

arena teaching system

Servlet和JSP（下）



Java企业应用及互联网
高级工程师培训课程

达内集团教学研发部 编著

目 录

Unit01	1
1. 状态管理-Cookie	2
1.1. 状态管理	2
1.1.1. 为什么需要状态管理	2
1.1.2. 什么是状态管理	2
1.1.3. 状态管理的两种常见模式	3
1.2. Cookie	3
1.2.1. 什么是 Cookie	3
1.2.2. Cookie 的原理	3
1.2.3. 如何创建 Cookie	4
1.2.4. 如何查询 Cookie	4
1.2.5. 如何修改 Cookie	4
1.2.6. Cookie 的生存时间	5
1.2.7. Cookie 编码	6
1.2.8. Cookie 解码	6
1.3. Cookie 的路径问题	6
1.3.1. 什么是 Cookie 的路径问题	6
1.3.2. 发送 Cookie 的条件	7
1.3.3. 如何设置 Cookie 的路径	7
1.3.4. Cookie 的限制	7
经典案例	8
1. 创建 Cookie	8
2. 查找 Cookie	13
3. 修改 Cookie	17
4. Cookie 的保存时间	20
5. Cookie 案例——中文编码	23
6. Cookie 案例——路径问题	27
课后作业	31
Unit02	32
1. 状态管理-Session	34
1.1. Session	34
1.1.1. 什么是 Session (会话)	34
1.1.2. Session 工作原理	34
1.1.3. 如何获得 Session	34
1.1.4. 如何使用 Session 绑定对象	35

1.1.5. 如何删除 Session 对象	35
1.1.6. Session 验证	36
1.2. Session 超时	36
1.2.1. 什么是 Session 超时	36
1.2.2. 如何修改 Session 的缺省时间限制	36
1.2.3. 浏览器禁用 Cookie 的后果	37
1.2.4. 什么是 URL 重写?	37
1.2.5. 如何实现 URL 重写?	38
1.2.6. Session 的优缺点	38
1.3. 验证码	38
1.3.1. 验证码的作用	38
1.3.2. 验证码的绘制	39
1.3.3. 验证码图片的绘制步骤	39
1.3.4. 验证码的验证流程	39
经典案例	40
1. 使用 Session 实现访问计数	40
2. 使用 Session 实现登录	43
3. Session 超时的设置	49
4. URL 重写	51
5. 验证码	53
6. 为登录添加验证码功能	56
课后作业	62
Unit03	63
1. 过滤器、监听器	65
1.1. 过滤器	65
1.1.1. 什么是过滤器	65
1.1.2. 如何编写过滤器	65
1.1.3. 编写一个 java 类实现 Filter 接口	65
1.1.4. 实现拦截处理逻辑	66
1.1.5. 将过滤器添加到 Web 应用中	66
1.1.6. 过滤器和 Web 应用一起打包部署	66
1.1.7. 过滤器的执行流程	67
1.1.8. 过滤器的优先级	67
1.1.9. 多个过滤器的执行流程	67
1.1.10. 过滤器的初始化参数	68
1.1.11. 初始化参数的配置	68
1.1.12. 读取初始化参数	68
1.1.13. 过滤器的优点	69

1.2. 监听器	69
1.2.1. 什么是监听器	69
1.2.2. 生命周期相关的事件	69
1.2.3. 绑定数据相关的事件	70
1.2.4. 如何编写监听器	70
1.2.5. 编写 Java 类	70
1.2.6. 实现处理逻辑	71
1.2.7. 注册监听器	71
1.2.8. 应用场景	71
经典案例	72
1. 过滤器——过滤敏感词汇	72
2. 过滤器——多个过滤器	77
3. 过滤器——过滤器的初始化参数	80
4. 监听器——统计在线人数	84
课后作业	89
Unit04	90
1. EL、JSTL	92
1.1. EL 表达式	92
1.1.1. 为什么需要 EL 表达式和 JSP 标签？	92
1.1.2. 什么是 EL 表达式	92
1.1.3. EL 表达式的作用	92
1.1.4. 使用 EL 表达式访问 Bean 属性	93
1.1.5. 方式一：\${对象名.属性名}	93
1.1.6. 方式二：\${对象名[“属性名”]}	94
1.1.7. 指定对象的查找范围	94
1.1.8. 使用 EL 表达式进行运算	95
1.1.9. 使用 EL 表达式进行运算（续）	95
1.1.10. 使用 EL 表达式获取请求参数值	95
1.2. JSTL	96
1.2.1. 什么是 JSTL	96
1.2.2. 如何使用 JSTL	96
1.2.3. 核心标签-if 标签	96
1.2.4. 核心标签-choose 标签	97
1.2.5. 核心标签-forEach 标签	97
1.2.6. 如何开发自定义标签	98
1.2.7. 标签的运行原理	98
1.2.8. JSTL 应用	98
经典案例	99

1. 访问 Bean 属性案例-1	99
2. 访问 Bean 属性案例-2	104
3. 使用 EL 表达式运算的案例	108
4. 使用 EL 表达式获取请求参数	110
5. if 标签案例	111
6. choose 标签案例	114
7. forEach 标签案例	116
8. 自定义标签案例	123
9. 员工管理——使用 EL 表达式和 JSTL 实现 JSP 页面	128
课后作业	135

Servlet和JSP(下)

Unit01

知识体系.....Page 2

状态管理-Cookie	状态管理	为什么需要状态管理
		什么是状态管理
		状态管理的两种常见模式
	Cookie	什么是 Cookie
		Cookie 的原理
		如何创建 Cookie
		如何查询 Cookie
		如何修改 Cookie
		Cookie 的生存时间
		Cookie 编码
		Cookie 解码
	Cookie 的路径问题	什么是 Cookie 的路径问题
		发送 Cookie 的条件
		如何设置 Cookie 的路径
		Cookie 的限制

经典案例.....Page 8


创建 Cookie	如何创建 Cookie
查找 Cookie	如何查询 Cookie
修改 Cookie	如何修改 Cookie
Cookie 的保存时间	Cookie 的生存时间
Cookie 案例——中文编码	Cookie 编码
	Cookie 解码
Cookie 案例——路径问题	什么是 Cookie 的路径问题
	发送 Cookie 的条件
	如何设置 Cookie 的路径

课后作业.....Page 31

1. 状态管理-Cookie


1.1. 状态管理

1.1.1. 【状态管理】为什么需要状态管理




为什么需要状态管理

- Web应用程序使用HTTP协议通信，而HTTP协议是“无状态”协议，即服务器一旦响应完客户的请求之后，就断开连接，而同一个客户的下一次请求将重新建立网络连接
- 服务器应用程序有时是需要判断是否为同一个客户发出的请求，比如客户的多次选购商品。因此，有必要跟踪同一个客户发出的一系列请求。





1.1.2. 【状态管理】什么是状态管理



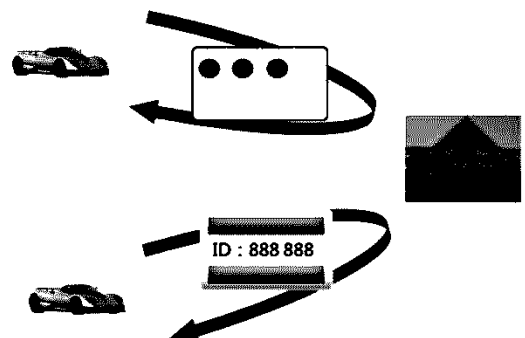
什么是状态管理


- 将客户端（浏览器）与服务器之间多次交互（一次请求，一次响应）当做一个整体来看待，并且将多次交互所涉及的数据即状态保存下来
- 状态指的是数据
- 管理指的是多次交互时对数据的修改





什么是状态管理（续）





1.1.3. 【状态管理】状态管理的两种常见模式

知识讲解

状态管理的两种常见模式

达内科技

- 客户端状态管理技术：将状态保存在客户端。代表性的是Cookie技术
- 服务器状态管理技术：将状态保存在服务器端。代表性的是Session技术

达内科技

+

1.2. Cookie

1.2.1. 【Cookie】什么是 Cookie

知识讲解

什么是Cookie

达内科技

- 浏览器向Web服务器发送请求时，服务器会将少量的数据以set-Cookie消息头的方式发送给浏览器，浏览器将这些数据保存下来；
- 当浏览器再次访问服务器时，会将这些数据以Cookie消息头的方式发送给服务器

达内科技

+

1.2.2. 【Cookie】Cookie 的原理

知识讲解

Cookie的原理

达内科技

保存在内存或硬盘

HTTP/1.1 200 ok
...
Set-Cookie : uname=xxx
...

GET /find HTTP/1.1
...
Cookie : uname=xxx
...

AddServlet

```
...
response.addCookie(c);
...
```

FindServlet

```
...
Cookie[] cs = request.getCookies();
...
```

Tomcat

+

1.2.3. 【Cookie】如何创建 Cookie

如何创建Cookie

- Servlet API为使用Cookie提供了javax.servlet.http.Cookie
- 创建：

```
Cookie c = new Cookie(String name , String value) ;  
response . addCookie( c ) ;
```

- name：用于区分不同Cookie的名字
- value：Cookie的值

1.2.4. 【Cookie】如何查询 Cookie

如何查询Cookie

- 获取客户端的所有Cookie对象：

```
Cookie[ ] request . getCookies( ) ;
```

注：该方法有可能返回null

- 获取一个Cookie对象的名称或值：

```
String Cookie . getName ( ) ;  
String Cookie . getValue ( ) ;
```

1.2.5. 【Cookie】如何修改 Cookie

如何修改Cookie

- step1，获取客户端发送的所有Cookie
- step2，根据name找到要修改的Cookie
- step3，调用Cookie的setValue (String newValue) 方法修改该Cookie的值
- step4，将修改后的Cookie加入到response发送回客户端

代码讲解

如何修改Cookie (续)

```

Cookie[ ] Cookies = request.getCookies( );
if(Cookies != null ){
    for(Cookie c : Cookies){
        String name = c.getName( ) ;
        if(name.equals( "city" )){
            c.setValue( "ShangHai" );
            response.addCookie( c );
        }
    }
}
                    
```

同名Cookie会覆盖，以达到修改的目的

1.2.6. 【Cookie】Cookie 的生存时间

代码讲解

Cookie的生存时间

- 默认情况下，浏览器会将Cookie保存在内存中，只要浏览器不关闭，Cookie就一直存在
- 如果希望关闭浏览器后Cookie仍在，可以通过设置过期时间

```
void Cookie . setMaxAge ( int seconds );
```

注：seconds 单位是秒，精度不是很高


代码讲解

Cookie的生存时间 (续)

- seconds > 0 : 浏览器要保存Cookie的最长时间为设置的参数值，如果超过指定的时间，浏览器 会删除这个Cookie。此时Cookie保存在硬盘上
- seconds = 0 : 删除Cookie。在修改Cookie的生存时间为0后，随着response发送回客户端，替换原有Cookie，因生命周期到了即将该Cookie删除。
- seconds < 0 : 缺省值，浏览器会将Cookie保存到内存中

5


1.2.7. 【Cookie】Cookie 编码



Cookie编码


- Cookie只能保存合法的ASCII字符。如果要保存中文，需要将中文转换成合法的ASCII字符，即编码。

```
Cookie c = new Cookie (
    "city",
    URLEncoder.encode( "北京", "utf-8" ) );
```



知识讲解

1.2.8. 【Cookie】Cookie 解码




Cookie解码

- 编码后的Cookie为了看到实际的中文，需要还原后再显示

```
Cookie[ ] Cookies = request.getCookies( );
if(Cookies != null){
    Cookie c = Cookies[ 0 ];
    String value = c.getValue( );
    value = URLDecoder.decode(value, "utf-8");
}
```


与编码的格式保持一致



知识讲解


1.3. Cookie 的路径问题

1.3.1. 【Cookie 的路径问题】什么是 Cookie 的路径问题





什么是Cookie的路径问题

- 浏览器在访问服务器上的某个地址时，会比较Cookie的路径与该路径是否匹配，只有匹配的Cookie才会发送给服务器
- Cookie的默认路径等于添加这个Cookie的Web组件的路径
- 如：/appName/file/addCookie.jsp添加了一个Cookie，则该Cookie的路径等于 /appName/file





知识讲解



1.3.2. 【Cookie 的路径问题】发送 Cookie 的条件

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4>发送Cookie的条件</h4> <ul style="list-style-type: none"> • 要访问的地址必须是Cookie的路径或者其子路径时，浏览器才会发送Cookie • 如： <ul style="list-style-type: none"> – Cookie的路径是 /appName/file – 则访问/appName/file/a.jsp 或 /appName/file/b.jsp时会发送Cookie – 如果访问 /appName/c.jsp则不会发送Cookie <div style="text-align: right;">  </div> <div style="text-align: right;">+</div>
---	--

1.3.3. 【Cookie 的路径问题】如何设置 Cookie 的路径

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4>如何设置Cookie的路径</h4> <ul style="list-style-type: none"> • 使用如下代码段可以设置Cookie的路径 <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <pre>Cookie c = new Cookie("uname" , "Jack"); c.setPath ("/appName"); response.addCookie (c);</pre> </div> <div style="text-align: right;">  </div> <div style="text-align: right;">+</div>
---	---

1.3.4. 【Cookie 的路径问题】Cookie 的限制

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4>Cookie的限制</h4> <ul style="list-style-type: none"> • Cookie可以被用户禁止 • Cookie会将状态保存在浏览器端，不安全。对于敏感数据，需要加密后再使用Cookie来保存 • Cookie只能保存少量的数据，大约4kb左右 • Cookie的个数是有限制的 • Cookie只能保存字符串 <div style="text-align: right;">  </div> <div style="text-align: right;">+</div>
---	--

经典案例

1. 创建 Cookie

- 问题

创建多个 Cookie，并使用浏览器内置功能实现查看。

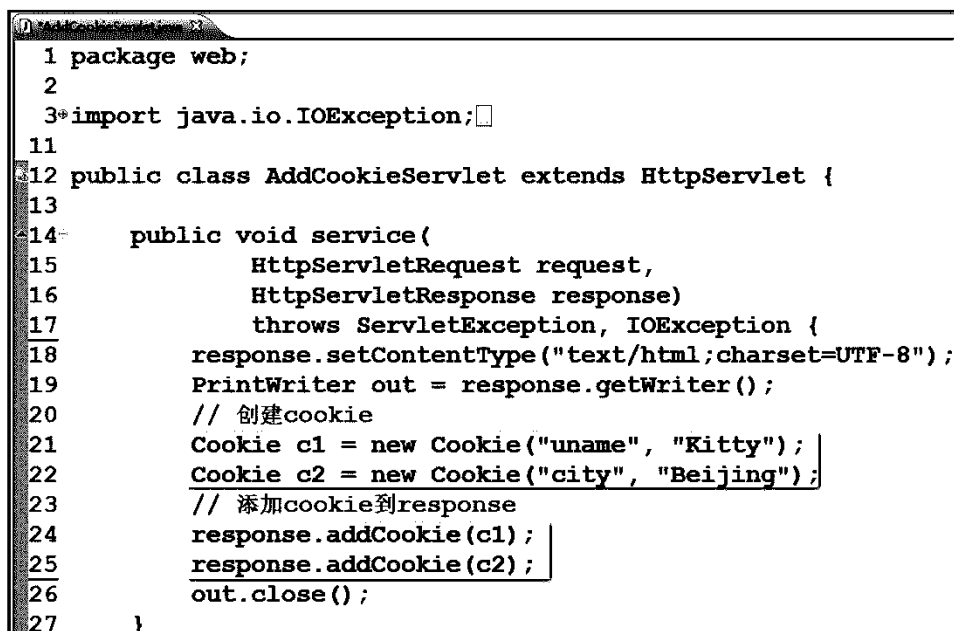
- 方案

创建 Cookie 对象，添加到响应对象 response 后返回给客户端。

- 步骤

步骤一：新建 AddCookieServlet

新建工程，在 src 下右键，新建 Servlet，点击 Next 以后，修改 url-pattern 为 addCookie，确定。



```
1 package web;
2
3 import java.io.IOException;
4
11
12 public class AddCookieServlet extends HttpServlet {
13
14     public void service(
15         HttpServletRequest request,
16         HttpServletResponse response)
17         throws ServletException, IOException {
18         response.setContentType("text/html;charset=UTF-8");
19         PrintWriter out = response.getWriter();
20         // 创建cookie
21         Cookie c1 = new Cookie("uname", "Kitty");
22         Cookie c2 = new Cookie("city", "Beijing");
23         // 添加cookie到response
24         response.addCookie(c1);
25         response.addCookie(c2);
26         out.close();
27     }
```

图 - 1

步骤二：部署应用，启动服务

部署应用，假定本次部署应用名称为 day06。启动 Tomcat，准备访问 AddCookieServlet.java。

步骤三：打开 Chrome 浏览器清空已有 Cookie

打开 Chrome 浏览器，点击菜单中设置，选择开发者工具。

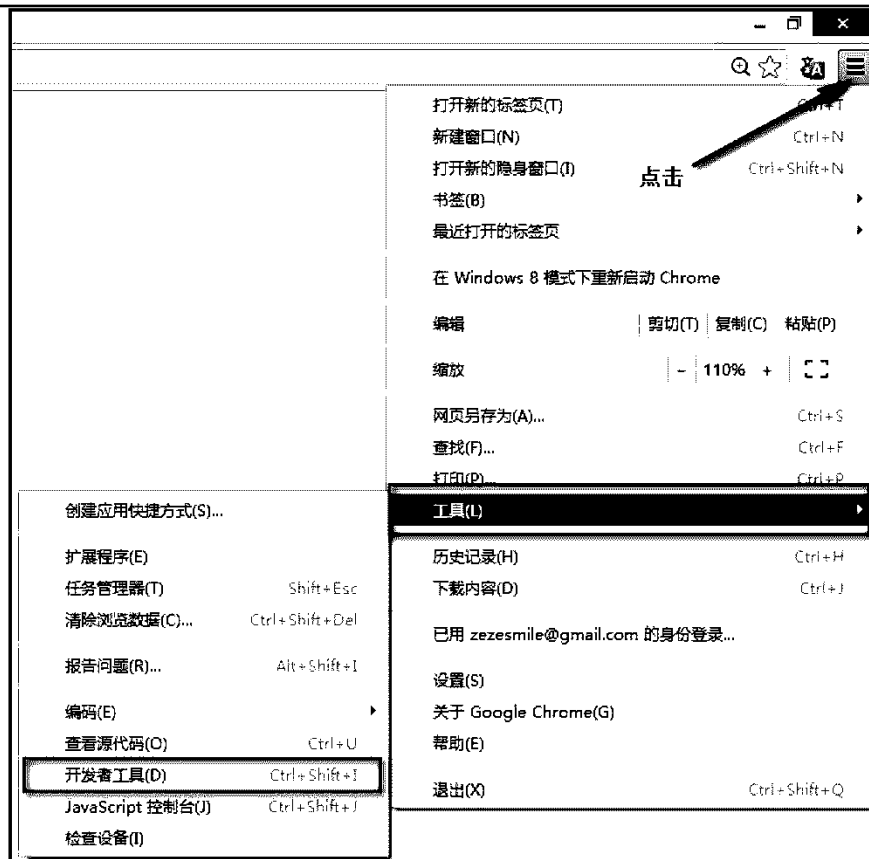


图 - 2

在打开的工具控制台中，选择 Resources 后，选择左边的 Cookie→Localhost→右键→Clear，代表清除浏览器当前已有的 Cookie，清除后，选中 localhost，右边的窗口内不再有 Cookie 列表。

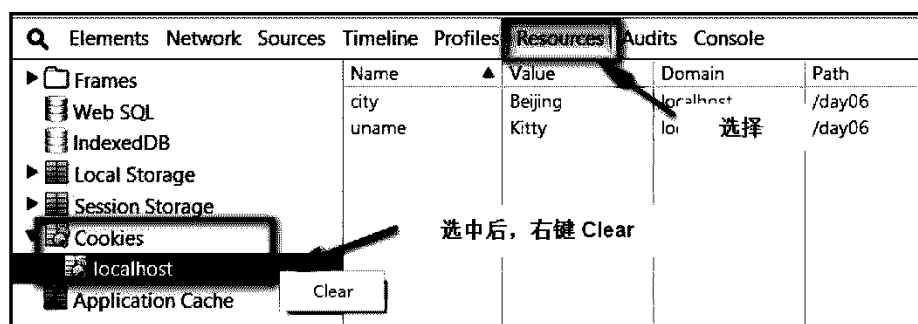


图 - 3

步骤四：访问 addCookie，查看请求及响应数据包

打开 Chrome 开发者工具窗口中的 Network，准备抓取请求响应数据包。

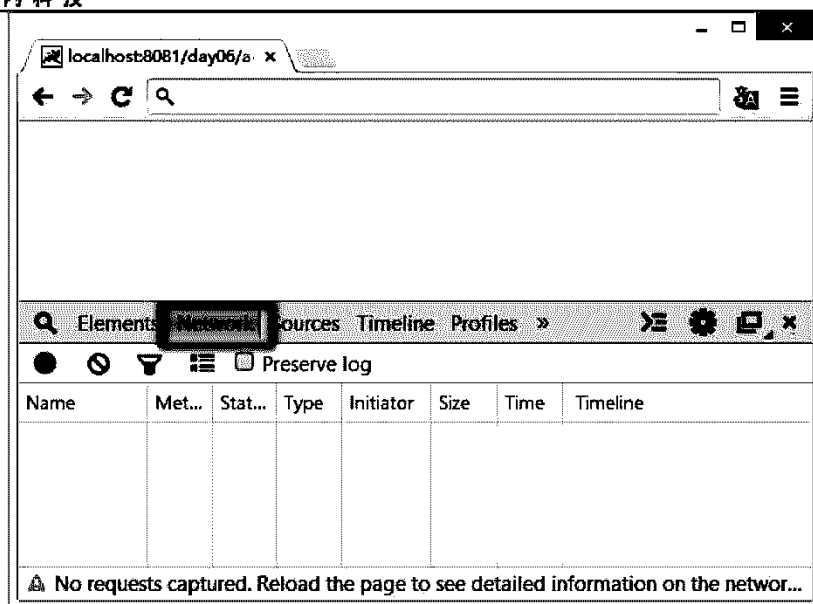


图 - 4

在浏览器地址栏中输入 `http://localhost:8080/day06/addCookie` 地址后确定，可以看见 Network 下面抓取了刚才的请求及响应数据包。点击 “addCookie” 这个名字后可以查看具体的包内数据。

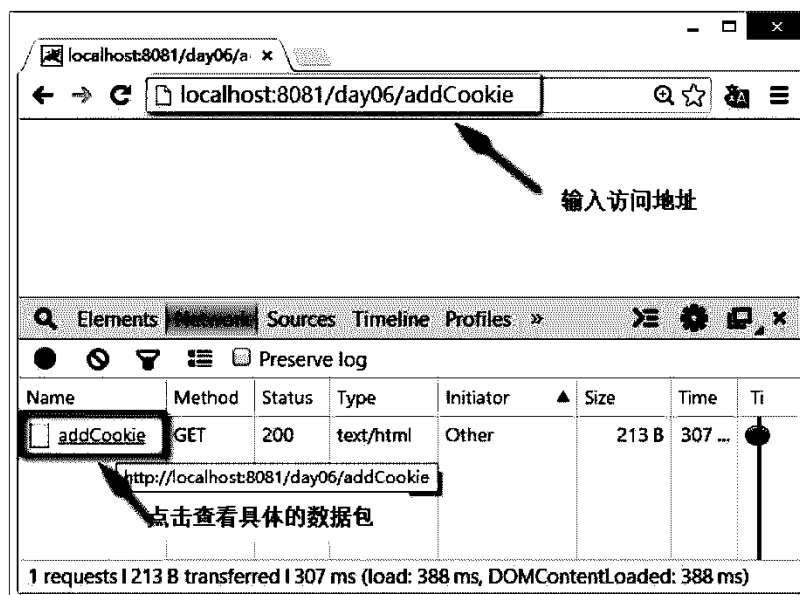


图 - 5

从响应数据包中可以看见以 Set-Cookie 头信息形式发回的 cookie 信息。如图-6 所示：

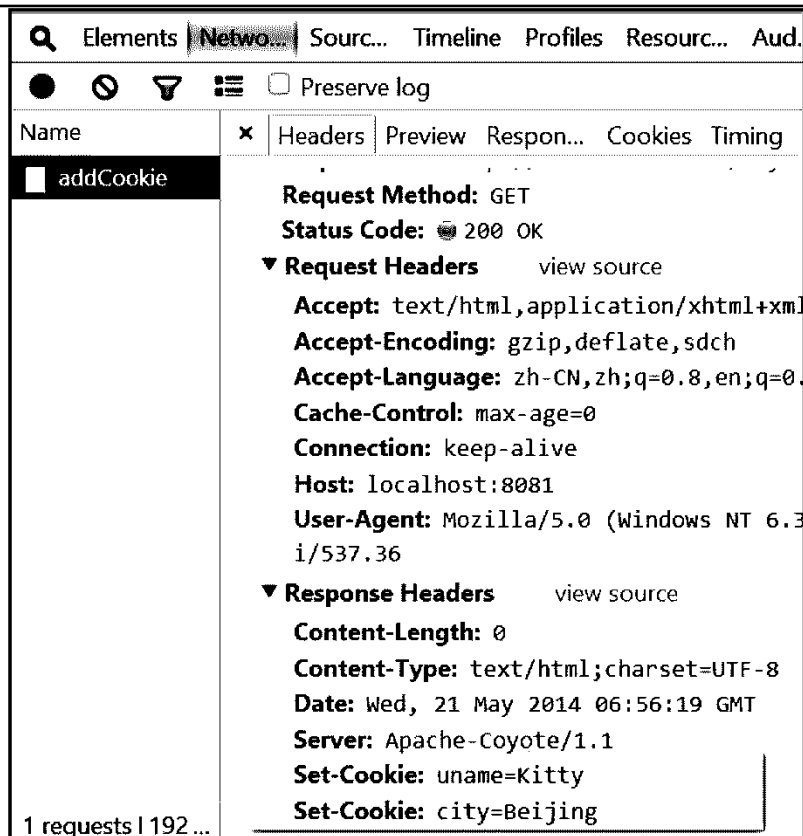


图 - 6

再次查看 Resources 下面的 cookie 信息, 会发现从空白增加为两对 Cookie 信息。如图-7 所示：

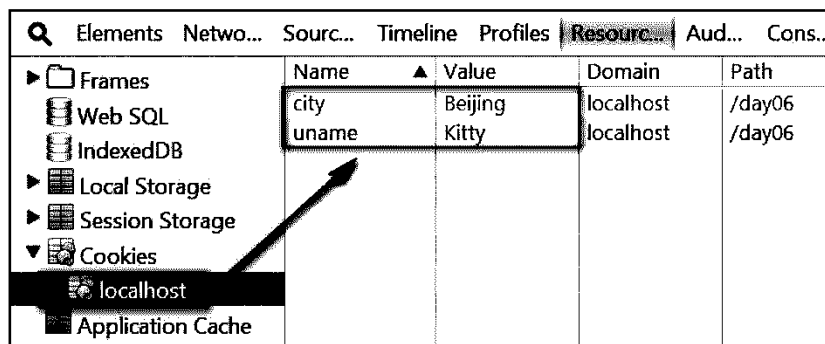


图 - 7

步骤五：再次刷新 addCookie，查看请求及响应数据包

再次回到 Network 区域，然后刷新页面，完成第二次对 addCookie 的访问，会发现在请求数据包中会自动发送存储的 Cookie 到服务器端，由于请求的 Servlet 中会执行 response.addCookie 这样的语句，所以在响应数据包中还是会以 Set-Cookie 的头信息发回两组添加到 response 中的信息。如图-8 所示：

