AN INTRODUCTION TO BACKEND FOR BEGINNERS

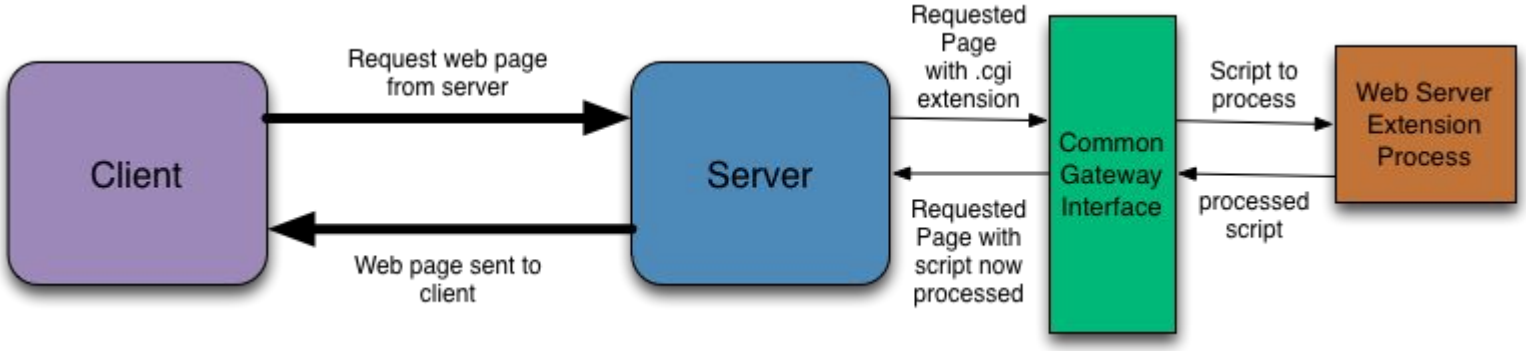
MODULO 3 – JAVA EE

Matteo Colombo Speroni
Armando Esposito

Objectives:

- Learn Java Web basics (Servlets, JSPs, …)
- Create a simple webapp using JEE components

CGI



***CGI** (Common Gateway Interface) is a standard **interface** by which the **Web Server** passes the **Client Request** to a **Program** and receives the response from it.*

PROS

- Dynamic server side content
- Universal support both language and systems (Java, C, Perl, ...)

CONS

- Not efficient, new process for each request
- No easily support to state maintenance
- Cannot change server's internal steps for handling a request

Web Server

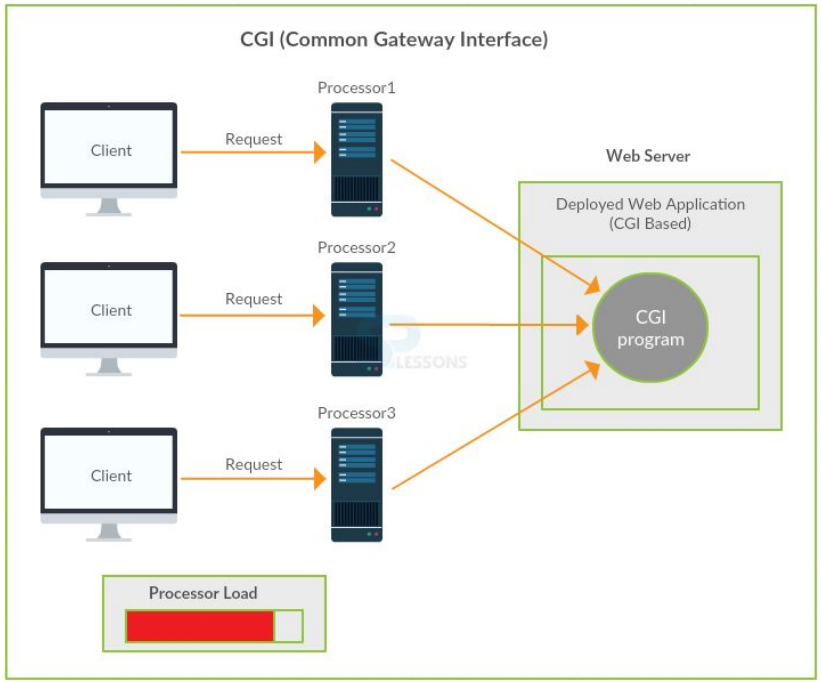
A web server is a software that **processes requests via HTTP**.

The most common use is to host websites **delivering HTML documents**, which may include images, style sheets and scripts in addition to text content.

