Java Programming : applets, servlets and JSP.

SR03

Dritan Nace

A summary of Java

Java is a language developed by Sun, which is designed to be object oriented and Simple, robust and secure, independent of hardware architectures and Multitasking.

Object oriented and simple : Simpler than C++, transparent memory managment…

Robust et secure : Data typing is extremely strict. For applets, it is in principle impossible to access the resources of the host machine.

Independant of hardware architectures : The compiler generates a universal code : the « byte-code ». An interpreter which is specific to the host machine, « virtual machine », executes the programs.

Multitasking : Java seemingly allows execution of several processes. In reality, a time slot is given to each process on the processor (Multithreaded).

J2SE versus J2EE

J2SE (standard edition) contains the basic usable components from both the client and server side,

- GUI, AWT/Swing for applications (client) or applets.
  - Currently J2SE v1.6 (ou V6)

J2EE (enterprise edition), is in a certain sense an extension of SE, designed for server side programming

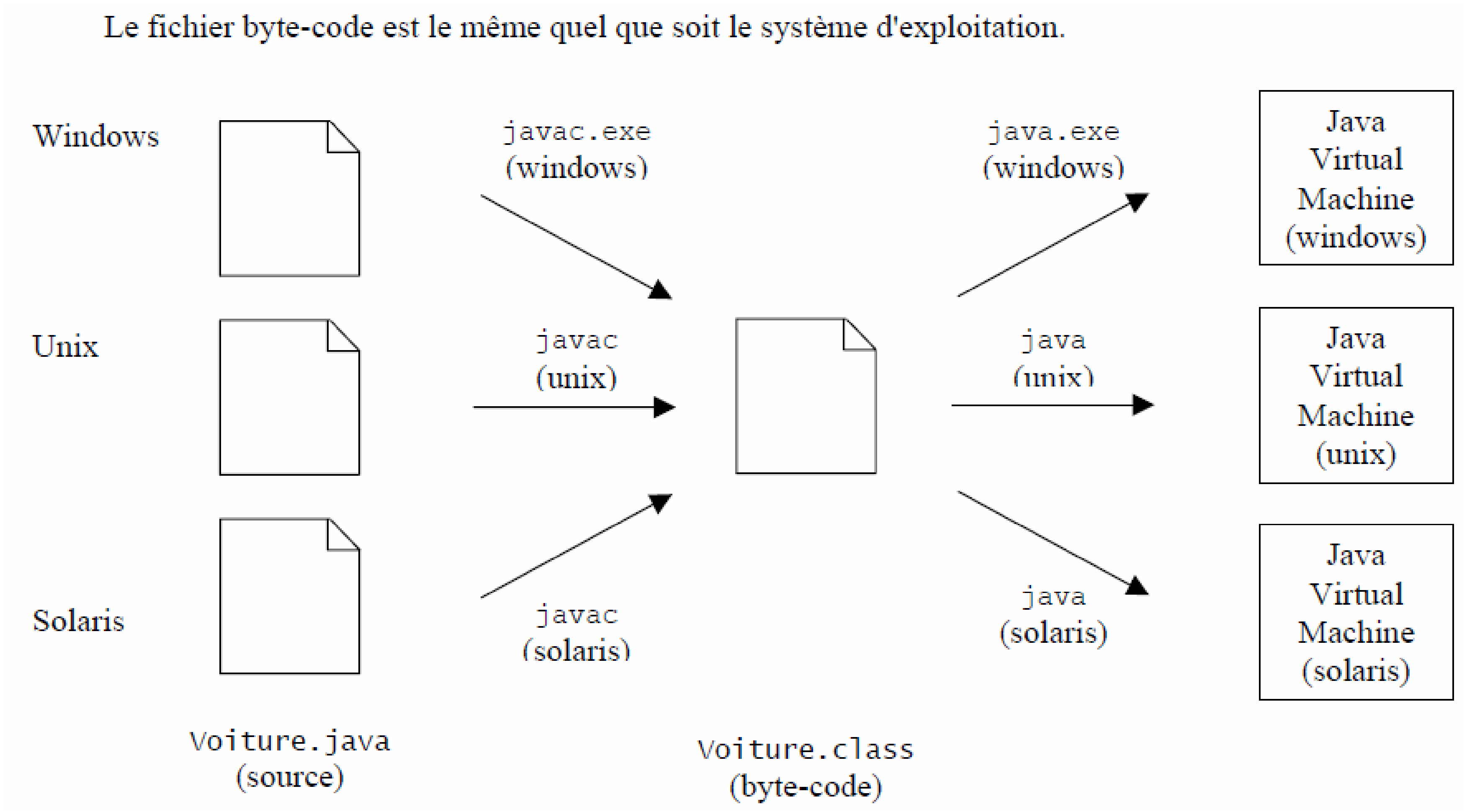
- Servlets, JSP, EJB, etc.
  - Currently J2EE v1.4

Java and object oriented programming

- Classes, and objects
  - The objects include data and processing for the data. Communication is done via messages (methods).
  - A class corresponds to an abstract model for object construction. A class is made up of:
    - attributes (static part)
    - Methods (dynamic part), which define the behaviour.
  - Inheritance : the « is a » relationship : a *car* is a *vehicule,*
  - Polymorphism : the same message can be recognised by several objects and entail different behaviour.

Portability: JVM

The compiler compiles the java source in byte code : *javac car.java => car.class*
Then, *java* is the name of the program which will interpret the generated byte code.


