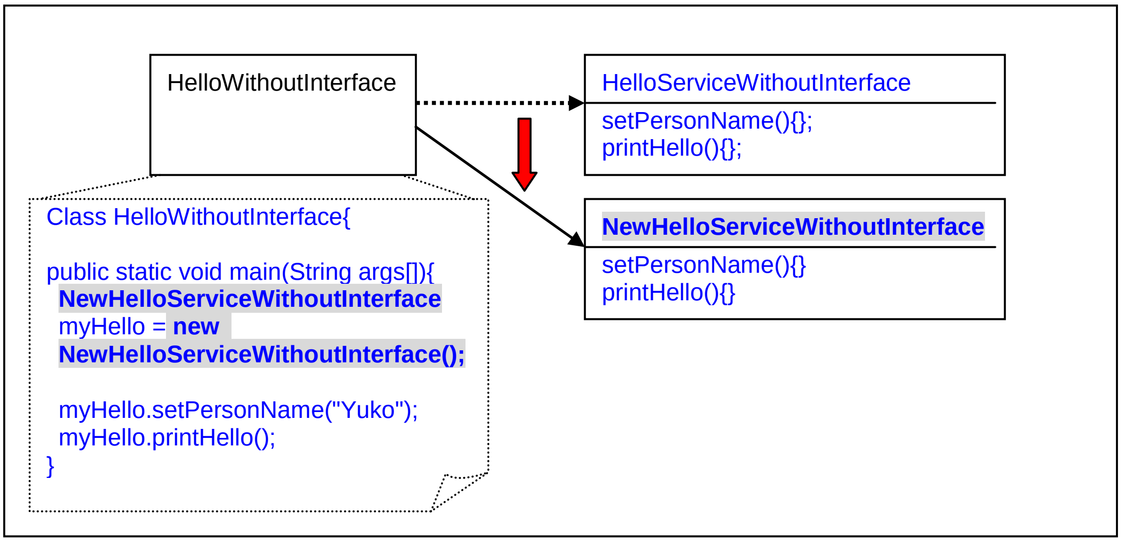

HelloWithoutInterface
HelloServiceWithoutInterface
setPersonName(){};
printHello(){};
NewHelloServiceWithoutInterface
setPersonName(){}
printHello(){}
Class HelloWithoutInterface{
public static void main(String args[]){
NewHelloServiceWithoutInterface
myHello = new
NewHelloServiceWithoutInterface();
myHello.setPersonName("Yuko");
myHello.printHello();
}

Figure 89 Program without Interface

☆ The above case seems that only the instantiation of the class is affected, but also the class type will be affected if it is used as parameter or return of a method. For example;

```
private NewHelloServiceWithoutInterface getService(){
    return new NewHelloServiceWithoutInterface();
…
NewHelloServiceWithoutInterface service = getService();
service.printService();
}
```

3) Let's separate interface and implementation to be more independent of the person name and of the way to output.

[HelloServiceInterface.java]

```
public interface HelloServiceInterface {
     public void setPersonName(String strName);
     public void printHello();
}
```

The following program implements the above interface.

[HelloServiceImpl.java]

```
public class HelloServiceImpl implements HelloServiceInterface {
    String personName;

    public void printHello() {
        System.out.println("Hello," +personName);
    }
    public void setPersonName(String strName) {
        personName = strName;
    }
}
```

The main program to instantiate the above service program is;

[HelloWithInterface.java]

```
public class HelloWithInterface {

    /**
     * Hello program without Interface
     */
    public static void main(String[] args) {
        // Instantiate HelloImpl
        HelloServiceInterface helloInterface =
                    new HelloServiceImpl();
        helloInterface.setPersonName("Pablo");
        helloInterface.printHello();
    }
}
```

The *HelloServiceImple* class is instantiated with *HelloServiceInterface* type. As long as the interface is defined, the method names are already known. Since all the reference to the implemented class is pointed to interface name, in case of change of implemented class, no modification is required except the following part ;

```
        HelloServiceInterface helloInterface =
                    new HelloServiceImpl();
```

Here the implemented class name is specified explicitly in the program and its reference is returned to the interface type.

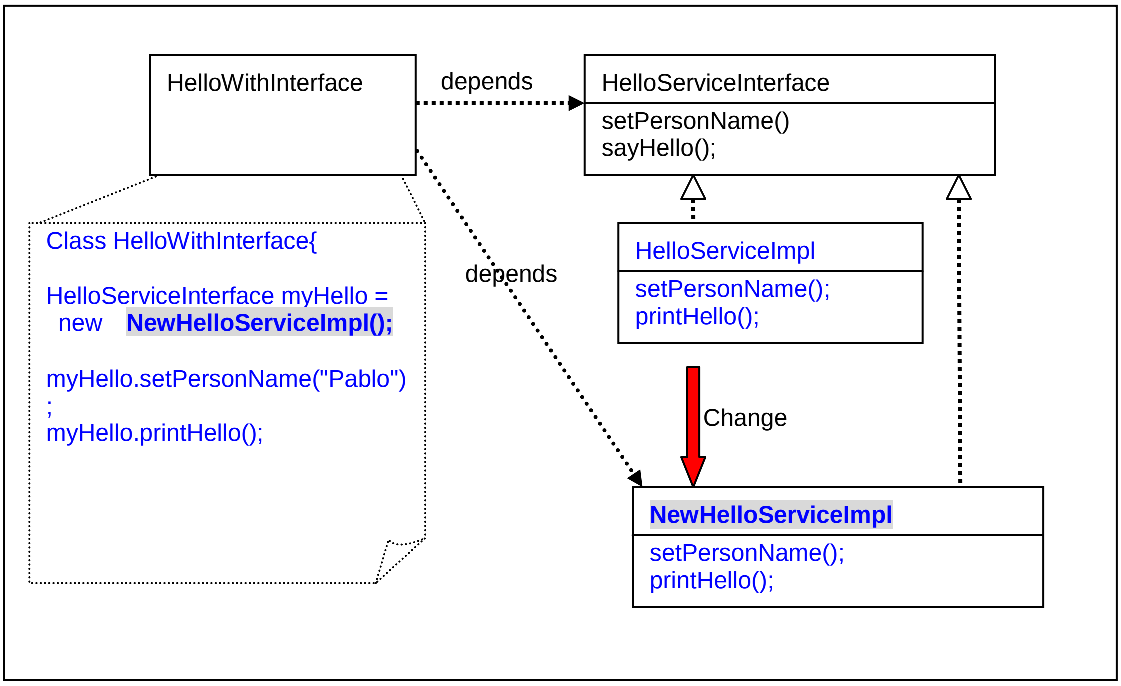


Figure 90 Program with Interface

4) Let's try DI container to see how the implemented class is defined in the configuration file.

☆ Set classpath to the following files;

- /dist/spring.jar
- /lib/Jakarta-commons/commons-logging.jar

[MySpringConfig.xml]

```
<?xml version="1.0" encoding="UTF-8"?>
 <beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xs
d">
  <bean id="serviceID" class="HelloServiceImpl">
     <property name="personName" value="Pablo"/>
   </bean>
 </beans>
```

The <bean> tag specifies the classes that will be instantiated by DI Container.

The "*HelloServiceImpl*" class is defined with the name "*serviceID*". The property "*personName*" of "*HelloServiceImpl*" will have the default value "*Pablo*". Place this file to where file path is passed (for example, the project root directory if you use Eclipse).

