1、 Web 应用概述

1. **URL 与 URI**

   URL: 统一资源定位器

   URI：统一资源定位符

2. 常见 **HTML** 标签
3. 表单的处理
4. 静态文档与动态文档的概念：静态文档是一种以文件的形式存放在服务器端的文档，客户发出对该文档的请求，服务器返回这个文档。动态文档是指文档的内容可根据需要动态生成，又可分为服务器端动态文档和客户端动态文档技术。
5. **Servlet** 概念：服务器端小程序，是使用 ServletAPI 以及相关类编写的 java 程序，主要用来扩展 web 服务器的功能。
6. 处理 **404** 错误：查看给定的路径名是否正确，查看 Servlet 类文件是否在 classes 目录下，查看 web.xml 文件内容是否正确，查看 tomcat 是否启动

2、 Servlet 技术模型

1. **Servlet** 的 **API**：**javax.servlet** 包，**javax.servlet.http** 包

   ❖ Servlet 接口及方法 ：

   1） public void init(ServletConfig config)： 完成 Servlet 初始化并准备提供服务。容器传给该方法一个 ServletConfig 类型的参数。

   2） public void service(ServletRequest req,ServletResponse res)throw ServletException,IOException：对每个客户请求容器调用一次该方法，它允许 Servlet 为请求提供响应。

   3） public void destroy()该方法由容器调用，指示 Servlet 清除本身，释放请求的资源并准备结束服务。

   4） public ServletConfig getServletConfig()返回关于 Servlet 的配置信息，如传递给 init()方法的参数。

   5） public String getServletInfo()返回关于 Servlet 的信息，如作者，版本及版权信息。

   ❖ ServleConfig 接口作用及方法

   ❖ HttpServlet 类 :新的 service 方法，doGet，doPost

   ❖ HttpServletRequest 接口及常用方法

   ❖ HttpServletResponse 接口及常用方法

2. **Servlet** 的开发步骤

   ❖ 编写一个 Servlet 类

   ❖ 编译

   ❖ 部署

   (1)在 tomcat 的虚拟目录下，创建 Web 目录

   (2)将编译好的.class 文件复制到 Web 目录的 WEB-INF\classes 目录

   (3)修改配置文件 WEB-INF\web.xml

   (4)启动 tomcat

   ❖ 运行

3. **Servlet** 的执行过程

1) 用户通过单击超链接或提交表单向容器请求访问 Servlet，容器分析这个请求，创建 request 和 response 两个对象
2) 容器根据请求的 URL 找到正确的 Servlet，为这个请求创建一个线程对象（每次请求都创建一个线程）
3) 容器调用 Servlet 的 service 方法，把请求和响应对象作为参数传递给该方法
4) 调用 Servlet 的 doGet()或 doPost()方法
5) 向客户发送响应：Servlet 使用相应对象获得输出流对象，调用有关方法将响应写给客户，响应通过容器发送给浏览器。

4. **Servlet** 的生命周期

加载实例化 Servlet——初始化 Servlet（init）——为客户提供服务（service）——销毁 Servlet（destroy）

5. 分析请求

http 请求结构：请求行——请求头——空行——数据

get 方法与 post 方法的对比

public String getParameter(String name)

public String[] getParameterValues(String name)

public Enumeration getParameterNames()

6. 请求转发

RequestDispatcher dispatcher=request.getRequestDispatcher(location);

Dispatcher. forward(request, response)

request.setAttribute

request.getAttribute

7. 发送响应

响应结构：状态行——响应头——空行——响应数据

response.setContentType(“text/html;charset=gb2312”);

PrintWriter out=response.getWriter();

响应重定向：response.sendRedirect(“URL”); return;

响应重定向与请求转发的比较：forward()方法转发请求是服务器端控制权的转向，客户端地址栏中不显示转发后的资源地址。sendRedirect()方法是服务器向浏览器发送一个特殊的响应，它使浏览器连接到新的位置，浏览器地址栏可看到地址的变化。使用重定向，资源不能位于 WEB-INF 目录下。