Le fichier byte-code est le même quel que soit le système d'exploitation.
Windows
Unix
Solaris
javac.exe (windows)
javac (unix)
javac (solaris)
java.exe (windows)
java (unix)
java (solaris)
Java Virtual Machine (windows)
Java Virtual Machine (unix)
Java Virtual Machine (solaris)
Voiture.java (source)
Voiture.class (byte-code)

Event managment via design patterns

- The management of events (mouse click, pressing a button, etc ...) is done using the Observer Pattern principle (or listener).
- Certain objects (**source**) can generate events.
- Any object which would like to be warned when an event of a source happens must implement an interface and methods for the processing <Type event>Listener and subscribe to this source using the source object's methods.
  - The subscription/retraction is achieved using add/remove<Type event>Listener()

- It then becomes a **target** (the source). When a source generates an event, concretely it will simply browse the list of subscribers and call one of the methods mentioned above.

Java and packages

- Packages
  - java.lang
  - java.util
  - java.io
  - java.math
  - java.net
  - java.security
  - java.sql
  - java.awt
  - javax.swing
  - java.applet

classpath: specifies to the virtual machine the location from which the resources (bytecode and others) should be taken.

- The directory **/project/classes**
- The archive **/project/lib/archive.jar**

import : command used for importing remote packages or classes…

Applets

- An applet is a special Java application that will run in an HTML document viewed by a browser.
  - Goal : transmit executable code to the client.
- The **Applet** class is a sub-class of the **Panel** class (from the java.awt package). (An applet is a graphic component container.)
- An applet will be able to respond to mouse and keyboard events and use graphic components like buttons, checkboxes etc..
- An applet runs in a specific JVM, the JVM of a web browser.

Applet specifics

An applet has special caracteristics compared with an independant application :

- **Limited access to the network :** An applet can only communicate with the machine from which it comes.
- **Limited access to the file system :** An applet which is loaded into a browser from the network has no access to the local file system.
- **Limited access to native methods.**

The java.applet package

The java.applet package:

allows programmers to integrate java applications (applets) into Web documents.

This package contains:

The **Applet** class

The following interfaces : **AppletContext, AppletStub** et **AudioClip**.

The **Applet** class is derived from the **java.awt.Panel** class.

```
Object
      |
      +----- Component { paint(); resize(); … }
                  |
                  +----- Container
                              |
                              +----- Panel
                                        |
                                        +--- Applet
```

Creating an applet

In order to create an applet, a class which inherits from the **Applet** class must be defined, and certain methods must be redefined.

It has four special methods, which are :

```
public void init()
public void start()
public void stop()
public void destroy()
```

It also inherits from the **public void paint(Graphics g)** methods declared in the **Component** class

The init() method

public void init()

The init() method, called by the browser, enables the initialisation of the applet, (the constructor will be called before this method).

This method is executed one time only at the startup of the applet.

It allows:

- parameter fetching
- instantiation of objects which are parameter-dependant
- the initialisation of parameter-dependat values
- police loading or loading images which are paramter dependant

Methods

public void start()

The start() method is executed :

- just after the init() method
- every time the browser comes back to the HTML page which contains the applet.

public void stop()

The stop() method is executed each time the user leaves the web page which contains the applet or whenever the page is no longer visible.

public void destroy()

The destroy() method is called at the end of the applet, so when the user leaves the window or the web page browser.

public void paint(Graphics g)

The paint() method is obtained by inheritance, and is declared in the **Component** class

The paint() method is called each time the window manager has to draw the content of the applet.

Other methods

Other methods allow obtaining of information about the applet which is running.

• **public String getParameter(String name)**
Recovers the parameters passed to the HTML document.

• **public AppletContext getAppletContext()**
Recovers the display context of the current applet.
It is generally a browser (Netscape etc.) or the appletviewer.

• **public URL getDocumentBase()**
Returns the **URL** of the HTML document which includes the applet.

• **public Image getImage(URL url)**
Loads an image which can be used afterwards. url is an absolute **URL**.

Parameter passing

Only one default constructor can be defined in an applet.

The browser would be unable to know which parameter to pass to another contructor.

On the other hand, it is possible to pass parameters to an applet designed for :

- instantiation of objects
- initialisation of values
- police or image loading

The <APPLET> tag

The <applet> tag enables the integration of a space for the executtion of a Java application in an HTML document.

```
<APPLET
CODE="AClassName"
HEIGHT= anInteger
WIDTH= anotherInteger
>
<PARAM
NAME="firstParameter"
VALUE="valueOfFirstParameter"
>
<PARAM
NAME= "secondParameter"
VALUE="valueOfSecondParameter"
>
…
</APPLET>
```

The HTML file has the ability to pass parameters to an applet (the PARAM tag).
The recovery of these parameters is done in the source of the applet using the following method
String getParameter(String name) of the**Applet** class.

Structure of an applet

```
import java.applet.*;
import java.awt.*;
public class <AppletName> extends Applet {
public void init() {
  <Initialisations>
  <Start of the processes>
  }
public void start() {
  <Start the applet, the Web page is visited or becomes visible again>
  }
public void paint(Graphics g) {
  <Draw the current content of the applet>
  }
public void stop() {
  <Stop the applet, the web page is no longer visible or the user leaves the navigator>
  }
public void destroy() {
  <Release the resources, the applet will leave the memory>
  }
}
```

Life cycle of an applet : init() (start() paint() stop()) destroy()

Example of a simple applet

```
import java.applet.*;
import java.awt.*;
       public class HelloSR03 extends Applet
       {
       String text;
       public void init()
         {
         texte = getParameter("text");
         }
       public void paint(Graphics g)
          {
          g.drawString(texte, 30, 30);
          }
       }
```

A minimal HTML page which holds an applet

```
<HTML>
<body>
<APPLET code="HelloSR03.class" width="500"
height="200">
<param name="text" value= "Hello SR03 !">
</applet>
</BODY>
</HTML>
```

In order to execute, use :

- either the browser by opening the HTML file
- or the *appletviewer*

Loading a JAR (Java Archive) file

- This file format de fichier allows the fusion of several files which are used by an applet (".class", sounds and images) into one file in the JAR format which will be loaded with a singe request by the HTTP protocol.
  - Creation : jar cfv file.jar file_1 … file_n
  - Appel :

```
<applet
code = "file.class"
archive="file.jar"
width= "200" height= "200" >
</applet>
```

Servlets and JSP

SERVLETS

- Servlets are alternative to the Java technology for programming CGI.
- What do they do?
  - Read data sent by the user. They are generally from a form on a Web page, an applet or any HTTP client program.
  - Acquire additional information about the request: Browser identification, type of method used, values of cookies, etc..
  - Generate results. This usually involves access to a business layer service.
  - Format the output into a document. In most cases the results are integrated into an HTML page..
  - Defining the parameters of the appropriate HTTP response. The type of document returned must be sent to the browser, cookies have to be set, a session must be established.
  - Return the document to the client. The format can be text (HTML), a binary format (image, for example), a zip etc.