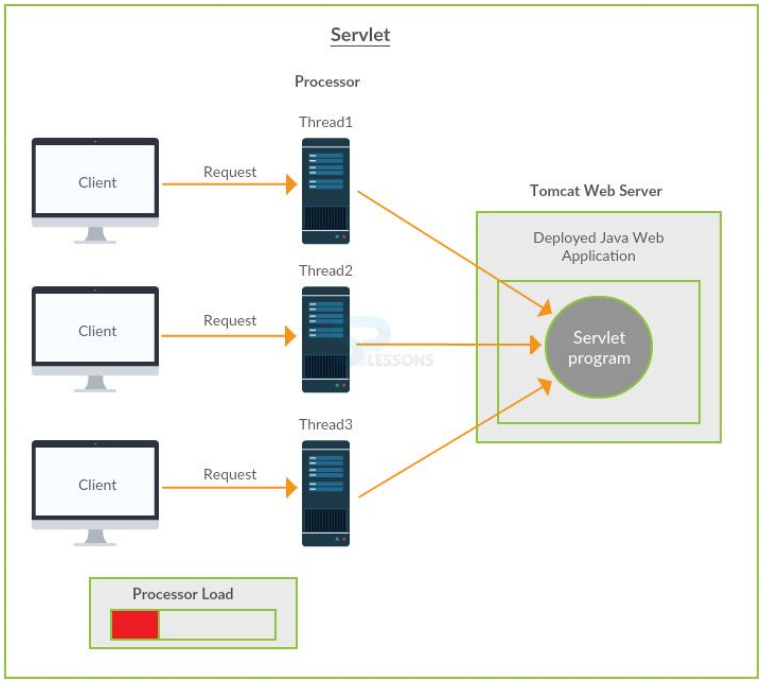
Many generic web servers also s**upport server-side scripting** using Active Server Pages (**ASP**), **PHP**, or other scripting languages.

... And in JAVA?

Servlet vs CGI



VS



JavaEE Web application components

Java Servlets: used to handling requests

Filters and Listeners: used to modify request lifecycle or to react to container events

Java Server Pages (JSP): used for presentation

Deployment descriptor: configuration file to tell the container how to deploy components.

Java Beans: used as value object for JSP

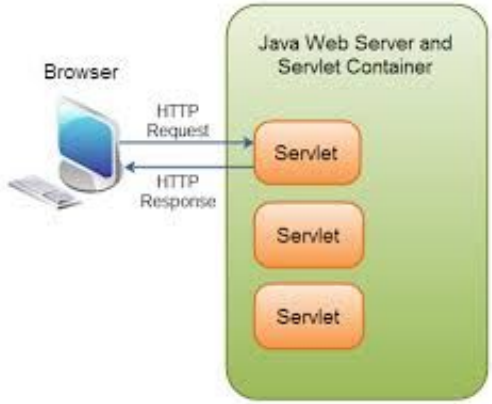
What is a Servlet?

Released in 1997 a as a replacement in JAVA world of the CGI scripts.

A servlet is a small JAVA program that runs within a Web server.

Basically it is an Object with a dedicated life cycle handled by the container.

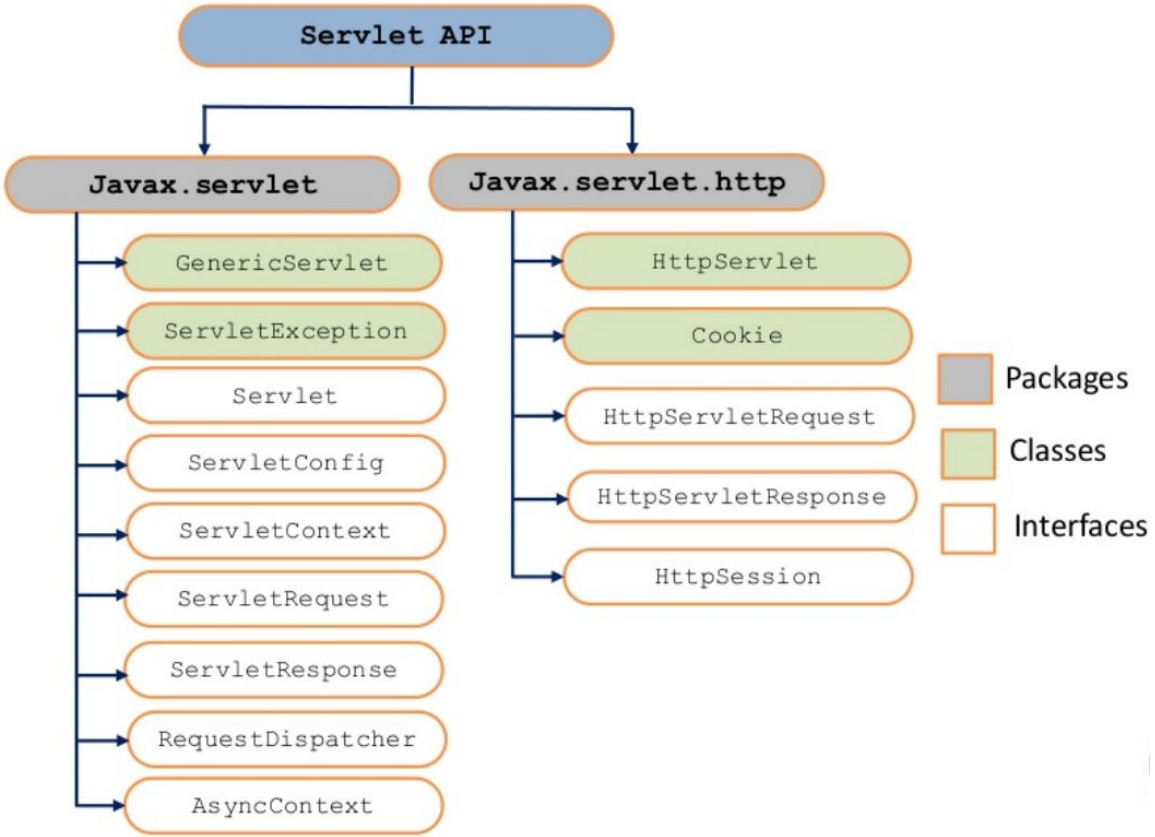
Servlets receive and respond to a web client usually across HTTP.



