Java EE基础与应用

翁秀木

第六回
Servlet+JSP+JSTL+EL

- JSP
- Servlet
- JSTL
- EL

学习目标

- 掌握什么是JSP、为什么用JSP、JSP怎么运作
- 掌握JSP的脚本元素（Scripting Element）
- 掌握JSP的隐含对象（implicit object），重点request/response/session
- 掌握JSP的指令
- 掌握JSP的动作
- 掌握什么是Servlet、为什么用Servlet、怎么用Servlet，即Servlet运作原理
- 掌握什么是Filter、为什么用Filter、怎么用Filter
- 掌握什么是JSTL、为什么用JSTL、怎么用JSTL
- 掌握什么是EL、为什么用EL、怎么用EL

JSP的简介

- 什么是JSP （**W**hat）
- 为什么用JSP （**W**hy）
- JSP怎么运作、怎么用JSP （ho**W**）

什么是JSP

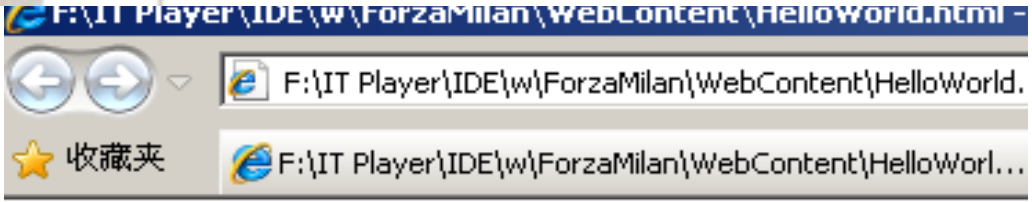
- **J**ava 技术
- **S**erver 端
- 动态的 web **P**age

为什么用JSP

HTML

HelloWorld.html

```
1 <html>
2 <body>
3 <font color = "red">Hello World</font>
4 </body>
5 </html>
```



Hello World

为什么用JSP

Servlet

```
public void service(HttpServletRequest request, HttpServletResponse response)
        throws IOException {

    int loopThreshold = Integer.parseInt(request.getParameter("loopThreshold"));

    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<body>");
    for (int x = 0; x < loopThreshold; x++) {
        out.println("<font color = 'red'>Hello World</font>");
    }
    out.println("</body>");
    out.println("</html>");
}
```

Hello World
Hello World
Hello World

为什么用JSP

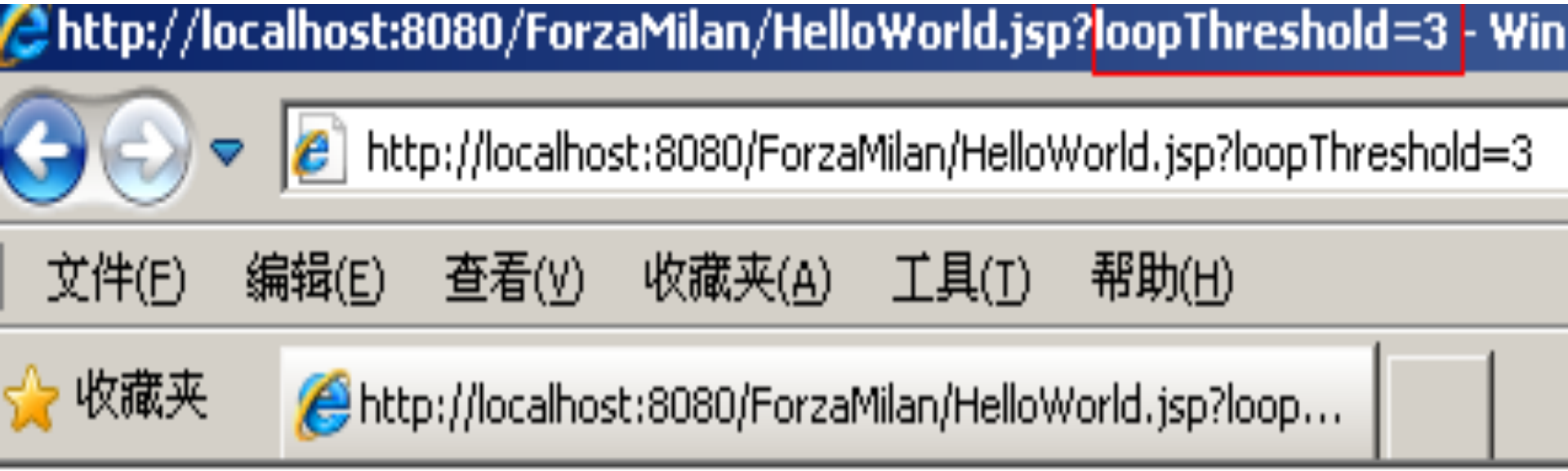
JSP

```
<%  int loopThreshold =
  Integer.parseInt(request.getParameter("loopThreshold"));%>
<html>
<body>
<%for (int x = 0; x < loopThreshold; x++) {%>
  <font color="red">Hello World</font><br>
<%}%>
</body>
</html>
```