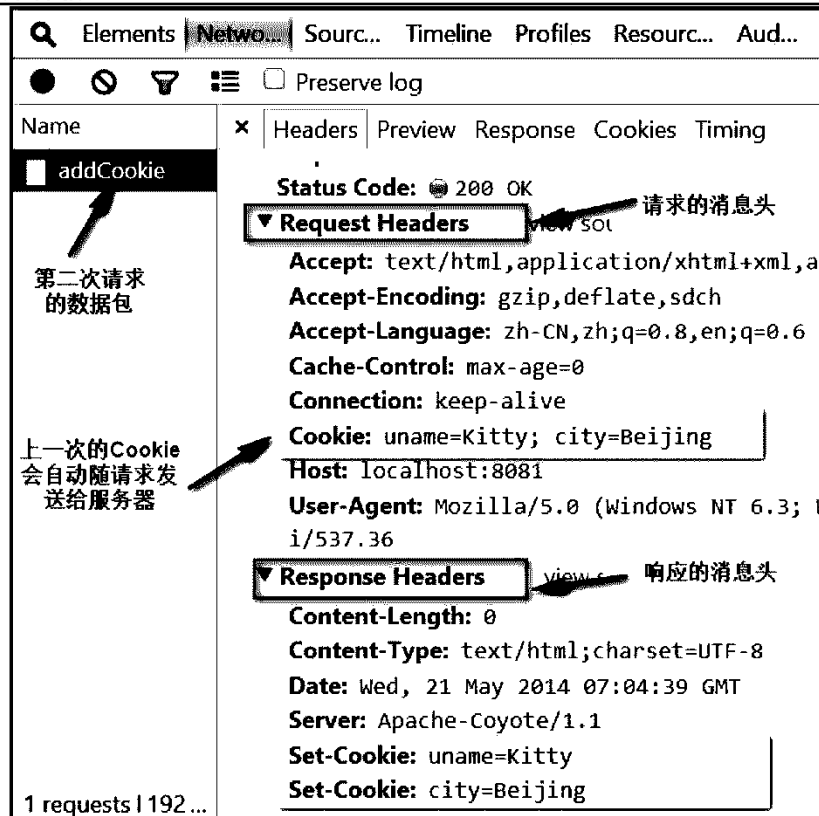


图 - 8

步骤六：关掉浏览器后再次打开，查看 Cookie

关闭浏览器，然后再次打开浏览器，不访问任何地址，查看 Resource 下面的 Cookie 节点，会发现刚刚通过访问 localhost 产生的两组 Cookie 信息已经不存在了，说明刚刚单独 Cookie 是保存在浏览器的内存中的，随着浏览器的关闭会被自动销毁掉。

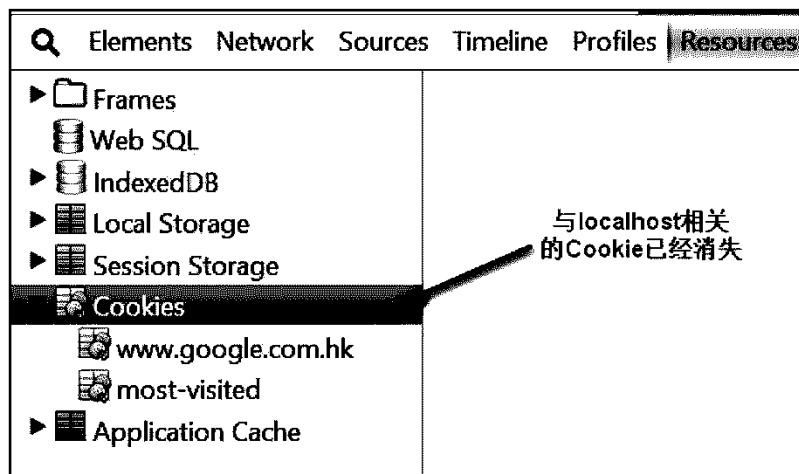


图 - 9

• 完整代码

AddCookieServlet.java 文件的代码如下：

```
package web;

import java.io.IOException;
import java.io.PrintWriter;

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

public class AddCookieServlet extends HttpServlet {
    public void service(
        HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        // 创建 cookie
        Cookie c1 = new Cookie("uname", "Kitty");
        Cookie c2 = new Cookie("city", "Beijing");
        // 添加 cookie 到 response
        response.addCookie(c1);
        response.addCookie(c2);
        out.close();
    }
}
```

web.xml 文件代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app 2.5.xsd">
    <display-name></display-name>
    <servlet>
        <servlet-name>AddCookieServlet</servlet-name>
        <servlet-class>web.AddCookieServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>AddCookieServlet</servlet-name>
        <url-pattern>/addCookie</url-pattern>
    </servlet-mapping>
</web-app>
```

2. 查找 Cookie

- 问题

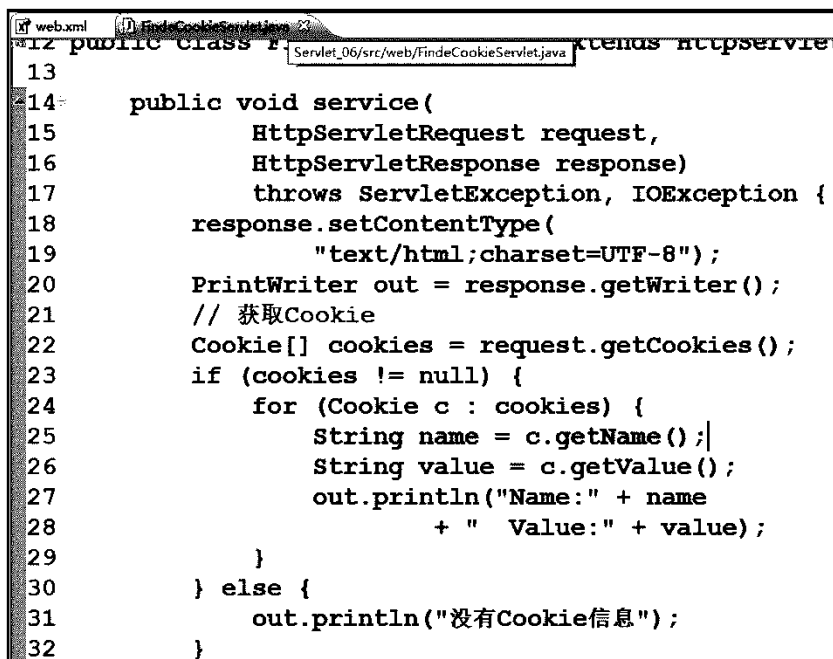
输出请求中发送的 Cookie 信息，如果没有 Cookie 则输出“无 Cookie 信息”。

- 方案

通过 request.getCookies() 方法获取 Cookie，判断是否为空，空代表没有 Cookie，非空则遍历数组，通过 getName 和 getValue 方法获取 cookie 的名称和值，并输出。

- 步骤

步骤一：创建 FindCookieServlet 类



```
13
14 public class FindCookieServlet extends HttpServlet {
15     public void service(
16         HttpServletRequest request,
17         HttpServletResponse response)
18         throws ServletException, IOException {
19         response.setContentType(
20             "text/html;charset=UTF-8");
21         PrintWriter out = response.getWriter();
22         // 获取Cookie
23         Cookie[] cookies = request.getCookies();
24         if (cookies != null) {
25             for (Cookie c : cookies) {
26                 String name = c.getName();
27                 String value = c.getValue();
28                 out.println("Name:" + name
29                     + " Value:" + value);
30             }
31         } else {
32             out.println("没有Cookie信息");
33         }
34     }
35 }
```

图 - 10

步骤二：先访问 addCookie，保证浏览器中有 Cookie

打开浏览器，先访问 addCookie，保证浏览器中有与 localhost 域名相关的 Cookie。
通过查看 Resources 内的 Cookie 节点可以确定是否存在。

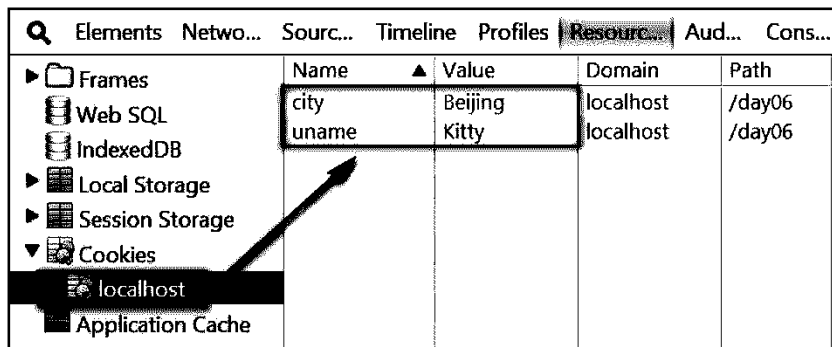


图 - 11

步骤三：再访问 findCookie，查看输出

在浏览器内输入 `http://localhost:8080/day06/findCookie`，查看页面输出。

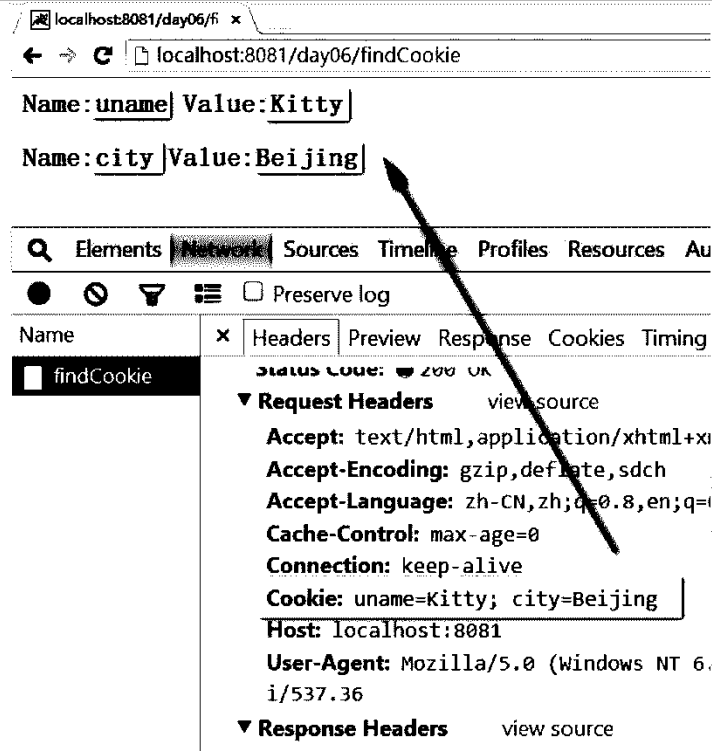


图 - 12

步骤四：重新打开浏览器，直接访问 findCookie，查看输出

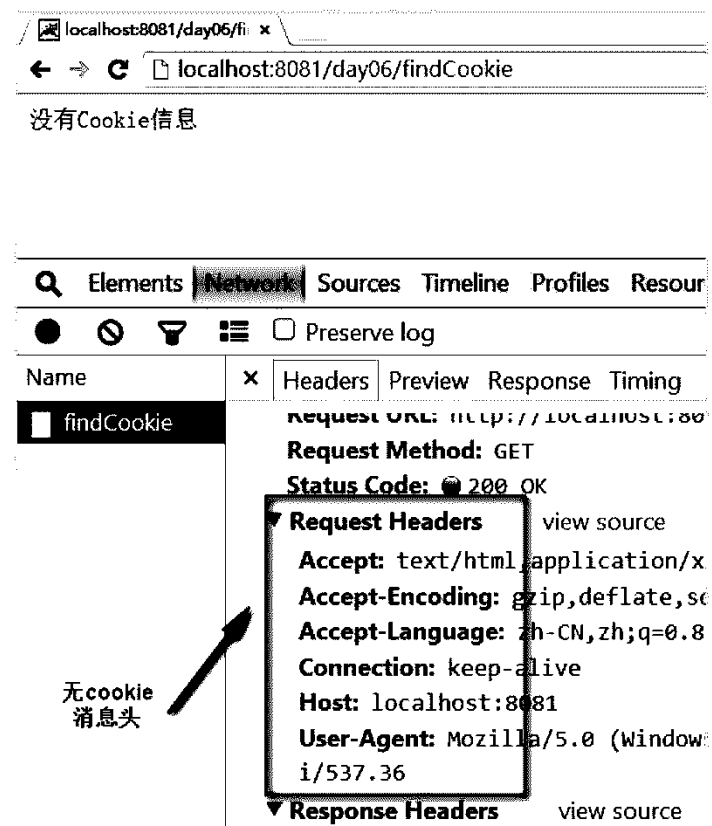


图 - 13

- **完整代码**

FindCookieServlet.java 文件代码如下：

```
package web;

import java.io.IOException;
import java.io.PrintWriter;

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

public class FindCookieServlet extends HttpServlet {

    public void service(
        HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType(
            "text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        // 获取Cookie
        Cookie[] cookies = request.getCookies();
        if (cookies != null) {
            for (Cookie c : cookies) {
                String name = c.getName();
                String value = c.getValue();
                out.println("<h3>Name:" + name
                    + " Value:" + value+"</h3>");
            }
        } else {
            out.println("没有Cookie信息");
        }
        out.close();
    }
}
```

web.xml 文件代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app 2.5.xsd">
    <display-name></display-name>
    <servlet>
        <servlet-name>AddCookieServlet</servlet-name>
        <servlet-class>web.AddCookieServlet</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>FindCookieServlet</servlet-name>
        <servlet-class>web.FindCookieServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>AddCookieServlet</servlet-name>
        <url-pattern>/addCookie</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
```

```
<servlet-name>FindCookieServlet</servlet-name>
<url-pattern>/findCookie</url-pattern>
</servlet-mapping>
</web-app>
```

3. 修改 Cookie

- 问题

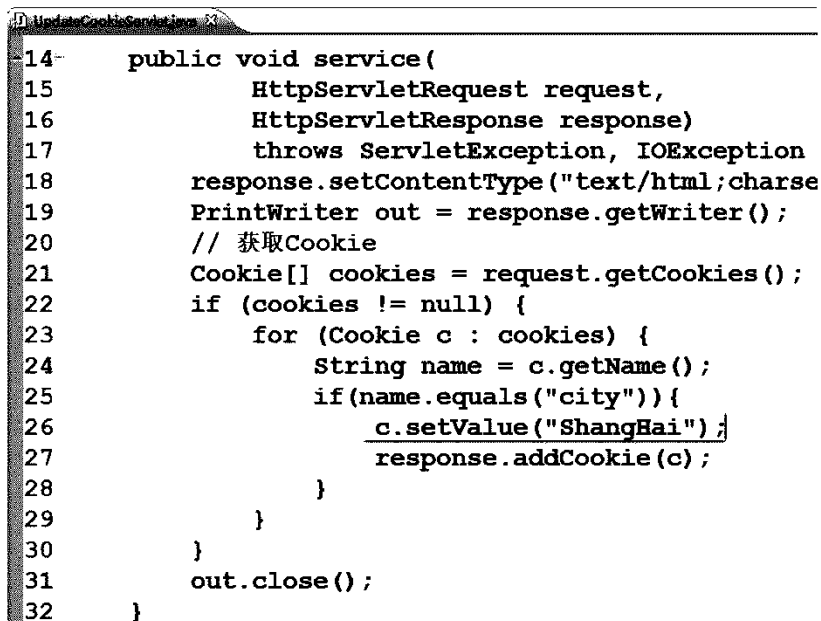
修改已有 cookie 的值。

- 方案

从 request 中获取 cookie 后，根据名字查找指定 cookie，使用 setValue 方法覆盖原有值，再将 cookie 添加到 response 中。同名 cookie 到达浏览器以后会覆盖以前的同名 cookie，效果等同为修改。

- 步骤

步骤一：新建 UpdateCookieServlet 类



```
14- public void service(
15-     HttpServletRequest request,
16-     HttpServletResponse response)
17-     throws ServletException, IOException
18- {
19-     response.setContentType("text/html;chase
20-     PrintWriter out = response.getWriter();
21-     // 获取Cookie
22-     Cookie[] cookies = request.getCookies();
23-     if (cookies != null) {
24-         for (Cookie c : cookies) {
25-             String name = c.getName();
26-             if (name.equals("city")) {
27-                 c.setValue("ShangHai");
28-                 response.addCookie(c);
29-             }
30-         }
31-     }
32-     out.close();
33- }
```

图 - 14

步骤二：先访问 addServlet，查看访问后浏览器中的 Cookie

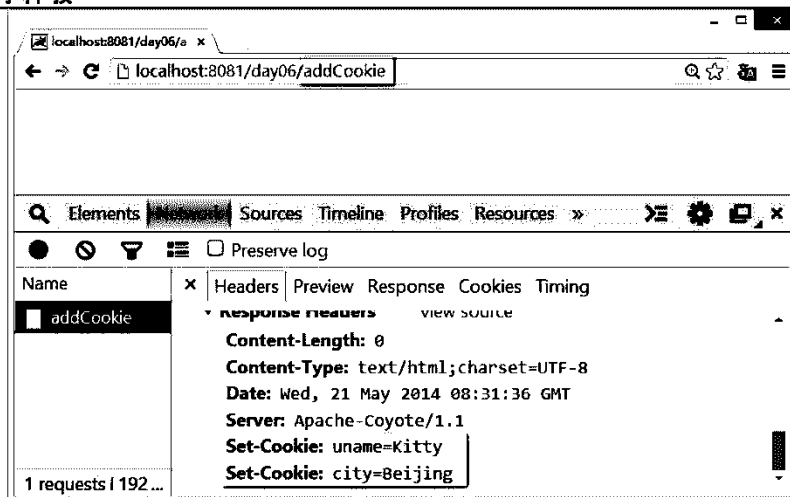


图 - 15

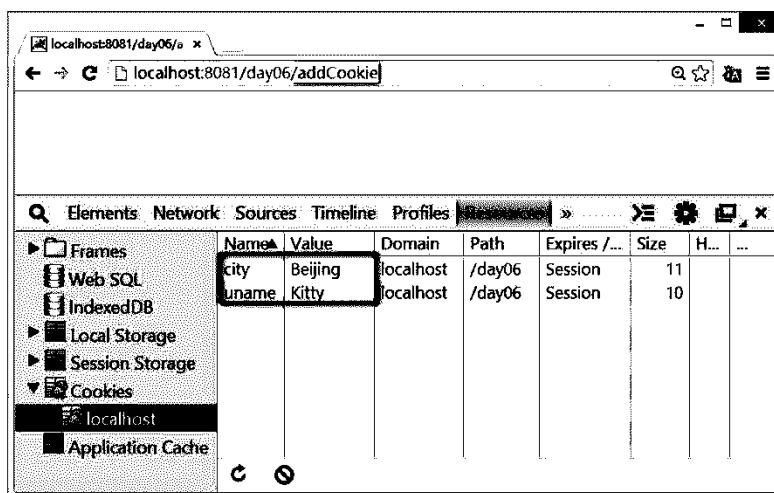


图 - 16

步骤三：再访问 updateServlet，查看访问后浏览器中的 Cookie

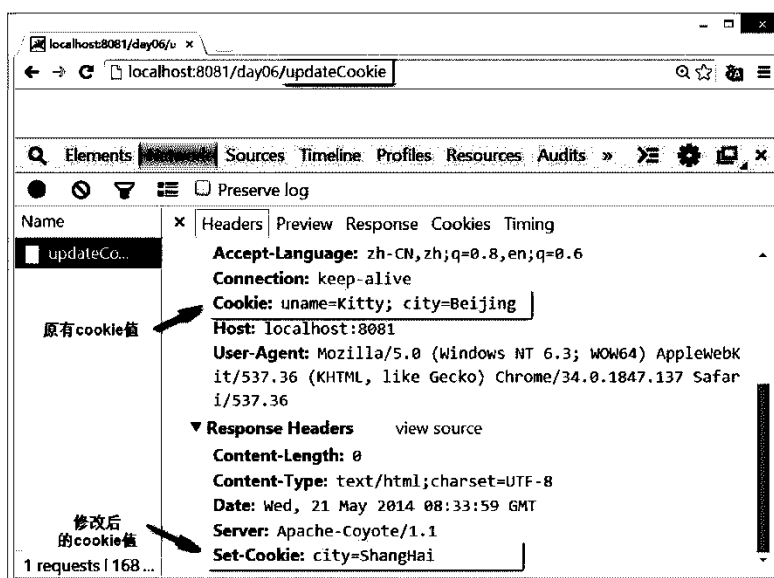


图 - 17

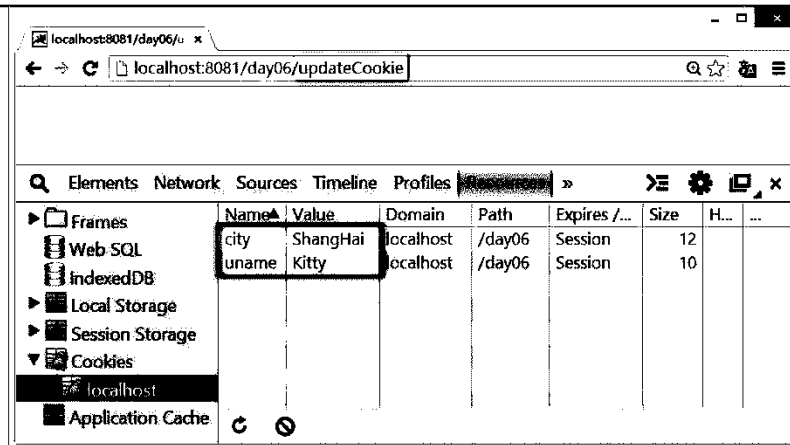


图 - 18

修改 Cookie 的流程：

第一次访问 addServlet 时，在响应头信息中使用 Set-Cookie 将 cookie 传送到客户端，浏览器在内存中保存了 Cookie 信息。接下来访问 updateServlet 时，在请求头信息中，使用 cookie 将内存中的 Cookie 信息发送到了服务器端，当服务器端执行到 response.add(c) 方法时，就将一对有新数据的 cookie 通过响应的 Set-Cookie 头发送回了客户端，于是同名 Cookie 覆盖原有信息，实现修改功能。

• 完整代码

UpdateCookieServlet.java 文件代码：

```
package web;

import java.io.IOException;
import java.io.PrintWriter;

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

public class UpdateCookieServlet extends HttpServlet {
    public void service(
        HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        // 获取 cookie
        Cookie[] cookies = request.getCookies();
        if (cookies != null) {
            for (Cookie c : cookies) {
                String name = c.getName();
                if (name.equals("city")) {
                    c.setValue("ShangHai");
                    response.addCookie(c);
                }
            }
        }
    }
}
```



```
        out.close();  
    }  
}
```

web.xml 文件代码：

```
<?xml version="1.0" encoding="UTF-8"?>  
<web-app version="2.5"  
    xmlns="http://java.sun.com/xml/ns/javaee"  
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee  
http://java.sun.com/xml/ns/javaee/web-app 2.5.xsd">  
    <display-name></display-name>  
    <servlet>  
        <servlet-name>AddCookieServlet</servlet-name>  
        <servlet-class>web.AddCookieServlet</servlet-class>  
    </servlet>  
    <servlet>  
        <servlet-name>FindCookieServlet</servlet-name>  
        <servlet-class>web.FindCookieServlet</servlet-class>  
    </servlet>  
    <servlet>  
        <servlet-name>UpdateCookieServlet</servlet-name>  
        <servlet-class>web.UpdateCookieServlet</servlet-class>  
    </servlet>  
  
    <servlet-mapping>  
        <servlet-name>AddCookieServlet</servlet-name>  
        <url-pattern>/addCookie</url-pattern>  
    </servlet-mapping>  
    <servlet-mapping>  
        <servlet-name>FindCookieServlet</servlet-name>  
        <url-pattern>/findCookie</url-pattern>  
    </servlet-mapping>  
    <servlet-mapping>  
        <servlet-name>UpdateCookieServlet</servlet-name>  
        <url-pattern>/updateCookie</url-pattern>  
    </servlet-mapping>  
</web-app>
```