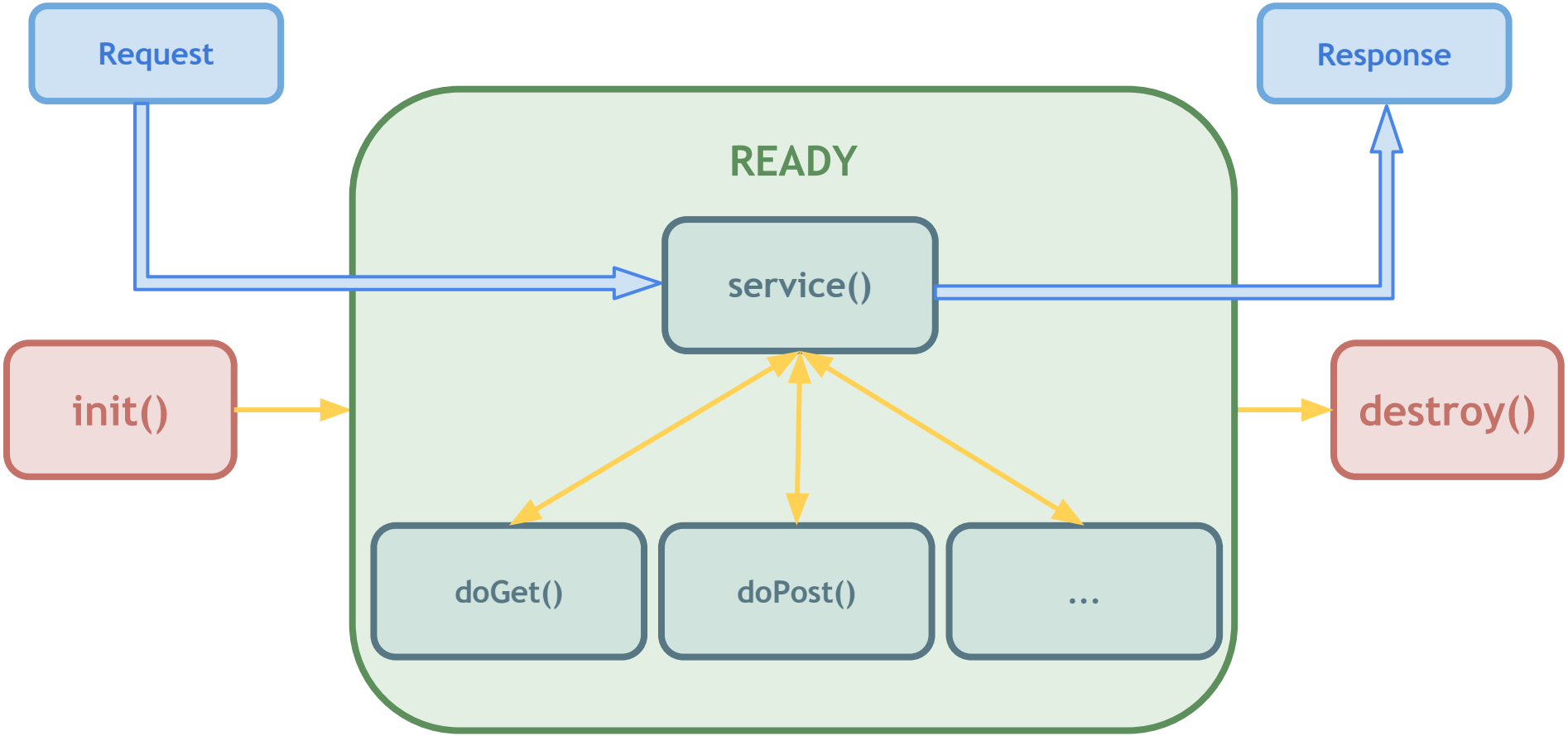
Servlet API history

Servlet API version	Released	Platform	Important Changes
Servlet 4.0	Under development	Java EE 8	HTTP/2
Servlet 3.1	May 2013	Java EE 7	Non-blocking I/O, HTTP protocol upgrade mechanism (WebSocket)[5]
Servlet 3.0	December 2009	Java EE 6, Java SE 6	Pluggability, Ease of development, Async Servlet, Security, File Uploading
Servlet 2.5	September 2005	Java EE 5, Java SE 5	Requires Java SE 5, supports annotation
Servlet 2.4	November 2003	J2EE 1.4, J2SE 1.3	web.xml uses XML Schema
Servlet 2.3	August 2001	J2EE 1.3, J2SE 1.2	Addition of `Filter`
Servlet 2.2	August 1999	J2EE 1.2, J2SE 1.2	Becomes part of J2EE, introduced independent web applications in .war files
Servlet 2.1	November 1998	Unspecified	First official specification, added `RequestDispatcher`, `ServletContext`
Servlet 2.0		JDK 1.1	Part of Java Servlet Development Kit 2.0
Servlet 1.0	June 1997		

Servlet API


Servlet API
Javax.servlet
Javax.servlet.http
GenericServlet
ServletException
Servlet
ServletConfig
ServletContext
ServletRequest
ServletResponse
RequestDispatcher
AsyncContext
HttpServlet
Cookie
HttpServletRequest
HttpServletResponse
HttpSession
Packages
Classes
Interfaces