HTML + Java

为什么用JSP

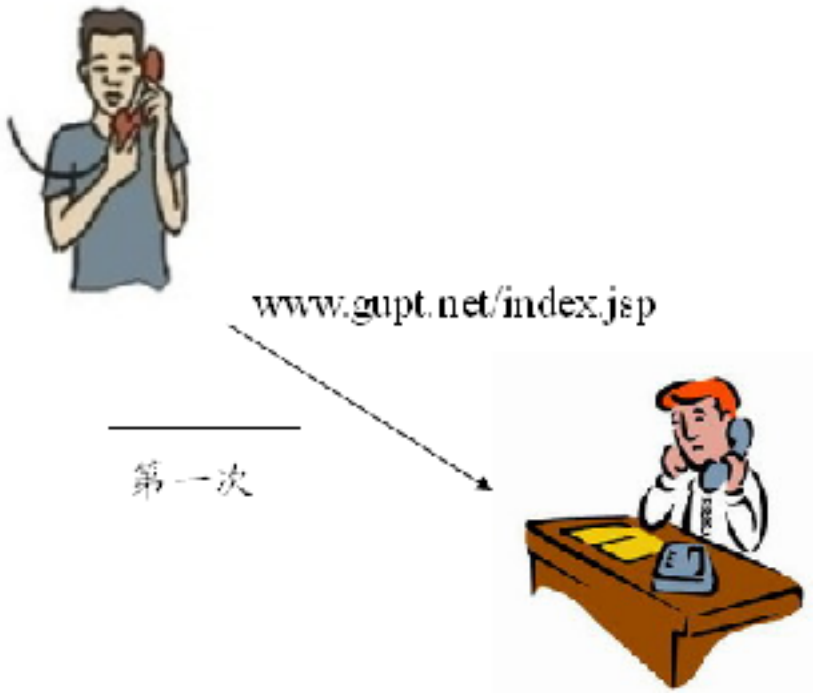
JS
P


http://localhost:8080/ForzaMilan/HelloWorld.jsp?loopThreshold=3 - Win
http://localhost:8080/ForzaMilan/HelloWorld.jsp?loopThreshold=3
文件(F) 编辑(E) 查看(V) 收藏夹(A) 工具(T) 帮助(H)
收藏夹
http://localhost:8080/ForzaMilan/HelloWorld.jsp?loop...
Hello World
Hello World
Hello World

为什么用JSP

- 更便利地在服务器端动态地生成HTML
  - 更清晰地将呈现逻辑与其他逻辑分开

JSP怎么运作



1> **转化** （translate）

index.jsp > index.java

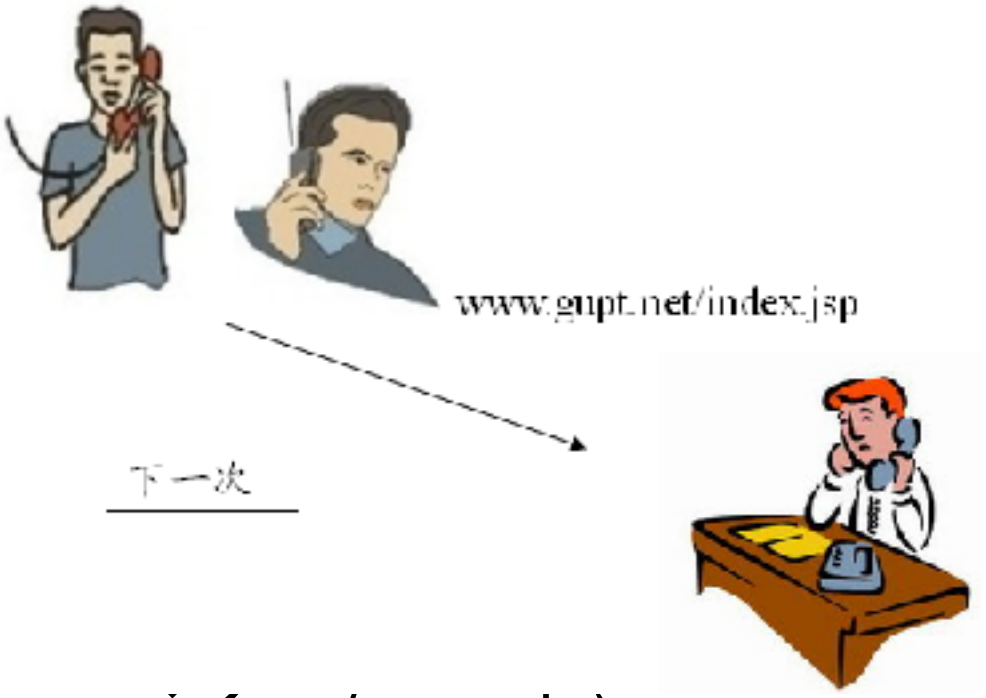
2> **编译** （compile）

index.java > index.class

3> **运行** （execute)