[HelloWithSpring.java]

```
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.FileSystemResource;

public class HelloWithSpring {

    public static void main(String[] args) {
        try {
            // get Configuration file
            XmlBeanFactory factory = new XmlBeanFactory(
                              new FileSystemResource(
                              "MySpringConfig.xml"));
            // get implemented class HelloServiceImpl
            HelloServiceInterface service =
            (HelloServiceInterface) factory.getBean("serviceID");
            // call printHello method of HelloSercieImpl
            service.printHello();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
            XmlBeanFactory factory = new XmlBeanFactory(
                              new FileSystemResource(
                              "MySpringConfig.xml"));
```

The bean configuration file "*MySpringConfig.xml*" is loaded using *FileSystemResource* which looks for the file from the file system path.

```
            HelloServiceInterface service =
            (HelloServiceInterface) factory.getBean("serviceID");
```

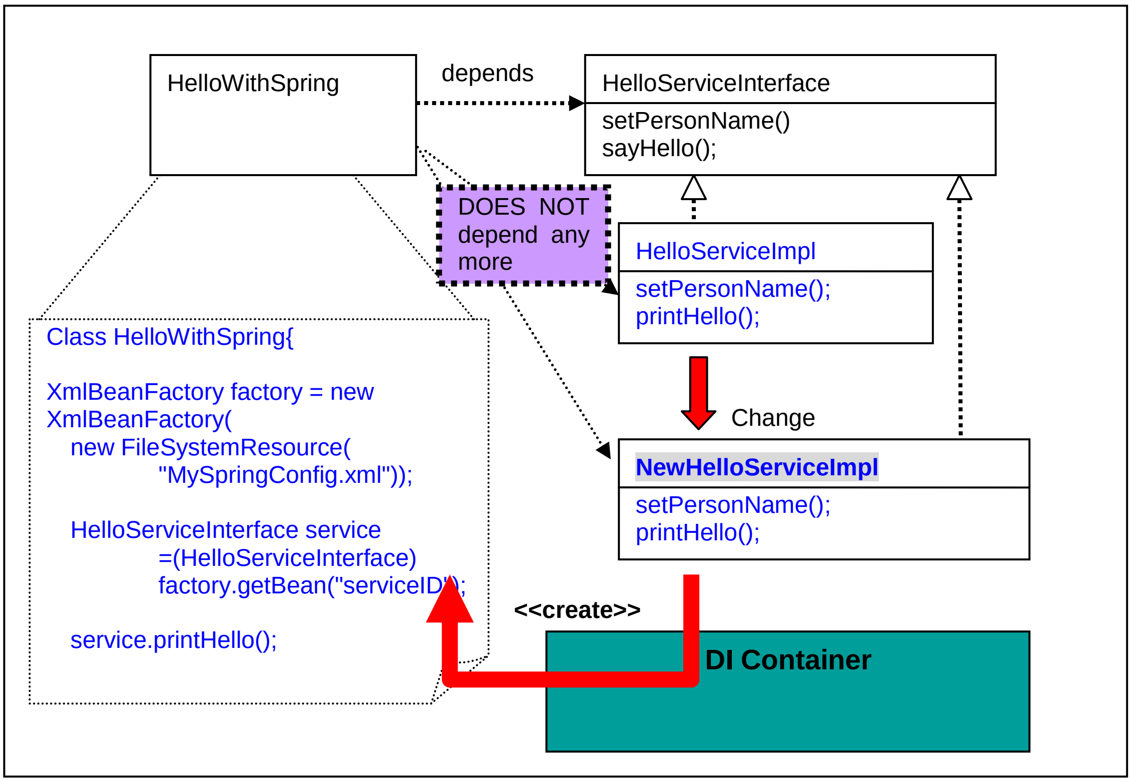
The bean class that is defined in the configuration file with the name "*serviceID*" is loaded that corresponds to "*HelloServiceImpl*" according to the definition. The default value "*Pablo*" for the property "*personName*" defined in the file also is set at the same time.

[result]

```
Hello,Pablo
```

```
2007/04/27 10:21:49
org.springframework.beans.factory.xml.XmlBeanDefinitionReader
loadBeanDefinitions
information: Loading XML bean definitions from file
[C:\Data\Myammer\ashare\workspaceWin\SpringTestPrj\MySpringConfig
.xml]
```

In this way, the implemented class name disappeared from the source code and only the interface name remains, which means that in case of replacing the implemented class to another, no any modification happens just changing the external configuration file.



```
Class HelloWithSpring{

XmlBeanFactory factory = new
XmlBeanFactory(
   new FileSystemResource(
              "MySpringConfig.xml"));

   HelloServiceInterface service
              =(HelloServiceInterface)
              factory.getBean("serviceID");

   service.printHello();
```

Figure 91 Spring DI Container

As a result instantiation of the class with "*new*" keyword does not exist since the implemented class is not created by caller program but by DI Container who passes the reference of the class to the caller program. The following figure demonstrates the class diagram to show the relationship between components.

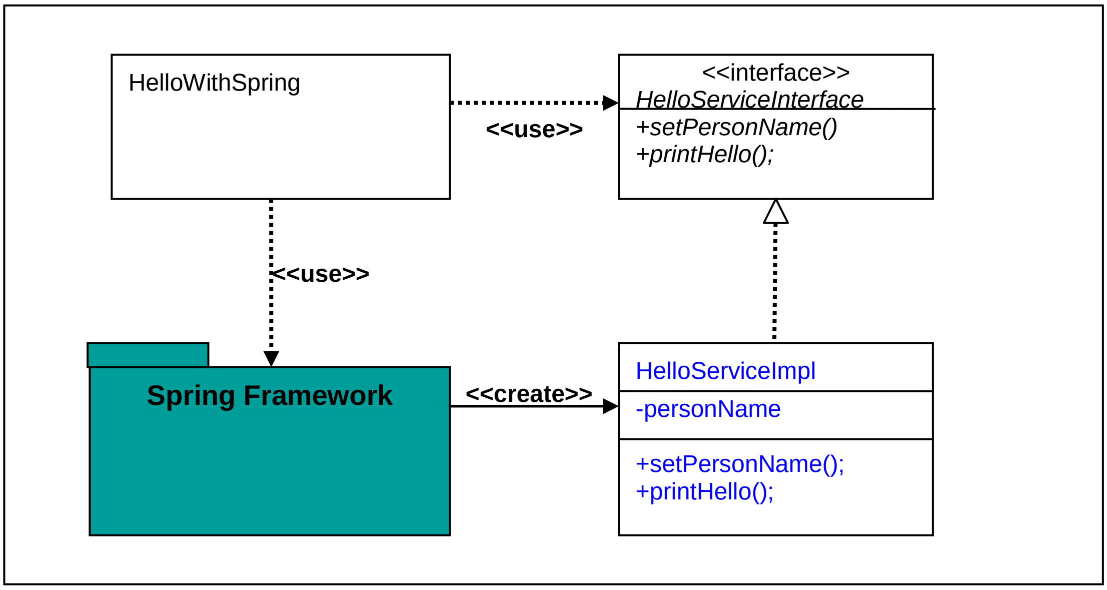


Figure 92 Class diagram of Spring Framework

Since the dependency of the components is injected by the Container, this mechanism is called Dependency Injection.

7.2.2. Why DI Container

- **Ease of Testing**
  Avoiding dependencies between components makes it easy to control unit test using mock objects (program just for test). Later the component will be replaced with the production version without any modification in the program. This also contributes the efficiency of development.
- **Maintainability**
  Since the relation between the components is loose coupling, the modification of one component does not affect to the other.
- **Reusability**
  Since each component is independent of each other, one component is easily reused in other system.

7.2.3. Spring Bean Factory

Spring Bean Factory administrates the structure of objects based on the bean configuration file which is loaded at run time.