HttpServlet life cycle



Servlet Class

```java
public class HelloServlet extends HttpServlet {

   @Override
   public void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException, ServletException {
       resp.setContentType("text/html");
       PrintWriter out = resp.getWriter();
       out.println("<html>");
       out.println("<body>");
       out.println("<h1>Hello World at " + req.getRequestURI() +"!!</h1>");
       out.println("<p>key is: " + req.getParameter("key") + "</p>");
       out.println("</body>");
       out.println("</html>");
   }

   @Override
   public void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
      // code
   }

   @Override
   public void init(ServletConfig config) throws ServletException {
       super.init(config);
   }


}
```

Servlet definition

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         mlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaeehttp://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
         version="3.1">


   <servlet>
       <servlet-name>hello-servlet</servlet-name>
       <servlet-class>package.HelloServlet</servlet-class>
   </servlet>

   <servlet-mapping>
       <servlet-name>hello-servlet</servlet-name>
       <url-pattern>/hello</url-pattern>
   </servlet-mapping>


</web-app>
```

Deployment descriptor

Must be called web.xml and must reside in the WEB-INF directory in the web application root

Defines servlets and URLs mapping

Defines filters and listeners

Defines resources available to the webapp

Defines security constraints

Defines also welcome files, session timeout, …

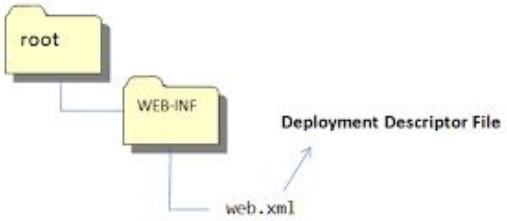


FIG : web.xml file

Welcome | index.jsp | myServlet.java | web.xml

General Servlets Filters Pages XML

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="h
  <servlet>
    <servlet-name>myServlet</servlet-name>
    <servlet-class>myPackage.myServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>myServlet</servlet-name>
    <url-pattern>/servlet/myServlet</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>
            30
        </session-timeout>
  </session-config>
  <welcome-file-list>
    <welcome-file>
            index.jsp
        </welcome-file>
  </welcome-file-list>
</web-app>
```

ServletConfig and ServletContext

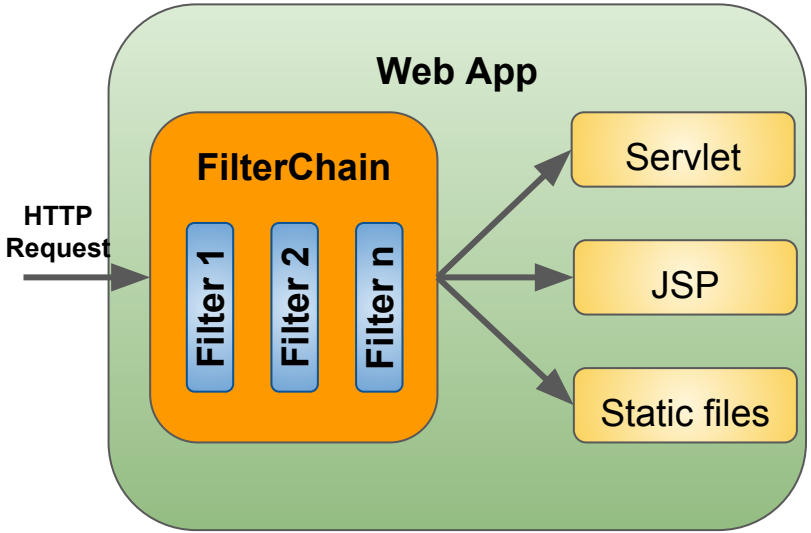
```
<servlet>
   <servlet-name>hello-servlet</servlet-name>
   <servlet-class>com.example.HelloServlet</servlet-class>
   <init-param>
       <param-name>key</param-name>
       <param-value>value</param-value>
   </init-param>
   <load-on-startup>1</load-on-startup>
</servlet>
```

```
<web-app>
   <context-param>
       <param-name>dburl</param-name>
       <param-value>jdbc:...</param-value>
   </context-param>
</web-app>
```

ServletConfig	ServletContext
web.xml *<init-param>* tag inside *<servlet-class>* tag	*<context-param>* tag under *<web-app>* tag
One for each Servlet	Global to webapp
Created during servlet initialization	Created at deployment
Visible only to his Servlet and has the same life	Visible to all Components and life equal to webapp

Filters

Object that can intercept HTTP requests targeted at your web application.




```
public class SampleFilter implements Filter {

   public void init(FilterConfig filterConfig)
       throws ServletException {
   }

   public void doFilter(ServletRequest request,
                        ServletResponse response,
                        FilterChain filterChain)
       throws IOException, ServletException {
      //code
       filterChain.doFilter(request,response);
   }

   public void destroy() {
   }
}
```

```
<filter>
   <filter-name>myFilter</filter-name>
 <filter-class>servlets.SampleFilter</filter-class>
</filter>

<filter-mapping>
   <filter-name>myFilter</filter-name>
   <url-pattern>*.sample</url-pattern>
</filter-mapping>
```

Listeners

Event handler get fired when events occurs inside the server.