运行index实例，处理请求。

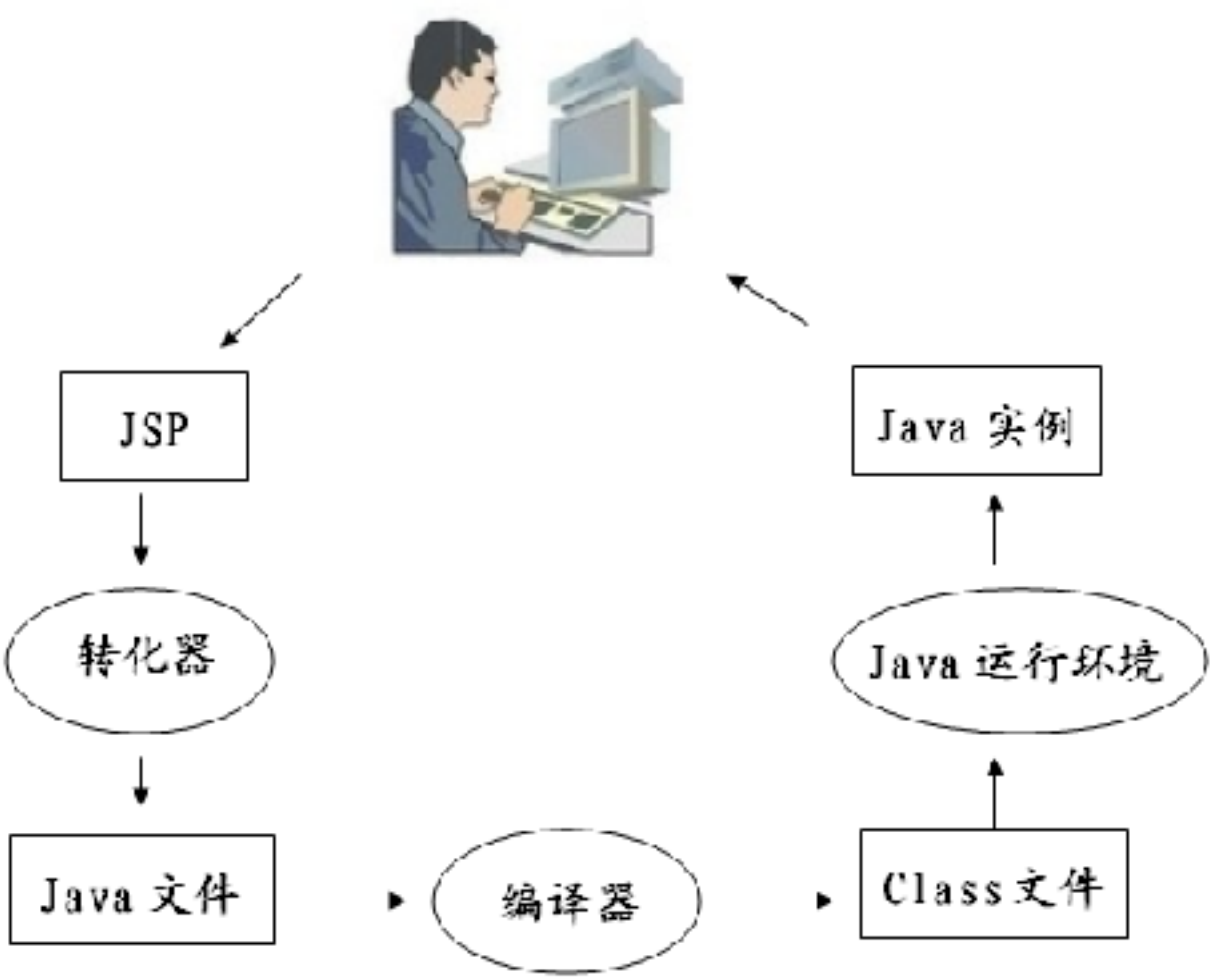
JSP怎么运作



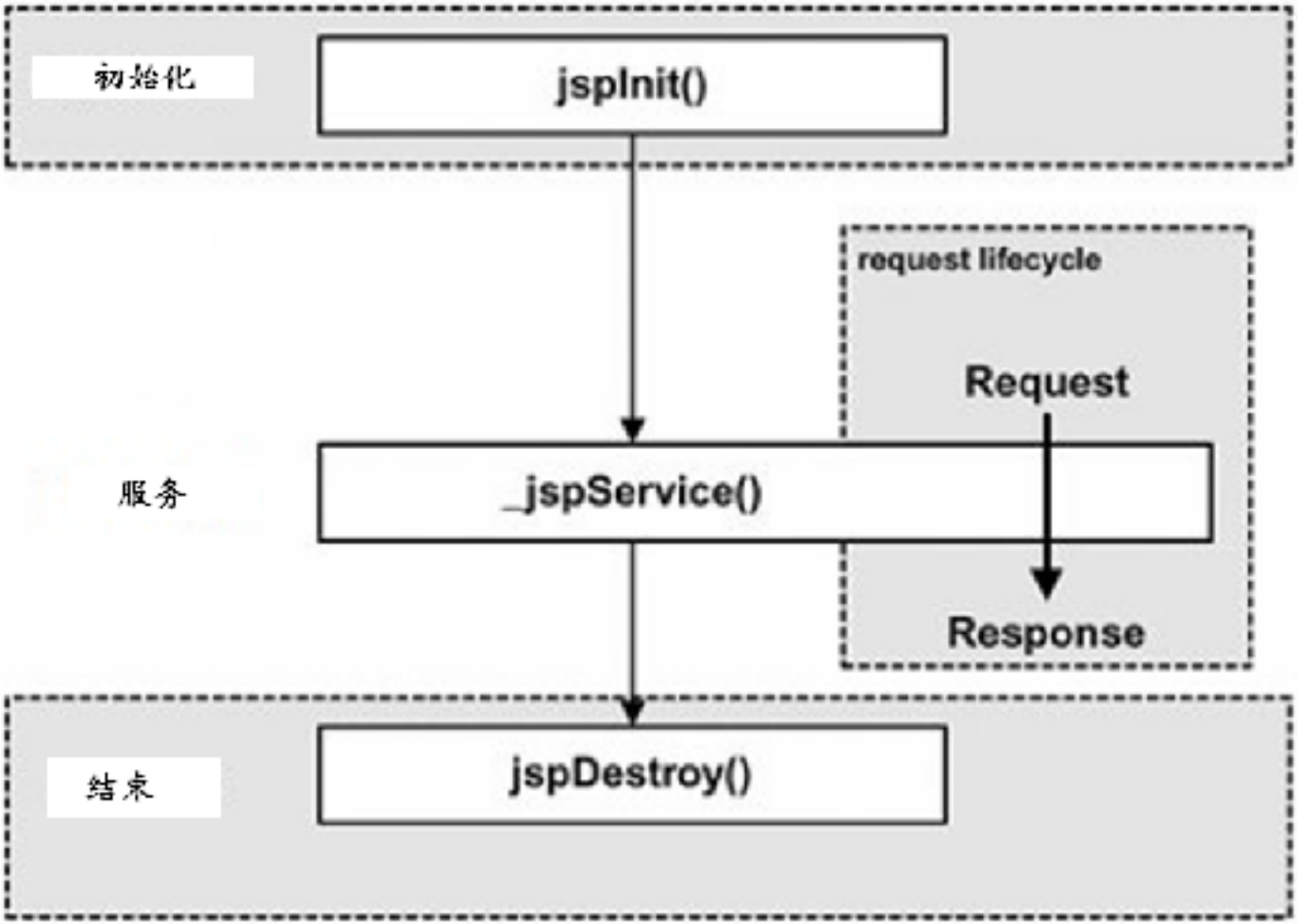
运行 （execute)

