

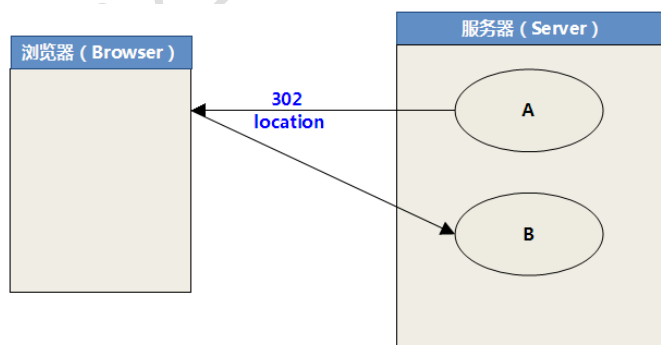
## 知识点列表

编号	名称	描述	级别
1	创建 cookie	掌握创建 Cookie 的语句	**
2	查找 cookie	掌握查找 Cookie 的语句	**
3	cookie 保存时的编码问题	了解并掌握 Cookie 操作的存在编码问题及解决方法	**
4	Cookie 的保存时间	了解 Cookie 的保存时间的操作	*
5	删除 cookie	掌握删除 Cookie 的方法	*
6	cookie 的路径问题	了解 Cookie 路径问题的存在及解决办法	*
7	Cookie 的限制	了解一般 Cookie 的限制,尤其在编程过程中注意 Cookie 是可以被禁用的	*
8	什么是 session?	了解	*
9	如何获得 session 对象	了解	*
10	HttpSession 接口提供的一些方法	了解并掌握常用的一些操作 Session 的方法	**
11	session 超时	了解什么是 Session 超时以及其原理,尤其注意 Session 的应用离不开 Cookie	*
12	删除 session	掌握删除 Session 的一般用法	*

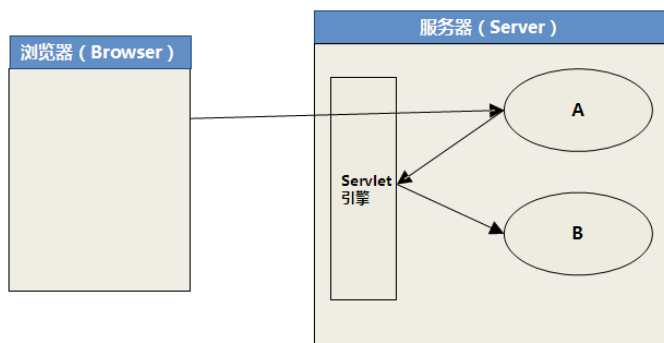
注:    \*\*"理解级别    \*\*\*"掌握级别    \*\*\*\*"应用级别