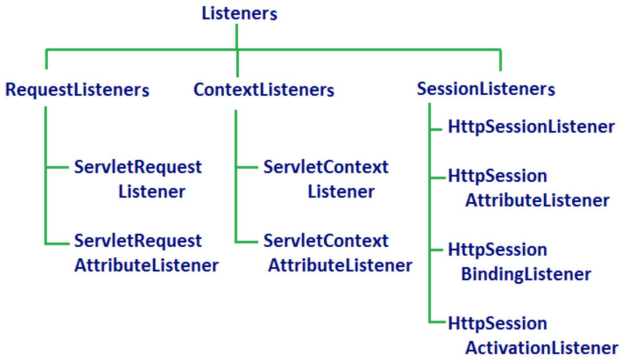
```
public class SessionMonitorListener
       implements HttpSessionListener, ServletContextListener {

   private Integer active=0, max=0;

   @Override
   public void contextInitialized(ServletContextEvent sce) {
        this.store(sce.getServletContext());
   }

   @Override
   public void sessionCreated(HttpSessionEvent httpSessionEvent) {
        this.active++;
        if(this.active > max) this.max = active;
        this.store(httpSessionEvent.getSession().getServletContext());
   }

   @Override
   public void sessionDestroyed(HttpSessionEvent httpSessionEvent) {
        this.active--;
        this.store(httpSessionEvent.getSession().getServletContext());
   }

   private void store(ServletContext servletContext) {
        servletContext.setAttribute("sessions_active", active);
        servletContext.setAttribute("sessions_max", max);
   }
```

```
<listener>
    <listener-class>SessionMonitorListener</listener-class>
</listener>
```

```
public class SessionMonitorServlet extends HttpServlet {

   protected void doGet(HttpServletRequest req,
                        HttpServletResponse resp)
           throws ServletException, IOException {

       Integer active = (Integer)
                        getServletContext()
                        .getAttribute("sessions_active");
       Integer max = (Integer)
                       getServletContext()
                       .getAttribute("sessions_max");

       resp.setContentType("text/html");
       PrintWriter out = resp.getWriter();
       out.println("<html>");
       out.println("<body>");
       out.println("<p>Active Sessions: "+ active+"</p>");
       out.println("<p>Max Sessions: "+ max +"</p>");
       out.println("</body>");
       out.println("</html>");
   }
}
```

Listeners Types

Listener Summary			
Object	**Event**	**Event class**	**Listener interface**
Web or ServletContext	attribute added,removed,replaced	ServletContextAttributeEvent	ServletContextAttributeListener
Web or ServletContext	initialize/destroy	ServletContextEvent	ServletContextListener
Request	attribute added,removed,replaced	ServletRequestsAttributeEvent	ServletRequestAttributeListener
Request	initialize/destroy	ServletRequestEvent	ServletRequestListener
Session	creation/invalidation/timeout	HttpSessionEvent	HttpSessionListener
Session	activation/passivation	HttpSessionEvent	HttpSessionActivationListener
Session	attribute added,removed,replaced	HttpSessionEvent	HttpSessionAttributeListener
Session	object bound/unbound from session	HttpSessionBindingEvent	HttpSessionBindingListener



Tomcat 7 and Servlet 3.0 containers

Use annotations to configure web application, these annotation must be supported by servlet container.

Omit web.xml entirely

- You are permitted to use web.xml even when using @WebServlet, but the entire file is completely optional.

```
@WebServlet(value = "/hello",name = "hello-servlet", initParams = {@WebInitParam(name="key", value="value")})
public class HelloServlet extends HttpServlet { ... }
```

@WebServlet

@WebFilter

@WebListener

@WebInitParam

@ServletSecurity

@MultipartConfig

@HttpConstraint

@HandlesTypes

@HttpMethodConstraint

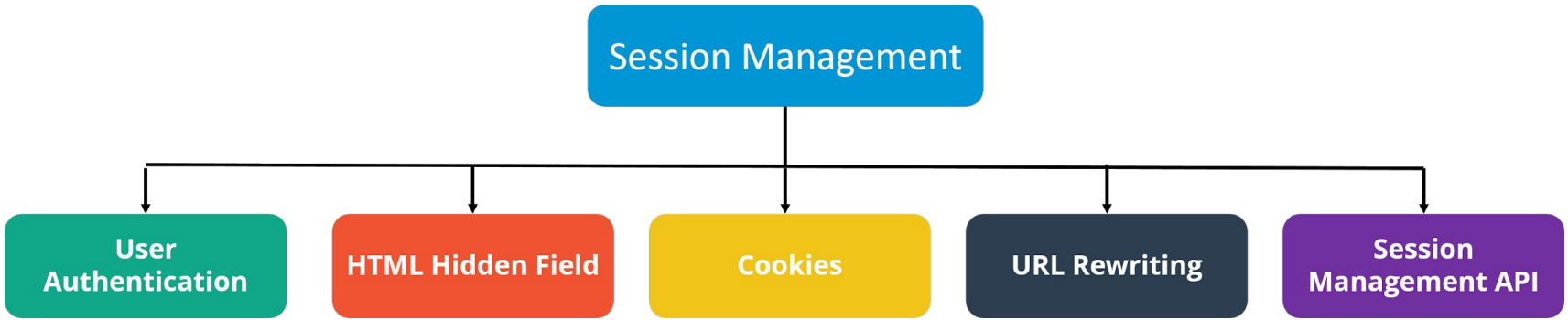
Session

HTTP protocol and Web Servers are stateless, but sometimes in web applications, we should know who the client is and process the request accordingly.