运行index实例，处理请求。

JSP怎么运作



JSP怎么运作


初始化
jspInit()
request lifecycle
Request
服务
_jspService()
Response
结束
jspDestroy()

怎么用JSP

- **JSP脚本元素**
  - JSP指令
  - JSP动作
- JSP隐含对象

JSP的脚本元素 Scripting element

- 声明 declaration

  <%! %>

- 脚本 Scriptlet

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

- 表达式 expression

  <%= %>

JSP的指令 directive

- page 页面

  <%@page %>

- include 包含

<%@include %>

- taglib 标签库

  <%@taglib %>

// JSP中的对象的范围 scope，

- page 页面
- request 请求
- session 会话
- application 应用

JSP的隐含对象 implicit object

1. **request 请求** : HttpServletRequest (request scope)
2. **response 响应** : HttpServletResponse (request scope)
3. **session 会话** : HttpSession (session scope)
4. **out 输出** : PrintWriter (request scope)
5. **application 应用** : ServletContext (application scope)
6. **config 配置** : ServletConfig (page scope)
7. **pageContext 页面环境** : PageContext (page scope)
8. **page 页面** : this (page scope)
9. **exception 异常** : Throwable (page scope)

+Page箱++++++

方式1)
pageContext.setAttribute(name, object);
pageContext.getAttribute(name);

方式2)
pageContext.setAttribute(name, object, PageContext.PAGE_SCOPE);
pageContext.getAttribute(name, PageContext.PAGE_SCOPE);

+Request箱++++++++++++++++++++++++++++++++

方式1)
request.setAttribute(name, object)
request.getAttribute(name)

方式2)
pageContext.setAttribute(name, object, PageContext.REQUEST_SCOPE);
pageContext.getAttribute(name, PageContext.REQUEST_SCOPE);

+Session箱++++++++++++++++++++++++

方式1)
session.setAttribute(name, object)
session.getAttribute(name)

方式2)
pageContext.setAttribute(name, object, PageContext.SESSION_SCOPE);
pageContext.getAttribute(name, PageContext.SESSION_SCOPE);

+Application箱++++++++++++++++++++++

方式1)
application.setAttribute(name, object)
application.getAttribute(name)

方式2)
pageContext.setAttribute(name, object, PageContext.APPLICATION_SCOPE);
pageContext.getAttribute(name, PageContext.APPLICATION_SCOPE);