4. Cookie 的保存时间

- 问题

设置 Cookie 的过期时间，使得 Cookie 保存在客户端硬盘中。

- 方案

使用 setMaxAge 方法设置，单位为秒。

seconds > 0：保存在硬盘上的时间；

seconds = 0：立即删除；

seconds < 0：默认，保存在浏览器的内存中。

- 步骤

步骤一：新建 PersistentCookieServlet 类

```
public void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    //创建Cookie
    Cookie c1 = new Cookie("uname", "Kitty");
    //设置c1的过期时间
    c1.setMaxAge(100);
    response.addCookie(c1);
    Cookie c2 = new Cookie("city", "Beijing");
    response.addCookie(c2);
    out.close();
}
```

图 - 19

步骤二：访问 persistentCookie，查看 Cookie 状态

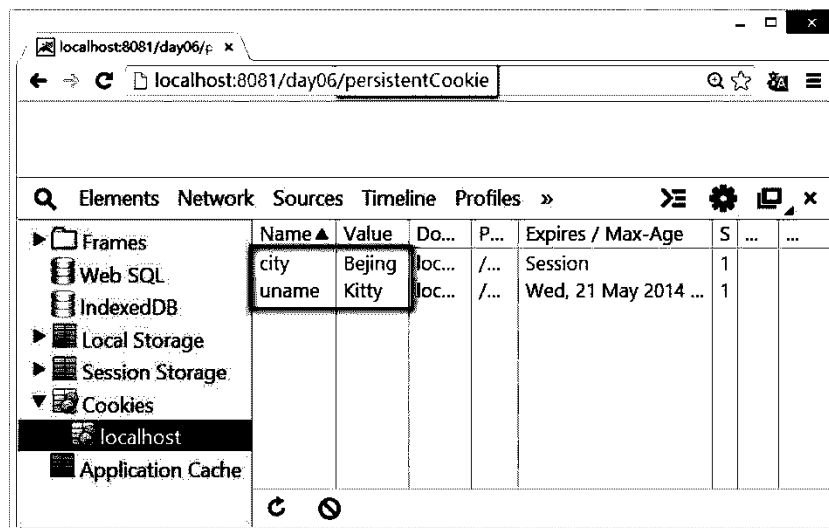


图 - 20

步骤三：重启浏览器，打开工具栏中的高级设置查看 Cookie 状态

重启浏览器，从菜单中选择“设置”→“显示高级设置”→“隐私设置”下的“内容设置”。

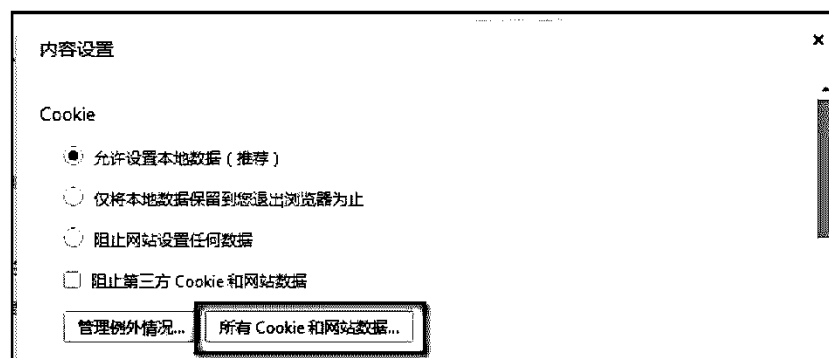


图 - 21



图 - 22

通过查看本地 Cookie 文件，发现关闭浏览器以后 city 已经消失，保留了设置过期时间的 uname 这个 Cookie 的值。

• 完整代码

PersistentCookieServlet.java 文件代码：

```
package web;

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

public class PersistentCookie extends HttpServlet {
    public void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        //创建 cookie
        Cookie c1 = new Cookie("uname", "Kitty");
        //设置 c1 的过期时间
        c1.setMaxAge(100);
        response.addCookie(c1);
        Cookie c2 = new Cookie("city", "Beijing");
        response.addCookie(c2);
        out.close();
    }
}
```

web.xml 文件代码：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
xmlns="http://java.sun.com/xml/ns/javaee"
```

```

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <display-name></display-name>
  <servlet>
    <servlet-name>AddCookieServlet</servlet-name>
    <servlet-class>web.AddCookieServlet</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>FindCookieServlet</servlet-name>
    <servlet-class>web.FindCookieServlet</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>UpdateCookieServlet</servlet-name>
    <servlet-class>web.UpdateCookieServlet</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>PersistentCookie</servlet-name>
    <servlet-class>web.PersistentCookie</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>AddCookieServlet</servlet-name>
    <url-pattern>/addCookie</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>FindCookieServlet</servlet-name>
    <url-pattern>/findCookie</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>UpdateCookieServlet</servlet-name>
    <url-pattern>/updateCookie</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>PersistentCookie</servlet-name>
    <url-pattern>/persistentCookie</url-pattern>
  </servlet-mapping>
</web-app>

```

5. Cookie 案例——中文编码

- 问题

在 cookie 中存储中文。

- 方案

cookie 的值只能是 ASCII 字符，中文需要转换成 ASCII 码形式，使用 `URLEncoder.encode()` 方法和 `URLDecoder.decode()` 方法实现。

- 步骤

步骤一：新建 EncodeCookieServlet

```
public void service(HttpServletRequest request,
                    HttpServletResponse response)
                    throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    //先将中文进行UTF-8编码，再将编码后的字节数组转变为字符串
    String city = URLEncoder.encode("北京", "UTF-8");
    //创建Cookie
    Cookie c1 = new Cookie("uname", "Kitty");
    Cookie c2 = new Cookie("city", city);
    response.addCookie(c1);
    response.addCookie(c2);
}
```

图 - 22

步骤二：新建 DecodeCookieServlet

```
public void service(HttpServletRequest request,
                    HttpServletResponse response)
                    throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    Cookie[] cookies = request.getCookies();
    if (cookies != null) {
        for (Cookie c : cookies) {
            String name = c.getName();
            String value = c.getValue();
            out.println("<h3>Name:" + name + "    Value:"
                + URLDecoder.decode(value, "UTF-8") + "</h3>");
        }
    } else {
        out.println("没有Cookie");
    }
    out.close();
}
```

图 - 23

步骤三：访问 encodeCookie，查看响应数据包

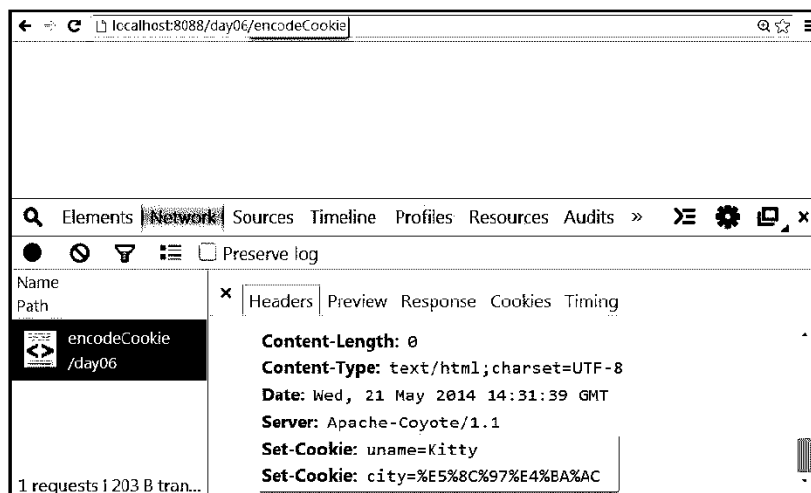


图 - 24

步骤四：访问 decodeCookie，查看页面输出

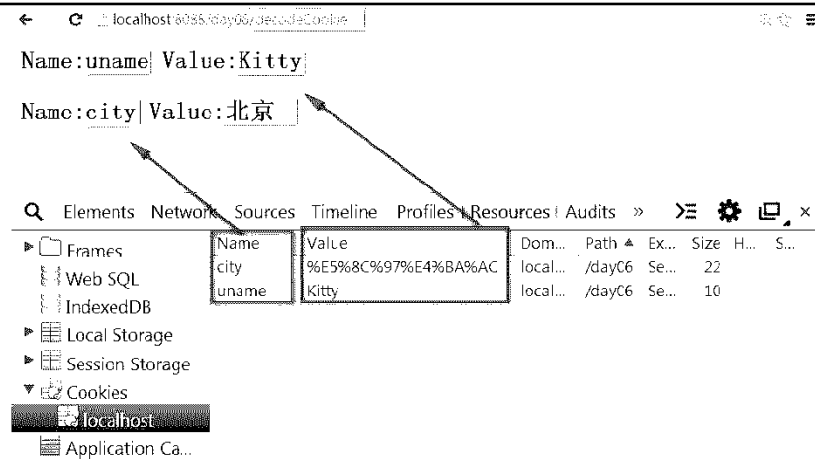


图 - 25

• 完整代码

EncodeCookieServlet.java 文件代码：

```
package web;

import java.io.IOException;
import java.io.PrintWriter;
import java.net.URLEncoder;

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

public class EncodeCookieServlet extends HttpServlet {
    public void service(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        //先将中文进行 UTF-8 编码，再将编码后的字节数组转变为字符串
        String city = URLEncoder.encode("北京", "UTF-8");
        //创建 Cookie
        Cookie c1 = new Cookie("uname", "Kitty");
        Cookie c2 = new Cookie("city", city);
        response.addCookie(c1);
        response.addCookie(c2);
    }
}
```

DecodeCookieServlet.java 文件代码：

```
package web;

import java.io.IOException;
import java.io.PrintWriter;
import java.net.URLDecoder;

import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
```

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

public class DecodeCookieServlet extends HttpServlet {

    public void service(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        Cookie[] cookies = request.getCookies();
        if (cookies != null) {
            for (Cookie c : cookies) {
                String name = c.getName();
                String value = c.getValue();
                out.println("<h3>Name:" + name + "    Value:"
                    + URLDecoder.decode(value, "UTF-8") + "</h3>");
            }
        } else {
            out.println("没有 Cookie");
        }
        out.close();
    }
}
```

web.xml 文件代码：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <display-name></display-name>
    <servlet>
        <servlet-name>AddCookieServlet</servlet-name>
        <servlet-class>web.AddCookieServlet</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>FindCookieServlet</servlet-name>
        <servlet-class>web.FindCookieServlet</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>UpdateCookieServlet</servlet-name>
        <servlet-class>web.UpdateCookieServlet</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>PersistentCookie</servlet-name>
        <servlet-class>web.PersistentCookie</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>EncodeCookieServlet</servlet-name>
        <servlet-class>web.EncodeCookieServlet</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>DecodeCookieServlet</servlet-name>
        <servlet-class>web.DecodeCookieServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>AddCookieServlet</servlet-name>
        <url-pattern>/addCookie</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>FindCookieServlet</servlet-name>
        <url-pattern>/findCookie</url-pattern>
    </servlet-mapping>
    </web-app>
```

```
</servlet-mapping>
<servlet-mapping>
  <servlet-name>UpdateCookieServlet</servlet-name>
  <url-pattern>/updateCookie</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>PersistentCookie</servlet-name>
  <url-pattern>/persistentCookie</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>EncodeCookieServlet</servlet-name>
  <url-pattern>/encodeCookie</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>DecodeCookieServlet</servlet-name>
  <url-pattern>/decodeCookie</url-pattern>
</servlet-mapping>
</web-app>
```

6. Cookie 案例——路径问题

• 问题

修改 Cookie 的默认路径，使得对应用的任何访问都能发送该站点的 Cookie。

• 方案

浏览器再向服务器的某个地址发请求时，会先比较 cookie 的路径与当前要访问的路径是否匹配，只有匹配的路径，才能发送 cookie 成功。cookie 的路径通过 setPath(String path)方法设置。缺省的 Cookie 路径为生成该 Cookie 的组件路径。只有路径相同或为其子路径时浏览器才会发送 Cookie。如图-26 所示即为 Cookie 的路径。



图 - 26

• 步骤

步骤一：新建 jsp/addCookie.jsp

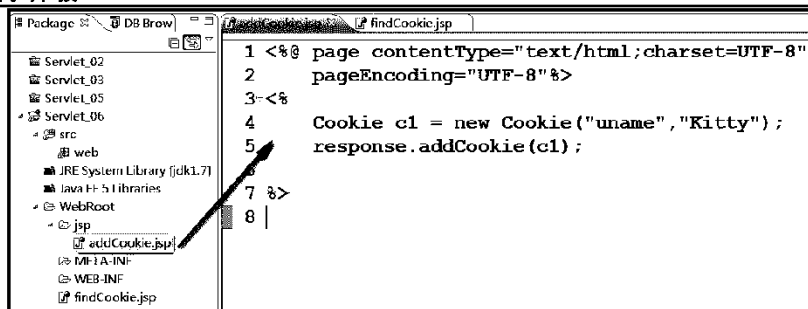


图 - 27

步骤二：新建 webRoot/findeCookie.jsp

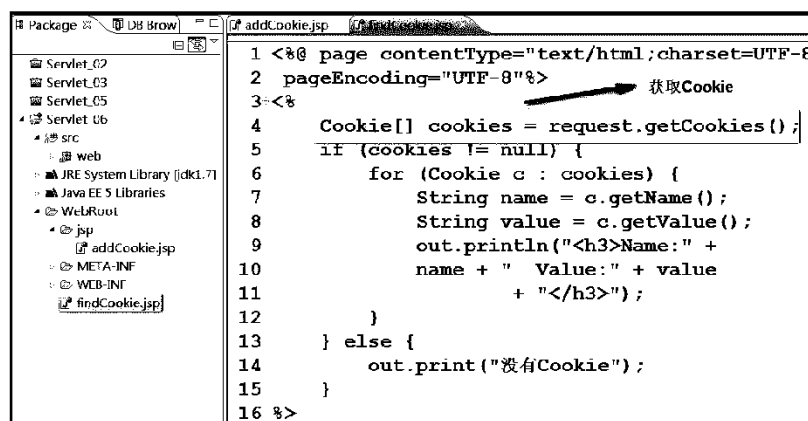


图 - 28

步骤三：访问 addCookie.jsp 页面

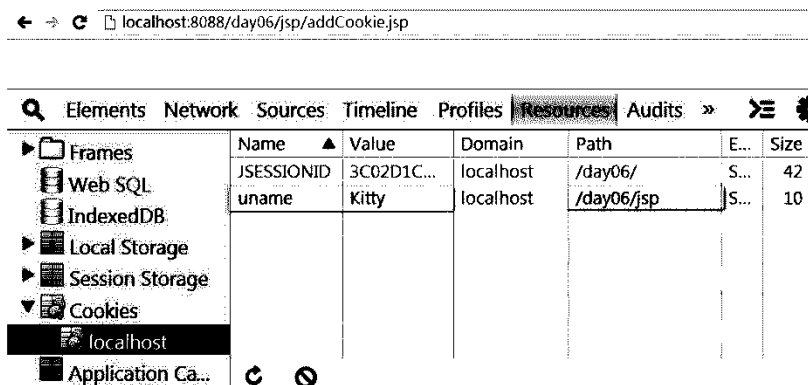


图 - 29

步骤四：访问 webRoot/findCookie.jsp 页面

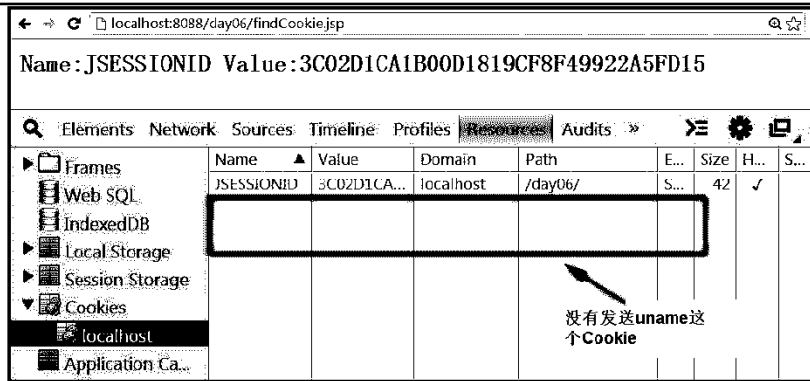


图 - 30

步骤五：修改 addCookie.jsp 的代码后再访问 findCookie.jsp

修改 addCookie.jsp 文件 添加 c.setPath() 方法。重新部署后 访问 addCookie.jsp , 会发现 path 的值与图-29 不再一样，由 “/day06/jsp” 变为 “/day06”。

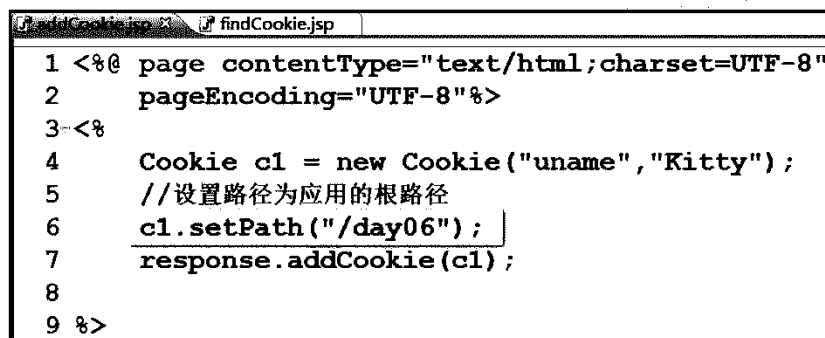


图 - 31

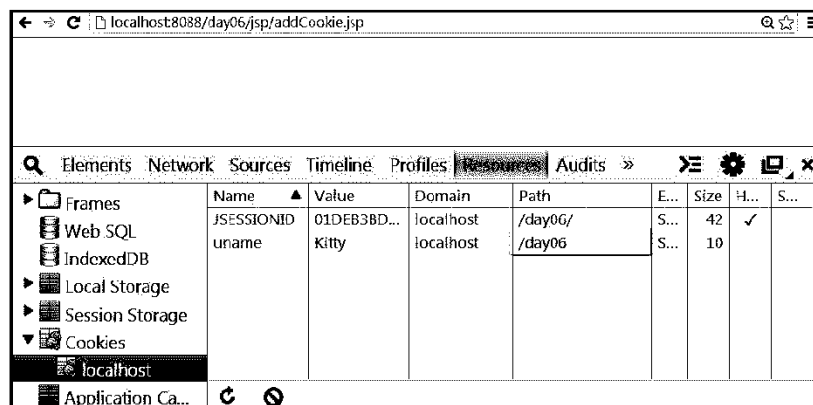


图 - 32

再次访问 findCookie.jsp 文件，运行如下：

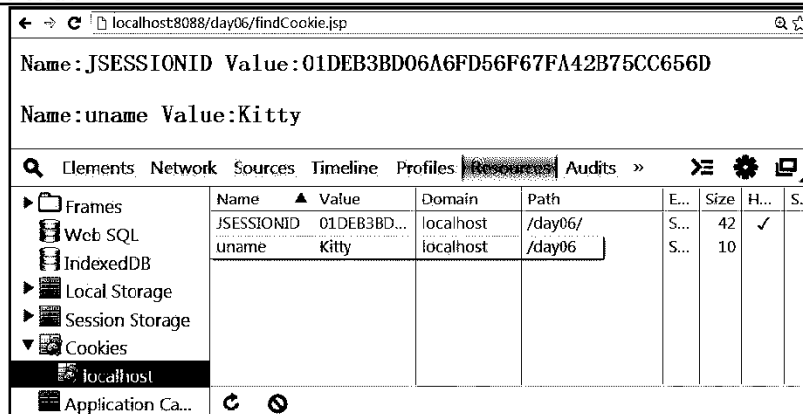


图 - 33

结论：只有访问的当前路径与 Cookie 路径一致或是其子目录时，浏览器才会发送 Cookie。需要修改 Cookie 路径时使用 `setPath()` 方法。

• 完整代码

`jsp/addCookie.jsp` 文件代码如下：

```
<%@ page contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%
    Cookie c1 = new Cookie("uname", "Kitty");
    //设置路径为应用的根路径
    c1.setPath("/day06");
    response.addCookie(c1);
%>
```

`findCookie.jsp` 文件代码如下：

```
<%@ page contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%
    Cookie[] cookies = request.getCookies();
    if (cookies != null) {
        for (Cookie c : cookies) {
            String name = c.getName();
            String value = c.getValue();
            out.println("<h3>Name: " +
                name + " Value: " + value
                + "</h3>");
        }
    } else {
        out.print("没有 Cookie");
    }
%>
```

课后作业

1. 简述什么是状态管理？
2. 简述什么是 Cookie？
3. 阅读下面的代码，说明序号处代码的含义。

```
Cookie[] cookies = request.getCookies();//-----1
if(cookies !=null){
    for(int i=0;i<cookies.length;i++){
        Cookie currCookie = cookies[i];
        String name = currCookie.getName();//-----2
        String value = currCookie.getValue();
        out.println("<h1>name:" + name
            + " value:" + value + "</h1>");
    }
}else{
    out.println("<h1>no cookie</h1>");
}
```

4. 使用 Cookie 实现登录功能 ,可以设置身份保存时间为“一次”或“永久”。

在课后练习“NETCTOSS 的登录功能和帐务帐号的 CRUD”的基础上，使用 Cookie 实现自动登录。

Servlet和JSP(下)

Unit02

知识体系.....Page 34

状态管理-Session	Session	什么是 Session (会话)
		Session 工作原理
		如何获得 Session
		如何使用 Session 绑定对象
		如何删除 Session 对象
		Session 验证
	Session 超时	什么是 Session 超时
		如何修改 Session 的缺省时间限制
		浏览器禁用 Cookie 的后果
		什么是 URL 重写 ?
		如何实现 URL 重写 ?
		Session 的优缺点
	验证码	验证码的作用
		验证码的绘制
		验证码图片的绘制步骤
		验证码的验证流程

经典案例.....Page 40

使用 Session 实现访问计数	如何获得 Session
	如何使用 Session 绑定对象
	如何删除 Session 对象
使用 Session 实现登录	Session 验证
Session 超时的设置	如何修改 Session 的缺省时间限制
URL 重写	什么是 URL 重写 ?
	如何实现 URL 重写 ?
验证码	验证码的绘制
	验证码图片的绘制步骤
为登录功能添加验证码	验证码的验证流程

1. 状态管理-Session

1.1. Session

1.1.1. 【Session】什么是 Session（会话）

Tarena
达内科技

什么是Session（会话）

- 浏览器访问Web服务器时，服务器会为每一个浏览器在服务器端的内存中分配空间，单独创建一个Session对象，该对象有一个Id属性，其值唯一，一般称之为SessionId，并且服务器会将这个SessionId（使用Cookie的方式）发送给浏览器；浏览器再次访问服务器时，会将SessionId发送给服务器，服务器可以依据SessionId找到对应的Session对象

+

1.1.2. 【Session】Session 工作原理

Tarena
达内科技

Session工作原理

+

1.1.3. 【Session】如何获得 Session


Tarena
达内科技

如何获得Session

```
HttpSession s = request.getSession(boolean flag);
```

- HttpSession是个接口，后面返回的是符合接口规范的对象
- 当flag为true时：先查看请求中有没有SessionId，如果没有SessionId，服务器创建一个Session对象；如果有SessionId，依据SessionId查找对应Session对象，找到则返回，找不到则创建一个新的Session对象，所以flag为true时，一定能得到一个Session对象
- 当flag为false时，没有SessionId及有SessionId但没有找到Session对象，均返回null；找到则返回

+




如何获得Session (续)

```
HttpSession s = request.getSession();
```

代码清单

- 等价于request.getSession (true) ;
- 提供该方法是为了代码书写更方便一些，大部分情况下是不管找没找到都需要返回一个Session对象



1.1.4. 【Session】如何使用 Session 绑定对象



如何使用Session绑定对象

```

绑定对象：
void    Session . setAttribute ( String name, Object
obj ) ;

获取绑定对象：
Object  Session . getAttribute ( String name ) ;

移除绑定对象：
void    Session . removeAttribute ( String name ) ;

```

代码清单

- 注：getAttribute方法的返回值是Object类型，在去除数据时要对其进行数据类型转换，且必须与我们存入的数据类型一致



1.1.5. 【Session】如何删除 Session 对象



如何删除Session对象

代码清单

- 立即删除Session对象：
Session . invalidate ()



1.1.6. 【Session】Session 验证

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h3>Session验证</h3> <ul style="list-style-type: none"> • 用户访问需要保护的资源时，可以使用Session验证的方式来保证其安全性，比如要求登陆后才能访问的资源 • 实现Session验证，遵循以下步骤 <ul style="list-style-type: none"> - 1、使用Session.setAttribute () 先绑定数据 - 2、使用Session.getAttribute () 方式来读取绑定值，如果没有，则跳转回登录页面 <div style="text-align: right;">+</div>
---	---

1.2. Session 超时

1.2.1. 【Session 超时】什么是 Session 超时

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h3>什么是Session超时</h3> <ul style="list-style-type: none"> • Web服务器会将空闲时间过长的Session对象删除掉，以节省服务器内存空间资源 • web服务器缺省的超时时间限制：一般是30分钟 <div style="text-align: right;">+</div>
---	---

1.2.2. 【Session 超时】如何修改 Session 的缺省时间限制

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h3>如何修改Session的缺省时间限制</h3> <ul style="list-style-type: none"> • 通过修改tomcat中 conf/web.xml 文件的设置 <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <pre><session-config> <session-timeout>30</session-timeout> </session-config></pre> </div> <div style="text-align: right;">+</div>
---	--

如何修改Session的缺省时间限制(续)

- 通过编程的方式来修改