Server will create session object:

```
HttpSession session = req.getSession();
HttpSession session = req.getSession(true);
```

- assign unique ID
- track sessions by some method such as cookies or URL rewriting
- Servlet can store anything in session context using session attributes



Session API

getId(): return the unique identifier

isNew(): determines if session is new to client

getCreationTime(): time at which session was first created

getLastAccessedTime(): time at which was last sent from client

invalidate(): invalidation and unbind of all objects related to it

setAttribute(key, value): set attribute into session object

getAttribute(key): retrieve attribute (if exist) from session object

setMaxInactiveInterval(): set timeout before consider object stale (override of default web.xml value)

Cookie

Small piece of information used to store shared information between Client and Server.

The class **javax.servlet.http.Cookie** is used to represent a cookie server side:

- Getter methods: **getName(), getValue(), getPath(), getDomain(), getMaxAge(), getSecure(), …**
- Setter methods: **setValue(), setPath(), setDomain(), setMaxAge(), …**

```
final Cookie[] cookies = req.getCookies();
String name = null;
for (Cookie c : cookies) {
   if (c.getName().equals("userName") && c.getDomain().equals("mydomain.com")) {
       name = c.getValue();
   }

}
PrintWriter out = resp.getWriter();
out.println("<html>");
out.println("<body>");
if (name != null) {
   out.println("<p>Welcome Back: " + name +"</p>");
} else {
   out.println("<p>Welcome!!!</p>");
}
out.println("</body>");
out.println("</html>");
```

Forward vs SendRedirect

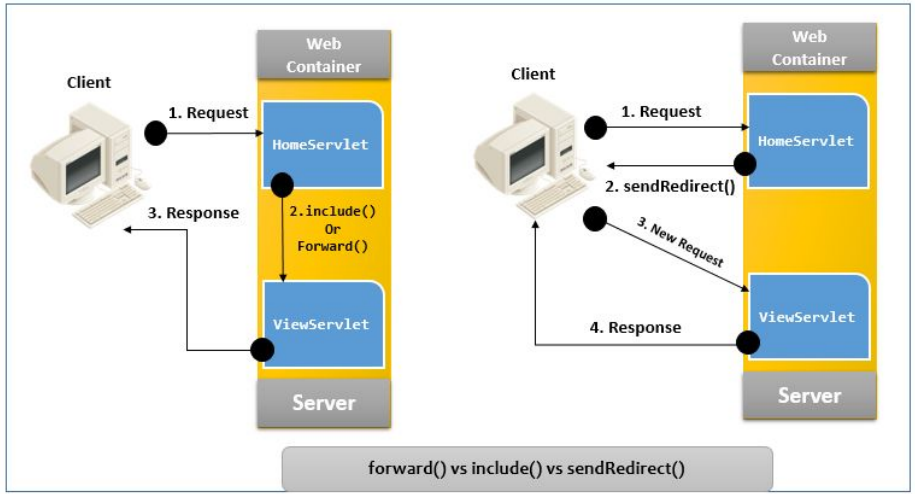
A request may need several servlets to cooperate, **RequestDispatcher** object can be used for “redirecting” a request from one servlet to another:

Forward: request is transferred to other resource within the same server for further processing giving control to the next servlet.

Include: request is transferred to other resource within the same server, the control remain in charge to the current servlet and the result is just included in the current one.

SendRedirect: request is transferred to another resource passing through the client.

```
RequestDispatcher rd =
getServletContext().getRequestDispatcher("/dispatchTo");

rd.forward(req,resp);

rd.include(req,resp);

resp.sendRedirect("/redirectTo");
```



Java Server Pages

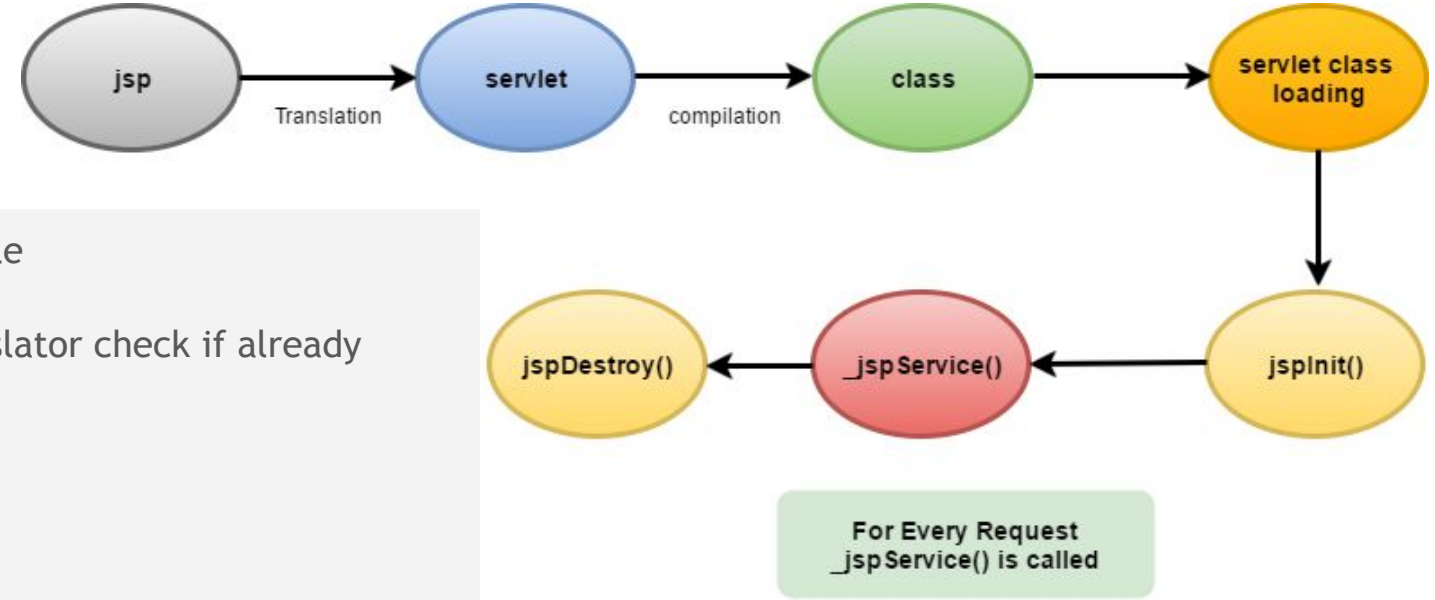
A simplified, fast way to create dynamic web content HTML pages with embedded Java Code or Java Beans.