SERVLETS : how do they work?

An application server can load and run servlets in a JVM. This is an extension of the web server. The application server contains, among other things, a servlet that manages the servlets it contains.

In order to exectue a servlet, a URL which corresponds to the server must be put into the browser.

- The server receives the http request which needs a servlet from the browser
- If this is the first request of the servlet, the server instantiates. Servlets are stored (as a .Class file) in a particular directory on the server and remains in memory until the server shuts down.
- The server creates an object that represents the HTTP request and an object that contains the response, and sends them to the servlet.
- The servlet creates the response dynamically in the form of an html page transmitted via a stream in the object containing the response. The creation of this response uses the client's request, but also a set of resources on the server such as files or databases.
- The server takes the response object and sends the html page to the client.

Environment

- The standard Edition J2SE v1.6 (or V6)
- The Entreprise Edition J2EE v1.4 which contains, amongst other things, the classes for servlet, with the 2.4 specification
- A servlet server (Tomcat) for Apache (**http://www.apache.org**) v5.x

Structure of theTomcat directory:

/bin	
/classes	
/common	
/classes	
/lib	
/conf	**Configuration files**
/lib	
/logs	
/server	
/classes	
/lib	
/webapps	**Contains a directory for every web application**
/examples	**Tomcat example application**
/manager	
/ROOT	**Serves as a model. Do not modify**
/tomcat-docs	
/webdav	
/work	
/localhost	**Contains a directory for each web application in which Tomcat generates the servlets which come from JSPs (very useful for debugging)**

Structure of the Tomcat directory, continued

```
/webapps
        /images                  for .gif and other .jpeg
        /WEB-INF
                /classes         for the .class
                /lib             for the .jar
                web.xml          the deployment file
        /special directories
        HTML and JSP directories
```

Deployment descriptor

A deployment descriptor is an XML file containing information about the WebApp which is necessary to the web server. The standard name is **web.xml**. In Tomcat, it should be placed in Webapps \ WEB-INF and is read at the startup of Tomcat.

In it, the following elements can be defined:

- Context parameters
- Servlet identifiers (**<servlet-name>)**.
- Parameter definitions (**<param-name>**) associated with servlet identifiers.
- Mappings between servelt identifiers and url (<**servlet-mapping**> and <**url-pattern**>).

An example of a Deployment descriptor

```
<?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_4.dtd">
<web-app>
<!--======================================== -->
<!– Definition of WEB parameters-->
<!-- Valid for all servlets -->
<context-param>
  <param-name>UV</param-name>
  <param-value>SR03</param-value>
</context-param>
<context-param>
  <param-name>Course</param-name>
  <param-value>Servlet and Jsp</param-value>
</context-param>
<!--===================================== ->
```

```
<!--======================================== -->
<!-- Definition pour Exemple 1 -->
<servlet>
  <servlet-name>Hello</servlet-name>
  <description>This is the first example</description>
  <servlet-class>Hello</servlet-class>
<servlet-mapping>
   <servlet-name>Hello</servlet-name>
   <url-pattern>/servlet/Hello</url-pattern>
</servlet-mapping>
<init-param>
<param-name … </param-name>
<param-value> …</param-value>
</init-param>
</servlet>
<!– Definition for example 2 -->
<servlet>
…
```