// JSP的动作 action *

- forward

  <jsp:forward>

  <jsp:param>

- include

  <jsp:include>

  <jsp:param>

- useBean

  <jsp:useBean>

  <jsp:setProperty> <jsp:getProperty>

JavaBean *

- 可序列化 implements Serializable
- public的get/set方法，以访问实例属性(field)
- public的无参数构造器

Servlet的简介

- 什么是Servlet （**W**hat）
- 为什么用Servlet（**W**hy）
- Servlet怎么运作，怎么用Servlet（ho**W**）

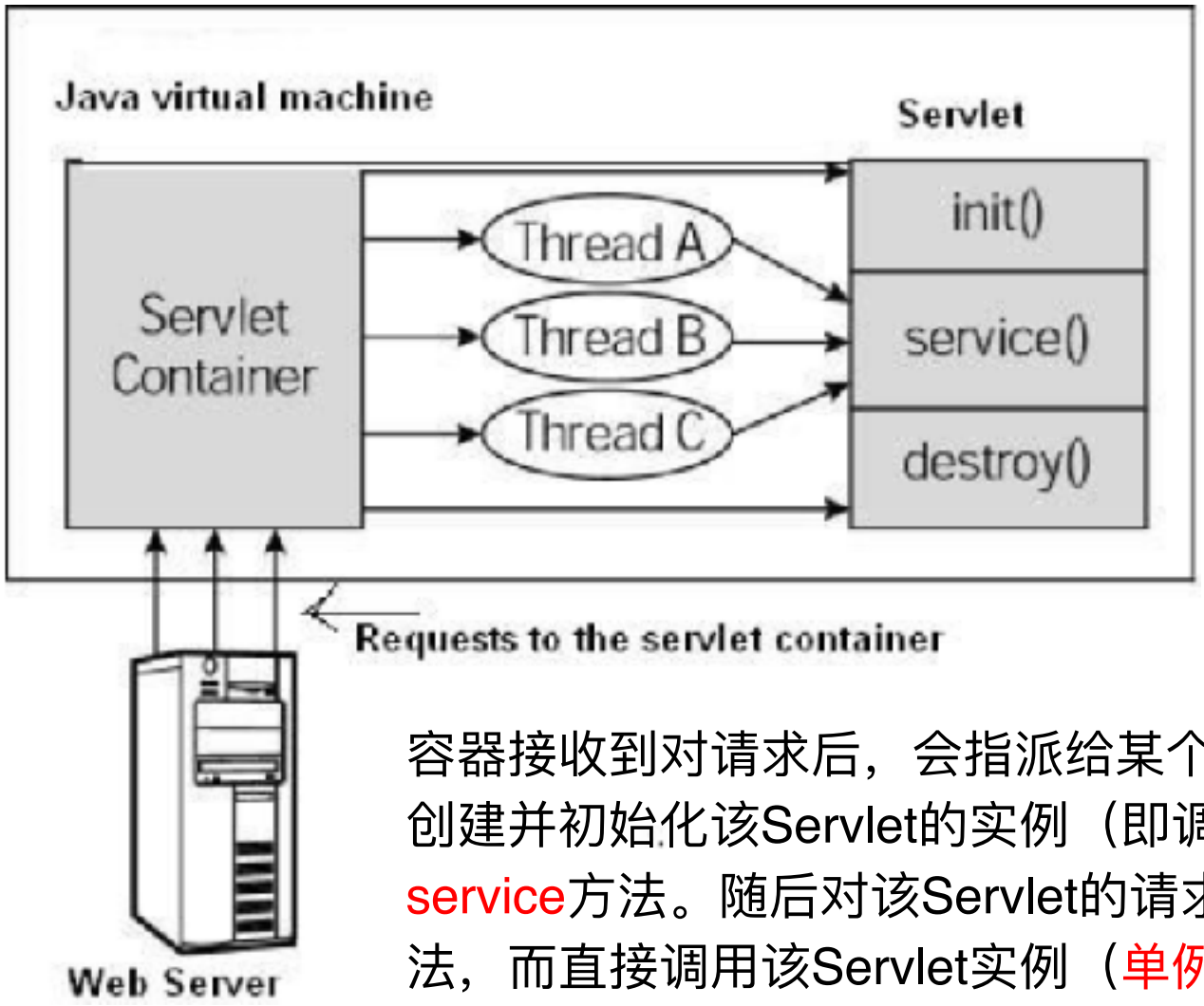
什么是Servlet

- Java applet是在客户（client）端web浏览器（browser）里运行的小程序，Servlet则可理解为在服务器（server）端运行的小程序
  - 其目的是处理各种互联网request请求（主要是http请求，并将结果返回给客户端。
- javax.servlet 和javax.servlet.http 包（packages）提供了写Servlet的接口和类。对提供通用服务的Servlet，可继承GenericServlet；对提供Http服务的Servlet，可继承HttpServlet（定义了doGet和doPost方法）

为什么用Servlet

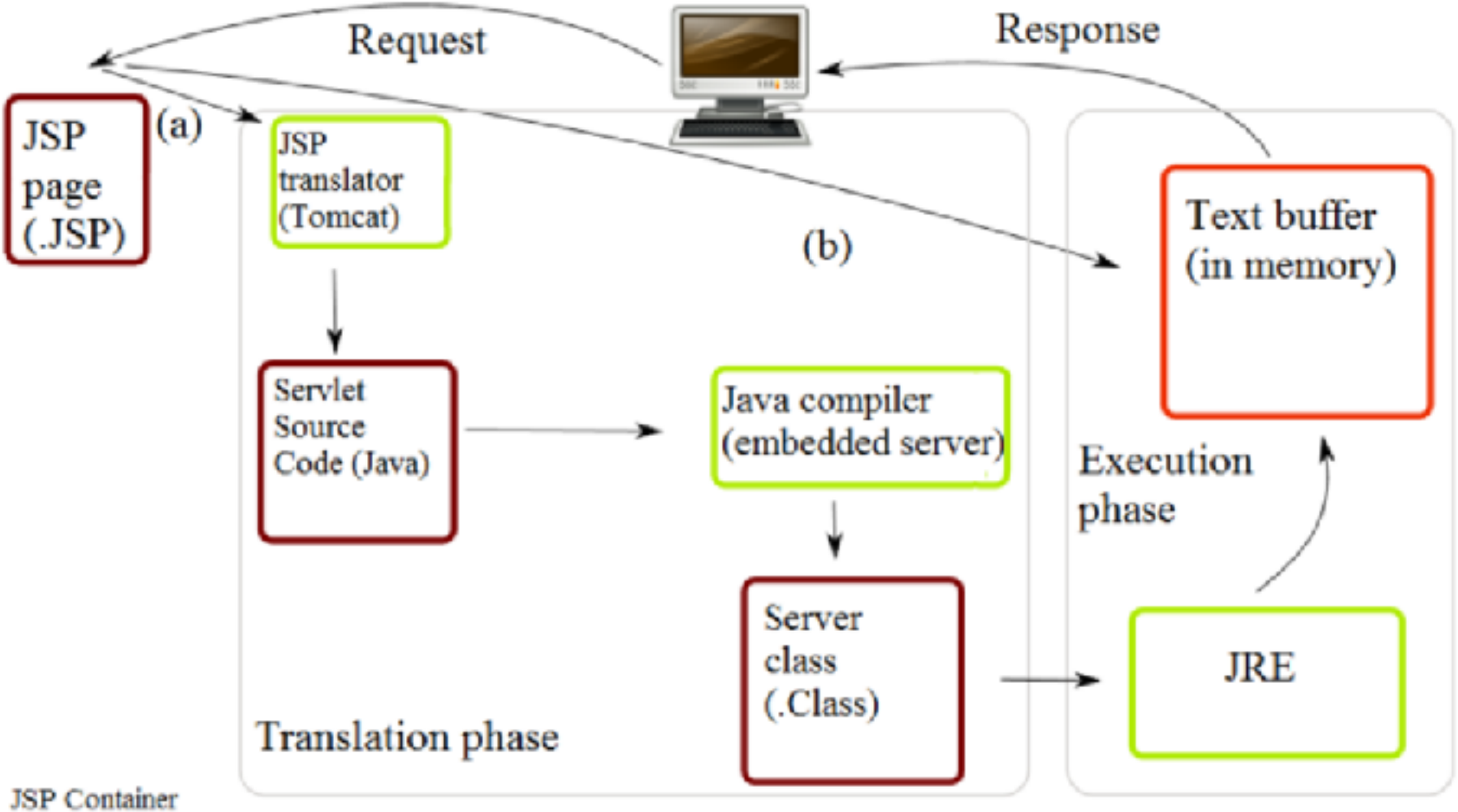
- 支持服务器端计算，扩展服务器端的能力，可理解为从客户端到服务器端的入口，接收客户端请求，处理后，返回结果到客户端。
  - 在Model-View-Controller（MVC）模式中，可当Controller的角色，主要职责是根据请求以控制页面流转、控制业务模块调度。

// Servlet怎么运作



容器接收到对请求后，会指派给某个Servlet处理，此时会先创建并初始化该Servlet的实例（即调用init方法），再调用service方法。随后对该Servlet的请求，都不会再调用init方法，而直接调用该Servlet实例（单例）的service方法。destroy方法在销毁该Servlet实例时用，如关闭服务器时。

Servlet怎么运作 - Servlet与JSP



JSP Container
(a) Translation occurs at this point, if JSP has been changed or is new.
(b) If not, translation is skipped.