[How to use]

```
Resource resource = new
ClassPathResource("Bean_configuration_file_name");
BeanFactory beanFactory = new XmlBeanFactory(resource);
class_name1 ObjectName =
              (class_name1)beanFactory.getBean(id_name1);
```

[Bean Configuration File]

```
<beans>
  <bean id="id_name1" class="package_name.class_name1">
    <property name="variable_name" value="default_string"
    <property name="variable_name" ref="id_name2">
  </bean>
  <bean id="id_name2" class="package_name.class_name2" />
<beans>
```

The bean objects declared in the <bean> tag are loaded when *getBean()* method is called with its id name. <ref> attribute is specified to refer to another object which is **also loaded** at the time of calling *getBean*() method according to the definition in <bean> tag.

☆ When property value or reference is set by Spring, setter method for each property is required since the container uses it.

7.3. Spring + Struts in Web Application

Struts is a framework that works in Presentation tier in Web Application. The roll of *Struts* is application controller to transfer the request from user to Model in MVC2 architecture. *Struts* has an *Action* class for application controller. To integrate *Struts* into *Spring*, it is necessary to construct the association between *Action* class and objects administrated by DI container. *Spring* provides several ways to associate *Spring* with *Struts* using *DelegationActionProxy* class and *ActionSupport* class

7.3.1. Spring + Struts using DelegationActionProxy class

One of the ways to integrate *Struts* with *Spring* is to use *DelegationActionProxy* class. This class is an agent class of *Action* class who calls the *Action* according to the Bean configuration of *Spring* Framework.

1) Set ContextLoaderPlugin in struts-config.xml

The *ContextLoaderPlugin* is a *Struts* plug-in that loads a *Spring* context file for the *ActionServlet* of *Struts*.

[struts-config.xml]

```
<plug-in
className="org.springframework.web.struts.ContextLoaderPlugIn"/>
```

2) Create Context file

The default name of the context file for *Spring* is the name of the mapped servlet, plus "*-servlet.xml*". If *ActionServlet* is defined in "*web.xml*" as <servlet-name>action</servlet-name>, the default context file name is "*action-servlet.xml*", and it is located under "/WEB-INF/" directory.

3) Set DelegatingActionProxy class

This class will be defined as Action class in struts-config.xml setting the class name in the "type" attribute to work as an agent.

[struts-config.xml]

```
<action-mappings>
  <action
       path="/LoginPath"
       name="LogForm"
       type="org.springframework.web.struts.DelegatingActionProxy"
       scope="request">
       <forward   name="forwardHello"
                  path="/jsp/ResultLogin.jsp" />
  </action>
</action-mappings>
```

4) Set Action Bean

Action bean in "*action-Servlet.xml"* is defined with the same path name in struts-config.xml indicating its *Action* class name.

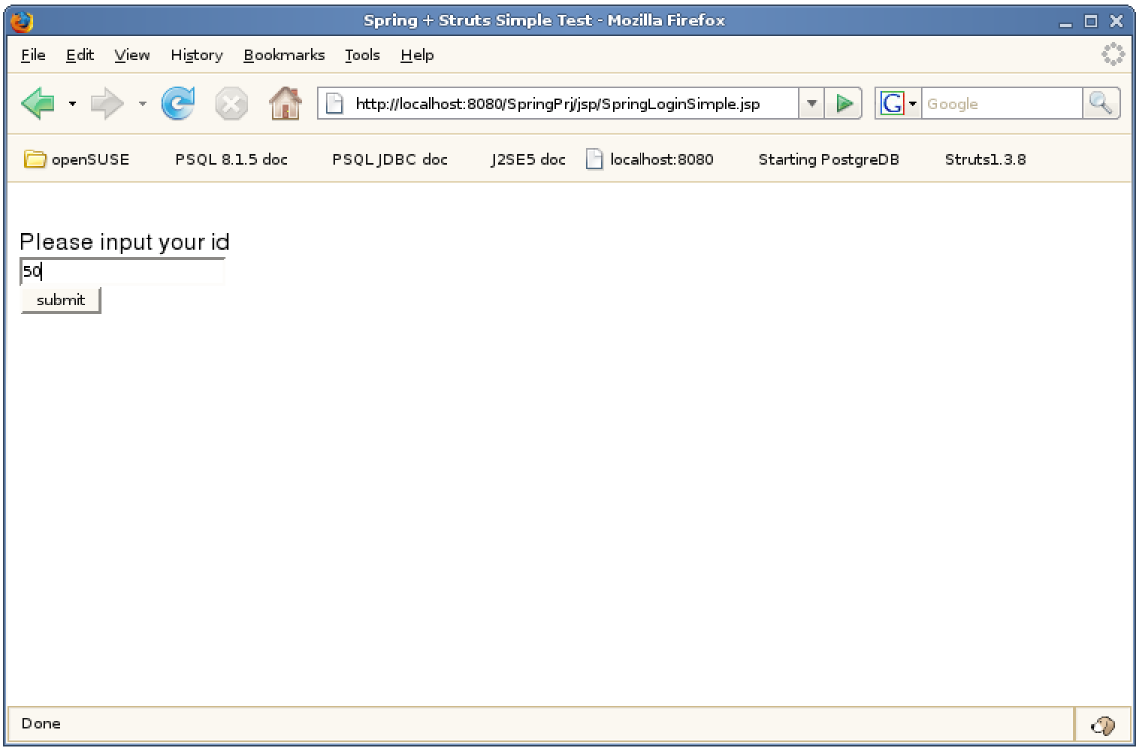
[action-servlet.xml]

```
<bean name="/LoginPath" class="action.HelloAction">
     <property name="myLogService" ref="idLoginService" />
</bean>
```

7.3.2. Sample program

Let's create a simple program of Web Application employing *Struts* and *Spring* Framework.

The first page requires you to input your id.



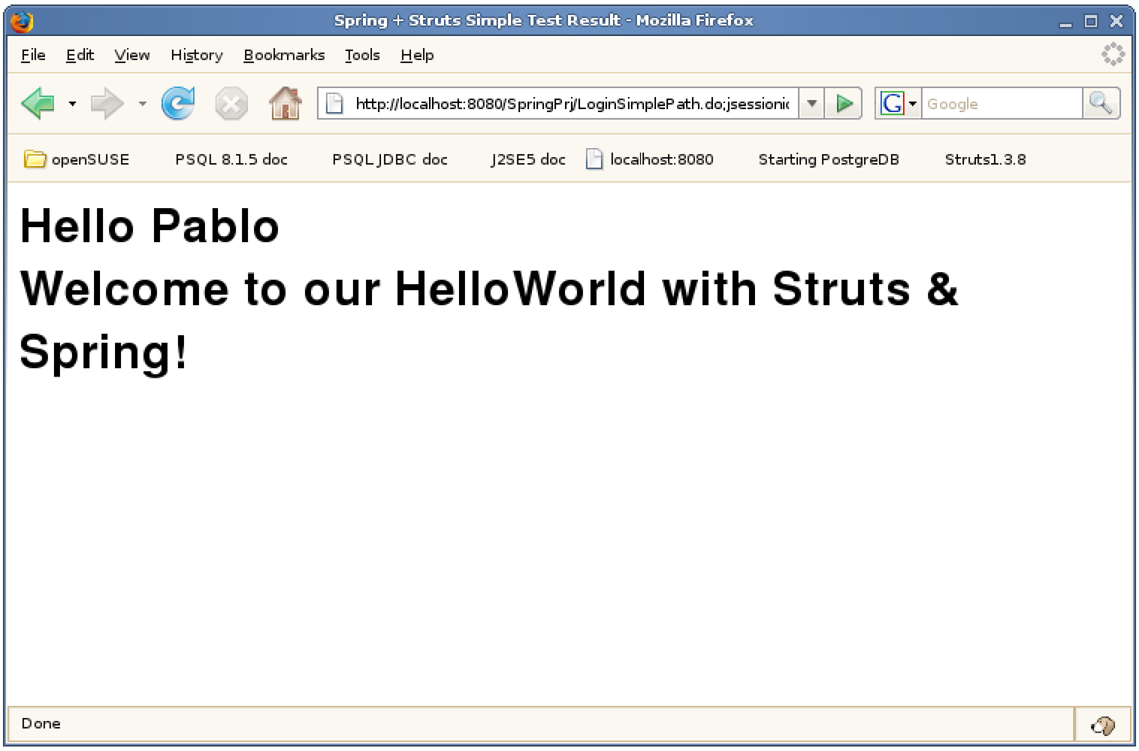
http://localhost:8080/SpringPrj/jsp/SpringLoginSimple.jsp

Please input your id

50

Figure 93 Spring + Struts sample program first page

Clicking the submit button, your name (Pablo as fixed name) will be shown with welcome message.