### 1. 转发与重定向的区别 \*

**重定向**



## 转发



### 重定向和转发的区别

#### 1) 地址

- ✓ 转发的地址必须是同一个应用内部的某个组件（不能跨应用，不能跨服务器）

比如：

地址 1 <http://localhost:8080/web06/aaa.jsp>

地址 2 <http://localhost:8080/web06/bbb.jsp>

地址 3 <http://localhost:8080/web07/ccc.jsp>

地址 4 <http://www.tarena.com.cn>

在应用 web06 内部的组件 aaa.jsp 可以将信息转发到地址 2（同一应用），但是不可以转发到地址 3（跨应用）和地址 4（跨服务器）

- ✓ 重定向的地址没有限制

#### 2) 能否共享 request

- ✓ 转发可以

- ✓ 重定向不行

原因是转发是一次请求，重定向为两次请求，Request 的生命周期只能在一次请求内，请求结束，Request 被删除

#### 3) 浏览器地址栏的地址是否变化

- ✓ 转发不变

- ✓ 重定向会变

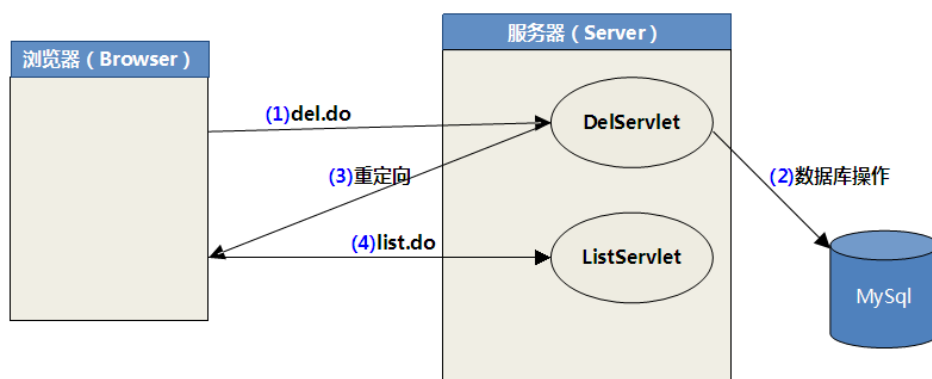
#### 4) 事件是否处理完毕

- ✓ 转发是一件事未做完

- ✓ 重定向是一件事已经做完

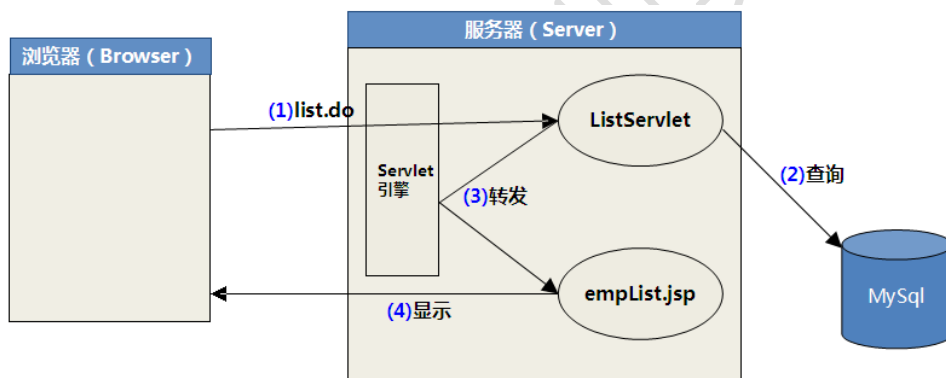
### 什么时候用重定向？

比如用户做删除操作时，删除操作已做完，重定向访问 list.do



### 什么时候用转发？

比如 (1) 用户调用 list.do, (2) 由 ListServlet 到数据库中查询数据, (3) ListServlet 将查询到的数据通过 Servlet 引擎转发给负责显示的 empList.jsp, (4) empList.jsp 将数据通过友好的界面显示给用户



## 2. 状态管理

### 1) 什么是状态管理

将客户端(一般是浏览器)与服务器之间的多次交互当作一个整体来看待,即将多次操作所涉及的数据记录下来。

### 2) 怎样进行状态管理

- ✓ 第一种方式, cookie (在客户端管理用户的状态)
- ✓ 第二种方式, session (在服务器端管理用户的状态)

### 3) cookie

#### a. 什么是 cookie

浏览器在访问服务器时,服务器将一些数据以 set-cookie 消息头的形式发送给浏览器。

浏览器会将这些数据保存起来。当浏览器再次访问服务器时，会将这些数据以 cookie 消息头的形式发送给服务器。通过这种方式，可以管理用户的状态。

**b. 创建 cookie**

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

**c. 查询 cookie**

```
//如果没有 cookie，则返回 null。
Cookie[] cookies = request.getCookies();
String name = cookie.getName();
String value = cookie.getValue();
```

**d. cookie 保存时的编码问题**

cookie 的值只能是 ascii 字符，如果是中文，需要将中文转换成 ascii 字符形式。可以使用 `URLEncoder.encode()` 方法和 `URLDecoder.decode()` 方法来进行这种转换。

**e. cookie 的保存时间**

```
cookie.setMaxAge(int seconds);
```

✓ **seconds > 0**

浏览器会将 cookie 以文件的方式保存在硬盘上。在超过指定的时间以后，会删除该文件。

✓ **seconds < 0**

**默认值**，浏览器会将 cookie 保存在内存里面。只有当浏览器关闭之后，才会删除。

✓ **seconds = 0**

立即删除该 Cookie

**f. 删除 cookie**

比如要删除一个 name 为 "username" 的 cookie。

```
Cookie c = new Cookie("username","");
c.setMaxAge(0);
response.addCookie(c);
```

**g. cookie 的路径问题**

浏览器在向服务器上的某个地址发送请求时，会先比较 cookie 的路径与向访问的路径(地址)是否匹配，只有匹配的 cookie，才会发送。cookie 的路径可以通过 `cookie.setPath(String path)` 方法来设置。

如果没有设置，则有一个缺省的路径，缺省的路径是生成该 cookie 的组件的路径。

比如：/appname/addCookie 保存了一个 cookie，

则该 cookie 的路径就是 /appname/addCookie。

**规则：**

cookie 的路径必须是要访问的路径的上层目录或者是与要访问的路径相等，浏览器才会将 cookie 发送给服务器。

一般可以设置 `setPath("/appname")`，表示访问该应用下的所有地址，均会发送

cookie。

#### h. cookie 的限制

- ✓ cookie 可以禁止
- ✓ cookie 的大小有限制(4k 左右)
- ✓ cookie 的数量也有限制(浏览器大约能保存 300 个)
- ✓ cookie 的值只能是字符串，要考虑编码问题。
- ✓ cookie 不安全