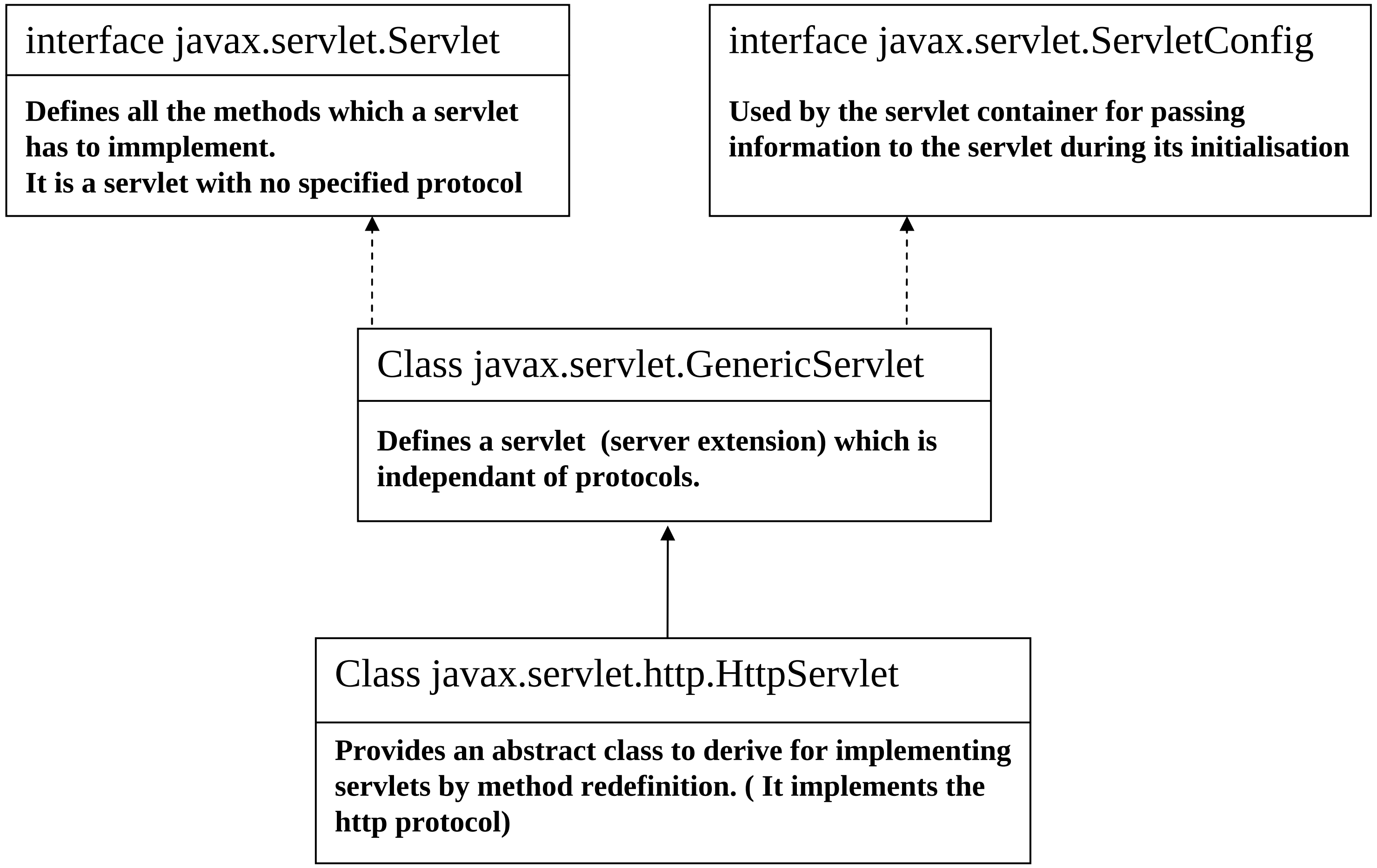
Calling a servlet from a browser

- With its <u>url mapping</u>, class name or identifier
  - Preferably, use the url-pattern : http://serveur:port/**servlet**/**Hello**

    (By default Tomcat receives on the 8080 port)

- Calling a servlet from an html page
  - On a <a > tag…
  - On a <form> tag…

Servlets : classes and interfaces


interface javax.servlet.Servlet
Defines all the methods which a servlet has to immplement.
It is a servlet with no specified protocol
interface javax.servlet.ServletConfig
Used by the servlet container for passing information to the servlet during its initialisation
Class javax.servlet.GenericServlet
Defines a servlet (server extension) which is independant of protocols.
Class javax.servlet.http.HttpServlet
Provides an abstract class to derive for implementing servlets by method redefinition. (It implements the http protocol)

javax.servlet package

javax.servlet package **Class** ***interface***	**Contains the classes and interfaces which define the servlets generically. In other words, with no imposed protocol.**
GenericServlet	Defines a generic servlet, which is protocol-independant.
RequestDispatcher	Defines an object which receives requests from a client and return them to any resource (Servlet, JSP or HTML) on the server. This is a redirection of the server.
Servlet	Defines the methods which all the servlets must implement. It defines thus the servlet's lifecycle.
ServletConfig	An object used by a servlet container for passing information (parameterisation) to a servlet during its initialisation.
ServletContext	Defines a set of methods used by a servlet to interact with its container. A ServletContext represents an area common to all servlets and JSPs of a Web application.
ServletException	Defines the general exception which can trigger a servlet when problems arise.
ServletRequest	Defines an object for providing information concerning the client's requests to the servlet
ServletResponse	Defines an object which assists the servlet in its response to the client.

javax.servlet.http package

Package javax.servlet.http **Class** ***interface***	**Contains the classes and interfaces which define the servlets which support the HTTP protocol.**
Cookie	Generates cookies.
HttpServlet	Provides an abstract class to derive, for creating an http servlet.
HttpServletRequest	Derived from *ServletRequest* to provide the information for the HTTP servlet.
HttpServletResponse	Derivde fro *ServletResponse* to provide the specific HTTP functionalities when sending a response.
HttpSession	Provides the means to identify a user, a request from one page to the next and to store information about this user.

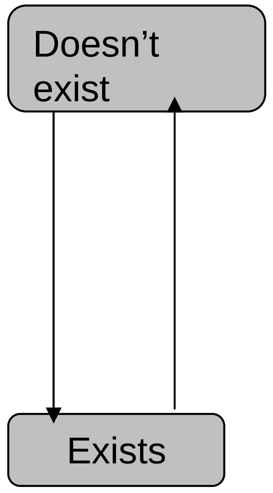
…

Lifecyle of a servlet

1 Constructeur
2 init (ServletConfig)

service(req,resp)
doPost(req,resp)
doGet(req,resp)
doPut(req,resp)
doDelete(req,resp)

Processing of the http message



Les methods (I)

The init method

- The **Init (ServletConfig)** method is called once after the creation of the servlet. (somewhat equivalent to the init method of the Applet class). The parameters set in the servlet web.xml in ServletConfig. can be recovered here

The service method

- Whenever the server receives a request for a servlet it creates a thread and it calls the service method. This method checks the type of the HTTP request (GET, POST, PUT, etc. ...) and calls the corresponding method (doGet, doPost, doPut, etc..)

Methods (II)

The doGet, doPost et doXxx methods...