页面错误：

200 表示请求成功，404 表示页面没有找到，500 表示服务器内部错误

3、 Servlet 容器模型

1. **Web** 应用程序的部署

(1)在 tomcat 的虚拟目录下，创建 Web 目录

(2)将编译好的.class 文件复制到 Web 目录的 WEB-INF\classes 目录

(3)修改配置文件 WEB-INF\web.xml

(4)启动 tomcat

2. 理解 **Web** 应用程序的部署描述文件 **web.xml**

下面的代码展示了在部署描述文件中<servlet>元素的一个典型的使用：

```
<servlet>
    <servlet-name>helloServlet</servlet-name>    定义 Servlet 名称
    <servlet-class>
        com.myserver.HelloServlet  指定 Servlet 类的完整名称
    </servlet-class>
  <init-param>向 Servlet 传递的初始化参数，可以定义多个
      <param-name>email</param-name>
      <param-value>hacker@163.com</param-value>
  </init-param>
            <servlet-mapping>  定义一个映射
                  <servlet-name>HelloServlet</servlet-name>
             <url-pattern>/Hello</url-pattern>
        </servlet-mapping>
   <load-on-startup>2</load-on-startup>
</servlet>
```

3. **ServletConfig**

得到 ServletConfig 接口对象的方法 ：

```
ServletConfig    config = getServletConfig();
```

覆盖 Servlet 的 init 方法。

ServletConfig 接口共定义了下面 4 个方法：

public String getInitParameter(String name)：返回指定名称的初始化参数值（是从 DD 文件中取出）

public Enumeration getInitParameterNames ()

public String getServletName() ：返回 DD 文件中<servlet-name>的名称

public ServletContext getServletContext()：返回 Servlet 所在上下文对象

4. **ServletContext** 接口

- 使用 RequestDispatcher 实现请求转发
- ServletRequest 的 getRequestDispatcher()方法，可以传递一个相对路径，
- ServletContext 的 getRequestDispatcher()方法只能传递以“/”开头的路径。
- 通过 ServletContext 对象共享数据

4、 会话管理

1. 会话管理

- 理解会话的基本思想和管理机制
- 了解会话对象 HttpSession 及常见方法

调用 request.getSession 获取 HttpSession 对象：

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

将信息存入会话

```
public void setAttribute (String name, Object value)
```

查找与会话相关联的信息

```
public Object getAttribute(String name)
public Enumeration getAttributeNames()
public void removeAttribute(String name)
```

- 使用 HttpSession 对象通常需要三步：

（1）为客户创建或获得与请求关联的会话对象；

（2）在会话对象中添加或删除名/值对属性；

（3）如果需要可使会话失效。

2. 通过 **Cookie**（一小段文字信息）支持会话

- Cookie 类的常用方法： getName、 getValue、setValue、setMaxAge 和 getMaxAge；
- 向客户端发送 Cookie
  - 创建 Cookie 对象
    - **Cookie c = new** Cookie("username", "hacker");
  - 将 Cookie 放入到 HTTP 响应中
    - **response.addCookie(c)**
    - 从客户端读取 Cookie

```
Cookie[] cookies = request.getCookies();
 if (cookies!=null){
     for(int i = 0;i<cookies.length;i++){
        Cookie cookie = cookies[i];
        if(cookie.getName().equals(cookieName))
           cookieValue = cookie.getValue();
     }
 }
```

3. **URL** 重写与隐藏表单域

重写正在重定向的 URL

调用 encodeRedirectURL( )方法

```
    response.sendRedirect(
response.encodeRedirectURL(http://localhost/store/catalog)
```

在 HTML 页面中，可以使用下面代码实现隐藏的表单域：

```
<input type="hidden" name="session" value="a1234">
```