```
void session . setMaxInactiveInterval ( int seconds )
```

代码片段

++

1.2.3. 【Session 超时】浏览器禁用 Cookie 的后果

浏览器禁用Cookie的后果

- 如果浏览器禁用Cookie，Session还能用吗？
- 答案：不能，但有其他的解决方案
- 服务器在默认情况下，会使用Cookie的方式将SessionId发送给浏览器，如果用户禁止Cookie，则SessionId不会被浏览器保存，此时，服务器可以使用如URL重写这样的方式来发送SessionId

代码片段

++

1.2.4. 【Session 超时】什么是 URL 重写？


什么是URL重写？

- 浏览器在访问服务器上的某个地址时，不再使用原来的那个地址，而是使用经过改写的地址（即，在原来的地址后面加上了SessionId）

代码片段

++

1.2.5. 【Session 超时】如何实现 URL 重写？


<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4>如何实现URL重写？</h4> <ul style="list-style-type: none"> • 如果是 链接地址和表单提交，使用 response.encodeURL (String url) 生成重写后的URL • 如果是重定向，使用 response.encodeRedirectURL (String url) 生成重写后的URL <div style="text-align: right;">+</div>
---	--

1.2.6. 【Session 超时】Session 的优缺点



<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4>Session的优缺点</h4> <ul style="list-style-type: none"> • 优点 <ul style="list-style-type: none"> - 安全（将状态保存在服务器端） - Session能够保存的的数据类型更丰富，Cookie只能保存字符串 - Session能够保存更多的数据，Cookie大约保存4k • 缺点 <ul style="list-style-type: none"> - Session将状态保存在服务器端，占用服务器的内存，如果用户量过大，会严重影响服务器的性能 <div style="text-align: right;">+</div>
---	--

1.3. 验证码



1.3.1. 【验证码】验证码的作用

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4>验证码的作用</h4> <ul style="list-style-type: none"> • 为了防止机器人的破坏性操作，可以使用验证码技术来防止恶意的发送数据 • 验证码本质上是一张动态产生的图片 • 图片的内容会随着程序的运行而随机生成 <div style="text-align: right;">+</div>
---	---



1.3.2. 【验证码】验证码的绘制

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4 style="text-align: center;">验证码的绘制</h4> <ul style="list-style-type: none"> • 验证码图片的生成需要使用java提供的与绘图有关的一系列API • 要想绘图，需要画板，画笔，颜料，背景色，字体等多种对象配合完成 <div style="text-align: right;">  </div> <div style="text-align: right;">+</div>
---	--

1.3.3. 【验证码】验证码图片的绘制步骤

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4 style="text-align: center;">验证码图片的绘制步骤</h4> <ol style="list-style-type: none"> ① 创建一个内存画板对象 ② 获取画笔 ③ 为画笔指定颜色 ④ 为画板设置背景色 ⑤ 绘制一个随机的字符串 ⑥ 修改画笔颜色 ⑦ 绘制多条随机干扰线 ⑧ 压缩图片并输出到客户端 <div style="text-align: right;">  </div> <div style="text-align: right;">+</div>
---	---

1.3.4. 【验证码】验证码的验证流程

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4 style="text-align: center;">验证码的验证流程</h4> <ul style="list-style-type: none"> • 请求带有验证码的页面时： <ul style="list-style-type: none"> - 通过img标签的src属性获取验证码图片 - 服务器端生成随机字符串，并绘制 - 服务器端将生成的随机字符串绑定到session中 • 提交表单及填写的验证码内容时： <ul style="list-style-type: none"> - 处理程序将session中绑定的正确的验证码字符串取出来 - 获取表单提交时填写的验证码内容 - 比较两者，根据结果做出判断 <div style="text-align: right;">  </div> <div style="text-align: right;">+</div>
---	--

经典案例

1. 使用 Session 实现访问计数

- 问题

记录不同客户端访问服务器的次数。

- 方案

由于对不同的客户端需要区分，所以使用 session 记录不同客户端的数据。由于访问次数需要保留，所以使用 setAttribute 方法进行记录。第一次访问时存储为 1，后续访问在此基础上自增。

- 步骤

步骤一：新建 CountServlet

```
11
12 public class CountServlet extends HttpServlet {
13
14     public void service(HttpServletRequest request,
15                         HttpServletResponse response)
16         throws ServletException, IOException {
17         response.setContentType("text/html;charset=UTF-8");
18         PrintWriter out = response.getWriter();    获取session对象
19         //获取session对象
20         HttpSession session = request.getSession();
21         //输出sessionID
22         System.out.println(session.getId());
23         //获取绑定的计数器
24         Integer count = (Integer)session.getAttribute("count");
25         if(count==null){//第一次访问
26             count = 1;
27         }else{//不是第一次访问
28             count++;    获取绑定数据
29         }
30         //在session中绑定计数器
31         session.setAttribute("count", count);
32         //输出提示信息
33         out.println("这是第"+count+"次访问");
34         out.close();
35     }
36 }
```

图 - 1

步骤二：部署应用，访问

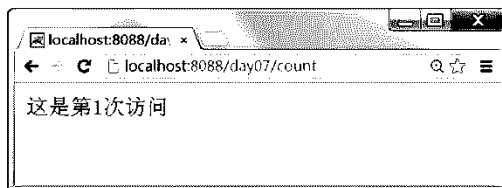


图 - 2

步骤三：多次刷新页面查看计数变化

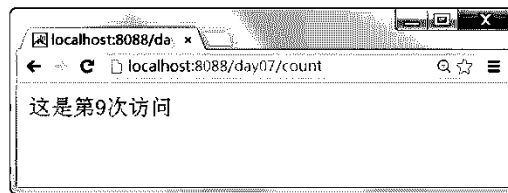


图 - 3

步骤四：重启浏览器，查看页面输出

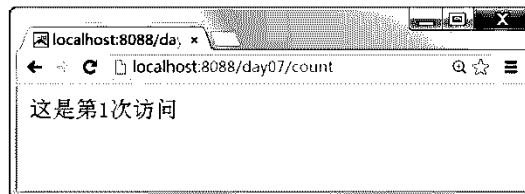


图 - 4

步骤五：查看控制台输出窗口

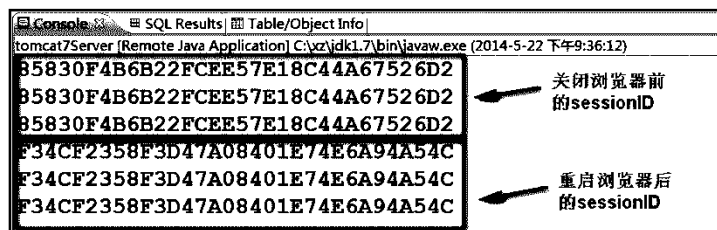


图 - 5

结论：sessionID 跟随一次会话，存储在客户端浏览器的内存中一个叫 JSESSIONID 的 Cookie 里面。第一次创建后，只要不关闭浏览器，那么每一次的请求 sessionID 都会被自动发送到服务器端。重启浏览器后，sessionID 会被销毁，重新创建新的数值。

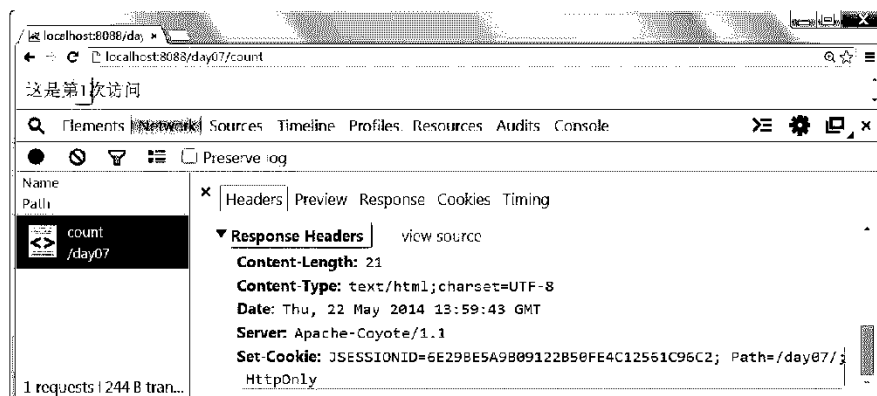


图 - 6

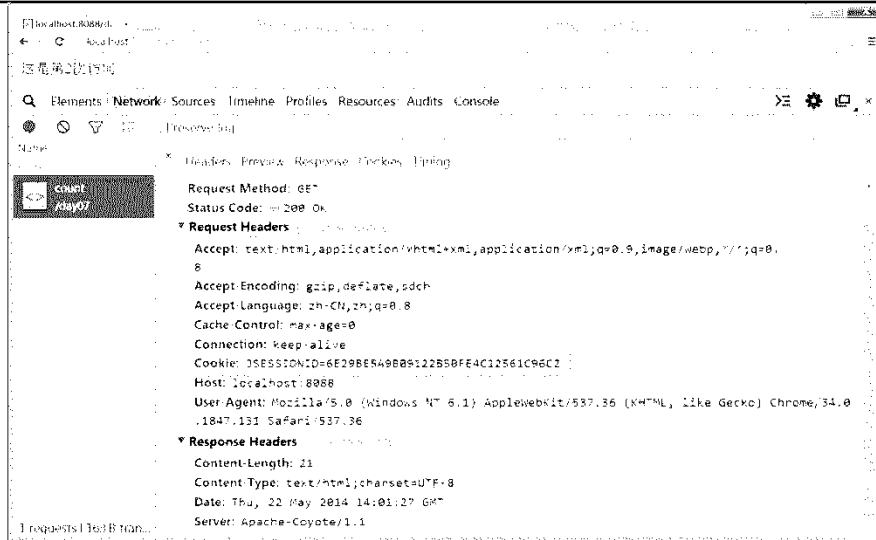


图 - 7

图-6 为第一次访问，JSESSIONID 在响应头中以 Set-Cookie 的形式传回客户端。图-7 为刷新页面时的数据包，在请求头中，JSESSIONID 以 Cookie 消息头的形式发给服务器端，以此同一个 ID 值找到对应的 session 对象。

• 完整代码

CountServlet.java 文件的代码：

```
package web;

import java.io.IOException;
import java.io.PrintWriter;

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

public class CountServlet extends HttpServlet {

    public void service(HttpServletRequest request,
                        HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out = response.getWriter();

        //获取 session 对象
        HttpSession session = request.getSession();

        //输出 sessionId
        System.out.println(session.getId());

        //获取绑定的计数器
        Integer count = (Integer) session.getAttribute("count");
        if(count==null){//第一次访问
            count = 1;
        }else{//不是第一次访问
            count++;
        }
    }
}
```

```
//在 session 中绑定计数器
session.setAttribute("count", count);
//输出提示信息
out.println("这是第"+count+"次访问");
out.close();
}
}
```

web.xml 文件代码：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <servlet>
    <servlet-name>CountServlet</servlet-name>
    <servlet-class>web.CountServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>CountServlet</servlet-name>
    <url-pattern>/count</url-pattern>
  </servlet-mapping>
</web-app>
```

2. 使用 Session 实现登录

• 问题

保存登录信息，保护某些资源只有在登录验证之后才能访问。

• 方案

登录信息提交后进行验证，验证通过则将用户信息保存在 session 中，验证失败则转发回登录页面。对于需要保护的资源，添加从 session 中获取绑定值的功能，以判断是否能够成功获取绑定的用户信息为判断依据，获取到则代表已验证，则允许访问，获取不到绑定在 session 中的用户信息，代码验证没有通过，则拒绝访问该资源，而将请求重定向到登录页面。

• 步骤

步骤一：新建 login.jsp 页面

```
<form action="login.do" method="post">
  用户名: <input name="uname" /><Br><br>
  密 码: <input name="pwd" type="password" /><br><br>
  <input type="submit" value="登录" />
</form>
```

图 - 8

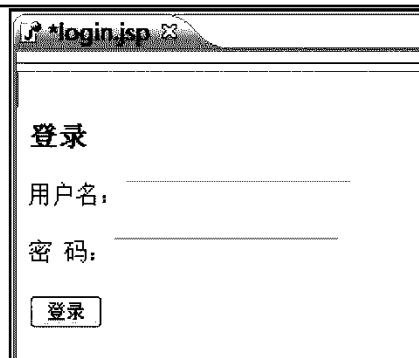


图 - 9

步骤二：新建 ActionServlet 类及配置

```
public void service(HttpServletRequest request, HttpServletResponse
    throws ServletException, IOException {
    request.setCharacterEncoding("UTF-8");
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    String uri = request.getRequestURI();
    String action = uri.substring(uri.lastIndexOf("/") + 1,
        uri.lastIndexOf("."));
    HttpSession session = request.getSession();
    System.out.println(session.getId());
    // 判断动作是否为登录
    if (action.equals("login")) {
        String name = request.getParameter("uname");
        String pwd = request.getParameter("pwd");
        if (name.equals("111") && pwd.equals("111")) {
            session.setAttribute("uname", name);
            // 重定向到首页
            response.sendRedirect("index.jsp");
        } else {
            // 登录失败
            request.setAttribute("msg", "用户名或密码错误");
            request.getRequestDispatcher("login.jsp")
                .forward(request, response);
        }
    }
}
```

图 - 10

```
<servlet>
  <servlet-name>ActionServlet</servlet-name>
  <servlet-class>web.ActionServlet</servlet-class>
</servlet>
```

```
<servlet-mapping>
  <servlet-name>ActionServlet</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

图 - 11

步骤三：新建 index.jsp 页面

```

login.jsp  ActionServlet.java  index.jsp
1 <%@ page contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8"%>
3 <%
4   //session验证
5   Object uname = session.getAttribute("uname");
6   if(uname == null){
7       response.sendRedirect("login.jsp");
8       return;
9   }
10
11 %>
12 <html>
13   <head>
14
15   </head>
16   <body>
17   <h3>首页</h3>欢迎你: <%=uname.toString()%><br><E

```

没有登录信息，拒绝访问

图 - 12

步骤四：部署后直接访问 index.jsp 页面

部署后访问 index.jsp 失败，会重定向到 login.jsp 页面。

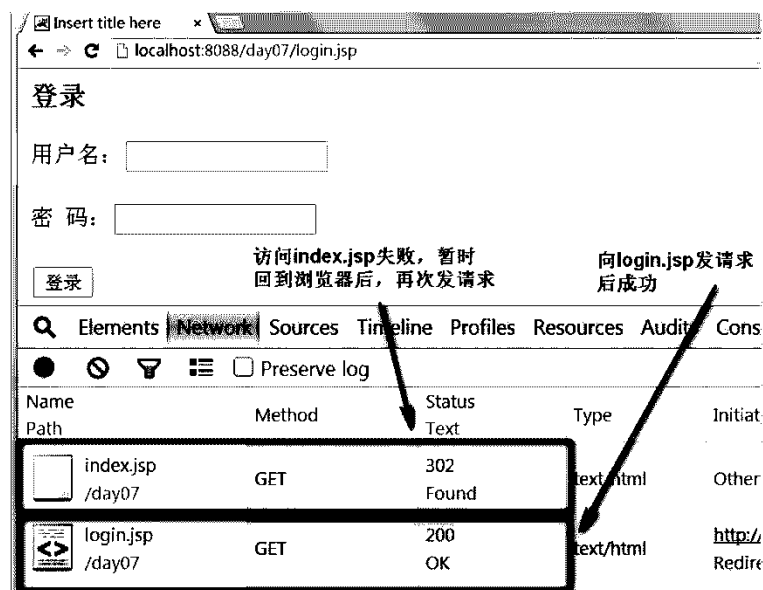


图 - 13

步骤五：在 login.jsp 页面进行登录

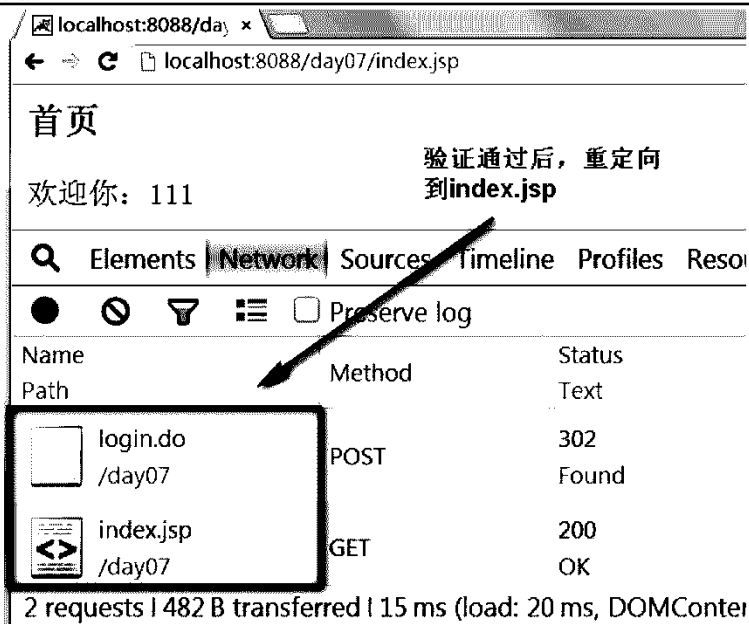


图 - 14

步骤六：新开页面后直接访问 index.jsp 页面

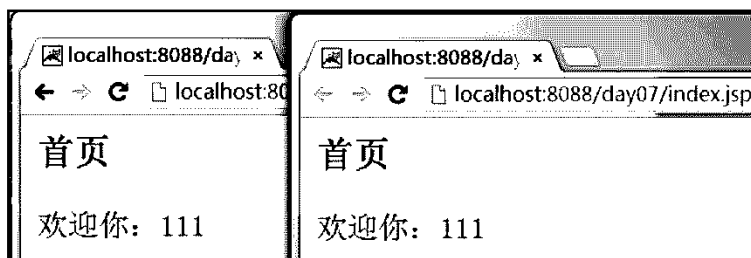


图 - 15

在新开的页面直接访问 index.jsp 页面也能够看到显示的登录信息。

步骤七：重启浏览器后访问 index.jsp 页面

重启浏览器后，直接访问 index.jsp 页面，结果与步骤四一致。因为，关闭浏览器后保存的 JSESSIONID 信息会丢失，以至于无法再在服务器中找到绑定了登录信息的 session 对象，于是认为没有登录过，进行了重定向操作。

步骤八：修改 index.jsp 页面，添加“登出”链接

```
<html>
  <head> |
</head>
<body>
<h3>首页</h3>欢迎你: <%=uname.toString()%><br>

  <a href="logout.do">登出</a>|
</body>
</html>
```

图 - 16

步骤九：修改 ActionServlet

```
} else if (action.equals("logout")) {
    // session失效
    session.invalidate();
    response.sendRedirect("login.jsp");
}
```

图 - 17

步骤十：重新访问 login.jsp 进行登录



图 - 18

步骤十一：点击“登出”后，再次访问 index.jsp

登出后，再次直接输入地址访问 index.jsp，结果同步骤四，重定向到 login.jsp 页面。符合登出功能的效果。

• 完整代码

ActionServlet.java 文件：

```
package web;

import java.io.IOException;
import java.io.PrintWriter;

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

public class ActionServlet extends HttpServlet {

    public void service(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        request.setCharacterEncoding("UTF-8");
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        String uri = request.getRequestURI();
        String action = uri.substring(uri.lastIndexOf("/") + 1,
            uri.lastIndexOf("."));
        HttpSession session = request.getSession();
        System.out.println(session.getId());
    }
}
```

```
// 判断动作是否为登录
if (action.equals("login")) {
    String name = request.getParameter("uname");
    String pwd = request.getParameter("pwd");
    if (name.equals("111") && pwd.equals("111")) {
        session.setAttribute("uname", name);
        // 重定向到首页
        response.sendRedirect("index.jsp");
    } else {
        // 登录失败
        request.setAttribute("msg", "用户名或密码错误");
        request.getRequestDispatcher("login.jsp").forward(request,
            response);
    }
} else if (action.equals("logout")) {
    // session 失效
    session.invalidate();
    response.sendRedirect("login.jsp");
}
out.close();
}
```

web.xml 文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <servlet>
        <servlet-name>CountServlet</servlet-name>
        <servlet-class>web.CountServlet</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>ActionServlet</servlet-name>
        <servlet-class>web.ActionServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>CountServlet</servlet-name>
        <url-pattern>/count</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>ActionServlet</servlet-name>
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>
</web-app>
```

login.jsp 文件：

```
<%@ page contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<html>
<head>
    <title>Insert title here</title>
</head>
<body>
<h3>登录</h3>
<form action="login.do" method="post">
    用户名:<input name="uname"/><Br><br>
```

```

        密 码 :<input name="pwd" type="password"/><br><Br>
        <input type="submit" value="登录"/>
    </form>
</body>
</html>

```

index.jsp 文件：

```

<%@ page contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%
    //session 验证
    Object uname = session.getAttribute("uname");
    if(uname == null){
        response.sendRedirect("login.jsp");
        return;
    }
%>
<html>
<head>
</head>
<body>
<h3>首页</h3>欢迎你 :<%=uname.toString()%><br><Br><Br>
<a href="logout.do">登出</a>
</body>
</html>

```

3. Session 超时的设置

- **问题**

使用编程式，修改 session 的超时时间为 10 秒。实现当用户登录后，如果 10 秒钟没有任何动作，则 session 失效，此时再访问 index.jsp 页面，则依然会重定向到 login.jsp 页面，重新登录。

- **方案**

使用 `setMaxInactiveInterval(10)` 方法实现编程式的 session 超时设置。时间单位为秒。

- **步骤**

步骤一：修改 ActionServlet

在创建 session 后，设置超时时间为 10 秒。

```
response.setContentType("text/html;charset=UTF-8");
PrintWriter out = response.getWriter();
String uri = request.getRequestURI();
String action = uri.substring(uri.lastIndexOf("/") + 1,
    uri.lastIndexOf("."));
HttpSession session = request.getSession();
System.out.println(session.getId());
// 编程式--设定session超时时间为10秒
session.setMaxInactiveInterval(10);
```

图 - 19

步骤二：访问 login.jsp 页面后 10 秒内不操作

访问 login.jsp 输入用户信息进行登录,到达 index.jsp 页面后,不再有任何动作,过 10 秒后,刷新 index.jsp 页面,请求会重定向到 login.jsp 页面。

如果以声明式的方式修改超时时间,则需要到 tomcat 的 conf 目录下,寻找 web.xml 文件,修改 session-timeout。

```
506 <!-- ===== Default Session Configuration =====>
507 <!-- You can set the default session timeout for all
508 <!-- created sessions by modifying this value here. 默认的超时设置
509
510 <session-config>
511     <session-timeout>30</session-timeout>
512 </session-config>
```

图 - 20

• 完整代码

ActionServlet.java 文件代码：

```
package web;

import java.io.IOException;
import java.io.PrintWriter;

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

public class ActionServlet extends HttpServlet {

    public void service(HttpServletRequest request, HttpServletResponse
        response)
        throws ServletException, IOException {
        request.setCharacterEncoding("UTF-8");
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        String uri = request.getRequestURI();
        String action = uri.substring(uri.lastIndexOf("/") + 1,
            uri.lastIndexOf("."));

        // 判断动作是否为登录
        if (action.equals("login")) {
            String name = request.getParameter("uname");
```

```
String pwd = request.getParameter("pwd");
if (name.equals("111") && pwd.equals("111")) {
    HttpSession session = request.getSession();
    System.out.println(session.getId());
    // 编程式--设定 session 超时时间为 10 秒
    session.setMaxInactiveInterval(10);
    session.setAttribute("uname", name);
    // 重定向到首页
    response.sendRedirect("index.jsp");
} else {
    // 登录失败
    request.setAttribute("msg", "用户名或密码错误");
    request.getRequestDispatcher("login.jsp").forward(request,
        response);
}
} else if (action.equals("logout")) {
    HttpSession session = request.getSession();
    // session 失效
    session.invalidate();
    response.sendRedirect("login.jsp");
}
}
out.close();
}
```

4. URL 重写

• 问题

如果浏览器禁用 Cookie，则 session 不能再继续使用。如何在禁用 Cookie 的情况下依然能继续使用 session。

• 方案

使用 URL 重写的方法，即改写原本的访问地址，在 URI 后面使用参数的方式携带上 sessionID。

• 步骤

步骤一：修改 ActionServlet

```
if (name.equals("111") && pwd.equals("111")) {
    session.setAttribute("uname", name);
    // 重定向到首页
    //response.sendRedirect("index.jsp");
    response.sendRedirect(
        response.encodeRedirectURL("index.jsp"));
}
```