- These are the methods which will need to be redefined, with respect to the request to process. They have the following form :

```
Public void doXxx (Httpservletrequest request, HttpServletResponse
   response) {
//Fetch the client requests on request…
//give the headers if necessary…
//fetch a response stream (reponse.getWriter())
//write the response (creation of the html document html for example)
…
}
```

An example

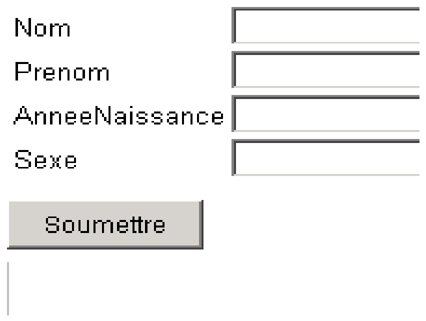
```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

public class Hello extends HttpServlet {
  private static final String CONTENT_TYPE = "text/html";
//==============================================================
/**Initialise global variables*/
 public void init() throws ServletException {
 }
//==============================================================
 /**Process the HTTP request Get*/
 public void doGet(HttpServletRequest request,HttpServletResponse response)
             throws ServletException, IOException {

 response.setContentType(CONTENT_TYPE);
 PrintWriter out = response.getWriter();
 out.println("<p>The servlet <b>Hello</b> received a <b>"+
                                +"<font color=\"red\">GET</font></b>. </p>");
 out.println("<h1>Hello. Hello.class. says hello</h1>");
}
//==============================================================
/**Process the HTTP request Post*/
public void doPost(HttpServletRequest request,HttpServletResponse response)
            throws ServletException, IOException {

 response.setContentType(CONTENT_TYPE);
 PrintWriter out = response.getWriter();
 out.println("<p>The servlet <b>Hello</b> received a<b>"+
                                "<font color=\"red\">POST</font></b>.</p>");
 out.println("<h1>Hello. Hello.class. says hello</h1>");
}
//==============================================================
 public void destroy() {  }
}
```

Fetching headers of the http resquest



Processing by doPost(HttpServletRequest req , HttpServletResponse rep)

```
{
String lastName;
String firstName;
int dateofBirth;
char gender;
...
   lastname=  req.getParameter ( "lastName");
   firstname =  req.getParameter ( "firstName" );
   dateofBirth= Integer.parseInt(req.getParameter("dateofBirth");
   gender = req.getParameter("gender").charAt(0);
...
}
```

Cookie managment

Insertion of cookies into response headers.

Interface HttpServletResponse	
addCookie (cookie)	Adds a cookie to the response.

Recovery of client cookies.

HttpServletRequest Interface	
Cookie [ ] getCookies ( )	Fetches a table containing all the cookies sent by the client

Session management

- Servlets offer a solution using the HttpSession API. This interface is located above the cookies and URL rewriting.
- Every input request is associated to an active session created by the client (the browser).
- The sesssion is destroyed if there are no more requests within 30 minutes (default value in tomcat).

I*nterface HttpServletRequest*	***Interface HttpSession***
HttpSession **getSession (boolean create)** ***HttpSession*** **getSession ()**	**Object getAttribute (String name)** **Enumeration getAttributeNames ( )**

Redirection on the server

To redirect to a login page or pass on the construction of the response, the servlet uses *forward*.

The servlet uses include when it needs a servlet to insert its answer inside its own.

Essential differences:

- Forward
  - Separate processing of the client's request (made on the caller servlet) from the generation of the response (done on the called servlet).
    - getServletContext().**getRequestDispatcher**(urlCible).**forward**(req,resp); (urlCible, eg. a JSP page, provides the response)
  - Share out the processing of the request on several servlets.
- Include
  - Share out the generation of the response on several several.

Servlets and databases

- One of the principles of a good architecture is that an application should be designed in three general layers :

  Presentation – Business – Persistence

- Servlets are part of the presentation layer: and therefore have, in principle, no access to JDBC in Servlets.

- JDBC accesses are processed in the persistence layer.

JSP

- Servlet are very flexible and have many advantages : object, portability, performance, extensibility, free, etc.

- The fact remains that when creating a direct response html it becomes extremely tedious. The Java developer does not necessarily want to master HTML, JavaScript and related tools.

- Similarly, le designer of a HTML page is not necessarily an object specialist.

The MVC pattern(I)

The **solution** proposed by Sun is to divide the web applications web into 3 parts :

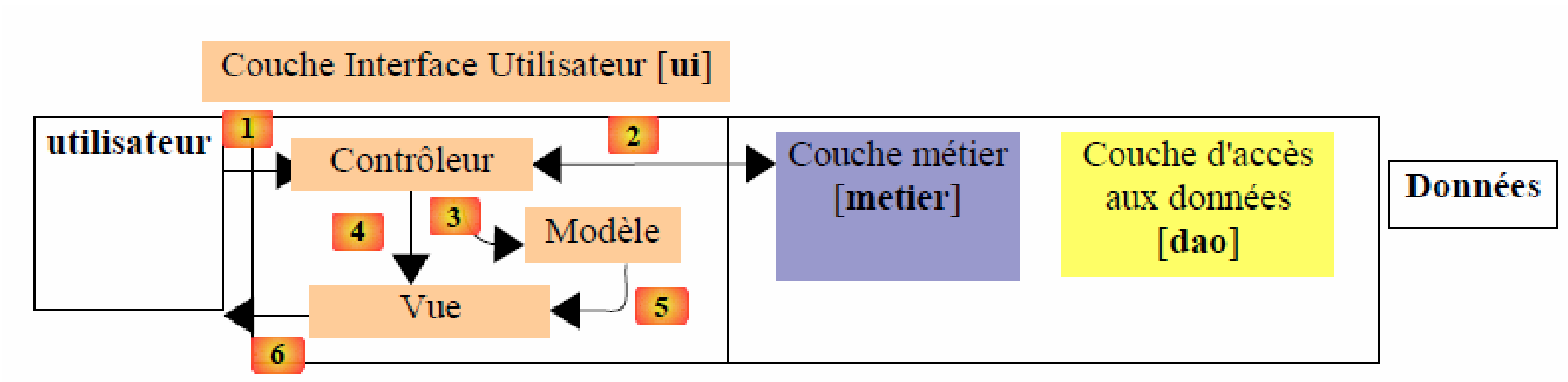
- The processing (the controller) which contains the control part only , chaining, etc.. This is a Java servlet which does not contain any HTML code generation, and is difficult to create "by hand". Its creation is handled by a Java developer.