怎么用Servlet - 配置Servlet

- 法1 web xml配置

```
<servlet>
    <servlet-name>helloServlet</servlet-name>
    <servlet-class>net.gupt.cs.jee.servlet.HelloServlet</servlet-class>
 </servlet>
 <servlet-mapping>
     <servlet-name>helloServlet</servlet-name>
    <url-pattern>/hello</url-pattern>
</servlet-mapping>
```

注意：url-pattern 必须以 "/" or "*."起头

- 法2 标注（annotation）

```
@WebServlet(name = "helloServlet", urlPatterns = { "/hello" })
public class HelloServlet extends HttpServlet {...}
```

怎么用Servlet - 定义Servlet类

```
public class HelloServlet extends HttpServlet {

protected void doGet(HttpServletRequest request,
                                    HttpServletResponse response)
                                  throws ServletException, IOException {...}

 protected void doPost(HttpServletRequest request,
                                     HttpServletResponse response)
                                   throws ServletException, IOException {...}
}
```

Filter的简介

- 什么是Filter （**W**hat）
- 为什么用Filter （**W**hy）
- Filter怎么运作、怎么用Filter （ho**W**）

什么是Filter

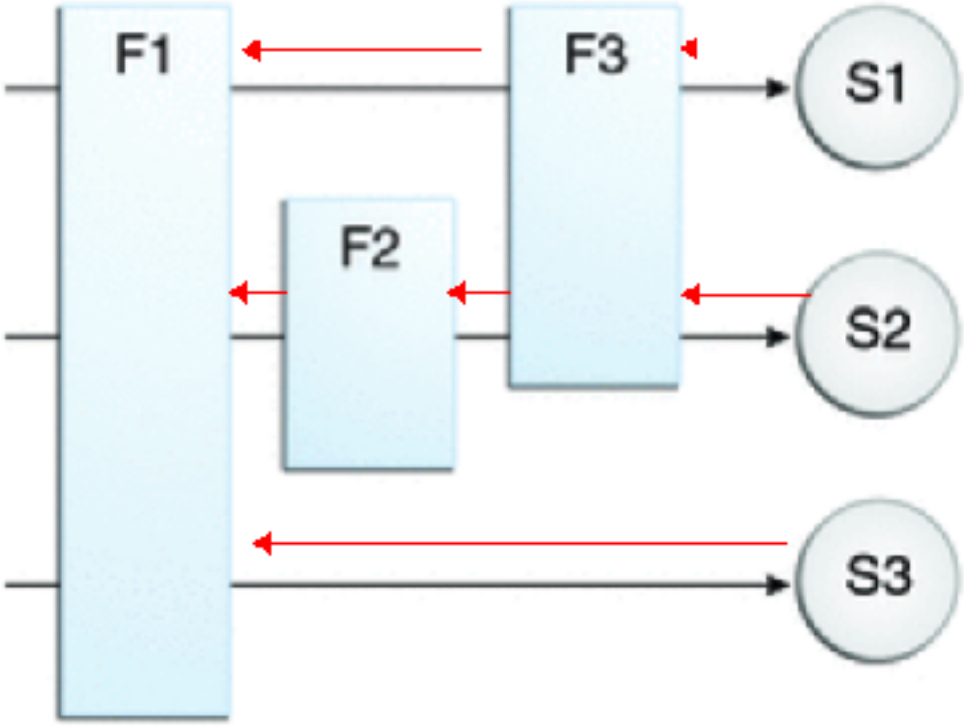
- 过滤器Filter，在Servlet2.3规范中引入，主要功能是拦截http请求，在该请求传递给目标资源（jsp、servlet或其他静态资源，如html）之前，或在结果返回给客户端之前，执行所需要的预处理。

// 为什么用Filter

- 过滤器Filter，其用途主要包括国际化时的编码设置、核对安全信息并将它转发给登录页面、记录系统日志（logging）、记录用户行为（tracing）、压缩下载数据、图像转化等。

Filter怎么运作

Figure 15-1 Filter-to-Servlet Mapping



F表示Filter，S表示Servlet。对S1的请求先要经过F1->F3这个Filter chain（过滤器链）的处理，S1处理完后在依照F3->F1而返回给客户端

怎么用Filter - 配置Filter

- 法1 web xml配置

```
<filter>
        <filter-name>myFaceFilter</filter-name>
        <filter-class>net.gupt.cs.jee.filter.MyFaceFilter</filter-class>
        </filter>
        <filter-mapping>
                <filter-name>myFaceFilter</filter-name>
                <url-pattern>/*</url-pattern>
        </filter-mapping>
```

注意：url-pattern 必须以 "/" or "*."起头

- 法2 标注（annotation）

```
@WebFilter(filterName = "myFaceFilter", urlPatterns = {"/*" })
public class MyFaceFilter implements Filter {...}
```

怎么用Filter - 定义Filter类

```
public class MyFaceServlet extends Filter {

    public void doFilter(ServletRequest request,
                                    ServletResponse response,
                                    FilterChain chain)
                throws IOException, ServletException {

        // 解决中文乱码
        request.setCharacterEncoding("UTF-8");

         // 调用下一个filter，若没有filter就会调用所请求的资源
         chain.doFilter(request, response);}
}
```