5、 JSP 技术模型

1. JSP 基本语法

<%@ %> JSP 指令：page 指令、include 指令和 taglib 指令

<%! %> JSP 声明：理解<%! int count = 0; %>和<% int count = 0; %>的区别

<% %> JSP 小脚本

<%= %> JSP 表达式

<jsp: > JSP 动作：jsp 标准动作、JSTL 中的动作、用户自定义动作

jsp 标准动作包括<jsp:forward>、<jsp:include>、<jsp:plugin>、<jsp:useBean> 、<jsp:getProperty>、<jsp:setProperty>

<%-- --%> JSP 注释

2. JSP 页面生命周期

❖ 理解 JSP 的执行过程

浏览器访问一个 jsp 页面，服务器端的 web 容器将 jsp 转换为 servlet，并处理执行此 servlet 中的相关内容，产生响应结果，然后再由 web 容器将响应结果返回客户端浏览器。

❖ 理解 JSP 页面转换

❖ 理解 JSP 页面的生命周期

页面转换——页面编译——加载类——创建实例——调用 jspInit()方法——调用_jspService()方法——调用 jspDestroy()方法

3. 理解 page 指令属性

❖ page 指令用于告诉容器关于 JSP 页面的全局属性，该指令适用于整个转换单元而不仅仅是它所声明的页面。

❖ 语法格式如下：

<%@ page

[language="java"]

[extends="package.class"]合法的实现了 javax.servlet.jsp.JspPage 接口的 java 类

[import="{package.class | package.*},..."] 导入在 jsp 中使用的 java 类和接口

[contentType="mimeType [;charset=characterSet]" |

"text/html; charset=ISO-8859-1"] 指定输出类型

[session="true | false"] 指定 jsp 是否参加 http 会话

[buffer="none | 8kb | sizekb"] 指定输出缓冲区的大小

[autoFlush="true | false"]

[isThreadSafe="true | false"]

[info="text"]

[errorPage="relativeURL"] 用相对 URL 指定另一个 jsp 页面来处理错误

[isErrorPage="true | false"]指定是否要用另一个 jsp 页面来处理错误

%>

4. JSP 隐含变量（内置对象）

❖ JSP 有以下九种内置对象，包括：

- **request**，请求对象（是 **HttpServletRequest** 的隐含变量）

- **session**，会话对象（是 **HttpSession** 的隐含变量）
- **application**，应用程序对象（是 **ServletContext** 的隐含变量）
- **page**，页面对象
- response，响应对象
- pageContext，页面上下文对象
- out，输出对象
- config，配置对象
- exception，例外对象

5. 作用域对象

应用作用域 **application**： **ServletContext** ： 在整个 **web** 应用程序有效

会话作用域 **session**： **HttpSession** ： 在一个用户会话范围内有效

请求作用域 **request: ServletRequest** ： 在用户的请求和转发的请求内有效

页面作用域 **page: PageContext** ： 只在当前页面内有效

6、 Web 组件重用与 JavaBeans

1. 静态包含和动态包含

❖ 静态包含是在 JSP 页面转换阶段将另一个文件的内容包含到当前 JSP 文件中产生一个 servlet。静态包含使用 include 指令，语法：

<%@ include file="relativeURL" %>

包含的文件在当前 Web 应用程序下，可以是任何基于文本的文件，如 HTML、JSP、XML 文件，甚至是简单的.txt 文件。

❖ 动态包含是通过 JSP 标准动作< jsp:include>实现的，它是在请求时将另一个页面的输出包含到主页面的输出中。

动态包含使用 jsp:include 标准动作，其语法如下所示：

<jsp:include page="relativeURL" flush="true" />

	<jsp:include>
<%@ include file=" "%>	**<jsp:include page=" " flush="true"/>**
修改了被包含的文件后，需更新源文件	修改了被包含的文件后，不需更新源文件
包含外部页面的过程在原 **JSP** 页面被编译成 **Servlet** 时进行	包含外部页面的过程在运行时进行

使 用

<jsp:forward>动作把请求转发到其他组件，然后由转发到的组件把响应发送给客户

❖ 该动作的格式为：

<jsp:forward page="relativeURL" />

- page 属性的值为转发到的组件的相对 URL，它可以使用请求时属性表达式。
- 它与<jsp:include>动作的不同之处在于，当转发到的页面处理完输出后，并不将控制转回主页面。
- 使用<jsp:forward>动作，主页面也不能包含任何输出。