- Presentation (a JSP Java Server Pages, which is an HTML extension with some JSP tags JSP added). It should not contain the business code (in the form of a "scriptlet"). However, it can call"scriptlets" for formatting. Its creation is the responsibility of a web developer who traditionally works with a tool such DreamWeaver.

- Access to data and to the business is done in Beans and/or EJB. Their creation is the responsibility of a Java developper.

The MVC pattern (II)

Presentation – Business – Persistence


Couche Interface Utilisateur [ui]
utilisateur
1
Contrôleur
2
4
3
Modèle
Vue
5
6
Couche métier [metier]
Couche d'accès aux données [dao]
Données

The MVC pattern (III)

The processing of a request from a client follows these steps :

1. the client makes a request of the controller. It sees all client requests pass. It is the entry to the application. This is the **C** of MVC.
2. the C controller processes this request. To do this, it may need help from the business layer. Once the client request is processed, it can invoke various responses. A classic example is :
   - an error page if the request could not be processed correctly
   - a confirmation page otherwise
3. the controller selects the response (= view) to send to the client. Choosing the answer for the client requires several steps :
   - choose the object that will generate the response. This is called the V view. This choice generally depends on the result of the execution of the action requested by the user.
   - provide the data needed to generate this response. In fact, it usually contains information calculated by the controller. This information form what is called the **M** of the View.
4. The **C** controller asks the chosen view to display itself. This mostly involves performing a particular method of the **V** view which generates the response for the client.
5. The generator of the **V** view uses the **M** model prepared by the controller C to initialise the dynamic parts of the response which he should send to the client.
6. The response is sent to the client. The exact form of the latter depends on the generator of view. This may be an HTML stream, PDF, Excel, ...

Lifecycle of a JSP

The JSP page is stored at the same location as the html pages.

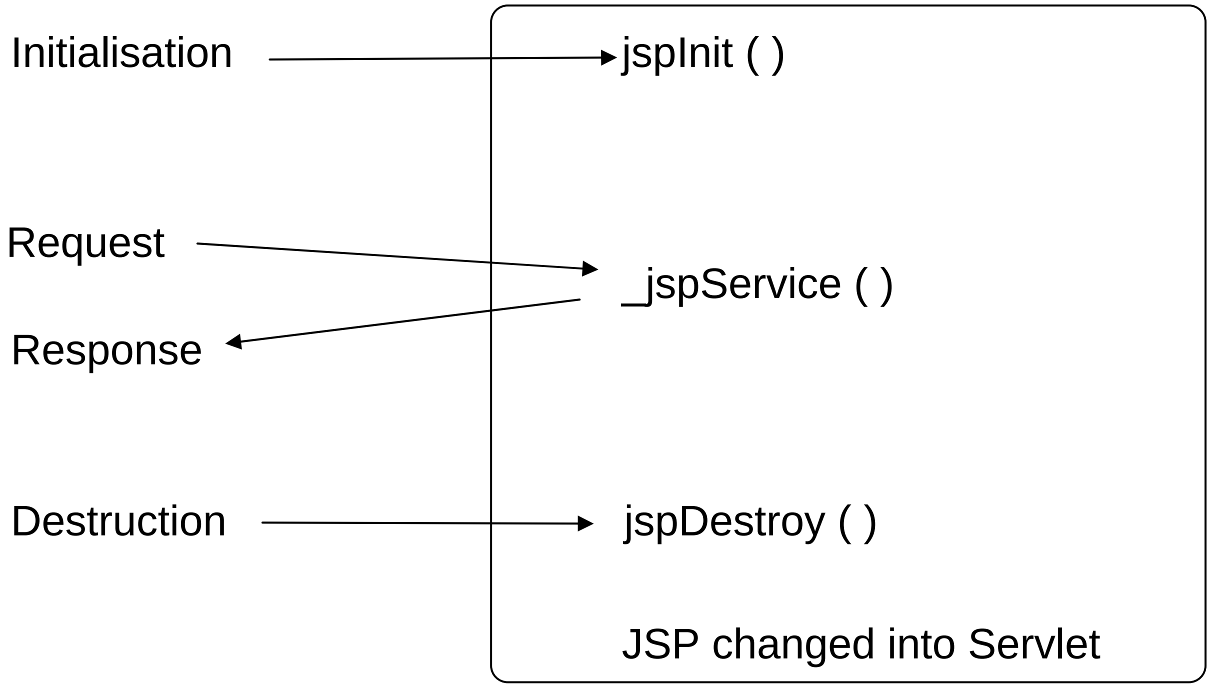
- The principle components of a JSP framework are :
  - A source generator which takes the JSP page as input and transforms it into a Servlet.
  - A java compiler to compile the generated servelet.
  - A set of execution support classes.
  - Tools for linking between different elements, such as tag libraries.