Spring + Struts Simple Test Result - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/SpringPrj/LoginSimplePath.do;jsessionid
Google
openSUSE PSQL 8.1.5 doc PSQL JDBC doc J2SE5 doc localhost:8080 Starting PostgreDB Struts1.3.8
Done

Hello Pablo

Welcome to our HelloWorld with Struts & Spring!

Figure 94 Spring + Struts sample program result

[struts-config.xml]

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE struts-config PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 1.3//EN"
    "http://struts.apache.org/dtds/struts-config_1_3.dtd">
<struts-config>
<!-- ========= Form Bean Definitions -->
  <form-beans>
      <form-bean
           name="LogSimpleForm"
           type="action.LoginForm"/>
  </form-beans>
<!-- ========= Action Mapping Definitions -->
  <action-mappings>
      <action
         name="LogSimpleForm"
    type="org.springframework.web.struts.DelegatingActionProxy"
         path="/LoginSimplePath"
         scope="request">
         <forward    name="forwardLogin"
                     path="/jsp/ResultLoginSimple.jsp" />
```

```
     </action>
  </action-mappings>

<!-- ================ Message Resources Definitions -->
   <message-resources parameter="java.MessageResources" />
<!-- ================ Plugin for Spring -->
   <plug-in 
className="org.springframework.web.struts.ContextLoaderPlugIn"/>
</struts-config>
```

In *"struts-config.xml"*, *DelegatingActionProxy* class is defined as type attribute of <action> tag to specify to use it as action instead of your proper action class.

Also *ContextLoaderPlugIn* is declared in the <plug-in> tag to load *Spring* context file.

[action-servlet.xml]

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd"
>
<bean name="/LoginSimplePath" class="action.LoginSimpleAction">
    <property name="personName" value="Pablo" />
</bean>

</beans>
```

In the *"action-Servlet.xml"*, the bean name is defined as the same path name declared in "*struts-config.xml*", so that *DelegatingActionProxy* can understand which class is user *Action* class. In the <property> tag, the default value is set for the variable "personName" in *LoginSimpleAction* class.

[/action/LoginSimpleAction.java]

```
package action;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

public class LoginSimpleAction extends Action {
    /* Action class that returns the default value
       of personName set in the action-servlet.xml.
     */
    private String personName;

    public ActionForward execute(ActionMapping mapping,
                ActionForm form,
                HttpServletRequest request,
                HttpServletResponse response)
                        throws Exception {
        // get LoginForm bean
        LoginForm loginForm = (LoginForm)form;

        // get UserID from the form
        String strID = loginForm.getId();
        System.out.println(strID);

        // set the user name to the form,
        // so that JSP can show the name
        loginForm.setName(personName);

        return mapping.findForward("forwardLogin");
    }
    public String getPersonName() {
        return personName;
    }
    public void setPersonName(String personName) {
        this.personName = personName;
    }
}
```

This is an *Action* class where business logic is executed. *Spring* also recognizes this class as *Action* class since it is also specified in "*action-servlet.xml*" with the same path name.

```
        LoginForm loginForm = (LoginForm)form;
        String strID = loginForm.getId();
```

Get the *Struts* "*Form Bean*" specified in the "*struts-config.xml*". In this form the input argument from *"id"* can be obtained by *getId()* method.

```
    private String personName;
…
    public void setPersonName(String personName) {
        this.personName = personName;
```

Instance value "*personName*" is declared, where its default value is set by *Spring* as *"Pablo"* using setter method.

[/action/LoginForm.java]

```
package action;

import org.apache.struts.action.ActionForm;

public class LoginForm extends ActionForm{
    private String name;
    private String id;

    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

LoginForm is a *Struts* "*Form Bean*" where data obtained from JSP is stored. This class is also used to transfer data to JSP to show the data.

[/jsp/SpringLoginSimple.jsp]

```
<%@ taglib uri="http://struts.apache.org/tags-html"
prefix="html" %>
<%@ taglib uri="http://struts.apache.org/tags-bean"
prefix="bean" %>
<html:html>
<Title>Spring + Struts Simple Test
 <BODY>
  <!-- Converted to Form tag -->
  <html:form action="/LoginSimplePath" >
  <BR>
   Please input your id<BR><html:text property="id" /><BR>
    <html:submit>
```

```
      <bean:message key="mysubmit" />
    </html:submit>
  </html:form>
 </BODY>
</html:html>
```

This is JSP for first page where your id is asked to input. <html:form> specifies "LoginSimplePath" as its action path which is a key to combine Struts and Spring framework. The value of textbox "id" is set to *LoginForm* bean.

[/jsp/ResultLoginSimple.jsp]

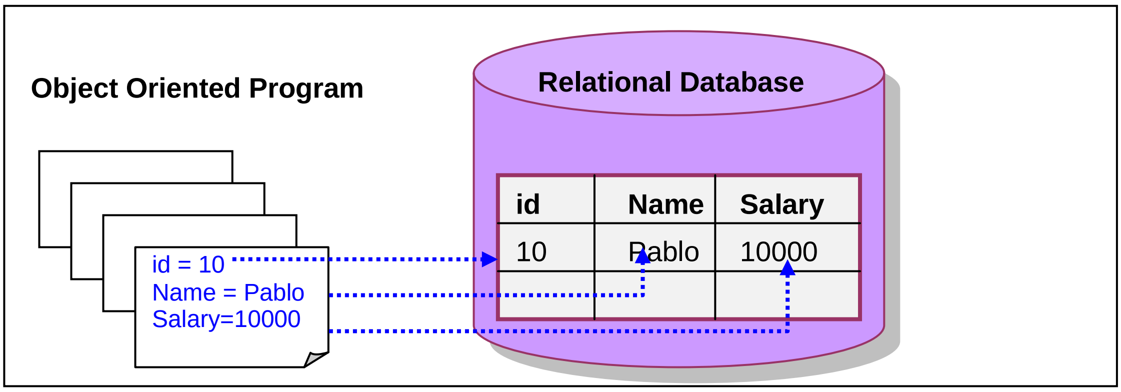
```
<%@ taglib uri="http://struts.apache.org/tags-html"
prefix="html" %>
<%@ taglib uri="http://struts.apache.org/tags-bean"
prefix="bean" %>