### 4) session

#### a. 什么是 session?

浏览器访问服务器时，服务器会创建一个 session 对象(该对象有一个唯一的 id，一般称为 sessionId)。服务器在缺省情况下，会将 sessionId 以 cookie 机制发送给浏览器。当浏览器再次访问服务器时，会将 sessionId 发送给服务器。服务器依据 sessionId 就可以找到对应的 session 对象。通过这种方式，就可以管理用户的状态。

#### b. 如何获得 session 对象

方式一:

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

当 flag = true:

服务器会先查看请求中是否包含 sessionId,

如果没有，则创建一个 session 对象。

如果有，则依据 sessionId 去查找对应的 session 对象，如果找到，则返回。

如果找不到，则创建一个新的 session 对象。

当 flag = false:

服务器会先查看请求中是否包含 sessionId,

如果没有,返回 null。

如果有，则依据 sessionId 去查找对应的 session 对象，如果找到，则返回。

如果找不到，返回 null。

方式二

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

与 request.getSession(true)等价。

#### c. HttpSession 接口提供的一些方法

```
//获得 sessionId。
```

```
String session.getId();
```

```
//绑定数据
```

```
session.setAttribute(String name,Object obj);
```

```
/* obj 最好实现 Serializable 接口
```

```
* (服务器在对 session 进行持久化操作时，
```

```
* 比如钝化、激活，会使用序列化协议)。 */
```

```
Object session.getAttribute(String name);
```

//如果 name 对应的值不存在，返回 null。

```
session.removeAttribute(String name);
```

#### d. session 超时

服务器会将超过指定时间的 session 对象删除(在指定的时间内，该 session 对象没有使用)。

方式一：

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

方式二：

服务器有一个缺省的超时限制，可以通过相应的配置文件来重新设置。

比如可以修改 tomcat 的 web.xml(tomcat\_home/conf 下面)。

```
<session-config>
    <session-timeout> 30</session-timeout>
</session-config>
```

另外，也可以只修改某个应用的 web.xml。

#### e. 删除 session

```
session.invalidate();
```

## 2.1. Cookie \*\*

### 2.1.1. 创建 cookie \*\*

#### 演示 1

##### 1) 新建 AddCookieServlet

添加 Cookie

```

AddCookieServlet.java  web.xml
1 package web;
2
3 import java.io.IOException;
4
5
6
7
8
9
10
11
12
13 public class AddCookieServlet extends HttpServlet {
14
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         Cookie cookie = new Cookie("username", "aaa");
23         Cookie cookie2 = new Cookie("pwd", "123");
24         response.addCookie(cookie);
25         response.addCookie(cookie2);
26         out.close();
27     }
28 }

```

## 2) web.xml

```

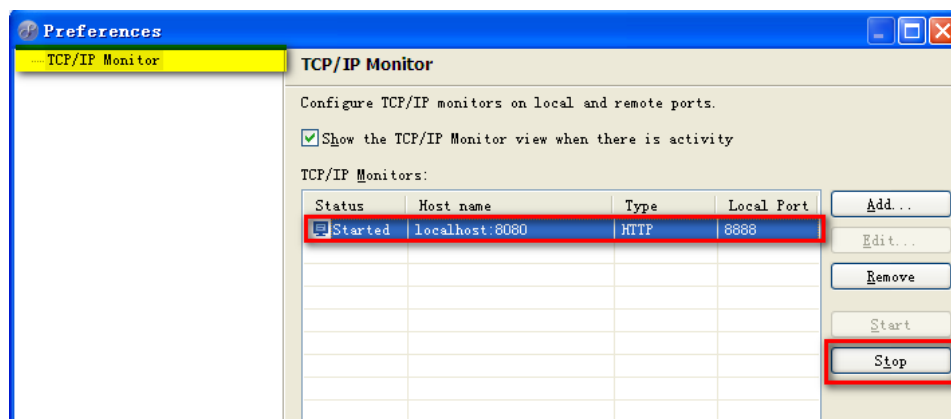
AddCookieServlet.java  web.xml
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app version="2.4"
3     xmlns="http://java.sun.com/xml/ns/j2ee"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5     xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
6     http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
7 >
8     <servlet>
9         <servlet-name>AddCookieServlet</servlet-name>
10        <servlet-class>web.AddCookieServlet</servlet-class>
11    </servlet>
12    <servlet-mapping>
13        <servlet-name>AddCookieServlet</servlet-name>
14        <url-pattern>/addCookie</url-pattern>
15    </servlet-mapping>
16 </web-app>