The generated servlet

In the end, a JSP is converted into a Servlet. All the concepts covered in previous chapters therefore also apply to JSP

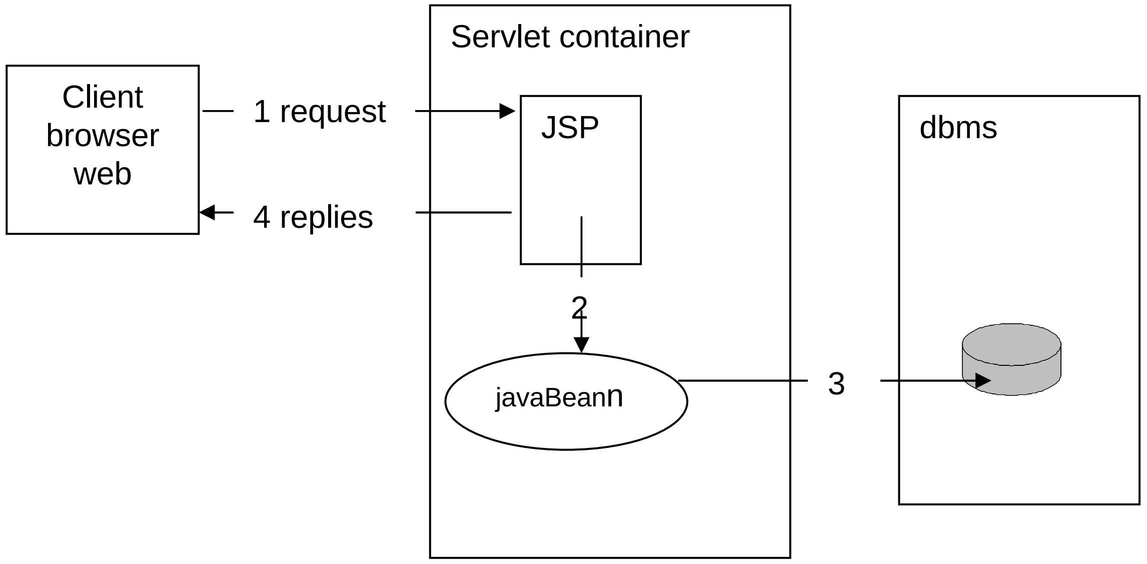
- The generated class derives from **javax.servlet.jsp.HttpJspBase.**
- The **_jspService(HttpServletRequest, HttpServletResponse)** method recovers the HTML code in a string format and writes it in a writer (JspWriter), as a servlet would do.
  It replaces the **service** and **doXXX** methods of servlets.
- The inherited methods jspInit () and jspDestroy () can be redefined. They are called respectively after creating and before deleting the servlet**.**

The generated servlet



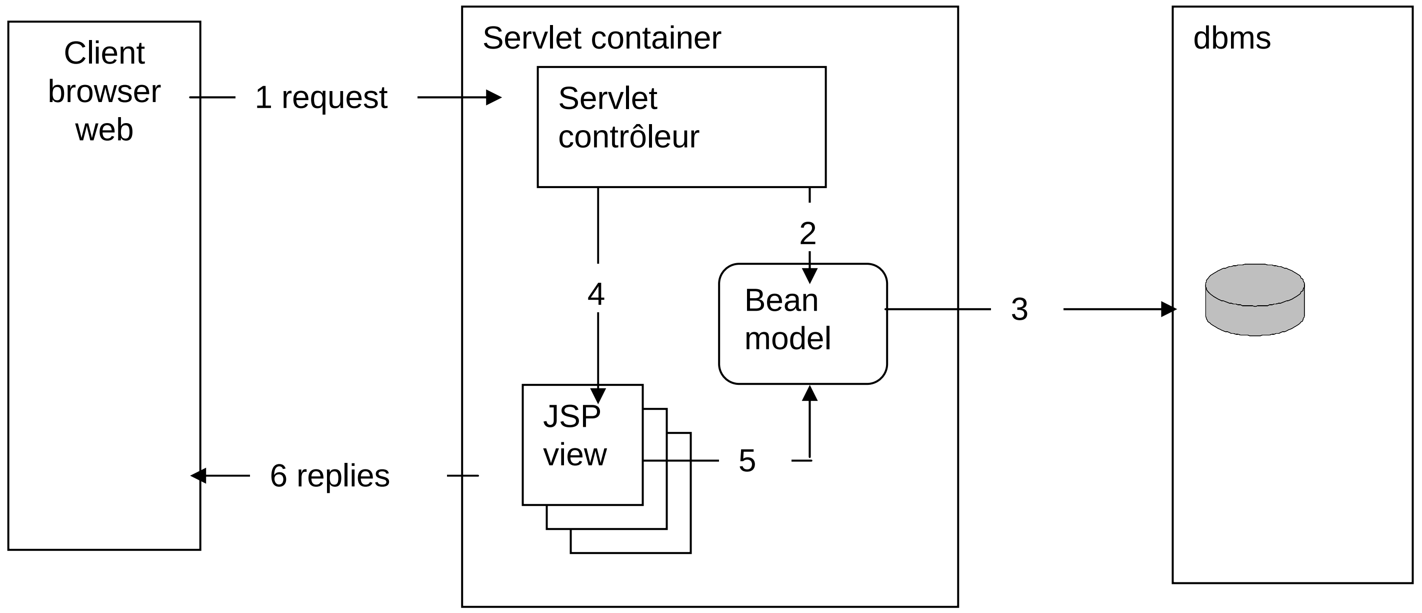
JSP architectures

- A request from the browser is sent to the JSP page. It examines the application (make checks such as authentication) and sends the response to the client.
- The separation between display and data is respected because JavaBeans address the content.
- This model is suitable for very simple applications. It is unadvisable because the JSP should not have a controller role, but only a view role.



MVC architecture (Model View Controller)

- The controller is a servlet that examines the HTTP request (authentication, sessions, ...) and instantiates JavaBeans or objects used by the view (presentation). The controller redirects the request to one of these objects according to the user actions. There is no generation of HTML code in the controller.
- The model is generally represented by JavaBeans (which also implement the job layer). They are aware of neither the controller nor the views.
- The view (presentation) is represented by JSPs that generate the user interface in HTML / XML. There is no job processing in the vue.