<html:html>
    <BODY>
      <H1>
        <bean:message key="hello" />
        <!-- Show the name obtained from Action Class -->
        <bean:write name="LogSimpleForm" property="name" />
       <br>Welcome to our HelloWorld with Spring & Struts!
      </H1>
    </BODY>
</html:html>
```

This is JSP for the result page. The user name obtained from Action class is shown.

7.4. Spring + Hibernate

7.4.1. What is O/R Mapping

Object-Relational mapping (O/R mapping) is a programming technique for associating data between data type in Relational Databases and Object Oriented programming languages. In Object-Oriented programming, data model is designed and implemented to manipulate objects, while Relational Databases are structured for better normalization and optimization of retrieving and saving data. The problem resides in how to convert the object values into database (and convert them back upon retrieval) in spite of the difference of the principal and philosophical design. Traditionally developers absorbed the difference by directly calling JDBC for its mapping. They had to implement typical work such as to establish the database connection, to retrieve the result data into Java Object. The purpose of O/R mapping is to solve the Object-Relational impedance mismatch issue and provides the seamless conversion between them reducing the typical and complicated work from developers.



id	Name	Salary
10	Pablo	10000

Figure 95 O/R Mapping

7.4.2. What is Hibernate

Hibernate is free open source software of the O/R mapping tool for the Java language developed by Red Hat. The characteristic features of Hibernate are;

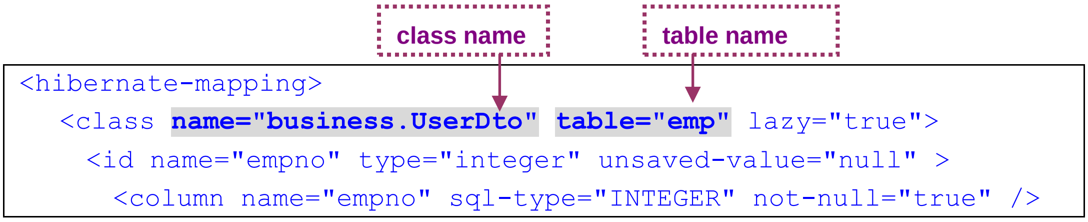
- Base on POJO(Plain Old Java Object)
- High performance
- Widely provided information about the product
- Use of HQL(Hibernate Query Language)
- Use of Mapping file for mapping

The following table shows some example of HQL comparing usual SQL;

Table 31 - Hibernate HQL

SQL	HQL
SELECT * FROM EMP	FROM Employ (Employ is NOT table but object name)
SELECT * FROM EMP WHERE ID='10'	FROM Employ p WHERE p.id='10'

In *Hibernate* Java objects are mapped to tables of Relational database. The following configuration is an example of Mapping in *Hibernate*. In the <class> tag, "*business.UserDto*" class is mapped to "emp" table. In the <id> tag shows the configuration of key in the "emp" table.



```
<hibernate-mapping>
   <class name="business.UserDto" table="emp" lazy="true">
     <id name="empno" type="integer" unsaved-value="null" >
       <column name="empno" sql-type="INTEGER" not-null="true" />
       <generator class="assigned" />
    </id>
    <property name="ename" column="ename" />
   </class>
</hibernate-mapping>
```

All the data access information is not any more described in the source code to load JDBC driver, but is specified in the configuration file instead. Introducing *Hibernate* releases developers from;

- Description about connection
- Description about SQL Statement
- Generating Java Bean to store the result set

The *Spring* Framework provides powerful support for O/R mapping including *Hibernate* in terms of resource management, DAO implementation support and transaction management. When *Hibernate* is integrated with DI container of Spring Framework, session or transaction can be obtained by way of Framework automatically.

[Benefits of using Spring Framework with O/R mapping]

- Ease of testing

  Spring's DI management makes it easy to swap the information of session instances, JDBC data source instances and O/R mapping object implementations.
- Common data access exceptions

  Spring wraps exception from individual O/R mapping object to a common runtime *DataAccessException*. This allows releasing developers from annoying description of *catch/throws* close and exception declarations.
- General resource management

  Spring can handle the location and configuration of data resources regarding to the connection or session management. For example for efficiency and transaction handling, *Hibernate* generally uses the same *Hibernate Session* similar to JDBC connection. *Spring* creates and binds the *Hibernate Session* easily.
- Transaction management

  Spring handles proper transaction including rollback in case of exceptions.

7.4.3. How to use Hibernate

1) How to Install

Hibernate can be downloaded from the following site;

http://www.hibernate.org/

Spring Framework provides dependent modules under "*lib/*" directory where necessary modules to execute *Hibernate* are found.

The following files are required to execute basic Hibernate features with Spring environment;

- hibernate3.jar	(%SpringDir%/lib/hibernate/)
- cglib-nodep-2.1.jar	(%SpringDir%/lib/cglib/)
- jta.jar	(%SpringDir%/lib/j2ee/)
- dom4j-1.6.jar	(%SpringDir%/lib/dom4j/
- commons-logging.jar	(%SpringDir%/lib/jakarta-commons/)
- commons-collection.jar	(%SpringDir%/lib/ jakarta-commons/)

☆ The required packages depend on what functions you use and in which environment. The version of each modules differs according to the *Hibernate* package you use. Please refer to the *Hibernate* manual to check required modules for your environment with corresponding version.

2) Hibernate configuration file

The following configuration demonstrates how to configure Database information for connection.

[hibernate.cfg.xml]

```
<?xml version="1.0" ?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd
">
<hibernate-configuration>
 <session-factory>
  <!--MySQL JDBC Driver connection -->
  <property
name="connection.driver_class">com.mysql.jdbc.Driver</property>
  <property
    name="connection.url">jdbc:mysql://localhost/ictti</property>
  <property name="connection.username">root</property>
  <property name="connection.password">root</property>
  <property
```

```
name="dialect">org.hibernate.dialect.MySQLInnoDBDialect</property
>
  <property name="show_sql">true</property>
  <property name="current_session_context_class">thread</property>
  <!-- Mapping files -->
  <mapping resource="EMPLOYEE.hbm.xml"/>
 </session-factory>
</hibernate-configuration>
```

SessionFactory is a global factory responsible for a particular database. The following table shows the property elements for *SessionFactory*.

Table 32 - Hibernate configuration file

element	description
connection	JDBC connection information
dialect	Specify SQL dialect for vendor specific RDBMS
show_sql	Output the generated SQL statement
current_session_context_class	Decides current scope of session. "thread" bounds to the current Java thread. There are "jta", "thread", and "managed". "Thread" enables Hibernate's automatic session context management
mapping resource	Specify mapping file name

3) Mapping configuration file

Mapping configuration file declares what the Java Class name is and what the table name is to be mapped. The file name should corresponds to what is specified in the <mapping resource> tag of *Hibernate* configuration file

[EMPLOYEE.hbm.xml]

```
<?xml version="1.0" ?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
   <class name="entity.EmpClass" table="EMP">
      <id name="empno" column="EMPNO" type="int" >
         <generator class="assigned" />
      </id>
      <property name="ename" type="string" column="ENAME" />
      <property name="job" type="string" column="JOB" />
      <property name="hiredate" type="date" column="HIREDATE" />
   </class>
</hibernate-mapping>
```

Column in EMP table

Property in EmpClass

The "*id*" element is the declaration of the identifier property, *name="empno"* declares the name of the property in "*EmpClass*". *Hibernate* will use the getter and setter methods to access the property. The column attribute tells *Hibernate* which column of the "EMP" table we use for this primary key.

<generator> tag specifies how to generate unique identifiers for instances of the persistent class.

Table 33 - Hibernate generator

generator	description
native	picks identity, sequence or hilo depending upon the capabilities of the underlying database.
assigned	lets the application to assign an identifier to the object before save() is called. default.
identity	supports identity columns in DB2, MySQL, MS SQL Server, Sybase and HypersonicSQL.
sequence	uses a sequence in DB2, PostgreSQL, Oracle, SAP DB, McKoi or a generator in Interbase.
hilo	uses a hi/lo (high and low) algorithm to efficiently generate identifiers of type long, short or int, given a table and column as a source of hi values.

☆ The name of mapping file has *"xxx.hbm.xml"* format.

4) Entity class (Persistent class)

Entity class is used for mapping data with Relational Database. For example the data retrieved from the database will be stored in this class and the data that will be saved to the database will be set in this class.

[entity.EmpClass.java]