2. JavaBeans 及序列化

- JavaBean 就是*使用* ***Java*** *语言定义的类*，而这种类的设计需要*遵循* ***Sun*** *制定的* ***JavaBean*** *规范*文档中描述的有关约定。
- 在 Java 模型中，通过 JavaBean 可以无限扩充 Java 程序的功能。
- JavaBean 的最大好处是可以*实现代码的重复利用*，另外在 JSP 页面中使用 JavaBean 可使代码更简洁，也易维护，也可充分利用面向对象语言的特性。
- 在 JSP 程序中常用 JavaBean 来封装业务逻辑、数据库操作等等，可以很好地*实现业务逻辑和表示逻辑的分离*。

3. Javabeans 规范

遵循下面 3 个规范的 Java 类作为 JavaBean

- 类必须是 public 的;类必须具有无参数的 public 构造方法，
- JavaBeans 类的成员变量一般称为属性（property）。对每个属性访问权限一般定义为 private 或 protected，而不是定义为 public 的。注意：属性名必须以小写字母开头。
- 对每个属性，一般定义两个 public 方法，它们分别称为访问方法（getXxx）和修改方法（setXxx），允许容器访问和修改 bean 的属性。

JavaBean 的主要特性

- 是一个 Java 类
- 有一个无参数的构造函数
- 不应该有公开的实例变量
- 对值的获取采用 getXxx 和 setXxx 方法来访问
- boolean 型属性，允许用 is 代替 get 和 set

4. 在 Servlet 中使用 JavaBeans

通过 JavaBeans 共享数据

```
// 创建实例并设置属性
CustomerBean customer = new CustomerBean();
customer.setCustName(request.getParameter("custName"));
customer.setEmail(request.getParameter("email"));
customer.setPhone(request.getParameter("phone"));
request.setAttribute("customer"customer);
getServletContext().getRequestDispatcher("/customer.jsp").forward(request,respo
nse);
```

5. 在 JSP 中使用 JavaBeans

- 使用<jsp:useBean>

- <jsp:useBean>动作通过五个属性来定制该动作的行为： id 、scope 、class 、type 、beanName
- <jsp:useBean id=”customer” class=”com.model.CustomerBean” scope="session" />

❖ 使用<jsp:setProperty>

- <%@ page import="com.model.CustomerBean" %>
- <jsp:useBean id="customer"
  class="com.model.CustomerBean" />
- <jsp:setProperty name="customer"
  property="custName" value=“zxm" />