Different JSP tags

Types	Syntax	Function
Comment	**<!--** Html **-->** **<%--** JSP **--%>**	Comments which are visible in the generated Html. Comments which are only visible in the JSP.
Instruction	**<%@ page** ... **%>** **<%@ include** ... **%>**	Definition of the structure of the JSP page Inclusion of an html page and/or JSP. This is a copy of the code.
Declaration	**<%!** attribut java **%>** **<%!** méthode java **%>**	Attribute declaration for the generated servlet. Method declaration for the generated servlet.
Scriptlet	**<%** code java ; **%>**	Insertion of java code in the service method of the generated servlet.
Action	<jsp:useBean .../>	Instantiation or recovery of Beans for use.
	<jsp:include .../>	Chaining on the server side to include the response from another JSP or servlet in the caller JSP.
	<jsp:forward .../>	Chaining on the server side for additional processing or generation of the response in another jsp or servlet.

Structure of a JSP page

A JSP page can be separated into several parts :

- Static data such as HTML,
  - Static data are written into the HTTP response exactly as they appear in the source file.
- instructions,
  - Instructions control the way in which the compiler generates the servlet.
    - <%@ page import="java.util.*" %> // import
    - <%@ page contentType="text/html" %> // contentType
- scripts and variables,
  - page, request, response, session …
- actions,
  - JSP actions are tags which call functions on the HTTP server.
    - jsp:useBean, jsp:include, jsp:forward
- Personnalised tags.
  - JSP tag libraries
    - In addition to actions JSP Predefined, custom actions can be added using the JSP Tag Extension API.

Structure of a JSP page (example)

```
<%-- This is a comment JSP --%>
<%@page contentType="text/html"%>
<%@page errorPage="error.jsp"%>
<%-- Importing a package --%>
<%@page import="java.util.*"%>
<html>
<head>
<title>Page JSP</title>
</head>
<body>
<%-- Declaration of a global variable of the class --%>
<%! int visitNumber = 0; %>
<%-- Definition of the Java code --%>
<% //Java code can be written here :
      Date date = new Date();
       visitNumber ++; %>
<h1>JSP page example </h1>
<%-- Printing the variables--%>
<p>At the execution of this script, we are <%= date %>.</p>
<p>This page was displayed <%= nombreVisites %> times!</p>
</body>
</html>
```

Transformation of a JSP page JSP -> Servlet -> html

Java Servlet generated by the compiler:

```
package org.apache.jsp;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import org.apache.jasper.runtime.*;
import java.util.*;
public class example_jsp extends HttpJspBase {
……
}
```

HTML code generated by the server

```
<html>
<head>
<title>Page JSP</title>
</head>
<body>
<h1>Exemple de page JSP</h1>
<p>At the execution of this script, we are Mon Apr 13 19:31:28 EST 2009.</p>
<p>This page was displayed 5 times!</p>
</body>
</html>
```

Implicit objects of the _jspService method

Objetc	Description
`HttpServletRequest` **`request`**	The request passed to _jspService.
`HttpServletResponse` **`response`**	The response passed to _jspService.
`javax.servlet.jsp.PageContext` **`pageContext`**	Associated to the servlet. Provides many methods for managing the servlet. Range : page
`ServletContext` **`application`**	Represents the servletContext. Range : application
`HttpSession` **`session`**	Represents the current session. Range : session
`ServletConfig` **`config`**	Represents the servletConfig Range : page
`JspWriter` **`out`**	The output flow Range: page
`Object` **`page`**	The page itself. Can be replaced by this Range : page

Use of javaBeans

- Allow the recovery of objects which display data from servlet.
- **<jsp:useBean id="myBean" class="exemples.TheBeans" scope="session" />**
  - **scope="page|request|session|application"**

Recovery of the beans attributes is done with the following action :

<jsp:getProperty name="myBean" property="course" />

Modification of the beans attributes is done with the following action :

<jsp:setProperty name="myBean" property="course" value="SR03"/>

…

JSP redirection

- **<jsp:forward page="anotherPage.jsp" />**
  - Stops the execution of the JSP page and redirects the request to another JSP (or servlet).

- **<jsp:include page="aJSPPage.jsp" />**
  - Behaves similarly to calling a subroutine. The control is temporarily given to another page or another JSP file, or a static file. After processing of the other page, the control is given back to the JSP during execution.
  - Using this feature, the Java code can be shared between two pages rather than being duplicated.

Personnalised tag libraries, TagLib

*The **JavaServer Pages Standard Tag Library** (JSTL) is a component of the J2EE development platform. It extends the JSP specification by adding a tag library for common tasks, such as work on XML files, conditional execution, loops and internationalisation (Wikipedia).*

- JSTL offers a method of developping different processing in a JSP page without using Java code directly.
- The taglib is a way to extend the capabilities of servlet servers.
  - manipulating the contents of the JSP page where they are inserted. That is, to recover the body of the tag, then transform it and display it instead of the JSP.
  - Mutualise the generation of html code common to many JSP. They are customisable, which is not possible with <jsp:include>

Bibliography

- Poly SR03, M. Vayssade, chapitres 16 et 17.
- JSP - Java Server Pages, Douanes K. Fields, Mark A. Kolb, Eyrolles editions, 2001.
- Programmer en java, C. Delannoy, Eyrolles editions, 2009.
- JSP avec Eclipse et Tomcat F.X. Sennesal, ENI editions, 2009.
- JSP et Servlets efficaces : Production de sites dynamiques Cas pratiques, J.L. Déléage, Dunod editions, 2009.

- Internet Sources :
  - Serge Tahe, http://tahe.developpez.com/
  - http://fr.wikipedia.org/wiki/JavaServer_Pages
  - http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Servlets.html
  - http://java.sun.com/products/jsp/
  - http://jakarta.apache.org/taglibs/
  - http://www.sybase.com/content/1015262/JSPs-Custom_Tag_Library_vs_JavaBeans.pdf
  - …