```
package entity;

import java.io.Serializable;
import java.util.Date;

public class EmpClass implements Serializable {

private Integer empno;
private String ename;

public EmpClass() {}

    public Integer getEmpno() {
        return this.empno;
    }
    public void setEmpno(Integer empno) {
        this.empno = empno;
    }
    public String getEname() {
        return this.ename;
    }
    public void setEname(String ename) {
        this.ename = ename;
    }
}
```

This class has standard Java Bean conventions for property getter/setter methods but this is not mandatory. The no-argument constructor is a requirement for all persistent classes;

5) Data Access Object class

[BaseDao.java]

```
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class BaseDao {
    private static final SessionFactory sessionFactory;
    static {

        try {
        // Create the SessionFactory from hibernate.cfg.xml
            sessionFactory =
            new Configuration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            // exception handling
            System.out.println(
                "Initial SessionFactory creation failed." + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }
    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

SessionFactory is built by according to the database information in Hibernate configuration;

```
            sessionFactory =
            new Configuration().configure().buildSessionFactory();
```

[HibernateStore.java]

```
import org.hibernate.Session;
import entity.EmpClass;

public class HibernateStore extends BaseDao{
     /**
      * Hibernate test for store data
      */
     public static void main(String[] args) {
               // get connection from super class
               Session session =
                    getSessionFactory().getCurrentSession();
               // set transaction
               session.beginTransaction();

               // set data to store in the emp table
               EmpClass emp = new EmpClass();
               emp.setEmpno(999);
               emp.setEname("Hibernate");

               // save data to the table
               session.save(emp);

               // commit the modification
               session.getTransaction().commit();
               // close the session
               getSessionFactory().close();
     }
}
```

This is an example of Data Access Object that access to the EMP table to store one record. The main DAO class extends *BaseDAO* to get SessionFactory by which Session is obtained using *getCurrentSession()* method.

```
               Session session =
                    getSessionFactory().getCurrentSession();
```

The necessary information to save data to the database is set in the entity class including its primary key.

Once the entity is prepared, "*save()*" method of "*Session*" is called to store data. Internally the "*save*()" method is converted to SQL insert statement.

```
               session.save(emp);
```

After transaction is committed, SessionFactory is required to close.

```
                    getSessionFactory().close();
```

Table 34 - Hibernate Session method examples

Data Access	Methods
Retrieve one record	Emp emp = (Emp)session.get(Emp.class, new Integer(id));
Retrieve multiple records	List list=session.find("from Emp p where p.name=?", name, Hibernate.STRING);
Save new record	session.save(emp);

Update record	Emp emp = (Emp)session.load(Emp.class, new Integer(1));
	emp.setName("HibernateNew");
	session.update(emp);

Delete record	session.delete(emp);

7.4.4. Sample Program of Spring + Hibernate

In the previous sample program, we have created program of *Spring* using *Struts* Framework. In the *Action* class, the fixed person name obtained from the *Spring* configuration file (always "Pablo") is set. Now let's include data access logic to retrieve person name from "emp" table of MySQL database, so that the person name that corresponds to the "id" in the first page is obtained and shown in the result page.

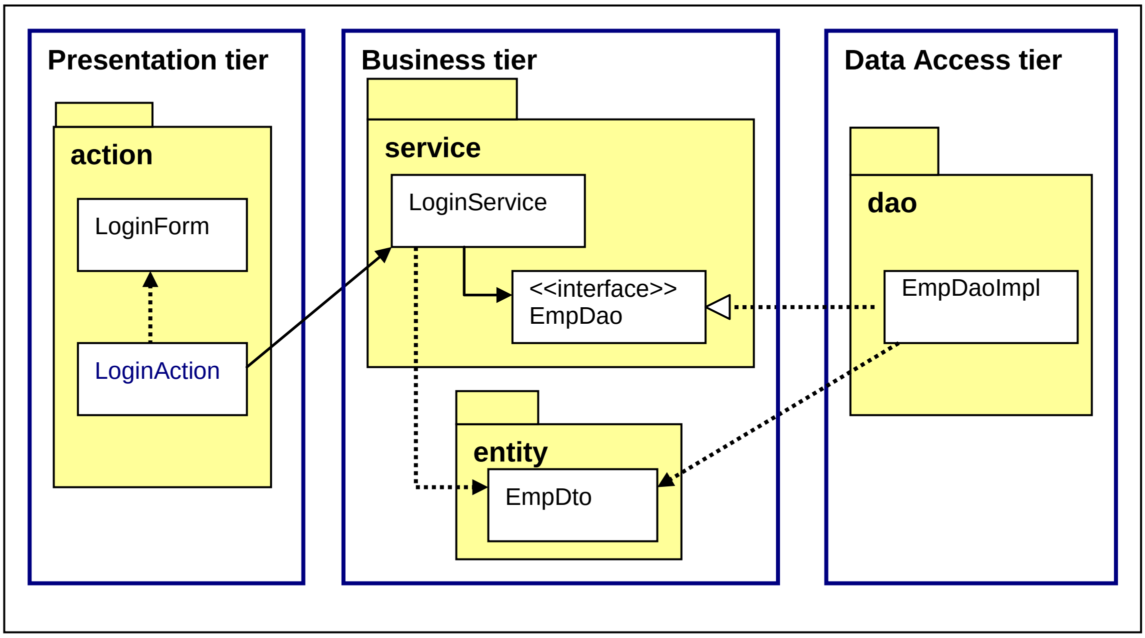


Figure 96 Class and package diagram of Sample Program Spring + Hibernate

When *Hibernate* is integrated into *Spring*, *Hibernate* configuration file is not required, but instead the resource information is absorbed in *Spring* configuration file.

1) Spring configuration file

[action-servlet.xml]

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd"
>
<!-- Bean Configuration for Spring + Struts sample program -->
<bean name="/LoginSimplePath" class="action.LoginSimpleAction">
    <property name="personName" value="Pablo" />
</bean>

<!-- Bean configuration for Spring + Hibernate sample -->
<bean name="/LoginPath" class="action.LoginAction">
    <property name="myLogService" ref="idLoginService" />
</bean>
<bean id="idLoginService" class="business.service.LoginService">
    <property name="myEmpDao" ref="idEmpDao" />
</bean>
<bean id="idEmpDao" class="dao.EmpDaoImpl">
    <property name="sessionFactory" ref="sessionFactory" />
</bean>

<!-- Database JDBC information -->
  <bean id="myDataSource"
        class="org.apache.commons.dbcp.BasicDataSource"
        destroy-method="close">
      <property name="driverClassName" value="${db.driver}"/>
      <property name="url" value="${db.url}"/>
      <property name="username" value="${db.username}"/>
      <property name="password" value="${db.password}"/>
  </bean>

<!-- definition of location of datasource external file -->
  <bean id="dbConf"
         class="org.springframework.core.io.ClassPathResource">
      <constructor-arg>
        <value>database.properties</value>
      </constructor-arg>
  </bean>
  <bean id="databaseConfPostProcessor"
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
     <property name="location">
       <ref bean="dbConf"/>
     </property>
   </bean>

<!-- sessionFactory definition -->
   <bean id="sessionFactory"

class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
     <property name="dataSource" ref="myDataSource"/>
     <property name="mappingResources">
       <list>
        <value>dao/emp.hbm.xml</value>
       </list>
     </property>
     <property name="hibernateProperties">
       <props>
         <prop key="hibernate.dialect">
                org.hibernate.dialect.MySQLInnoDBDialect</prop>
         <prop key="hibernate.show_sql">true</prop>
       </props>
     </property>
  </bean>
</beans>
```

The following part shows data resource information for *Hibernate*. The information is defined in the external file named "database.properties".