❖ 使用<jsp:getProperty>

<jsp:getProperty name="customer" property="email" />

6. 理解 **MVC** 设计模式及其开发步骤

❖ 理解 MVC 设计模式

❖ Model—代表了数据对象，用 JavaBeans 实现

❖ View—用来在屏幕上显示数据对象的当前状态，是应用程序的外观，用 JSP 页面实现

❖ Controller—定义了用户接口对用户输入反应的方式，它处理数据对象，用 Servlet 实现

❖ 开发步骤

❖ 1. 定义 JavaBeans 表示数据

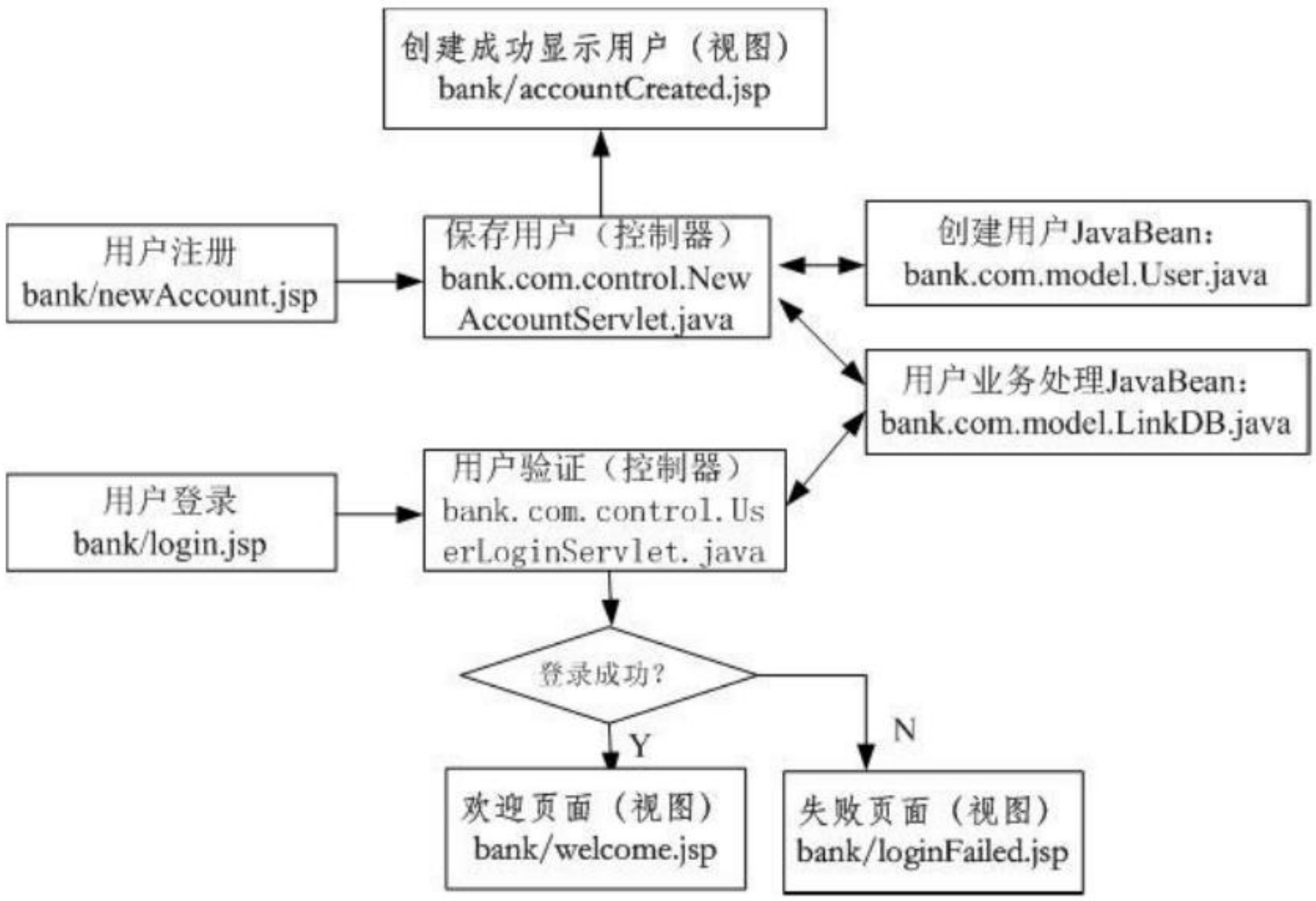
❖ 2. 使用 Servlet 处理请求

❖ 3.调用业务逻辑完成相应功能，填写 JavaBeans 对象数据

❖ 4.结果的存储

❖ 5. 转发请求到 JSP 页面

❖ 6. 从 JavaBeans 对象中提取数据



7、 JDBC 数据库访问

1. JDBC 数据库访问的一般步骤

1) 加载驱动程序：使用 Class 类的 forName() 静态方法，如 Class.forName("com.mysql.jdbc.Driver");

2) 建立连接对象：使用 DriverManager 类的 getConnection() ，如

```
String dburl = jdbc:mysql://localhost:3306/bank
Connection conn = DriverManager.getConnection(dburl, "root", "111111");
```

3) 创建语句对象：使用 Connection 接口的不同方法创建，如

```
Statement stmt = conn.createStatement();
```