JSTL的简介

- 什么是JSTL （**W**hat）
- 为什么用JSTL（**W**hy）
- 怎么用JSTL（ho**W**）

什么是JSTL

- JSTL=JavaServer Pages Standard Tag Library

- 提供标签库，在JSP中封装通用功能，包括核心标签库、国际化标签库、SQL标签库、XML标签库、函数标签库

什么是JSTL

- 核心标签库：包括变量支持（如定义变量）、流程控制、URL管理及其他。
- 国际化标签库：包括地域设置（如中文支持）、输出地域性字符串、格式化地域性数字和日期等。
- SQL标签库：包括设置数据源、事务管理和数据操作（查找和更新）
- XML标签库：包括输出、解析和设置XML元素等
- 函数标签库：包括主要的字符串操作函数，类似length(), contains(), startsWith()等

为什么用JSTL

- 提供标准的标签库，提高软件质量（如可读性、可维护性）
- 尽量减少在JSP中使用嵌套的java代码，以更清晰地将呈现逻辑与其他逻辑分开
- 便于习惯tag的web页面开发人员使用
- 提高安全性，可避免一些XSS攻击（XSS=Cross Site Scripting）

怎么用JSTL

使用JSTL等自定义的标签库，需要在网页前面声明Tag标记，例如若要用JSTL的core标签，则需要加入以下标记。

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

// 怎么用JSTL

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<html>
<body>
        <c:set var="userName" value="zhang san" />
        <c:set value="16" var="age" />
        欢迎您,
        <c:out value="${userName}" /><hr>
 <!--从1到5依次取一个数字赋于变量i-->
        <c:forEach var="i" begin="1" end="5">
                ${i}  
        </c:forEach>
        <br>
         <c:if test="${age<18}">
              对不起，你不能访问这个网页
         </c:if><br>
 </body>
</html>
```

// 怎么用JSTL

JSTL核心标签库

1、表达式操作out、set、remove

(1) <c:out>：输出一个特定范围里面的属性。 类似jsp的out.println()。有两种方式：

Without a body
<c:out value="value" [escapeXml="{truelfalse}"]
[default="defaultValue"] />

或者With a body
<c:out value="value" [escapeXml="{truelfalse}"]>
default value
</c:out>

怎么用JSTL

JSTL核心标签库

(1) <c:out>

例1 <c:out value=“Hello JSTL”/>　　打印出Hello JSTL

例2 <jsp:useBean id="singer"
class="net.gupt.cs.jee.vo.Singer">
　　</jsp:useBean>
　　<jsp:setProperty name="singer" property="name" value="动力火车"/>

　　<c:out value="Hi ${singer.name}"/>

打印出Hi 动力火车

// 怎么用JSTL

(2) <c:set>设置某个特定对象的一个属性（默认scope=page）

<c:set value="value" var="varName" [scope=
"{pageIrequestIsessionIapplication}"]/ >

例1 <c:set value="Kaka" var="name"/>
给属性name赋值Kaka，并把它放入"page"箱子里（page scope）

例2<c:set value="Pippo" var="name" scope="session"/>
给变量name赋值Pippo，并存入session

// 怎么用JSTL

(2) <c:set>

例3 <c:set value="${name1}" var="name2" />
取出name1的值，赋给name2

例4：<c:set value="${param.name1}" var="name3" />
取出request中参数name1值，赋给name3

例5 <jsp:useBean id= "singer" class="gupt.Singer"></jsp:useBean>

<c:set target="${singer}" property="name" value="GEM" />
<c:set target="${singer}" property="rank" value="1" />

给名为singer的属性对应的对象赋值，该对象可以是JavaBean（即给其属性赋值），也可以是Map（即给其中key赋值）。

// 怎么用JSTL

(3) <c:remove>它的作用是删除某个变量或者属性

<c:remove var="varName"

[scope= "{pageIrequestIsessionIapplication}"]/ >

如<c:remove var="singer" scope="session"/>从session中删除属性名为singer的对象。

怎么用JSTL

JSTL核心标签库

2、流程控制<c:if>、<c:choose>、<c:when>、<c:otherwise>、<c:forEach>

怎么用JSTL

(1) <c:if>; 它用来做条件判断，相当于java中的if

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<html>
<body>
    <c:set var="score" value="81"/>
    <c:if test="${score>=90}">
      成绩优秀
    </c:if>
    <c:if test="${score>=80 && score<90}">
      成绩良好
    </c:if>
    <c:if test="${score>=60 && score<80}">
      成绩及格
    </c:if>
    <c:if test="${score<60}">
      成绩不及格
    </c:if>
  </body>
</html>
```

怎么用JSTL

(2) <c:choose><c:when><c:otherwise>; 它用来做条件判断，相当于Java中的switch case default

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<html>
<body>
    <c:set var="score" value="81"/>
    <c:choose>
      <c:when test="${score>=90}">成绩优秀</c:when>
      <c:when test="${score>=80 && score<90}">成绩良好</c:when>
      <c:when test="${score>=60 && score<80}">成绩及格</c:when>
      <c:otherwise>成绩不及格</c:otherwise>
     </c:choose>
  </body>
</html>
```

怎么用JSTL

（3）迭代操作forEach、forTokens

<c:forEach>是最常用的，几乎能够完成所有的迭代任务，类似于JSP脚本中的for(int i=j;i<k;i++)

基本语法：

```
<c:forEach [var="varName"]
items="collection"  [varStatus="varStatusName"]
          [begin="begin"] [end="end"] [step="step"]>
         Body 内容
 </c:forEach>
```

怎么用JSTL

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

<html>
  <body>
   固定次数的循环
   <c:forEach var="count" begin="50" end="60“ >
      <c:out value="${count}"/>
   </c:forEach>

  </body>
 </html>
```