```
  <bean id="myDataSource"
        class="org.apache.commons.dbcp.BasicDataSource"
        destroy-method="close">
      <property name="driverClassName" value="${db.driver}"/>
      <property name="url" value="${db.url}"/>
      <property name="username" value="${db.username}"/>
      <property name="password" value="${db.password}"/>
  </bean>
  <bean id="dbConf"
         class="org.springframework.core.io.ClassPathResource">
      <constructor-arg>
        <value>database.properties</value>
      </constructor-arg>
  </bean>
  <bean id="databaseConfPostProcessor"
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
     <property name="location">
       <ref bean="dbConf"/>
     </property>
   </bean>
```

In the *"database.properties"* information for JDBC connection is described.

[database.properties]

```
db.driver=com.mysql.jdbc.Driver
db.url=jdbc:mysql://localhost/ictti
db.username=root
db.password=root
db.dialect=org.hibernate.dialect.MySQLInnoDBDialect
```

This external file is obtained by *"org.springframework.core.io.ClassPathResource"* class which looks for resources in ClassPath directory. Therefore **"database.properties" will be placed where the class path is defined.**

The *"sessionFactory"* bean tag represents which data resource to use, and which is the mapping file for data access. In this case, *"myDataSource"* is referred to get JDBC connection. Mapping file name is *"dao/emp.hbm.xml"*. You will notice that multiple mapping files can be defined as <value> tag of <list>.

```
<bean id="sessionFactory"
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
     <property name="dataSource" ref="myDataSource"/>
     <property name="mappingResources">
       <list>
        <value>dao/emp.hbm.xml</value>
       </list>
     </property>
     <property name="hibernateProperties">
       <props>
         <prop key="hibernate.dialect">
                org.hibernate.dialect.MySQLInnoDBDialect </prop>
        <prop key="hibernate.show_sql">true</prop>
       </props>
     </property>
  </bean>
```

User bean *"/LoginPath"* is the name defined as "Action Path" in *Spring*. The corresponding action class name is specified in <class> tag, same as configured in the previous chapter. The "*action.LoginAction*" class refers to "*business.service.LoginService*" class by variable "*idLoginService*". "*business.service.LoginService*" also contains "*dao.EmpDaoImpl*" class by variable name "*myUserDao*". Since these two beans are dependent with other bean, Spring can inject dependencies.

```
<!-- Bean configuration for Spring + Hibernate sample -->
<bean name="/LoginPath" class="action.LoginAction">
   <property name="myLogService" ref="idLoginService" />
</bean>
<bean id="idLoginService" class="business.service.LoginService">
   <property name="myEmpDao" ref="idEmpDao" />
</bean>
<bean id="idEmpDao" class="dao.EmpDaoImpl">
   <property name="sessionFactory" ref="sessionFactory" />
</bean>
```

The final *"dao.EmpDaoImpl"* class is a DAO class who needs to get access to Relational Database according to the configuration. For that purpose "*sessionFactory"* bean should be declared as its property.

```
<bean id="idUserDao" class="business.UserDaoImpl">
   <property name="sessionFactory" ref="sessionFactory" />
</bean>
```

2) Spring Action class

[action/LoginAction.java]

```
package action;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import business.service.LoginService;

public class LoginAction extends Action {
    /* Service to get access emp table via hibernate.
     This class will be generated by Spring DI
     */
    private LoginService myLogService;

    public ActionForward execute(ActionMapping mapping,
                        ActionForm form,
                        HttpServletRequestrequest,
                        HttpServletResponse response)
                              throws Exception {
        // get LoginForm bean
        LoginForm loginForm = (LoginForm)form;

        // get UserID from the form
        String strID = loginForm.getId();
        System.out.println(strID);

        // Access to emp table via hibernate and
        // get user name from id
        String strName= myLogService.getUserName(strID);
        System.out.println(strName);

        // set the user name to the form,
        // so that JSP can show the name
        loginForm.setName(strName);

        return mapping.findForward("forwardHello");
    }

    public LoginService getMyLogService() {
        return myLogService;
    }
    public void setMyLogService(LoginService myLogService) {
        this.myLogService = myLogService;
    }
}
```

"*LoginAction*" class contains the Service bean class "*LoginService*" which is declared in the *Spring* configuration file, so that it is instantiated by DI container and the reference is set to its variable name "myLogService".

```
    private LoginService myLogService;
```

Now the Action class calls service logic where business logics are controlled.

```
        String strName= myLogService.getUserName(strID);
```

3) Service logic class

"*LoginService*" class is a service logic class that serves as an entry point to manage various business logics. This class belongs to business tier and is called from *Action* class (Presentation tier) to call DAO interface class.

[business/service/LoginService.java]

```
package business.service;

import business.entity.EmpDto;

public class LoginService {
    private EmpDao myEmpDao;

    public EmpDao getMyEmpDao() {
        return myEmpDao;
    }

    public void setMyEmpDao(EmpDao userDao) {
        this.myEmpDao = userDao;
    }

    public String getUserName(String id) {
        EmpDto user = myEmpDao.get(id);
        return user.getEname();
    }
}
```

"*LoginService*" contains "*EmpDao*" interface class which **implementation** is declared in the *Spring* configuration file, so that it is instantiated by DI container and the reference of implementation object is set to its variable name "*myEmpDao*".

```
    private EmpDao myEmpDao;
```

Now the service class calls business logic where database is really accessed.

```
        EmpDto user = myEmpDao.get(id);
```

☆ Let's think what are rolls of service logic class. Why is it better not to describe the business logic directly inside the Action class?

4) DAO interface

"*EmpDao*" is an interface to data access to get data from database. The retrieved data is set in entity class "*EmpDto*".

[business/service/EmpDao.java]

```
package business.service;

import business.entity.EmpDto;

public interface EmpDao {
    public EmpDto get(Integer id);
    public EmpDto get(String id);
}
```

5) DAO Implemented class

"*EmpDaoImpl*" is an implemented class of "*EmpDao*" to get access to emp table by Hibernate and returns retrieved data in entity class.

[dao/EmpDaoImpl.java]

```
package dao;

import
org.springframework.orm.hibernate3.support.HibernateDaoSupport;

import business.entity.EmpDto;
import business.service.EmpDao;
/**
 * Implemented class of EmpDao to get record from emp table.
 * Use Hibernate as Data Access.
 */
public class EmpDaoImpl extends HibernateDaoSupport implements EmpDao
{
    public EmpDto get(Integer id) {
        System.out.println("EmpDto id=" + id);
        // Get access to emp table and returns retrieved data
        return
    (EmpDto) super.getHibernateTemplate().get(EmpDto.class, id);
    }
    public EmpDto get(String id) {
        return get(Integer.valueOf(id));
    }
}
```

"*EmpDaoImpl*" class extends "*HibernateDaoSupport*" class provided by *Spring* package.

"*HibernateDaoSupport*" has *setSessionFactory()* method to get "*sessionFactory*" bean created by *Spring* according to the *"action-servlet.xml"*.

```
public final void setSessionFactory(SessionFactory sessionFactory)
```

That's why *"EmpDao"* bean has property for "*sessionFactory*" in *"action-servlet.xml"*.

```
<bean id="idEmpDao" class="dao.EmpDaoImpl">
    <property name="sessionFactory" ref="sessionFactory" />
</bean>
```

As a result, there is **no need** to get/close "SessionFactory" or "session" of *Hibernate* which is done automatically by *Spring* DI container.

The following code shows how to get "*emp"* table using *Hibernate*.