图 - 21

步骤二：禁用浏览器 Cookie

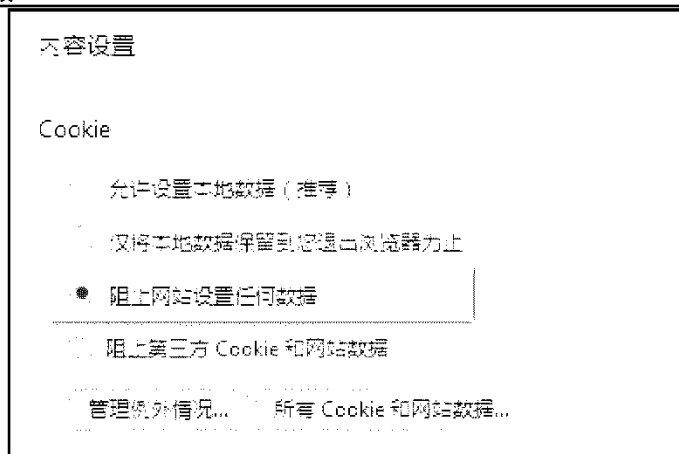


图 - 22

步骤三：使用 login.jsp 完成登录

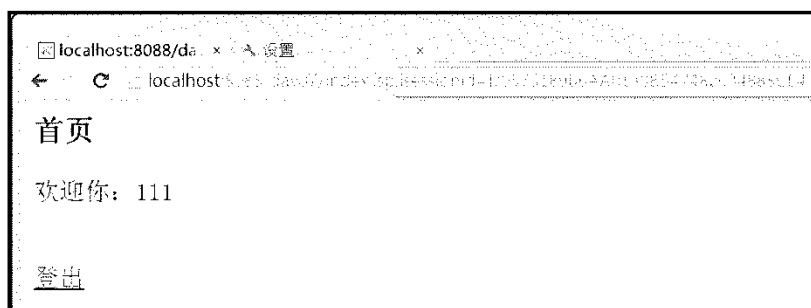


图 - 23

由于禁用了 cookie，所以要想继续使用 session 那么就可以通过如上方式，在 URL 中添加 sessionId 信息，以达到传递 id 标识的作用。

• 完整代码

ActionServlet.java 文件代码：

```
package web;

import java.io.IOException;
import java.io.PrintWriter;

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

public class ActionServlet extends HttpServlet {

    public void service(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        request.setCharacterEncoding("UTF-8");
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        String uri = request.getRequestURI();
```

```
String action = uri.substring(uri.lastIndexOf("/") + 1,
    uri.lastIndexOf("."));
HttpSession session = request.getSession();
System.out.println(session.getId());
// 编程式--设定 session 超时时间为 10 秒
//session.setMaxInactiveInterval(10);
// 判断动作是否为登录
if (action.equals("login")) {
    String name = request.getParameter("uname");
    String pwd = request.getParameter("pwd");
    if (name.equals("111") && pwd.equals("111")) {
        session.setAttribute("uname", name);
        // 重定向到首页
        response.sendRedirect(
            response.encodeRedirectURL("index.jsp"));
    } else {
        // 登录失败
        request.setAttribute("msg", "用户名或密码错误");
        request.getRequestDispatcher("login.jsp").forward(request,
            response);
    }
} else if (action.equals("logout")) {
    // session 失效
    session.invalidate();
    response.sendRedirect("login.jsp");
}
out.close();
}
}
```

5. 验证码

- **问题**

根据随机产生的字符串绘制图片，并输出到 jsp 页面中。

- **方案**

使用 java 的绘图技术，动态绘制一张图片，由 Servlet 完成后，通过 OutputStream 将其内容输出到客户端。浏览器通过 img 标签的 src 属性访问该 Servlet 以获得动态图片。

- **步骤**

步骤一：创建 validateCode.jsp 页面

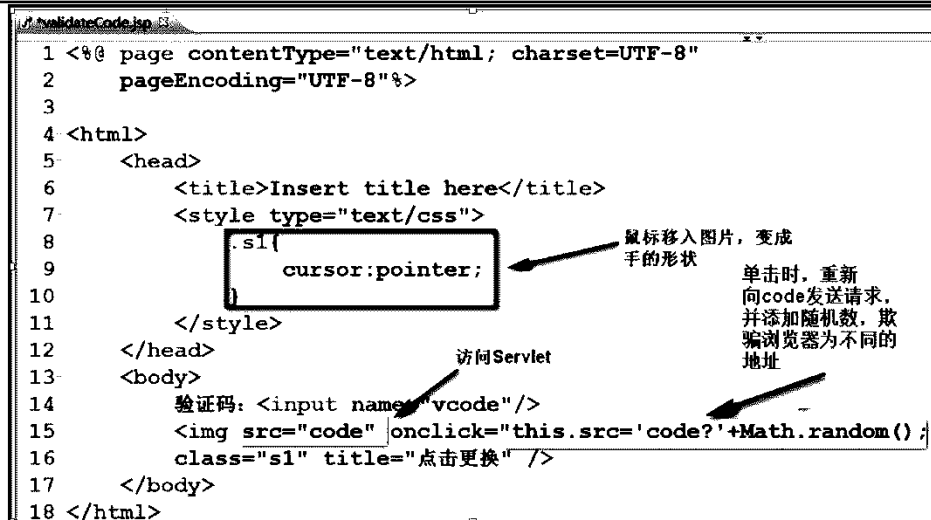


图 - 24

步骤二：创建 ValidateCode.java 文件

```

public void service(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    //0.创建空白图片
    BufferedImage image = new BufferedImage(100,30,BufferedImage.TYPE_INT_RGB);
    //1.获取图片画笔
    Graphics g = image.getGraphics();
    Random r = new Random();
    //2.设置画笔颜色
    g.setColor(new Color(r.nextInt(255), r.nextInt(255), r.nextInt(255)));
    //3.绘制矩形的背景
    g.fillRect(0, 0, 100, 30);
    //4.调用自定义的方法，获取长度为5的字母数字组合的字符串
    String number = getNumber(5);
    g.setColor(new Color(0,0,0));
    g.setFont(new Font(null,Font.BOLD,24));
    //5.设置颜色字体后，绘制字符串
    g.drawString(number, 5, 25);
    //6.绘制8条干扰线
    for(int i=0;i<8;i++){
        g.setColor(new Color(r.nextInt(255), r.nextInt(255), r.nextInt(255)));
        g.drawLine(r.nextInt(100), r.nextInt(30), r.nextInt(100), r.nextInt(30));
    }
    response.setContentType("image/jpeg");
    OutputStream ops = response.getOutputStream();
    ImageIO.write(image, "jpeg", ops);
    ops.close();
}

```

图 - 25

步骤三：部署后访问查看图片



图 - 26

• 完整代码

ValidateCode.java 文件代码：

```
package web;

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.image.BufferedImage;
import java.io.IOException;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.util.Random;

import javax.imageio.ImageIO;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class ValidateCode extends HttpServlet {
    public void service(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        //0.创建空白图片
        BufferedImage image = new BufferedImage(100,30,BufferedImage.TYPE_INT_RGB);
        //1.获取图片画笔
        Graphics g = image.getGraphics();
        Random r = new Random();
        //2.设置画笔颜色
        g.setColor(new Color(r.nextInt(255), r.nextInt(255), r.nextInt(255)));
        //3.绘制矩形的背景
        g.fillRect(0, 0, 100, 30);
        //4.调用自定义的方法，获取长度为5的字母数字组合的字符串
        String number = getNumber(5);
        g.setColor(new Color(0,0,0));
        g.setFont(new Font(null,Font.BOLD,24));
        //5.设置颜色字体后，绘制字符串
        g.drawString(number, 5, 25);
        //6.绘制8条干扰线
        for(int i=0;i<8;i++){
            g.setColor(new Color(r.nextInt(255),r.nextInt(255),r.nextInt(255),r.nextInt(255)));
            g.drawLine(r.nextInt(100), r.nextInt(30), r.nextInt(100), r.nextInt(30));
        }
    }
}
```

```
response.setContentType("image/jpeg");
OutputStream ops = response.getOutputStream();
ImageIO.write(image, "jpeg", ops);
ops.close();
}

private String getNumber(int size){
    String str = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
    String number = "";
    Random r = new Random();
    for(int i=0;i<size;i++){
        number+=str.charAt(r.nextInt(str.length()));
    }
    return number;
}
}
```

validateCode.jsp 文件代码：

```
<%@ page contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<html>
<head>
    <title>Insert title here</title>
    <style type="text/css">
        .s1{
            cursor:pointer;
        }
    </style>
</head>
<body>
    验证码:<input name="vcode"/>
    
</body>
</html>
```

web.xml 文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <servlet>
        <servlet-name>ValidateCode</servlet-name>
        <servlet-class>web.ValidateCode</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>ValidateCode</servlet-name>
        <url-pattern>/code</url-pattern>
    </servlet-mapping>
</web-app>
```

6. 为登录添加验证码功能

- 问题

在登录时添加验证码功能。

• 方案

在产生验证码的随机字符串之后，绑定到 session 中，验证过程中将用户输入和 session 绑定中的验证码比对一致后再判断用户名和密码。

• 步骤

步骤一：修改 ValidateCode.java 文件

```
//3. 绘制矩形的背景
g.fillRect(0, 0, 100, 30);
//4. 调用自定义的方法，获取长度为5的字母数字组合的字符串
String number = getNumber(5);
HttpSession session = request.getSession();
session.setAttribute("code", number);
g.setColor(new Color(0,0,0));
g.setFont(new Font(null,Font.BOLD,24));
//5. 设置颜色字体后，绘制字符串
```

图 - 27

步骤二：修改 login.jsp 页面

```
<head>
  <title>Insert title here</title>
  <style type="text/css">
    .s1{
      cursor:pointer;
    }
  </style>
</head>
<body>
<h3>登录</h3>
<form action="login.do" method="post">
  用户名: <input name="uname"/><br><br>
  密 码: <input name="pwd" type="password"/><br><br>
  验证码: <input name="vcode"/>
  <br><br>
  <input type="submit" value="登录"/>
</form>
</body>
```

图 - 28

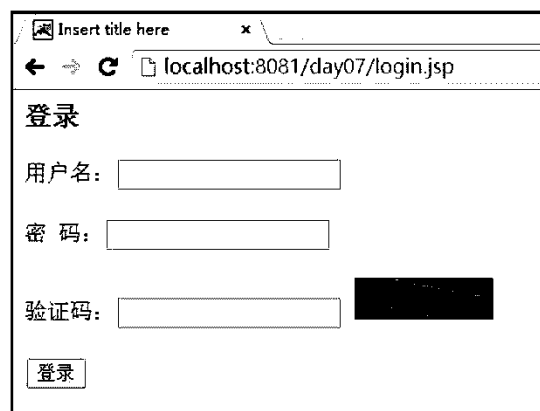


图 - 29

步骤三：修改 ActionServlet.java 文件

```
// 判断动作是否为登录
if (action.equals("login")) {
    String name = request.getParameter("uname");
    String pwd = request.getParameter("pwd");
    String number = request.getParameter("vcode");
    HttpSession session = request.getSession();
    String code = session.getAttribute("code").toString();
    if (number.equals(code) && name.equals("111") && pwd.equals("111")) {
        // 编程式--设定session超时时间为10秒
        //session.setMaxInactiveInterval(10);
        session.setAttribute("uname", name);
        // 重定向到首页
    }
}
```

图 - 30

步骤四：重新部署，访问应用

用户名和密码以及验证码均正确的情况下登录成功。验证码区分大小写。登出后直接访问 index.jsp 会自动回到 login.jsp 页面。

• 完整代码

ValidateCode.java 文件：

```
package web;

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.image.BufferedImage;
import java.io.IOException;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.util.Random;

import javax.imageio.ImageIO;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class ValidateCode extends HttpServlet {

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

        //0. 创建空白图片
        BufferedImage image = new BufferedImage(100, 30, BufferedImage.TYPE_INT_RGB);

        //1. 获取图片画笔
        Graphics g = image.getGraphics();
        Random r = new Random();

        //2. 设置画笔颜色
        g.setColor(new Color(r.nextInt(255), r.nextInt(255), r.nextInt(255)));

        //3. 绘制矩形的背景
```

```

g.fillRect(0, 0, 100, 30);
//4.调用自定义的方法,获取长度为5的字母数字组合的字符串
String number = getNumber(5);
HttpSession session = request.getSession();
session.setAttribute("code", number);
g.setColor(new Color(0,0,0));
g.setFont(new Font(null,Font.BOLD,24));
//5.设置颜色字体后,绘制字符串
g.drawString(number, 5, 25);
//6.绘制8条干扰线
for(int i=0;i<8;i++){
    g.setColor(new
Color(r.nextInt(255),r.nextInt(255),r.nextInt(255),r.nextInt(255)));
    g.drawLine(r.nextInt(100),      r.nextInt(30),      r.nextInt(100),
r.nextInt(30));
}
response.setContentType("image/jpeg");
OutputStream ops = response.getOutputStream();
ImageIO.write(image, "jpeg", ops);
ops.close();
}

private String getNumber(int size){
    String str = "ABCDEFGHJKLMNOPQRSTUVWXYZ0123456789";
    String number = "";
    Random r = new Random();
    for(int i=0;i<size;i++){
        number+=str.charAt(r.nextInt(str.length()));
    }
    return number;
}
}

```

login.jsp 文件：

```

<%@ page contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<html>
<head>
    <title>Insert title here</title>
    <style type="text/css">
        .s1{
            cursor:pointer;
        }
    </style>
</head>
<body>
<h3>登录</h3>
<form action="login.do" method="post">
    用户名:<input name="uname"/><br><br>
    密 码:<input name="pwd" type="password"/><br><br>
    验证码:<input name="vcode"/>
    <br><br>
    <input type="submit" value="登录"/>
</form>
</body>
</html>

```

ActionServlet 文件：


```
package web;

import java.io.IOException;
import java.io.PrintWriter;

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

public class ActionServlet extends HttpServlet {

    public void service(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        request.setCharacterEncoding("UTF-8");
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        String uri = request.getRequestURI();
        String action = uri.substring(uri.lastIndexOf("/") + 1,
            uri.lastIndexOf("."));

        // 判断动作是否为登录
        if (action.equals("login")) {
            String name = request.getParameter("uname");
            String pwd = request.getParameter("pwd");
            String number = request.getParameter("vcode");
            HttpSession session = request.getSession();
            String code = session.getAttribute("code").toString();
            if (number.equals(code) && name.equals("111") && pwd.equals("111"))
            {

                // 编程式--设定 session 超时时间为 10 秒
                //session.setMaxInactiveInterval(10);
                session.setAttribute("uname", name);

                // 重定向到首页
                //response.sendRedirect("index.jsp");
                response.sendRedirect(
                    response.encodeRedirectURL("index.jsp"));
            } else {
                // 登录失败

                request.setAttribute("msg", "用户名或密码错误");
                request.getRequestDispatcher("login.jsp").forward(request,
                    response);
            }
        } else if (action.equals("logout")) {
            HttpSession session = request.getSession();
            // session 失效
            session.invalidate();
            response.sendRedirect("login.jsp");
        }
        out.close();
    }
}
```

web.xml 文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app 2.5.xsd">
    <servlet>
        <servlet-name>CountServlet</servlet-name>
```

```
<servlet-class>web.CountServlet</servlet-class>
</servlet>
<servlet>
  <servlet-name>ActionServlet</servlet-name>
  <servlet-class>web.ActionServlet</servlet-class>
</servlet>
<servlet>
  <servlet-name>ValidateCode</servlet-name>
  <servlet-class>web.ValidateCode</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>CountServlet</servlet-name>
  <url-pattern>/count</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>ActionServlet</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>ValidateCode</servlet-name>
  <url-pattern>/code</url-pattern>
</servlet-mapping>
</web-app>
```

课后作业

1. 简述 Session 的工作原理。
2. 下列说法正确的是（ ）。
 - A. 在 Servlet 中，Session 对象不需要创建，直接就可以使用。
 - B. 使用 Session 对象的 setAttribute 方法进行对象的绑定。
 - C. Session 对象不能手动删除，只能等待系统删除。
 - D. 禁用 Cookie 后，Session 功能将失效。
3. 简述什么是 Session 超时，如何修改缺省的时间限制？
4. 为 NETCTOSS 系统的登录增加验证码功能。

在课后练习 “使用 Cookie 实现登录功能，可以设置身份保存时间为“一次”或“永久”的基础上为登录添加验证码功能。

Servlet和JSP(下)

Unit03

知识体系.....Page 65

过滤器、监听器	过滤器	什么是过滤器
		如何编写过滤器
		编写一个 java 类实现 Filter 接口
		实现拦截处理逻辑
		将过滤器添加到 Web 应用中
		过滤器和 Web 应用一起打包部署
		过滤器的执行流程
		过滤器的优先级
		多个过滤器的执行流程
		过滤器的初始化参数
		初始化参数的配置
		读取初始化参数
		过滤器的优点
	监听器	什么是监听器
		生命周期相关的事件
		绑定数据相关的事件
		如何编写监听器
		编写 Java 类
		实现处理逻辑
		注册监听器
		应用场景

经典案例.....Page 72

过滤器——过滤敏感词汇	过滤器的执行流程
过滤器——多个过滤器	过滤器的优先级
	多个过滤器的执行流程
过滤器——过滤器的初始化参数	过滤器的初始化参数
	初始化参数的配置
	读取初始化参数



监听器——统计在线人数	编写 Java 类
	实现处理逻辑
	注册监听器
	应用场景

课后作业.....Page 89



1. 过滤器、监听器

1.1. 过滤器



1.1.1. 【过滤器】什么是过滤器

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4>什么是过滤器</h4> <ul style="list-style-type: none"> 过滤器是Servlet2.3规范中定义的一种小型的、可插入的Web组件。用来拦截Servlet容器的请求和响应过程，以便查看、提取或以某种方式操作正在客户机和服务器之间交换的数据。 过滤器通常是封装了一些功能的Web组件，这些功能很重要，但对于处理客户端请求或发送响应来说不是决定性的。 典型的应用包括记录请求和响应的数据、管理会话属性等 <div style="text-align: right;">  </div>
---	---

1.1.2. 【过滤器】如何编写过滤器

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4>如何编写过滤器</h4> <ol style="list-style-type: none"> ① 编写一个Java类，实现Filter接口 ② 在doFilter方法中实现拦截处理逻辑 ③ 将过滤器添加到Web程序中 ④ 把过滤器和Web应用一起打包部署 <div style="text-align: right;">  </div>
---	--

1.1.3. 【过滤器】编写一个 java 类实现 Filter 接口

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4>① 编写一个java类实现Filter接口</h4> <ul style="list-style-type: none"> 过滤器API中有3个常用的接口，位于javax.Servlet包中。 Filter、FilterChain、FilterConfig 编程中，过滤器类要实现Filter接口 该接口中包含三个必须实现的方法 <ul style="list-style-type: none"> - void init (FilterConfig filterConfig) - void doFilter (ServletRequest request , ServletResponse response , FilterChain chain) - void destroy() <div style="text-align: right;">  </div>
---	---

1.1.4. 【过滤器】实现拦截处理逻辑

Tarena
达内科技

② 实现拦截处理逻辑

```

public class CommentFilter implements Filter{
    /**创建实例后，调用init方法，只执行一次*/
    public void init ( FilterConfig arg0 ) throws ServletException
    {    // ...    }

    /**容器调用doFilter方法处理请求*/
    public void doFilter( ServletRequest arg0 , ServletResponse arg1 ,
        FilterChain arg2 ) throws IOException , ServletException
    {    //... ...
        arg2.doFilter(arg0 , arg1);    }

    /**容器删除过滤器实例之前调用该方法，只执行一次*/
    public void destroy()( ...    }
    }
                    
```

知识扩展

++

1.1.5. 【过滤器】将过滤器添加到 Web 应用中

Tarena
达内科技

③ 将过滤器添加到Web应用中

- 修改web.xml文件，增加注册过滤器的节点

```

<filter>
    <filter-name>filter1</filter-name>
    <filter-class>web.xxxServlet</Servlet-
class>
</filter>
<filter-mapping>
    <filter-name>filter1</filter-name>
    <url-pattern>/xxx</url-pattern>
</filter-mapping>
                    
```

知识扩展

++

1.1.6. 【过滤器】过滤器和 Web 应用一起打包部署

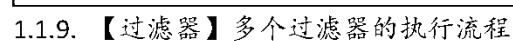
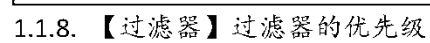
Tarena
达内科技

④ 过滤器和Web应用一起打包部署



- 与 Web 应用程序一起部署过滤器时，只需把过滤器类和其他 Web 组件类包括在一起，把 web.xml 文件（连同过滤器注册）放进 Web 应用程序结构中，Servlet 容器将处理之后的其他所有事情

知识扩展


++




1.1.10. 【过滤器】过滤器的初始化参数

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h3 style="text-align: center;">过滤器的初始化参数</h3> <ul style="list-style-type: none"> • 容器启动之后，会创建过滤器实例 • 接下来，容器会调用过滤器的init方法，而容器会事先创建FilterConfig对象。该对象可以访问在web.xml文件中配置的一些参数 • 这些在web.xml文件中存储，由FilterConfig对象读取，在执行init方法时能够访问的参数值，叫初始化参数 • 通过这些初始化参数可以方便快捷的配置及修改一些辅助参数 <div style="text-align: right;">  </div>
---	--



1.1.11. 【过滤器】初始化参数的配置

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h3 style="text-align: center;">初始化参数的配置</h3> <pre style="border: 1px solid black; padding: 10px;"> <filter> <filter-name>filter1</filter-name> <filter-class>web.xxxFilter</filter-class> <!-- 初始化参数 --> <init-param> <param-name>illegalStr</param-name> <param-value>xxx</param-value> </init-param> </filter> </pre>
---	--

1.1.12. 【过滤器】读取初始化参数



<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h3 style="text-align: center;">读取初始化参数</h3> <ul style="list-style-type: none"> • 使用FilterConfig对象可以读取在web.xml中配置的初始化参数 <div style="border: 1px solid black; padding: 10px; margin-top: 10px;"> <pre>String config . getInitParamter ("illegalStr")</pre> </div>
---	---

1.1.13. 【过滤器】过滤器的优点

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4>过滤器的优点</h4> <ul style="list-style-type: none"> • 实现代码的“可插拔性”，即增加或减少某个功能模块，不会影响程序的正常执行 • 可以将多个相同处理逻辑的模块集中写在过滤器里面，方便代码的维护 <div style="text-align: right;">  </div>
---	---

1.2. 监听器

1.2.1. 【监听器】什么是监听器

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4>什么是监听器</h4> <ul style="list-style-type: none"> • Servlet规范中定义的一种特殊的组件，用来监听Servlet容器产生的事件并进行相应的处理 • 容器产生的两大类事件 <ul style="list-style-type: none"> – 生命周期相关的事件 – 绑定数据相关的事件 <div style="text-align: right;">  </div>
---	---



1.2.2. 【监听器】生命周期相关的事件

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4>生命周期相关的事件</h4> <ul style="list-style-type: none"> • 容器创建或者销毁 request , session , ServletContext 时产生的事件 • ServletRequestListener <ul style="list-style-type: none"> – requestDestroyed (ServletRequestEvent sre) – requestInitialized (ServletRequestEvent sre) • HttpSessionListener <ul style="list-style-type: none"> – sessionCreated (HttpSessionEvent se) – sessionDestroyed (HttpSessionEvent se) • ServletContextListener <ul style="list-style-type: none"> – contextDestroyed (ServletContextEvent sce) – contextInitialized (ServletContextEvent sce) <div style="text-align: right;">  </div>
---	--


1.2.3. 【监听器】绑定数据相关的事件

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4 style="text-align: center;">绑定数据相关的事件</h4> <ul style="list-style-type: none"> • 调用了request, session, ServletContext 的 setAttribute、removeAttribute方法时产生的事件 • ServletRequestAttributeListener <ul style="list-style-type: none"> – attributeAdded(ServletRequestAttributeEvent srae) – attributeRemoved(ServletRequestAttributeEvent srae) – attributeReplaced(ServletRequestAttributeEvent srae) • HttpSessionAttributeListener <ul style="list-style-type: none"> – 参考 API Document • ServletContextAttributeListener <ul style="list-style-type: none"> – 参考 API Document <div style="text-align: right;">  </div>
---	---

1.2.4. 【监听器】如何编写监听器

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4 style="text-align: center;">如何编写监听器</h4> <ul style="list-style-type: none"> • step1, 编写一个Java类, 依据监听的事件类型选择实现相应的监听器接口。如, 要监听Session对象的创建和销毁, 要实现HttpSessionListener • step2, 在监听器接口方法中, 实现相应的监听处理逻辑。 • step3, 在web.xml文件中注册该监听器 <div style="text-align: right;">  </div>
---	--

1.2.5. 【监听器】编写 Java 类

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4 style="text-align: center;">① 编写Java类</h4> <pre> public class CouListener implements HttpSessionListener { // 实现方法 } </pre> <div style="position: relative; height: 100px;"> <div style="position: absolute; top: 0; right: 0; border: 1px solid black; padding: 2px;">选择相应的接口</div> </div>
---	---

1.2.6. 【监听器】实现处理逻辑

Tarena
达内科技

② 实现处理逻辑

```
public class CouListener implements HttpSessionListener
{
    public void sessionCreate(HttpSessionEvent arg0){
        //... ..
        HttpSession session = arg0.getSession( );
        ServletContext ctx =
            session.getServletContext( );
        //... ..
    }
}
```

HttpSession session = arg0.getSession();
 ServletContext ctx =
 session.getServletContext();

获取session或上
上下文对象的方法

Tarena
达内科技

1.2.7. 【监听器】注册监听器

Tarena
达内科技

③ 注册监听器

在web.xml文件中，增加以下节点：

```
<!-- 监听器 -->
<listener>
    <listener-class>web.CouListener</listener-class>
</listener>
```

<listener>
 <listener-class>web.CouListener</listener-class>
 </listener>

<!-- 实现相同的Listener接口多个监听器，
在执行时是按web.xml注册出现的顺序来决定 -->

Tarena
达内科技

1.2.8. 【监听器】应用场景

Tarena
达内科技

应用场景

- 由于ServletRequest、HttpSession、ServletContext对象都是容器创建的，通过对这些对象注册监听器，就可以得知何时创建了。比如：
 - 1) 在contextDestroyed方法中对应用级别的资源进行释放。
 - 2) 统计在线人数可以通过HttpSessionListener监听器的sessionCreated方法监听session的创建动作。

Tarena
达内科技

经典案例

1. 过滤器——过滤敏感词汇

- 问题

针对用户输入的评论内容进行敏感词汇的过滤，如果包含“damn”字样，则提示用户评论已关闭。

- 方案

在评论内容到达处理 Servlet 之前，由过滤器拦截后先进行内容的过滤，如果包含敏感词汇则直接返回给客户端相应提示，不包含敏感词汇则可以将此请求传递给下一个过滤器或用于处理的 Servlet。