怎么用JSTL

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%
 List <String> users =new ArrayList<String>();
            users.add("user1");
            users.add("user2");
     }
  pageContext.setAttribute(“userList”,users); // 把对象users放入page scope
   %>

   <table border="1">
   <c:forEach var="user" items="${userList}" >
      <tr>
         <td><c:out value="${user}"/></td>
      </tr>
   </c:forEach>
   </table>
```

怎么用JSTL

假设存在User类，其属性为name，方法为getName()和setName(String)

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%
 List <User> users =new ArrayList<User>();
            users.add(new User().setName("user1"));
            users.add(new User() .setName("user2"));
     }
     pageContext.setAttribute("userList",users);
  %>

  <table border="1">
  <c:forEach var="user" items="${userList}" >
      <tr>
        <td><c:out value="${user.name}"/></td>
      </tr>
  </c:forEach>
  </table>
```

EL的简介

- 什么是EL （**W**hat）
- 为什么用EL（**W**hy）
- 怎么用EL（ho**W**）

什么是EL

- EL= Expression Language （又叫Unified Expression Language） 表达式语言
  - 出现于JSTL1.0，最初是为了支持JSTL的使用，现在可以不依赖于JSTL来使用EL。

为什么用EL

- 使得web页面的作者可以更方便地访问各种数据对象。
  - 使用EL，web页面的作者可以减少页面内的脚本代码量，从而提高生产率，提高可维护性，并且易学习。

怎么用EL

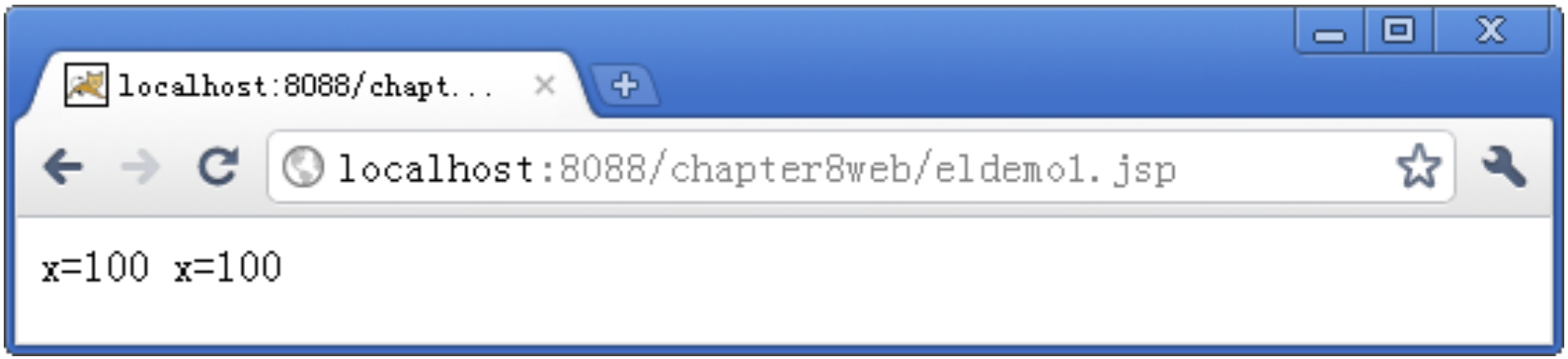
- 例1 ${singer.name}， 访问id为singer的JavaBean对象，并调用其getName方法取得name值
  - 例2 嵌套于JSTL，取得request中name参数值，并打印出来，相当于request.getParameter(“name”)

```
<c:out value="${param.name}" />
${param.name}
```

怎么用EL

```
<%
 int x=100 ;
pageContext.setAttribute("ax",x);
%>
                <!--使用JSP脚本输出-->
<%
out.println("x="+ pageContext.getAttribute("ax"));
%>
<!--使用EL表达式输出-->
```

x=${ax}



怎么用EL

1、输出某一个范围内的变量值

${name}。它的意思是读出某一范围中名称为name的变量。

因为我们并没有指定哪一个范围的name，所以它会依序从Page、Request、Session、Application范围查找。

我们要取得session中储存一个属性name的值，可以利用下列方法：

<%=session.getAttribute("name")%> 取得name的值，

在EL中则使用下列方法

${sessionScope.name}

sessionScope是EL中与范围有关的隐含对象。与范围有关的EL隐含对象包含以下四个：pageScope、requestScope、sessionScope 和applicationScope。

2、输出页面之间传的值

与输入有关的隐含对象有两个：param和paramValues，它们是EL中比较特别的隐含对象。

例如我们要取得用户的请求参数时，可以利用下列方法：

```
String name = request.getParameter(String name)
String[] names = request.getParameterValues(String name)
```

在EL中则可以使用param和paramValues两者来取得数据。

```
${param.name}
${paramValues.name}
```

怎么用EL

EL运算符

- 算术运算符有5个：+、-、*或$、/或div、%或mod
- 关系运算符有6个：==或eq、!=或ne、<或lt、>或gt、<=或le、>=或ge
- 逻辑运算符有3个：&&或and、||或or、!或not
- 其它运算符有3个：Empty运算符、条件运算符、()运算符

怎么用EL

EL输出JavaBean中属性值

1、输出JavaBean中的属性

```
<jsp:useBean id="singer" class="gupt.Singer"></jsp:useBean>

<jsp:setProperty name="singer" property="name" param="name"/>
<jsp:setProperty name="singer" property="rank" param="rank"/>

歌手: <jsp:getProperty name="singer" property="name" /><br />
排名: <jsp:getProperty name="singer" property="rank" /><br />
```

若用EL:

```
歌手 : ${singer.name}
    排名: ${singer.rank}
```

未完待续，谢谢！