A JSP is a compiled to a Java Servlet automatically by the Servlet container, it is then cached.

It represent the presentation layer in a n-tier architecture.

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Hello World JSP</title>
    </head>
    <body>

    <h1>Hello World JSP</h1>


    <% String userName = request.getParameter("name");
       if (userName == null) userName = "World"; %>

    Hello, <%= userName %>!


    </body>
</html>
```

JSP life cycle



Similar to Servlet life cycle

When requested the translator check if already compiled or it does:

- Parsing
- Transformation
- Compile

Initialization / destruction

Handling requests with _jsp_Service method

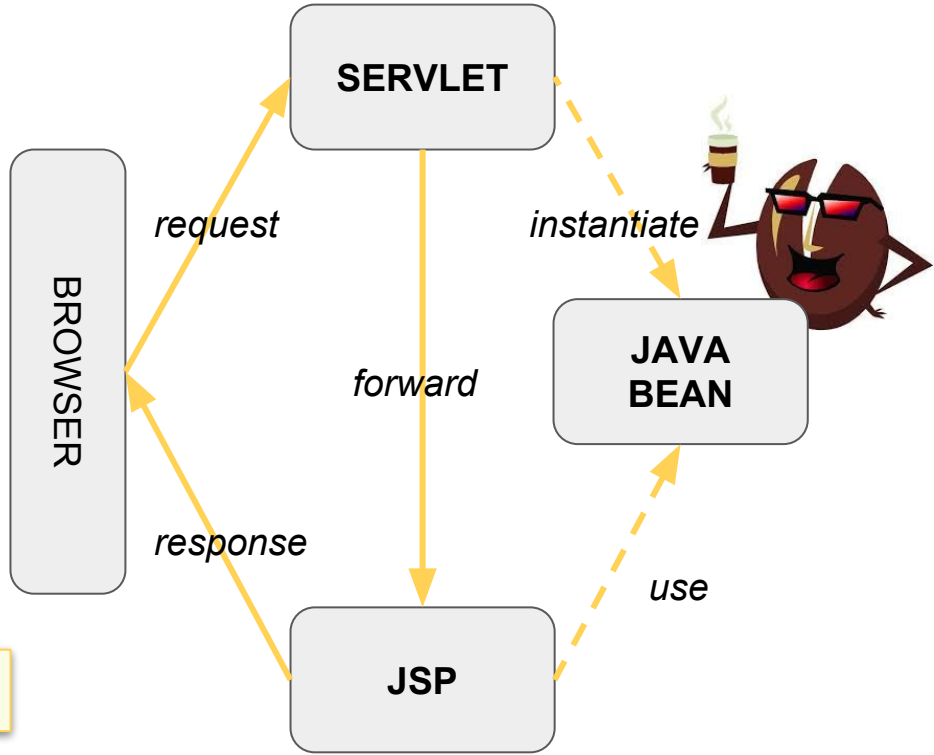
Java Bean

Class that encapsulate many objects into a single object (the bean).

They must have these characteristics:

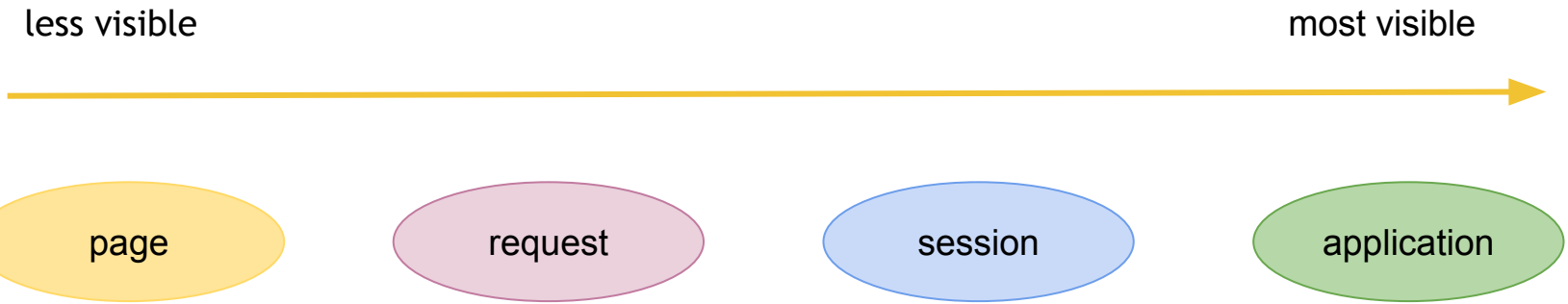
- **serializable**
- **zero-argument constructor**
- **access to properties using getter and setter methods**

In a 3-tier architecture they are the objects shared between Business Logic Layer and Presentation Layer




```
<jsp:useBean id="beanName" scope="session" class="javaclass" />
```

Scopes



Available to the handling **page only**

Available to the handling page and to **any servlet or JSP that control is forwarded to or included**

Available to any servlet or JSP **within the same session** (across repeat visits to web application)

Available to any servlet or JSP within **the same web application**

Use of tags, scriptlet vs action elements

JSP scripting elements require that developers mix Java code with HTML.

Each JSP tag is associated with a Java class that contains the logic without exposing it.