- 步骤

步骤一：新建 comment.jsp 页面

comment.jsp 页面用于收集用户评论信息，显示效果及页面源码如下：



图 - 1

```
<form action="comment" method="post">
  <fieldset>
    <legend>评论</legend>
    请输入评论: <textarea name="comment"
      style="vertical-align:middle;
      width:140px;height:55px"></textarea><Br>
    <input type="submit" value="发表评论" />
  </fieldset>
</form>
```

图 - 2

步骤二：新建 CommentServlet.java 文件

创建处理评论及显示的 Servlet。并部署工程，测试评论是否能够正确发布。

```
public void service(HttpServletRequest request,
                    HttpServletResponse response)
                    throws ServletException, IOException {
    request.setCharacterEncoding("UTF-8");
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    //获取评论内容
    String str = request.getParameter("comment");
    //显示评论内容
    out.println("<h3>评论内容: " + str + "</h3>");
    out.close();
}
```

图 - 3

web.xml 文件配置如下：

```
<servlet>
    <servlet-name>CommentServlet</servlet-name>
    <servlet-class>web.CommentServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>CommentServlet</servlet-name>
    <url-pattern>/comment</url-pattern>
</servlet-mapping>
```

图 - 4

步骤三：新建 CommentFilter 类

```
public class CommentFilter1 implements Filter{
    private String illegalWord;

    * 容器删除过滤器实例之前调用，只执行一次
    public void destroy() {}

    * 用于处理请求的主要方法
    public void doFilter(ServletRequest request

    /**
     * 容器启动之后，创建过滤器实例
     * 然后调用init方法，只会调用一次
     * 容器会将已经创建好的FilterConfig对象作为参数传入
     * 可以从该参数中获取初始化的配置信息
     */
    public void init(FilterConfig filterConfig)
```

图 - 5

步骤四：配置 CommentFilter 过滤器

```
<!-- 过滤器 -->
<filter>
  <filter-name>filter1</filter-name>
  <filter-class>web.CommentFilter1</filter-class>
</filter>
<filter-mapping>
  <filter-name>filter1</filter-name>
  <url-pattern>/comment</url-pattern>
</filter-mapping>
```

与哪个Servlet路径一致，代表针对这个路径的请求都要进行过滤

图 - 6

步骤五：部署应用，输入评论查看结果



图 - 7



图 - 8

• 完整代码

comment.jsp 页面的代码：

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<html>
<head>
<title>评论</title>
</head>
<body>
<form action="comment" method="post">
  <fieldset>
```

```

        <legend>评论</legend>

        请输入评论 : <textarea name="comment"
        style="vertical-align:middle;
        width:140px;height:55px"></textarea><Br>
        <input type="submit" value="发表评论" />
    </fieldset>
</form>
</body>
</html>

```

CommentServlet.java 文件代码：

```

package web;
import java.io.IOException;
import java.io.PrintWriter;

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

public class CommentServlet extends HttpServlet {
    public void service(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        request.setCharacterEncoding("UTF-8");
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        //获取评论内容
        String str = request.getParameter("comment");
        //显示评论内容
        out.println("<h3>评论内容：" + str + "</h3>");
        out.close();
    }
}

```

CommentFilter1.java 文件代码：

```

package web;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class CommentFilter1 implements Filter{
    private String illegalWord;

    /**
     * 容器删除过滤器实例之前调用，只执行一次
     */
    public void destroy() {
    }
}

```



```
/**
 * 用于处理请求的主要方法
 */
public void doFilter(ServletRequest request, ServletResponse response,
    FilterChain chain) throws IOException, ServletException {
    HttpServletRequest req = (HttpServletRequest)request;
    HttpServletResponse resp = (HttpServletResponse)response;
    req.setCharacterEncoding("UTF-8");
    resp.setContentType("text/html;charset=UTF-8");
    PrintWriter out = resp.getWriter();
    String comment = req.getParameter("comment");
    if(comment.indexOf("damn")!=-1){
        //有敏感词汇

        out.print("<h3>评论内容已关闭</h3>");
    }else{
        //没有敏感词汇则向后处理，交给其他过滤器或 Servlet
        chain.doFilter(req, resp);
    }
}

/**
 * 容器启动之后，创建过滤器实例
 * 然后调用 init 方法，只会调用一次
 * 容器会将已经创建好的 FilterConfig 对象作为参数传入
 * 可以从该参数中获取初始化的配置信息
 */
public void init(FilterConfig filterConfig) throws ServletException {
    illegalWord = filterConfig.getInitParameter("illegalWord");
    System.out.println("Filter1:init is runnming... " + illegalWord);
}
}
```

web.xml 文件代码：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app 2.5.xsd">
    <display-name></display-name>
    <!-- 过滤器 -->
    <filter>
        <filter-name>filter1</filter-name>
        <filter-class>web.CommentFilter1</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>filter1</filter-name>
        <url-pattern>/comment</url-pattern>
    </filter-mapping>
    <!-- Servlet -->
    <servlet>
        <servlet-name>CommentServlet</servlet-name>
        <servlet-class>web.CommentServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>CommentServlet</servlet-name>
        <url-pattern>/comment</url-pattern>
    </servlet-mapping>
</web-app>
```

2. 过滤器——多个过滤器

- 问题

在进行过滤评论时，如果发布内容多过 20 个字，则不允许发布，并出现相应提示。

- 方案

增加过滤器，用于检查评论内容的长度。

- 步骤

步骤一：新建 CommentFilter2.java 文件

```
public void doFilter(ServletRequest request, ServletResponse response) throws IOException, ServletException {
    FilterChain chain = request.getFilterChain();
    HttpServletRequest req = (HttpServletRequest) request;
    HttpServletResponse resp = (HttpServletResponse) response;
    req.setCharacterEncoding("UTF-8");
    resp.setContentType("text/html; charset=UTF-8");
    PrintWriter out = resp.getWriter();
    String str = req.getParameter("comment");
    if (str.length() > 20) {
        out.print("评论太长，请重试");
    } else {
        chain.doFilter(request, response);
    }
}
```

保证其他的Filter和Servlet可以运行

图 - 9

步骤二：配置 CommentFilter 过滤器

```
<filter>
  <filter-name>filter2</filter-name>
  <filter-class>web.CommentFilter2</filter-class>
</filter>
<filter-mapping>
  <filter-name>filter2</filter-name>
  <url-pattern>/comment</url-pattern>
</filter-mapping>
```

与filter1过滤同样的请求路径

图 - 10

步骤三：运行查看结果



图 - 11



图 - 12



图 - 13



图 - 14

从运行结果中的，如果有多个过滤器存在，则执行顺序按<filter-mapping>的顺序执行。谁在前面，谁先执行。

- **完整代码**

CommentFilter.java 文件代码：

```
package web;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class CommentFilter2 implements Filter {

    public void destroy() {

    }

    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain) throws IOException, ServletException {
        HttpServletRequest req = (HttpServletRequest)request;
        HttpServletResponse resp = (HttpServletResponse)response;
        req.setCharacterEncoding("UTF-8");
        resp.setContentType("text/html;charset=UTF-8");
        PrintWriter out = resp.getWriter();
        String str = req.getParameter("comment");
        if(str.length()>20){
            out.print("评论太长，请重试");
        }else{
            chain.doFilter(request, response);
        }
    }

    public void init(FilterConfig filterConfig) throws ServletException {

    }

}
```

web.xml 文件代码：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <display-name></display-name>
    <!-- 过滤器 -->
    <filter>
        <filter-name>filter1</filter-name>
        <filter-class>web.CommentFilter1</filter-class>
    </filter>
```

```
<filter>
  <filter-name>filter2</filter-name>
  <filter-class>web.CommentFilter2</filter-class>
</filter>
<filter-mapping>
  <filter-name>filter2</filter-name>
  <url-pattern>/comment</url-pattern>
</filter-mapping>
<filter-mapping>
  <filter-name>filter1</filter-name>
  <url-pattern>/comment</url-pattern>
</filter-mapping>
<!-- Servlet -->
<servlet>
  <servlet-name>CommentServlet</servlet-name>
  <servlet-class>web.CommentServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>CommentServlet</servlet-name>
  <url-pattern>/comment</url-pattern>
</servlet-mapping>
</web-app>
```

3. 过滤器——过滤器的初始化参数

- 问题

动态设置敏感词汇及所限制的评论长度。

- 方案

使用过滤器的初始化参数，在 web.xml 文件设定敏感的词汇及限制的评论长度。需要修改时，只需要使用记事本修改 web.xml 即可。

- 步骤

步骤一：修改 web.xml 文件

```
<filter>
  <filter-name>filter1</filter-name>
  <filter-class>web.CommentFilter1</filter-class>
  <!-- 初始化参数 -->
  <init-param>
    <param-name>illegalWord</param-name>
    <param-value>damn</param-value>
  </init-param>
</filter>
<filter>
  <filter-name>filter2</filter-name>
  <filter-class>web.CommentFilter2</filter-class>
  <!-- 初始化参数 -->
  <init-param>
    <param-name>illegalLength</param-name>
    <param-value>30</param-value>
  </init-param>
</filter>
```

图 - 15

步骤二：修改 CommentFilter1 类、CommentFilter2 类读取初始化参数

```
public class CommentFilter1 implements Filter{
    private String illegalWord;           私有属性
}
```

图 - 16

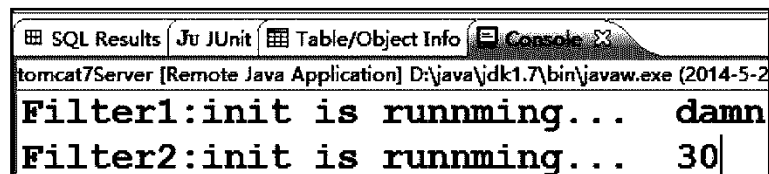
```
public void init(FilterConfig filterConfig) throws ServletException {
    illegalWord = filterConfig.getInitParameter("illegalWord");
    System.out.println("Filter1:init is running... " + illegalWord);
}
```

图 - 17

```
if (comment.indexOf(illegalWord) != -1) {
    //有敏感词汇
    out.print("<h3>评论内容已关闭</h3>");
} else {
    //没有敏感词汇则向后处理，交给其他过滤器或Servlet
    chain.doFilter(request, response);
}
```

图 - 18

步骤三：运行，查看是否能正确输出获取到的初始化参数



```
tomcat7Server [Remote Java Application] D:\java\jdk1.7\bin\javaw.exe (2014-5-2)
Filter1:init is running... damn
Filter2:init is running... 30
```

图 - 19

• 完整代码

web.xml 文件代码：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <display-name></display-name>
    <!-- 过滤器 -->
    <filter>
        <filter-name>filter1</filter-name>
        <filter-class>web.CommentFilter1</filter-class>
        <!-- 初始化参数 -->
        <init-param>
            <param-name>illegalWord</param-name>
            <param-value>damn</param-value>
        </init-param>
    </filter>
    <filter>
        <filter-name>filter2</filter-name>
        <filter-class>web.CommentFilter2</filter-class>
```

```
<!-- 初始化参数 -->
<init-param>
    <param-name>illegalLength</param-name>
    <param-value>30</param-value>
</init-param>
</filter>
<filter-mapping>
    <filter-name>filter2</filter-name>
    <url-pattern>/comment</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>filter1</filter-name>
    <url-pattern>/comment</url-pattern>
</filter-mapping>
<!-- Servlet -->
<servlet>
    <servlet-name>CommentServlet</servlet-name>
    <servlet-class>web.CommentServlet</servlet-class>
</servlet>
</servlet>
<servlet-mapping>
    <servlet-name>CommentServlet</servlet-name>
    <url-pattern>/comment</url-pattern>
</servlet-mapping>
</web-app>
```

CommentFilter1.java 文件代码：

```
package web;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class CommentFilter1 implements Filter{
    private String illegalWord;

    /**
     * 容器删除过滤器实例之前调用，只执行一次
     */
    public void destroy() {
    }

    /**
     * 用于处理请求的主要方法
     */
    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain) throws IOException, ServletException {
        HttpServletRequest req = (HttpServletRequest)request;
        HttpServletResponse resp = (HttpServletResponse)response;
        request.setCharacterEncoding("UTF-8");
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        String comment = request.getParameter("comment");
        if(comment.indexOf(illegalWord)!=-1){
```

```

        //有敏感词汇
        out.print("<h3>评论内容已关闭</h3>");
    }else{
        //没有敏感词汇则向后处理，交给其他过滤器或 Servlet
        chain.doFilter(request, response);
    }
}

/**
 * 容器启动之后，创建过滤器实例
 * 然后调用 init 方法，只会调用一次
 * 容器会将已经创建好的 FilterConfig 对象作为参数传入
 * 可以从该参数中获取初始化的配置信息
 */
public void init(FilterConfig filterConfig) throws ServletException {
    illegalWord = filterConfig.getInitParameter("illegalWord");
    System.out.println("Filter1:init is running... " + illegalWord);
}
}

```

CommentFilter2.java 文件代码：

```

package web;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class CommentFilter2 implements Filter {
    private int length;
    public void destroy() {
    }
    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain) throws IOException, ServletException {
        HttpServletRequest req = (HttpServletRequest)request;
        HttpServletResponse resp = (HttpServletResponse)response;
        req.setCharacterEncoding("UTF-8");
        resp.setContentType("text/html;charset=UTF-8");
        PrintWriter out = resp.getWriter();
        String str = req.getParameter("comment");
        if(str.length()>length){
            out.print("评论太长，请重试");
        }else{
            chain.doFilter(request, response);
        }
    }

    public void init(FilterConfig filterConfig) throws ServletException {
        length =
        Integer.parseInt(filterConfig.getInitParameter("illegalLength"));
        System.out.println("Filter2:init is running... " + length);
    }
}

```


4. 监听器——统计在线人数

- 问题

实现对当前在线人数的统计。

- 方案

通过监听 session 的创建动作，修改存储在线总人数的计数器。这个计数器保存在 Servlet 上下文中。

- 步骤

步骤一：新建 CountListener 类

```
public class CountListener implements HttpSessionListener {  
    private int count = 0;  
  
    public void sessionCreated(HttpSessionEvent se) {  
        count++;  
        HttpSession session = se.getSession();  
        ServletContext sct = session.getServletContext();  
        sct.setAttribute("count", count);  
    }  
  
    public void sessionDestroyed(HttpSessionEvent se) {  
        count--;  
        HttpSession session = se.getSession();  
        ServletContext sct = session.getServletContext();  
        sct.setAttribute("count", count);  
    }  
}
```

监听Session, 需要实现HttpSession接口

创建Session, 计数器+1

销毁Session, 计数器-1

图 - 20

步骤二：配置监听器

```
<!-- 监听器 -->  
<listener>  
    <listener-class>web.CountListener</listener-class>  
</listener>  
<!-- 过滤器 -->  
<filter>
```

监听器在过滤器之前

图 - 21

步骤三：添加 index.jsp 页面

```
<body>
    当前共有<%=application.getAttribute("count").toString()%>人在线
    <a href="logout">登出</a>
</body>
```

图 - 22

步骤四：添加 LogoutServlet 类

```
public void service(HttpServletRequest request,
                    throws ServletException, IOException {

    response.setContentType("text/html;charset=
    PrintWriter out = response.getWriter();
    HttpSession session = request.getSession();
    session.invalidate();
    out.close();
}
```

登出时销毁Session

图 - 23

步骤四：配置 LogoutServlet

```
<servlet>
    <servlet-name>LogoutServlet</servlet-name>
    <servlet-class>web.LogoutServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>LogoutServlet</servlet-name>
    <url-pattern>/logout</url-pattern>
</servlet-mapping>
```

图 - 24

步骤五：运行查看结果

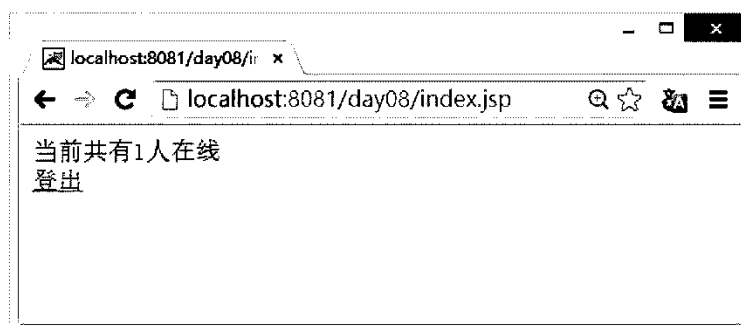


图 - 25

由于 Chrome 浏览器再打开选项卡也是与第 1 个窗口共用 sessionId ,所以使用火狐浏览器来模拟第 2 个上线的用户，运行效果如图-26 所示：



图 - 26

回到 Chrome 浏览器，点击刷新，查看结果如图-27 所示：

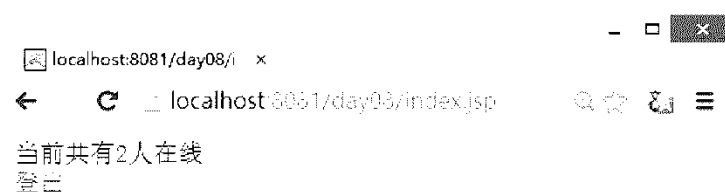


图 - 27

点击火狐浏览器中的登出，然后回到 Chrome 浏览器刷新页面，查看结果与图-25 结果一样。

• 完整代码

CountListener.java 文件代码：

```
package web;

import javax.servlet.ServletContext;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpSessionEvent;
import javax.servlet.http.HttpSessionListener;

public class CountListener implements HttpSessionListener {
    private int count = 0;

    public void sessionCreated(HttpSessionEvent se) {
        count++;
        HttpSession session = se.getSession();
        ServletContext sct = session.getServletContext();
        sct.setAttribute("count", count);
    }

    public void sessionDestroyed(HttpSessionEvent se) {
        count--;
        HttpSession session = se.getSession();
        ServletContext sct = session.getServletContext();
        sct.setAttribute("count", count);
    }
}
```

```
}  
}
```

web.xml 文件代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>  
<web-app version="2.5"  
  xmlns="http://java.sun.com/xml/ns/javaee"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee  
http://java.sun.com/xml/ns/javaee/web-app 2.5.xsd">  
  <display-name></display-name>  
  <!-- 全局初始化参数 -->  
  <context-param>  
    <param-name>count</param-name>  
    <param-value>1000</param-value>  
  </context-param>  
  <!-- 监听器 -->  
  <listener>  
    <listener-class>web.CountListener</listener-class>  
  </listener>  
  <!-- 过滤器 -->  
  <filter>  
    <filter-name>filter1</filter-name>  
    <filter-class>web.CommentFilter1</filter-class>  
    <!-- 初始化参数 -->  
    <init-param>  
      <param-name>illegalWord</param-name>  
      <param-value>damn</param-value>  
    </init-param>  
  </filter>  
  <filter>  
    <filter-name>filter2</filter-name>  
    <filter-class>web.CommentFilter2</filter-class>  
    <!-- 初始化参数 -->  
    <init-param>  
      <param-name>illegalLength</param-name>  
      <param-value>30</param-value>  
    </init-param>  
  </filter>  
  <filter-mapping>  
    <filter-name>filter2</filter-name>  
    <url-pattern>/comment</url-pattern>  
  </filter-mapping>  
  <filter-mapping>  
    <filter-name>filter1</filter-name>  
    <url-pattern>/comment</url-pattern>  
  </filter-mapping>  
  <!-- Servlet -->  
  <servlet>  
    <servlet-name>CommentServlet</servlet-name>  
    <servlet-class>web.CommentServlet</servlet-class>  
  </servlet>  
  <servlet>  
    <servlet-name>LogoutServlet</servlet-name>  
    <servlet-class>web.LogoutServlet</servlet-class>  
  </servlet>  
  <servlet-mapping>  
    <servlet-name>CommentServlet</servlet-name>  
    <url-pattern>/comment</url-pattern>  
  </servlet-mapping>  
  <servlet-mapping>  
    <servlet-name>LogoutServlet</servlet-name>
```

```
<url-pattern>/logout</url-pattern>
</servlet-mapping>
</web-app>
```

index.jsp 文件代码如下：

```
<%@ page contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<html>
<head>
    <title></title>
</head>
<body>
    当前共有<%=application.getAttribute("count").toString()%>人在线<br>
    <a href="/logout">登出</a>
</body>
</html>
```

课后作业

1. 简述什么是过滤器？
2. 简述过滤器的开发步骤。
3. 下列说法正确的是（ ）。
 - A．存在多个过滤器时，执行的顺序由系统随机决定。
 - B．过滤器只会过滤请求动作，不会过滤响应动作。
 - C．使用 filter 标记的 init-param 子标记进行过滤器的初始化参数设置。
 - D．编写过滤器时需要实现 javax.servlet.FilterChain 接口。
4. 简述什么是监听器。
5. 编写程序，监听用户访问 NETCTOSS 系统的频度。

Servlet和JSP(下)

Unit04

知识体系.....Page 92

EL、JSTL	EL 表达式	为什么需要 EL 表达式和 JSP 标签？
		什么是 EL 表达式
		EL 表达式的作用
		使用 EL 表达式访问 Bean 属性
		方式一：\${对象名.属性名}
		方式二：\${对象名["属性名"]}
		指定对象的查找范围
		使用 EL 表达式进行运算
		使用 EL 表达式获取请求参数值
	JSTL	什么是 JSTL
		如何使用 JSTL
		核心标签-if 标签
		核心标签-choose 标签
		核心标签-forEach 标签
		如何开发自定义标签
		标签的运行原理
		JSTL 应用

经典案例.....Page 99

访问 Bean 属性案例-1	方式一：\${对象名.属性名}(续 2)
访问 Bean 属性案例-2	方式二：\${对象名["属性名"]}
	指定对象的查找范围
使用 EL 表达式运算的案例	使用 EL 表达式进行运算
使用 EL 表达式获取请求参数	使用 EL 表达式获取请求参数值
if 标签案例	核心标签-if 标签
choose 标签案例	核心标签-choose 标签
forEach 标签案例	核心标签-forEach 标签
自定义标签案例	如何开发自定义标签
员工管理——使用 EL 表达式和 JSTL 实现 JSP 页面	JSTL 应用

1. EL、JSTL

1.1. EL 表达式

1.1.1. 【EL 表达式】为什么需要 EL 表达式和 JSP 标签？

为什么需要EL表达式和JSP标签？

达内科技

- JSP中嵌套的大量Java代码增加了页面内容的复杂度，使得页面不够简洁，不方便代码的维护
- 为此Sun公司制定了JSP标签（类似于html标签）来代替Java代码
- Apache组织开发的一套标签库被Sun公司整合后，称为标准标签库（JSP Standard Tag Library 即 JSTL），配合EL表达式，以达到减轻JSP文件的复杂度、方便维护JSP文件的目的

知识讲解

+

+

1.1.2. 【EL 表达式】什么是 EL 表达式

什么是EL表达式

达内科技

- EL(Expression Language)表达式是一套简单的计算规则，用于给JSP标签的属性赋值，也可以直接用来输出。
- 表达式也可以脱离标签单独使用

知识讲解

+

+

1.1.3. 【EL 表达式】EL 表达式的作用

EL表达式的作用

达内科技


- EL表达式的作用可分为以下几类：
 - 访问Bean的属性
 - 输出简单的运算结果
 - 获取请求参数值

知识讲解


+


+

1.1.4. 【EL 表达式】使用 EL 表达式访问 Bean 属性

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h3>使用EL表达式访问Bean属性</h3> <ul style="list-style-type: none"> • 通常情况下的JavaBean指的是一个公共的类，含有一个空参的构造方法，一些属性以及访问这些属性提供的get/set方法，方法名与属性名需要符合一定的规范。 • 使用EL表达式访问Bean属性时可使用如下两种方式 <ul style="list-style-type: none"> – 方式一：\${对象名.属性名} – 方式二：\${对象名["属性名"]} <div style="text-align: right;">+</div>
---	---

1.1.5. 【EL 表达式】方式一：\${对象名.属性名}


<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h3>方式一：\${对象名.属性名}</h3> <ul style="list-style-type: none"> • 如：\${user.name} • 执行过程：容器会依次从pageContext，request，session，application中查找绑定名称为“user”的对象。找到后，调用“getName”方法，然后输出。 • 其等价代码如下： <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <pre><% User user = (User) request . getAttribute("user"); out.print(user . getName()); %></pre> </div> <div style="text-align: right;">+</div>
---	--

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h3>方式一：\${对象名.属性名}（续1）</h3> <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <pre><% User user = (User) request . getAttribute("user"); out.print(user . getName()); %></pre> </div> <ul style="list-style-type: none"> • 上述代码存在的问题： <ul style="list-style-type: none"> – 如果request中没有user对象，会报500错误 – 没有为name属性赋过值，页面会打印出“null”字样 – 取值时绑定名写错，会报空指针500的异常 <div style="text-align: right;">+</div>
---	---

代码讲解

方式一：\${对象名.属性名} (续2)

- 以\${ user.name }方式访问会避免上段代码产生的问题
 - 如果没有为name属性赋值，页面输出空白，不会出现“null”的字样
 - 如果取值时绑定名写错，页面输出空白，不会报空指针异常
- 两种写法对比后，EL表达式具备了简洁，以及会将null转换成“”输出的特性
- 注意：属性名不能写错，否则报错**



+

1.1.6. 【EL 表达式】方式二：\${对象名[“属性名”]}


代码讲解

方式二：\${对象名[“属性名”]}

- 方括号中的属性名可以使用单引号或双引号
- 此种方式允许[]中出现绑定名，还可以允许[]中出现从0开始的下标，用于访问数组中的某个元素的值
- 如：

`${ user ["name"] }`
- 或：