```
        return
    (EmpDto) super.getHibernateTemplate().get(EmpDto.class, id);
```

"*HibernateTemplate*" is a *Spring Helper* class that simplifies *Hibernate* data access code. Automatically converts *HibernateExceptions* into *DataAccessExceptions*, following the *org.springframework.dao* exception hierarchy providing many methods that mirror the methods exposed on the *Hibernate* Session interface.

In combination of "*HibernateDaoSupport*" and "*HibernateTemplate*", this allows for very simple DAO implementations for typical requirements

6) Mapping entity class

The following class is an entity class that is used for mapping with "*emp*" table of Relational Database.

[business/entity/EmpDto.java]

```
package business.entity;

public class EmpDto {
    private Integer empno;
    private String ename;

    public void setEmpno(Integer id) {
        this.empno = id;
    }
    public Integer getEmpno() {
        return (this.empno);
    }
    public void setEname(String name) {
        this.ename = name;
    }
    public String getEname() {
        return (this.ename);
    }
}
```

7) Mapping configuration file

The mapping configuration file for *Hibernate* specifies the Java class "*business.entity.EmpDto*" and "*emp*" table.

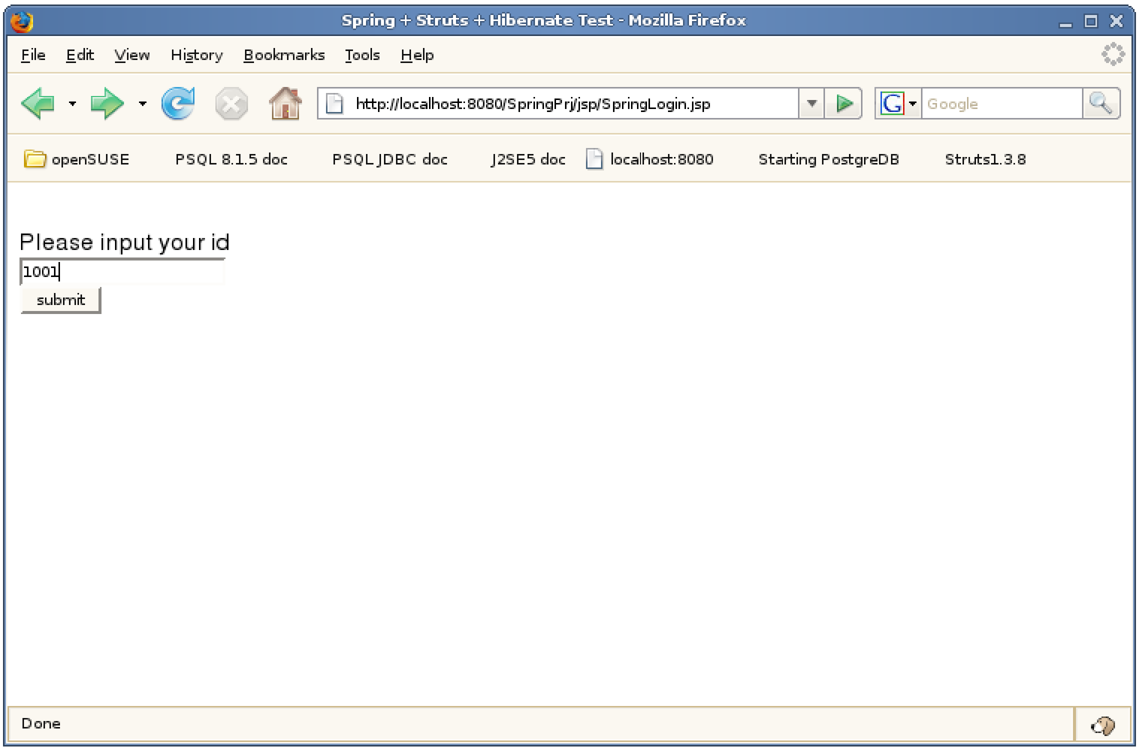
[dao/emp.hbm.xml]

```
<?xml version="1.0"?>
 <!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping
DTD//EN"
 "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
 <hibernate-mapping>
   <class name="business.entity.EmpDto" table="emp" lazy="true">
     <id name="empno" type="integer" unsaved-value="null" >
       <column name="empno" sql-type="INTEGER" not-null="true" />
        <generator class="assigned" />
     </id>
     <property name="ename" column="ename" />
   </class>
</hibernate-mapping>
```

Exercise 7: Spring + Hibernate

1. Create "*SpringPrj*" project in *Eclipse* with *Tomcat*. Locate necessary files to create a program that shows the name of corresponding "id" described in the textbook using *Hibernate*. Execute it from the browser to see if the programs and configuration files are set correctly.

[First page]

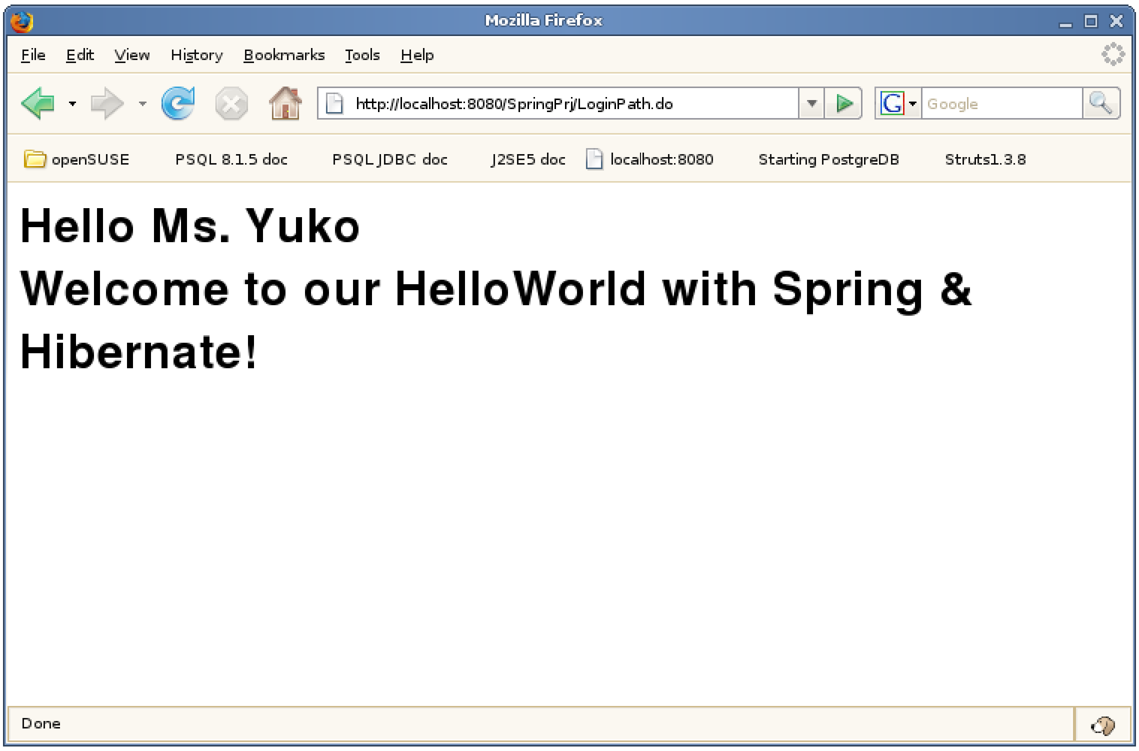

Spring + Struts + Hibernate Test - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/SpringPrj/jsp/SpringLogin.jsp
Google
openSUSE
PSQL 8.1.5 doc
PSQL JDBC doc
J2SE5 doc
localhost:8080
Starting PostgreDB
Struts1.3.8
submit
Done

Please input your id
1001

Figure 97 Exercise Spring + Hibernate first page

The employee name of empno "1001" is shown as a result.

[Result page]



Hello Ms. Yuko

Welcome to our HelloWorld with Spring & Hibernate!

Figure 98 Exercise Spring + Hibernate result page

Tables and Figures

Figures

Tables

Indexes

Keywords