```
<html>
   <head>
       <title>Count to 10 in JSP scriptlet</title>
   </head>
   <body>
   <%
       for(int i=1;i<=10;i++)
       {
   %>
   <%= i %><br/>
   <%
   }
   %>
   </body>
</html>
```

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<html>
   <head>
       <title>Count to 10 Example (using JSTL)</title>
   </head>
   <body>
       <c:forEach var="i" begin="1" end="10" step="1">
           <c:out value="${i}" /> <br />
       </c:forEach>
   </body>
</html>
```

JSP action elements

Action elements are an important syntax element represented by tag introduced to avoid the use of scriptlets.

```
<prefix:action_name attr1 = “value” attr2 = “value2”>
    action_body
</prefix:action_name>

<prefix:action_name attr1 = “value” attr2 = “value2” />
```

JSP predefined tags

Tag prefix is:

<jsp: …>

```
<jsp:include page="scripts/login.jsp" />
```

External tag library (JSTL) or Custom

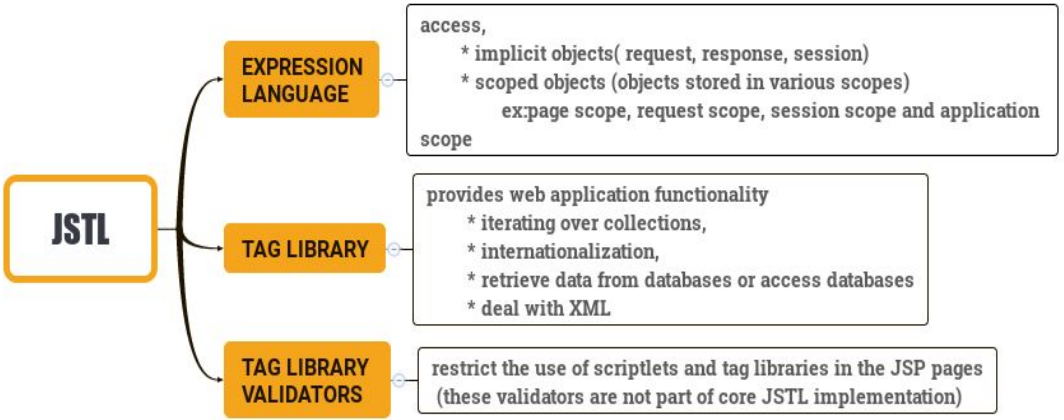
Tag prefix is defined by developer:

```
<%@ taglib prefix="spring"
uri="http://www.springframework.org/tags"%>

<spring:url value="/resource" var="resourceUrl" />
```

Jstl standard library

JSTL (JSP Standard Tag Libraries) is a collection of JSP custom tags developed by Java Community Process, (www.jcp.org).

It is a specification, the most famous implementation is Apache Standard Taglibs (https://tomcat.apache.org/taglibs/standard/).



Functional Area	URI	Prefix
Core	http://java.sun.com/jsp/jstl/core	c
Xml Processing	http://java.sun.com/jsp/jstl/xml	x
I18N capable formatting	http://java.sun.com/jsp/jstl/fmt	fmt
Relational database accesing (SQL)	http://java.sun.com/jsp/jstl/sql	sql
Functions	http://java.sun.com/jsp/jstl/functions	fn

Expression language

The Expression Language (EL) is a special purpose programming language for embedding expressions into Web pages.

All EL expressions are evaluated at runtime

The EL usually handles data type conversion and null values

An EL expression always starts with a ${ and ends with a }

The expression can include:

- literals (“1”, “100” etc)
- variables
- implicit variables (cookie,pageContext,sessionScope, …)

```
<%-- evaluate expressione --%>
<c:out value = '${1+2+3}' />

<%-- Access request parameters by name --%>
<c:out value='${param.lastName}'/>
<c:out value='${param.firstName}'/>

<%-- Access implicit object cookie --%>
<c:forEach items='${cookie}' var='c'>
   <c:out value='${"Key: " + c.key + " Value: "
c.value.value}'/>
</c:forEach>

<%-- Use operator --%>
<c:if test="${empty param.name}">
&& is the same as and, || is the same as or
</c:if>
```

You can use standard java operators or their JSTL form (empty, or, and, +, -, ||, &&, …)

Servlet-Based frameworks

OCI | HOME TO GRAILS

spring

Servlet alone are no more used as alone technology, but…

Struts

Useful Links

reactive web and Servlet 4

annotation vs deployment descriptor

JSTL and Tags

Github project course repository and Lessons documentation:
https://github.com/mcolombosperoni/an-introduction-to-backend-for-beginners