```
<%
    request.setAttribute( "propName", "age" );
%>
${ user [ propName ] }
```
- 注：相当于表达式中写一个变量



+

1.1.7. 【EL 表达式】指定对象的查找范围

代码讲解

指定对象的查找范围


- 在编写EL表达式时，可以指定查找对应绑定名对象的范围
- 如，在session中查找绑定名为user的对象时，可编写如下代码

`${ sessionScope . user . name }`

pageScope



requestScope

applicationScope





+



1.1.8. 【EL 表达式】使用 EL 表达式进行运算

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4 style="text-align: center;">使用EL表达式进行运算</h4> <ul style="list-style-type: none"> • EL表达式可以做一些简单的计算，可将结果直接输出或给JSP标签的属性赋值 <ul style="list-style-type: none"> - 算术运算：“+” “-” “*” “/” “%” 注意，“+”号只能求和，不能够连接字符串 - 逻辑运算：“&&” “ ” “!” - 关系运算：“>” “>=” “<” “<=” “==” “!= - empty：用来判断一个字符串是否为空，或者一个集合是否为空，以下四种情况结果为true：空字符串，空的集合，值为null，找不到对应的值 <div style="text-align: right;">  </div>
---	--

1.1.9. 【EL 表达式】使用 EL 表达式进行运算（续）

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4 style="text-align: center;">使用EL表达式进行运算（续）</h4> <ul style="list-style-type: none"> • 算术运算： <ul style="list-style-type: none"> - <code>\${ 1 + 1 }</code> - <code>\${ "100" + " 200" }</code> • 关系运算 <ul style="list-style-type: none"> - <code>\${ 1 > 2 }</code> - <code>\${ str == "abc" }</code> 或 <code>\${ str eq "abc" }</code> • 逻辑运算 <ul style="list-style-type: none"> - <code>\${ 1 > 0 && 2 < 3 }</code> • empty运算 <ul style="list-style-type: none"> - <code>\${empty str1}</code> - <code>\${empty null}</code> <div style="text-align: right;">  </div>
---	--

1.1.10. 【EL 表达式】使用 EL 表达式获取请求参数值


<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4 style="text-align: center;">使用EL表达式获取请求参数值</h4> <ul style="list-style-type: none"> • <code>\${ param . username }</code> 等价于 <code>request.getParameter ("username") ;</code> • <code>\${ paramValues . city }</code> 等价于 <code>request.getParameterValues("city") ;</code> <div style="text-align: right;">  </div>
---	---

1.2. JSTL


1.2.1. 【JSTL】什么是 JSTL

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4>什么是JSTL</h4> <ul style="list-style-type: none"> • JSTL : (JSP Standard Tag Library) JSP标准标签库 • JSP标签是Sun公司定义的一套标准，由Apache组织基于这套标准开发的一套标签库后又转给Sun公司，被称为JSTL，成为了JavaEE5.0的核心 <div style="position: absolute; left: 455px; top: 215px; background-color: black; color: white; padding: 2px;">知识讲解</div> <div style="text-align: right;">+</div>
---	--



1.2.2. 【JSTL】如何使用 JSTL

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4>如何使用JSTL</h4> <ul style="list-style-type: none"> • step1, 将JSTL标签对应的jar文件拷贝到WEB-INF/lib目录下 • step2, 使用taglib指令导入要使用的JSP标签 <code><%@taglib uri= "" prefix= "" %></code> uri : JSP标签的命名空间 prefix : 命名空间的前缀 <div style="position: absolute; left: 455px; top: 455px; background-color: black; color: white; padding: 2px;">知识讲解</div> <div style="text-align: right;">+</div>
---	--



1.2.3. 【JSTL】核心标签-if 标签



<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4>核心标签-if标签</h4> <ul style="list-style-type: none"> • 语法 : <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <code><c:if test = "" var= "" scope= "" ></code> <code></c:if ></code> </div> • 当test属性值为true时，执行标签体的内容。test属性可以使用EL表达式赋值 • var属性：指定一个绑定名称 • scope属性：指定绑定的范围（ page , request , session , application ） • 注：var和scope要配合使用 <div style="position: absolute; left: 455px; top: 695px; background-color: black; color: white; padding: 2px;">知识讲解</div> <div style="text-align: right;">+</div>
---	--

1.2.4. 【JSTL】核心标签-choose 标签

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h3>核心标签-choose标签</h3> <ul style="list-style-type: none"> 语法： <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <pre><c:choose> <c:when test = "" ></c:when> ... <c:otherwise> </c:otherwise> </c:choose></pre> </div> when表示一个处理分支，当test属性为true时会执行该分支，可以出现1次或者多次 otherwise表示例外，可以出现0次或1次 <div style="text-align: right;">  </div>
---	---

1.2.5. 【JSTL】核心标签-forEach 标签

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h3>核心标签-forEach标签</h3> <ul style="list-style-type: none"> 用来遍历集合或者数组 语法： <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <pre><c:forEach var= "" items= "" > ... </c:forEach></pre> </div> items属性：指定要遍历的集合，一般使用EL表达式来复制 var属性：指定一个绑定名称，容器每次从集合中取一个对象，然后绑定到pageContext对象上 varStatus属性：指定一个绑定名称，绑定值是一个由容器创建的对象，该对象封装了当前迭代的状态 <div style="text-align: right;">  </div>
---	---

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h3>核心标签-forEach标签（续）</h3> <ul style="list-style-type: none"> varStatus属性：指定一个绑定名称，绑定值是一个由容器创建的对象，该对象封装了当前迭代的状态 index返回正在被迭代的对象的下标，下标从0开始 count返回是第几次迭代，从1开始 <div style="text-align: right;">  </div>
---	---

经典案例

1. 访问 Bean 属性案例-1

- 问题

使用 EL 表达式访问 Bean 的各个属性。

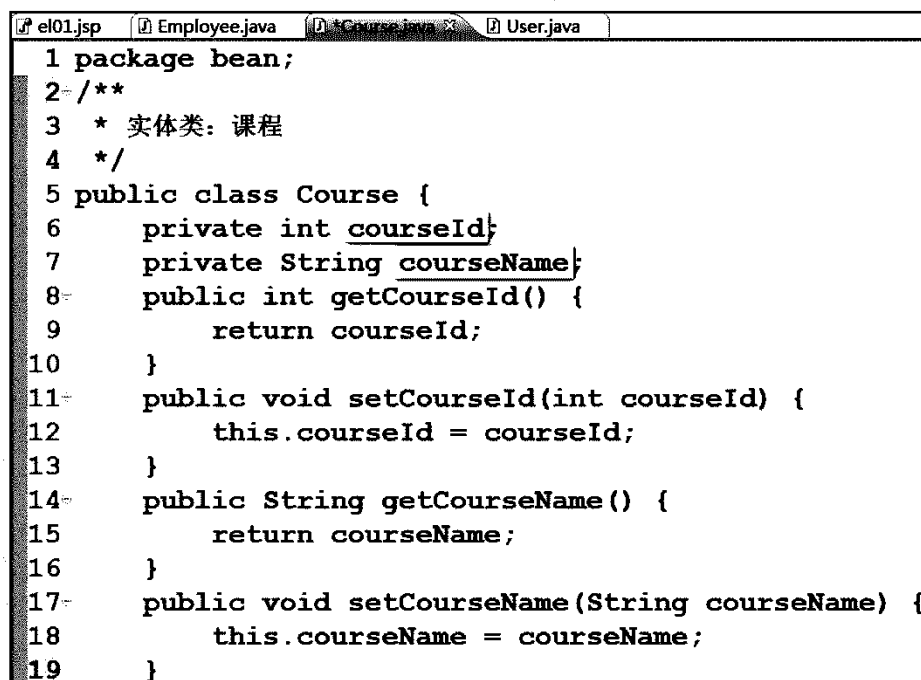
- 方案

创建 User 类型对象,绑定在 request 对象中,输出基本类型的属性值以及引用类型的属性值。

- 步骤

步骤一：创建 Course 类和 User 类

创建 Course 课程类,包括课程 Id 和课程名称属性,分别为各属性添加 get/set 方法。如图-1 所示：



```

1 package bean;
2 /**
3  * 实体类: 课程
4  */
5 public class Course {
6     private int courseId;
7     private String courseName;
8     public int getCourseId() {
9         return courseId;
10    }
11    public void setCourseId(int courseId) {
12        this.courseId = courseId;
13    }
14    public String getCourseName() {
15        return courseName;
16    }
17    public void setCourseName(String courseName) {
18        this.courseName = courseName;
19    }

```

图 - 1

创建 User 用户类,包括姓名,年龄,课程,兴趣爱好属性。其中课程属性为引用类型,兴趣爱好为数组类型。分别为各属性添加 get/set 方法。如图-2 所示：


```
package bean;
/**
 * 实体类：用户
 */
public class User {
    private String name;
    private int age;
    private Course course;
    private String[] interest;

    public String[] getInterest() {
        return interest;
    }
    public void setInterest(String[] interest) {
        this.interest = interest;
    }
    public Course getCourse() {
        return course;
    }
    public void setCourse(Course course) {
        this.course = course;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
}
```

图 - 2

步骤二：创建 el01.jsp 页面

新建工程后，添加 el01.jsp 页面，使用小脚本创建 User 类型的对象。将对象绑定在 request 对象中，使用 Java 代码输出绑定的对象的各个属性。如图-3 所示：

```

1 <%@ page contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8"%>
3 <%@ page import="bean.*" %>
4 <html>
5   <head></head>
6   <body>
7     <!-- 访问Bean属性 -->
8     <%
9       Course course = new Course();
10      course.setCourseId(1);
11      course.setCourseName("Servlet");
12      User u1 = new User();
13      u1.setName("Luffy");
14      u1.setAge(17);
15      u1.setCourse(course);
16      u1.setInterest
17        (new String[]{"Sleeping","Eating"});
18      request.setAttribute("user",u1);
19    %>
20    <!-- 1.Java代码 -->
21    <%
22      User u = (User)request.getAttribute("user");
23      out.println(u.getName());
24      out.println(u.getAge());
25    %><hr/>

```

图 - 3

步骤三：使用 EL 表达式输出属性

在 e101.jsp 页面中，添加 EL 表达式，输出姓名、年龄属性的值，输出引用类型属性课程的课程名称的值。如图-4 所示：

```

<!-- 2. EL表达式1 -->
2. EL表达式1<hr/>
<!-- 2.1 基本类型 -->
姓名: ${user.name}<br>
年龄: ${user.age}<br>
<!-- 2.2 引用类型 -->
课程名: ${user.course.courseName}<br> <br/>

```

图-4

步骤四：部署工程，查看 e101.jsp 页面的输出结果

部署工程，访问 e101.jsp 页面，结果如图-5 所示。上面输出的值是使用 java 代码，下面输出的值使用的是 EL 表达式的方式。从代码角度可以看出 EL 表达式大大简化了 JSP 页面的代码形式。

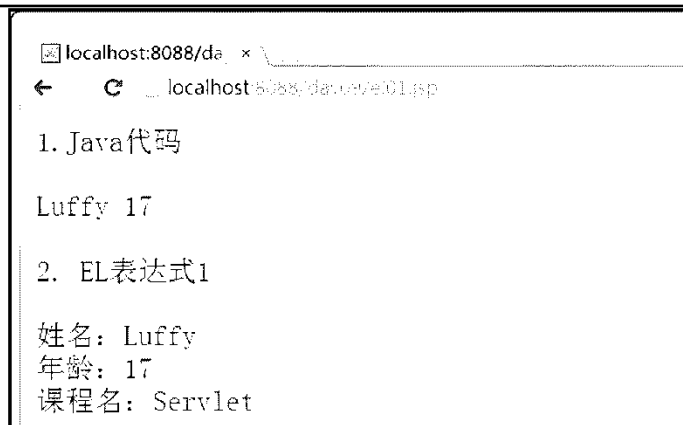


图 - 5

- **完整代码**

Course.java 文件代码如下：

```
package bean;
/**
 * 实体类：课程
 */
public class Course {
    private int courseId;
    private String courseName;
    public int getCourseId() {
        return courseId;
    }
    public void setCourseId(int courseId) {
        this.courseId = courseId;
    }
    public String getCourseName() {
        return courseName;
    }
    public void setCourseName(String courseName) {
        this.courseName = courseName;
    }
}
```

User.java 文件代码如下：

```
package bean;
/**
 * 实体类：用户
 */
public class User {
    private String name;
    private int age;
    private Course course;
    private String[] interest;

    public String[] getInterest() {
        return interest;
    }
    public void setInterest(String[] interest) {
        this.interest = interest;
    }
}
```

```
public Course getCourse() {
    return course;
}
public void setCourse(Course course) {
    this.course = course;
}
public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}
public int getAge() {
    return age;
}
public void setAge(int age) {
    this.age = age;
}
}
```

e101.jsp 文件代码如下：

```
<%@ page contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="bean.*" %>
<html>
<head></head>
<body>
<!-- 访问 Bean 属性 -->
<%
    Course course = new Course();
    course.setCourseId(1);
    course.setCourseName("Servlet");
    User u1 = new User();
    u1.setName("Luffy");
    u1.setAge(17);
    u1.setCourse(course);
    u1.setInterest
        (new String[]{"Sleeping","Eating"});
    request.setAttribute("user",u1);
%>
<!-- 1.Java 代码 -->
1.Java 代码<hr/>
<%
    User u = (User)request.getAttribute("user");
    out.println(u.getName());
    out.println(u.getAge());
%><br/><br/>
<!-- 2. EL 表达式 1 -->
2. EL 表达式 1<hr/>
<!-- 2.1 常用类型-->
姓名：${user.name}<br>
年龄：${user.age}<br>
<!-- 2.2 引用类型 -->
课程名：${user.course.courseName}<br>    <br/>
</body>
</html>
```

2. 访问 Bean 属性案例-2

- 问题

使用方括号形式的 EL 表达式访问 Bean 属性。

- 方案

访问 User 对象的姓名属性，使用变量的形式来访问 User 对象的年龄，使用下标的形式访问爱好属性中的第二个爱好。

- 步骤

步骤一：新建 Course 类、User 类

同上例中的 Course、User 类。

步骤二：创建 User 类型对象，绑定到 request 对象中

如图-6 所示：

```
<%@ page contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="bean.*" %>
<html>
    <head></head>
    <body>
        <!-- 访问Bean属性 -->
        <%
            Course course = new Course();
            course.setCourseId(1);
            course.setCourseName("Servlet");
            User u1 = new User();
            u1.setName("Luffy");
            u1.setAge(17);
            u1.setCourse(course);
            u1.setInterest
                (new String[]{"Sleeping","Eating"});
            request.setAttribute("user",u1);
        %>
```

图- 6

步骤三：修改 e101.jsp 内容，使用 EL 表达式访问属性

在 EL 表达式中，使用方括号形式来访问绑定对象的属性。如图-7 所示：

```
<%@ page contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="bean.*" %>
<html>
    <head></head>
    <body>
        <!-- 访问Bean属性 -->
        <%
            Course course = new Course();
            course.setCourseId(1);
            course.setCourseName("Servlet");
            User u1 = new User();
            u1.setName("Luffy");
            u1.setAge(17);
            u1.setCourse(course);
            u1.setInterest
                (new String[]{"Sleeping","Eating"});
            request.setAttribute("user",u1);
        %>
        <!-- 3. EL表达式2 -->
        姓名: ${user["name"]} <br>
```

图 - 7

可以在 request 对象中绑定一个属性名称,然后在 EL 表达式中通过这个绑定名来获取到属性名称,进而间接访问对象属性。如图-8 所示:

```
<%@ page contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="bean.*" %>
<html>
    <head></head>
    <body>
        <!-- 访问Bean属性 -->
        <%
            Course course = new Course();
            course.setCourseId(1);
            course.setCourseName("Servlet");
            User u1 = new User();
            u1.setName("Luffy");
            u1.setAge(17);
            u1.setCourse(course);
            u1.setInterest
                (new String[]{"Sleeping","Eating"});
            request.setAttribute("user",u1);
        %>
        <!-- 3. EL表达式2 -->
        姓名: ${user["name"]} <br>
        <%
            request.setAttribute("userAge","age");
        %>
        年龄: ${user[userAge]} <br/>
```

图 - 8

可以在方括号内填写下标来访问对象的数组属性中的某一个值。如图-9 所示:

```
<%@ page contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="bean.*" %>
<html>
    <head></head>
    <body>
        <!-- 访问Bean属性 -->
        <%
            Course course = new Course();
            course.setCourseId(1);
            course.setCourseName("Servlet");
            User u1 = new User();
            u1.setName("Luffy");
            u1.setAge(17);
            u1.setCourse(course);
            u1.setInterest
                (new String[]{"Sleeping","Eating"});
            request.setAttribute("user",u1);
        %>
        3. EL表达式2<hr/>
        <!-- 3. EL表达式2 -->
        姓名: ${user["name"]} <br>
        <%
            request.setAttribute("userAge","age");
        %>
        年龄: ${user[userAge]} <br/>
        爱好: ${user.interest[1]}<br><br/>
```

图 - 9

步骤四：运行查看结果

3. EL表达式2

姓名: Luffy
年龄: 17
爱好: Eating

图 - 10

步骤五：修改 e101.jsp 内容，设定具体的访问范围

如图-11 所示，可以设定检索对象的范围，如 request 对应 requestScope，session 对应 sessionScope。

```
<%@ page contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="bean.*" %>
<html>
    <head></head>
    <body>
        <!-- 访问Bean属性 -->
        <%
            Course course = new Course();
            course.setCourseId(1);
            course.setCourseName("Servlet");
            User u1 = new User();
            u1.setName("Luffy");
            u1.setAge(17);
            u1.setCourse(course);
            u1.setInterest
                (new String[]{"Sleeping","Eating"});
            request.setAttribute("user",u1);
        %>
        4. 访问范围<hr/>
        <!-- 4. 访问范围 -->
        用户姓名: ${requestScope.user.name}
    </body>
</html>
```

图 - 11

步骤六：运行查看结果

4. 访问范围

用户姓名: Luffy

图 - 12

• 完整代码

el01.jsp 文件代码如下：

```
<%@ page contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="bean.*" %>
<html>
    <head></head>
    <body>
        <!-- 访问 Bean 属性 -->
        <%
            Course course = new Course();
            course.setCourseId(1);
            course.setCourseName("Servlet");
            User u1 = new User();
            u1.setName("Luffy");
            u1.setAge(17);
            u1.setCourse(course);
            u1.setInterest
                (new String[]{"Sleeping","Eating"});
```



```

request.setAttribute("user",u1);
%>
<!-- 1.Java 代码 -->
1.Java 代码<hr/>
<%
    User u = (User)request.getAttribute("user");
    out.println(u.getName());
    out.println(u.getAge());
%><br/><br/>
<!-- 2. EL 表达式 1 -->
2. EL 表达式 1<hr/>
<!-- 2.1 常用类型-->
姓名:${user.name}<br>
年龄:${user.age}<br>
<!-- 2.2 引用类型 -->
课程名:${user.course.courseName}<br>    <br/>

3. EL 表达式 2<hr/>
<!-- 3. EL 表达式 2 -->
姓名:${user["name"]} <br>
<%
    request.setAttribute("userAge","age");
%>
年龄:${user[userAge]} <br/>
爱好:${user.interest[1]}<br><br/>

4. 访问范围<hr/>
<!-- 4. 访问范围 -->
用户姓名:${requestScope.user.name}
</body>
</html>

```

3. 使用 EL 表达式运算的案例

- 问题

使用 EL 表达式进行算数运算、关系运算、empty 运算。

- 方案

进行 empty 运算时，分别对空字符串、空集合、null 进行判断。

- 步骤

步骤一：新建 e102.jsp 文件

在 e102.jsp 页面中使用不同的运算符进行运算，并输出结果查看。如图-13 所示：

```
<%@ page contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="java.util.*" %>
<html>
    <head></head>
    <body>
        <!-- 使用EL表达式进行运算 -->
        <!-- 1. 算术运算 -->
        加: ${1+2}<br>
        加: ${"123" + "456"}<br><hr/>
        <!-- 加号只能进行加法运算，不能连接 -->

        <!-- 2. 关系运算 -->
        ${"123" == "123"}<br>
        ${"123" eq "123" }<br>
        <% pageContext.setAttribute("str","abc"); %>
        ${"abc" eq str }<br><hr/>

        <!-- 3. empty运算 -->
        <%
            request.setAttribute("str1","");
            List list = new ArrayList();
            request.setAttribute("list1", list);
            request.setAttribute("obj", null);
        %>
        空字符串: ${empty str1}<br>
        找不到绑定名对象: ${empty xxx }<br>
        集合内容为空: ${empty list1 }<br>
        null的结果: ${empty obj }
    </body>
</html>
```

图 - 13

步骤二：运行查看结果

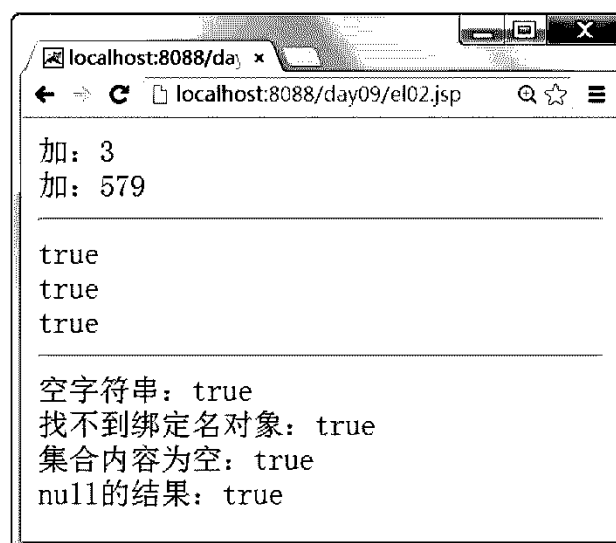


图 - 14

- **完整代码**

e102.jsp 文件代码如下：

```
<%@ page contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="java.util.*" %>
<html>
<head></head>
<body>
<!-- 使用 EL 表达式进行运算 -->
<!-- 1. 算术运算 -->
加：${1+2}<br>
加：${"123" + "456"}<br><hr/>
<!-- 加号只能进行加法运算，不能连接 -->

<!-- 2. 关系运算 -->
${"123" == "123"}<br>
${"123" eq "123" }<br>
<% pageContext.setAttribute("str","abc"); %>
${"abc" eq str }<br><hr/>

<!-- 3. empty 运算 -->
<%
    request.setAttribute("str1","");
    List list = new ArrayList();
    request.setAttribute("list1", list);
    request.setAttribute("obj", null);
%>
空字符串：${empty str1}<br>
找不到绑定名对象：${empty xxx }<br>
集合内容为空：${empty list1 }<br>
null 的结果：${empty obj }
</body>
</html>
```

4. 使用 EL 表达式获取请求参数

- **问题**

使用 EL 表达式获取请求参数的值。

- **方案**

在地址栏中直接输入 GET 方式提交的多个 Name-Value，使用 EL 表达式进行输出。

- **步骤**

步骤一：新建 e103.jsp 页面

如图-15 所示，使用 param 访问 Name-Value 中的 Value，使用 paramValues 访问 Name-Values 中的多值。

```

1 <%@ page contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8"%>
3 <html>
4   <head>
5     <title>Insert title here</title>
6   </head>
7   <body>
8     <!-- 获取请求参数 -->
9     姓名: ${param.name}<br/><hr/>
10    爱好: ${paramValues.interest[1]}
11    爱好: ${paramValues.interest[0]}
12  </body>
13 </html>

```

图 - 15

步骤二：在地址栏输入地址，查看输出结果

在地址栏中输入 `http://localhost:8080/day09/e103.jsp?name=Luffy&interest=Sleeping&interest=Eating` 后，查看运行结果如图-16 所示：

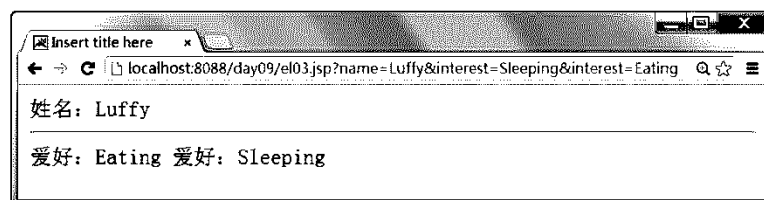


图 - 16

• 完整代码

e103.jsp 文件的代码如下：

```

<%@ page contentType="text/html; charset=UTF-8"
  pageEncoding="UTF-8"%>
<html>
<head>
  <title>Insert title here</title>
</head>
<body>
  <!-- 获取请求参数 -->
  姓名: ${param.name}<br/><hr/>
  爱好: ${paramValues.interest[1]}
  爱好: ${paramValues.interest[0]}
</body>
</html>

```

5. if 标签案例

• 问题

使用 if 标签输出员工性别。

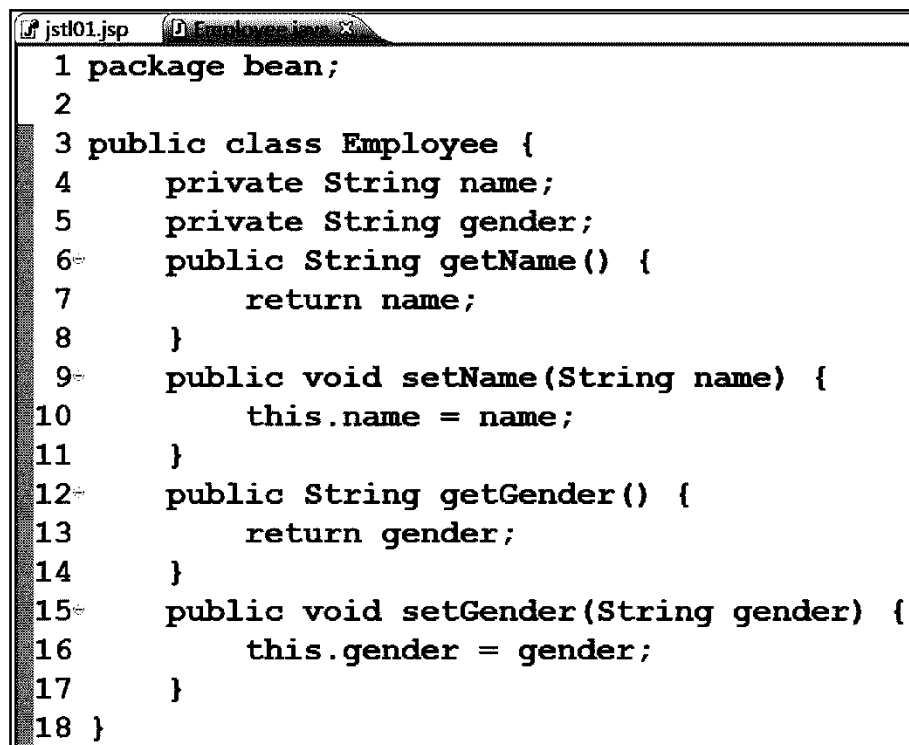
- 方案

添加变量，保留与“男”进行比对的结果，从而判断是否应该输出“女”。

- 步骤

步骤一：新建 Employee.java 文件

新建具有姓名、性别的实体类 bean.Employee 类，添加 get/set 方法。如图-17 所示：