4) 获得 SQL 语句的执行结果：对于查询语句，调用 executeQuery(String sql)方法,如：

```
String sql = "SELECT * FROM books" ;
ResultSet rst = stmt.executeQuery(sql) ;
while(rst.next()){
   out.print(rst.getString(1)+"\t") ;
}
```

对于语句如 CREATE、 ALTER、 DROP、 INSERT、 UPDATE、 DELETE 等须 executeUpdate(String sql)方法。

5) 关闭建立的对象： close()方法释放资源

2. JDBC API：Connection 接口、Statement 接口、ResultSet 接口

3. 理解 DAO 设计模式

❖ DAO（Data Access Object）称为数据访问对象。

❖ DAO 设计模式可以在使用数据库的应用程序中实现业务逻辑和数据访问逻辑分离，从而使应用的维护变得简单。

❖ 它通过将数据访问实现（通常使用 JDBC 技术）封装在 DAO 类中，提高应用程序的灵活性。

8、 自定义标签

1. 自定义标签的开发步骤

创建标签处理类（tag handler）；

创建标签库描述文件（TLD）；

在 JSP 文件中导入标签库和使用标签。

2. 理解 **TLD** 文件

```
<?xml version="1.0" encoding="UTF-8" ?>
<taglib version="2.0">
        <tlib-version>1.0</tlib-version>
        <uri>http://localhost/sampleLib</uri>
        <tag>
        <name>hello2</name>
        <tag-class>sampleLib.HelloTag2</tag-class>
        <body-content>empty</body-content>
        <description>Prints Hello user! </description>
        <attribute>
            <name>user</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
    </tag>
    </taglib>
```

3. 空标签、带属性标签和带标签体的开发

在 **JSP** 页面要使用自定义标签，需要通过**<taglib>**指令声明自定义标签的前缀和标签库的 **URI**，格式如下所示：

```
<%@ taglib prefix= "test"
  uri="http://localhost/sampleLib"%>
```

hello.jsp 的代码如下：

```
<%@ page contentType="text/html" %>
<%@ taglib prefix="test" uri="http://localhost/sampleLib"%>
<html><head>
<title>Using Custom Tag</title></head>
```

```
<body>
<<h3><test:hello2    /></h3>
<h3><test:hello2 user="john" /></h3>
<h3><test:hello2 user=
   '<%= request.getParameter("username") %>'/> </h3>
</body>
</html>
```

9、 Web 事件处理与过滤器

1. 监听并处理 **Servlet** 上下文事件的例子
   - 实现 ServletContextListener 接口： MyServletContextListener.java
   - 在 web.xml 添加以下代码注册事件监听器

```
<listener>
    <listener-class>
        event.MyServletContextListener
    </listener-class>
</listener>
```

   - listenerTest.jsp 页面对监听器进行测试：数据库连接对象的使用：

2. 理解 **Web** 应用的事件模型

3. 处理监听事件的开发

4. 过滤器的概念、作用和工作原理

❖ 过滤器是 Web 服务器上的组件，它们对客户和资源之间的请求和响应进行过滤。

❖ 过滤器的一些常见应用包括：

- 登录和审计过滤器
- 验证过滤器
- 图像转换过滤器
- 数据压缩过滤器
- 加密过滤器
- XSLT（eXtensible Stylesheet Language Transformation，可扩展样式表语言转换）过滤器

❖ 当容器接收到对某个资源的请求，它要检查是否有过滤器与之关联。如果有过滤器与该资源关联，容器将把该请求发送给过滤器，而不是直接发送给资源。在过滤器处理完请求后，它将做下面 3 件事：

- 产生响应并将其返回给客户；
- 如果有过滤器链，它将把（修改过或没有修改过）请求传递给下一个过滤器；
- 将请求传递给不同的资源。

❖ 当请求返回到客户时，它是以相反的方向经过同一组过滤器返回。过滤器

链中的每个过滤器够可能修改响应。

5. 开发过滤器的步骤
   编写代码，编译，部署和运行