```
1 package bean;
2
3 public class Employee {
4     private String name;
5     private String gender;
6     public String getName() {
7         return name;
8     }
9     public void setName(String name) {
10        this.name = name;
11    }
12    public String getGender() {
13        return gender;
14    }
15    public void setGender(String gender) {
16        this.gender = gender;
17    }
18 }
```

图 - 17

步骤二：新建 jstl01.jsp 页面

新建 jstl01.jsp 页面，添加 taglib 指令后，使用 if 标签判断绑定在 request 对象中的 Employee 对象的性别属性，并输出中文对应文本。如图-18 所示：

```

1 <%@ page contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8"%>
3 <%@ page import="bean.Employee" %>
4 <%@ taglib uri="http://java.sun.com/jsp/jstl/core"
5   prefix="c" %>
6 <html>
7   <head></head>
8   <body>
9   <%
10       Employee emp = new Employee();
11       emp.setName("Luffy");
12       emp.setGender("m");
13       request.setAttribute("emp", emp);
14   %>
15   if标签: <hr/>
16   员工姓名: ${emp.name}<br/>
17   员工性别: <c:if test="${emp.gender=='m'}"
18 记录test结果 var="rs" scope="request">男</c:if>
19 的临时变量 <c:if test="${!rs}">女</c:if>
20   </body>
21 </html>

```

引入标准标签库

绑定对象

图 - 18

步骤三：运行查看结果



图 - 19

• 完整代码

Employee.java 文件代码如下：

```

package bean;

public class Employee {
    private String name;
    private String gender;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getGender() {

```

```
        return gender;
    }
    public void setGender(String gender) {
        this.gender = gender;
    }
}
```

jstl01.jsp 文件代码如下：

```
<%@ page contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="bean.Employee" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core"
    prefix="c" %>
<html>
<head></head>
<body>
<%
    Employee emp = new Employee();
    emp.setName("Luffy");
    emp.setGender("m");
    request.setAttribute("emp", emp);
%>
if 标签：<hr/>
员工姓名：${emp.name}<br/>
员工性别：<c:if test="${emp.gender=='m'}"
    var="rs" scope="request">男</c:if>
    <c:if test="${!rs}">女</c:if>
</body>
</html>
```

6. choose 标签案例

- 问题

使用 choose 标签输出 Employee 对象的性别属性值。

- 方案

通过子标签 when 进行判断后输出性别属性对应的中文说明。

- 步骤

步骤一：新建 jstl02.jsp 文件

引入标准标签库后，使用 choose 标签对绑定的对象的属性进行判断并输出对应的文字说明。如图-20。

```

1 <%@ page contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8"%>
3 <%@ page import="bean.Employee" %>
4 <%@ taglib uri="http://java.sun.com/jsp/jstl/core"
5   prefix="c"%>
6 <html>
7   <head>
8     <title>Insert title here</title>
9   </head>
10  <body>
11    <%
12      Employee emp = new Employee();
13      emp.setName("Luffy");
14      emp.setGender("m");
15      request.setAttribute("emp", emp);
16    %>
17    choose标签: <hr/>
18    员工性别:
19    <c:choose>
20      <c:when test="${emp.gender=='m'}">男</c:when>
21      <c:otherwise>女</c:otherwise>
22    </c:choose>
23  </body>
24 </html>

```

图 - 20

步骤二：运行查看结果

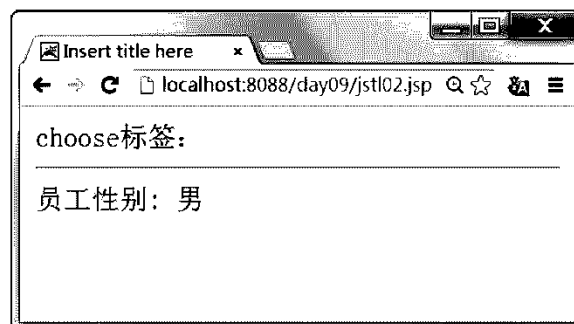


图 - 21

• 完整代码

jstl02.jsp 文件代码如下：

```

<%@ page contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<%@ page import="bean.Employee" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core"
   prefix="c"%>
<html>
<head>
  <title>Insert title here</title>
</head>
<body>
<%
  Employee emp = new Employee();

```



```
emp.setName("Luffy");
emp.setGender("m");
request.setAttribute("emp", emp);
%>
choose 标签 : <hr/>
员工性别:
<c:choose>
    <c:when test="{emp.gender=='m'}">男</c:when>
    <c:otherwise>女</c:otherwise>
</c:choose>
</body>
</html>
```

7. forEach 标签案例

- 问题

使用 forEach 标签输出绑定的集合对象中的 Employee 对象的属性。

- 方案

使用 forEach 标记的 var 属性指定每次从集合中获取的一个对象，使用 items 属性指定需要遍历的集合，使用 varStatus 属性访问遍历集合过程中下标及记录的数目。

- 步骤

步骤一：新建 jst103.jsp 文件

绑定具有两个元素的集合到 request 对象中，使用 forEach 标签输出对象的属性。如图-22 所示：

[illegible]

图 - 26

步骤六：运行查看结果

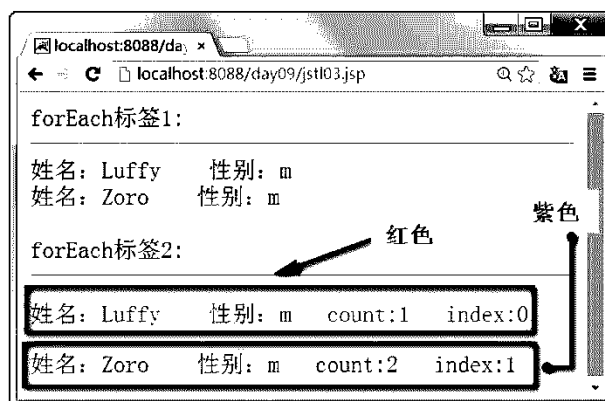


图 - 27

步骤七：增加 `ActionServlet` 类

创建 Employee 对象的集合后绑定到 request 中 转发至 jstl04.jsp 页面进行输出。



- 完整代码

jstl03.jsp 文件代码如下：

121

```
</c:forEach>
</body>
</html>
```

ActionServlet.java 文件代码如下：

```
package web;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.List;

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

import bean.Employee;

public class ActionServlet extends HttpServlet {

    public void service(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        String uri = request.getRequestURI();
        String path = uri.substring
            (uri.lastIndexOf("/") + 1,
            uri.lastIndexOf("."));
        //判断路径
        if (path.equals("list")) {
            List<Employee> emps = new ArrayList<Employee>();
            Employee e1 = new Employee();
            e1.setName("Luffy");
            e1.setGender("m");
            emps.add(e1);
            Employee e2 = new Employee();
            e2.setName("Zoro");
            e2.setGender("m");
            emps.add(e2);
            //绑定集合到 request
            request.setAttribute("emps", emps);
            request.getRequestDispatcher("jstl04.jsp")
                .forward(request, response);
        }
    }
}
```

web.xml 文件代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <display-name></display-name>
    <servlet>
        <servlet-name>ActionServlet</servlet-name>
        <servlet-class>web.ActionServlet</servlet-class>
    </servlet>
    <servlet-mapping>
```

[illegible]

- 问题

• 方案

• 步骤

新建 `HelloTag` 类，继承自 `SimpleTagSupport` 类，为了接收页面中指定的参数，需要定义两个属性，并添加 `get/set` 方法。注意：属性名称一定与使用标签时，暴露的属性

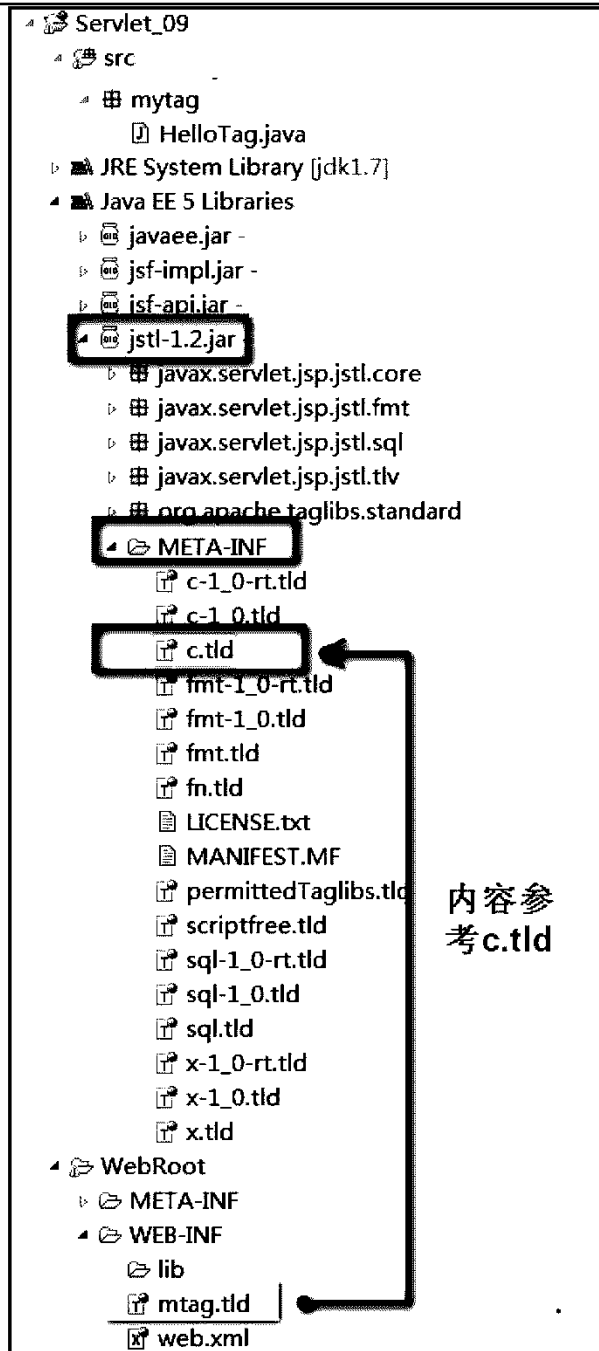
名称一致。重写 doTag 方法，将标签要执行的动作在该方法中进行定义。如图-32 所示：

```
public class HelloTag extends SimpleTagSupport {
    private String str;
    private int count;
    @Override
    public void doTag() throws JspException, IOException {
        PageContext ctx=(PageContext) getJspContext();
        JspWriter out = ctx.getOut();
        for(int i=0;i<count;i++){
            out.println(str+"<br/>");
        }
    }
    public String getStr() {
        return str;
    }
    public void setStr(String str) {
        this.str = str;
    }
    public int getCount() {
        return count;
    }
    public void setCount(int count) {
        this.count = count;
    }
}
```

图 - 32

步骤二：新建 mtag.tld 文件

在 WEB-INF 目录下，新建 mtag.tld 文件，内容可以参考标准标签库中的 c.tld 文件的内容。c.tld 文件的目录如图-33 所示：



内容参
考c.tld

图 - 33

在 mtag.tld 文件中，设定一对 tag 标记即可。如图-34 所示：

```
<?xml version="1.0" encoding="UTF-8" ?>
<taglib xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-jsp2_1.xsd"
  version="2.1">
  <tlib-version>1.1</tlib-version>
  <short-name>c3</short-name>
  <uri>http://www.tarena.com.cn/mytag</uri>
  <tag>
    <description>根据指定的次数打印文本</description>
    <name>pt</name>
    <tag-class>mytag.HelloTag</tag-class>
    <body-content>empty</body-content>
    <attribute>
      <description>Content</description>
      <name>str</name>
      <required>false</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
    <attribute>
      <description>Count</description>
      <name>count</name>
      <required>false</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
  </tag>
</taglib>
```

自定义的prefix

自定义的URI

图 - 34

步骤三：新建 myTag.jsp 文件

```
1 <%@ page contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8"%>
3 <%@ taglib uri="http://www.tarena.com.cn/mytag"
4   prefix="c3" %>
5 <html>
6   <head>
7     <title>Insert title here</title>
8   </head>
9   <body>
10    <c3:pt count="10" str="Hello Goddess"/>
11  </body>
12 </html>
```

与tld文件内容保持一致

使用自定义标记

图 - 35

步骤四：运行，查看结果

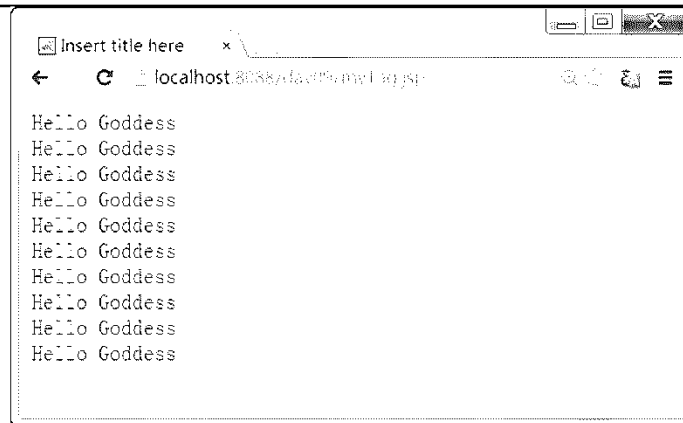


图 - 36

• 完整代码

mytag.HelloTag.java 文件代码如下：

```
package mytag;

import java.io.IOException;

import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class HelloTag extends SimpleTagSupport {
    private String str;
    private int count;
    @Override
    public void doTag() throws JspException, IOException {
        PageContext ctx=(PageContext)getContext();
        JspWriter out = ctx.getOut();
        for(int i=0;i<count;i++){
            out.println(str+"<br/>");
        }
    }
    public String getStr() {
        return str;
    }
    public void setStr(String str) {
        this.str = str;
    }
    public int getCount() {
        return count;
    }
    public void setCount(int count) {
        this.count = count;
    }
}
```

mtag.tld 文件代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<taglib xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-jsp-taglibrary_2_1.xsd"
```

```
version="2.1">
<tlib-version>1.1</tlib-version>
<short-name>c3</short-name>
<uri>http://www.tarena.com.cn/mytag</uri>
<tag>
  <description>根据指定的次数打印文本</description>
  <name>pt</name>
  <tag-class>mytag.HelloTag</tag-class>
  <body-content>empty</body-content>
  <attribute>
    <description>Content</description>
    <name>str</name>
    <required>false</required>
    <rtexprvalue>false</rtexprvalue>
  </attribute>
  <attribute>
    <description>Count</description>
    <name>count</name>
    <required>false</required>
    <rtexprvalue>false</rtexprvalue>
  </attribute>
</tag>
</taglib>
```

myTag.jsp 文件的代码如下：

```
<%@ page contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib uri="http://www.tarena.com.cn/mytag"
    prefix="c3" %>
<html>
<head>
  <title>Insert title here</title>
</head>
<body>ac
<c3:pt count="10" str="Hello Goddess"/>
</body>
</html>
```

9. 员工管理——使用 EL 表达式和 JSTL 实现 JSP 页面

- 问题

使用 EL 表达式和 JSTL 改写员工管理的增删改查功能。

- 方案

entity、dao、web 包下面的类不用修改，参考前面案例拷贝。web.xml 文件无需修改，参考前面案例拷贝。updateEmp.jsp 页面、listEmp.jsp、error.jsp 页面中的 Java 代码由 EL 表达式和 JSTL 进行替换。addEmp.jsp 页面中无 Java 代码，无需修改，参考前面案例拷贝即可。

- 步骤

步骤一： 参考前面案例中的代码，构建如下结构的 Web 项目

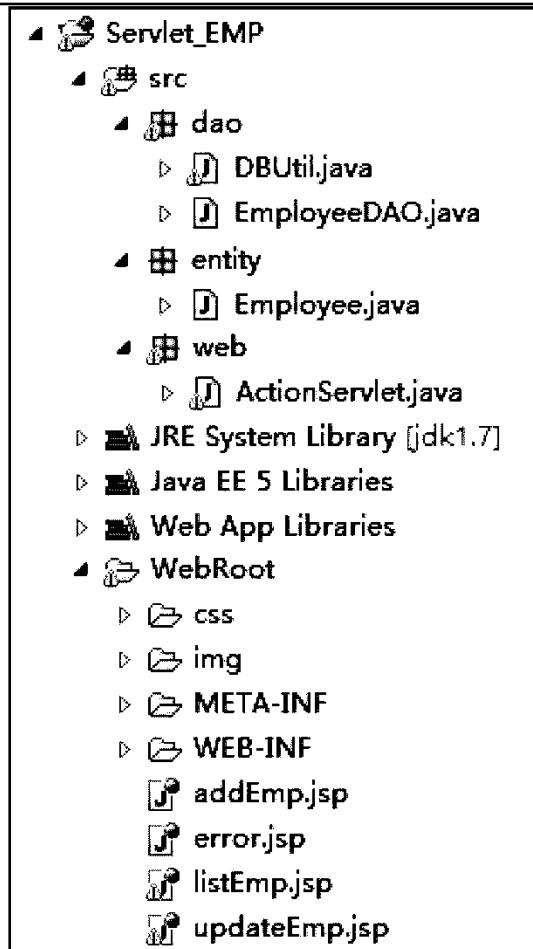


图 - 37

步骤二：修改 listEmp.jsp 文件

删除 listEmp.jsp 中的小脚本，替换为 JSTL，如图-38 所示：

```
<c:forEach var="emp" items="${emps}" varStatus="s">
  <tr class="row${s.index%2+1}">
    <td>${emp.id}</td>
    <td>${emp.name}</td>
    <td>${emp.salary}</td>
    <td>${emp.age}</td>
    <td>
      <a href="delete.do?id=${emp.id}"
        onclick="return confirm('是否确认删除${emp.name}信息? ');">删除</a>
      <a href="load.do?id=${emp.id}">修改</a>
    </td>
  </tr>
</c:forEach>
```

图 - 38

步骤三：修改 uploadEmp.jsp 文件

删除 uploadEmp.jsp 文件中的小脚本，替换为 JSTL 的。如图-39 所示：

```
<form action="update.do?id=${emp.id}" method="post">
  <table cellpadding="0" cellspacing="0" border="0"
    class="form_table">
    <tr>
      <td valign="middle" align="right">
        编号:
      </td>
      <td valign="middle" align="left">
        ${emp.id}
      </td>
    </tr>
    <tr>
      <td valign="middle" align="right">
        姓名:
      </td>
      <td valign="middle" align="left">
        <input type="text" class="inputgri"
          name="name" value="${emp.name}"/>
      </td>
    </tr>
    <tr>
      <td valign="middle" align="right">
        薪水:
      </td>
      <td valign="middle" align="left">
        <input type="text" class="inputgri"
          name="salary" value="${emp.salary}"/>
      </td>
    </tr>
    <tr>
      <td valign="middle" align="right">
        年龄:
      </td>
      <td valign="middle" align="left">
        <input type="text" class="inputgri"
          name="age" value="${emp.age}"/>
      </td>
    </tr>
  </table>
  <p>
    <input type="submit" class="button" value="修改" />
  </p>
</form>
```

图 - 39

步骤四：修改 error.jsp 页面

```
<%@ page contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c"
    uri="http://java.sun.com/jsp/jstl/core" %>
<html>
    <head>
        <title>错误提醒</title>
    </head>
    <body style="font-size:24px">

        <c:if test="${err_msg==null}" var="msg"
            scope="request"></c:if>
        <c:if test="${!msg}">${err_msg}</c:if>

        系统异常 <a href="list.do">返回</a>
    </body>
</html>
```

图 - 40

• 完整代码

listEmp.jsp 文件代码如下：

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ page import="entity.*,java.util.*"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
<title>员工列表</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<link rel="stylesheet" type="text/css" href="css/style.css" />
</head>
<body>
    <div id="wrap">
        <div id="top_content">
            <div id="header">
                <div id="righthead">
                    <p>
                        2009/11/20 <br />
                    </p>
                </div>
                <div id="topheader">
                    <h1 id="title">
                        <a href="#">main</a>
                    </h1>
                </div>
            <div id="navigation"></div>
        </div>
        <div id="content">
            <p id="whereami"></p>
            <h1>员工信息</h1>
            <table class="table">
                <tr class="table_header">
                    <td>编号</td>
                    <td>姓名</td>
                    <td>薪水</td>
                    <td>年龄</td>
```



```
<td>操作</td>  
</tr>  
<c:forEach var="emp" items="${emps}" varStatus="s">  
    <tr class="row${s.index%2+1}">  
        <td>${emp.id}</td>  
        <td>${emp.name}</td>  
        <td>${emp.salary}</td>  
        <td>${emp.age}</td>  
        <td>  
            <a href="delete.do?id=${emp.id}"  
                onclick="return confirm('是否确认删除${ emp.name }  
信息? '); ">删除</a>&nbsp;   <a href="load.do?id=${emp.id}">修改</a>  
        </td>  
    </tr>  
</c:forEach>  
</table>  
<p>  
    <input type="button" class="button" value="增加员工"  
        onclick="location='addEmp.jsp'" />  
</p>  
</div>  
</div>  
<div id="footer">  
    <div id="footer bg">ABC@126.com</div>  
</div>  
</div>  
</body>  
</html>
```

uploadEmp.jsp 文件代码如下：

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ page import="entity.*" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
    <title>修改员工信息</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <link rel="stylesheet" type="text/css" href="css/style.css" />
</head>

<body>
    <div id="wrap">
        <div id="top_content">
            <div id="header">
                <div id="righthead" >
                    <p>
                        2009/11/20
                    <br />
                </p>
            </div>
            <div id="topheader">
                <h1 id="title">
                    <a href="#">Main</a>
                </h1>
            </div>
            <div id="navigation">
            </div>
        </div>
        <div id="content">
            <p id="whereami">
            </p>
        </div>
    </div>
</body>
</html>
```

```

<h1>
    更新员工
</h1>
<form action="update.do?id=${emp.id}" method="post">
    <table cellpadding="0" cellspacing="0" border="0"
        class="form table">
        <tr>
            <td valign="middle" align="right">
                编号:
            </td>
            <td valign="middle" align="left">
                ${emp.id}
            </td>
        </tr>
        <tr>
            <td valign="middle" align="right">
                姓名:
            </td>
            <td valign="middle" align="left">
                <input type="text" class="inputgri"
                    name="name" value="${emp.name}" />
            </td>
        </tr>
        <tr>
            <td valign="middle" align="right">
                薪水:
            </td>
            <td valign="middle" align="left">
                <input type="text" class="inputgri"
                    name="salary" value="${emp.salary}" />
            </td>
        </tr>
        <tr>
            <td valign="middle" align="right">
                年龄:
            </td>
            <td valign="middle" align="left">
                <input type="text" class="inputgri"
                    name="age" value="${emp.age}" />
            </td>
        </tr>
    </table>
    <p>
        <input type="submit" class="button" value="修改" />
    </p>
</form>
</div>
<div id="footer">
    <div id="footer bg">
        ABC@126.com
    </div>
</div>
</div>
</body>
</html>

```

error.jsp 文件的代码如下：

```

<%@ page contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c"
    uri="http://java.sun.com/jsp/jstl/core" %>

```

```
<html>
<head>
  <title>错误提醒</title>
</head>
<body style="font-size:24px">

  <c:if test="${err msg==null}" var="msg"
    scope="request"></c:if>
  <c:if test="${!msg}">${err_msg}</c:if>

  系统异常<a href="list.do">返回</a>
</body>
</html>
```

课后作业

1. 简述什么是 EL 表达式？
2. 简述 EL 表达式的作用。
3. 下列说法正确的是 ()。
 - A. EL 表达式查找对象的范围依次是 request ,pageContext ,session ,application。
 - B. 使用 EL 表达式输出对象的属性值时，如果属性值为空，则输出空白。
 - C. 如果指定了对象的查找范围，那么如果在该范围内没有找到绑定的对象则不会再去其他范围进行查找了。
 - D. 使用 EL 表达式输出 Bean 属性时，不允许使用下标的形式。
4. 简述 JSTL 的作用。
5. 下列说法正确的是 ()。
 - A. 核心标签中的 if 标签、choose 标签、forEach 标签都有 test 属性。
 - B. 核心标签中的 choose 标签内可以包含 when 和 otherwise 子标签。
 - C. 无法获取 forEach 标签迭代时的下标。
 - D. 自定义标签时可以继承自 javax.servlet.jsp.tagext.SimpleTagSupport 类。
6. 简述自定义标签的步骤及标签的运行原理。
7. 使用 JSTL 实现 NETCTOSS 中账务账号的 CRUD。