```

### 3) 部署项目

### 4) 启动 TCP/IP Monitor

请参考 Servlet day04 第 2.4 章节 MyEclipse 工具演示 : TCP/IP Monitor



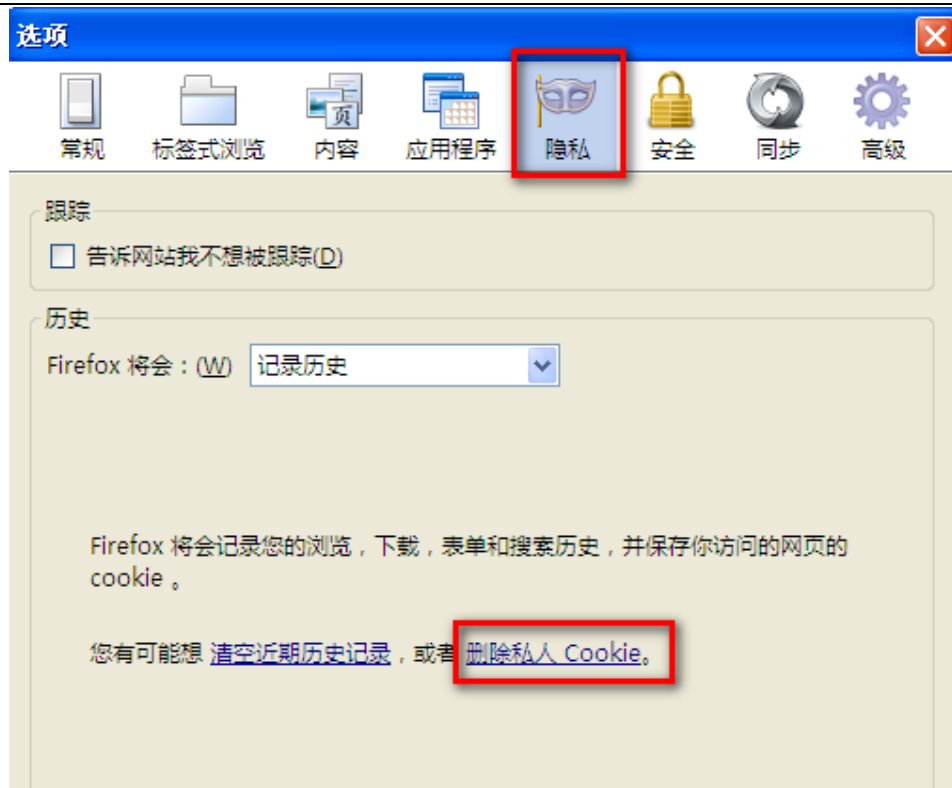
### 5) 使用 FireFox 浏览器查看 Cookie

#### a. “工具” -- “选项”



#### b. “隐私” -- “删除私人 Cookie”



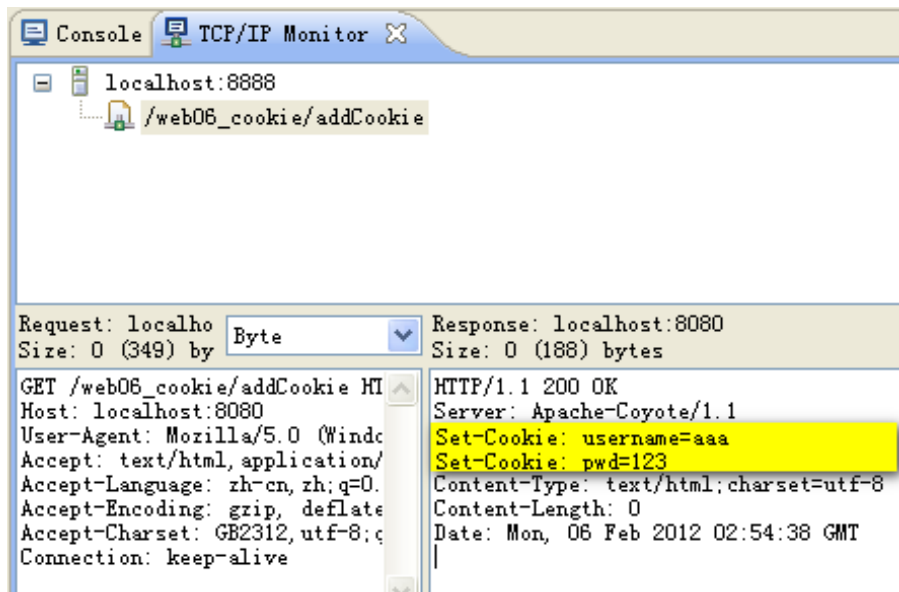


c. “Cookies”

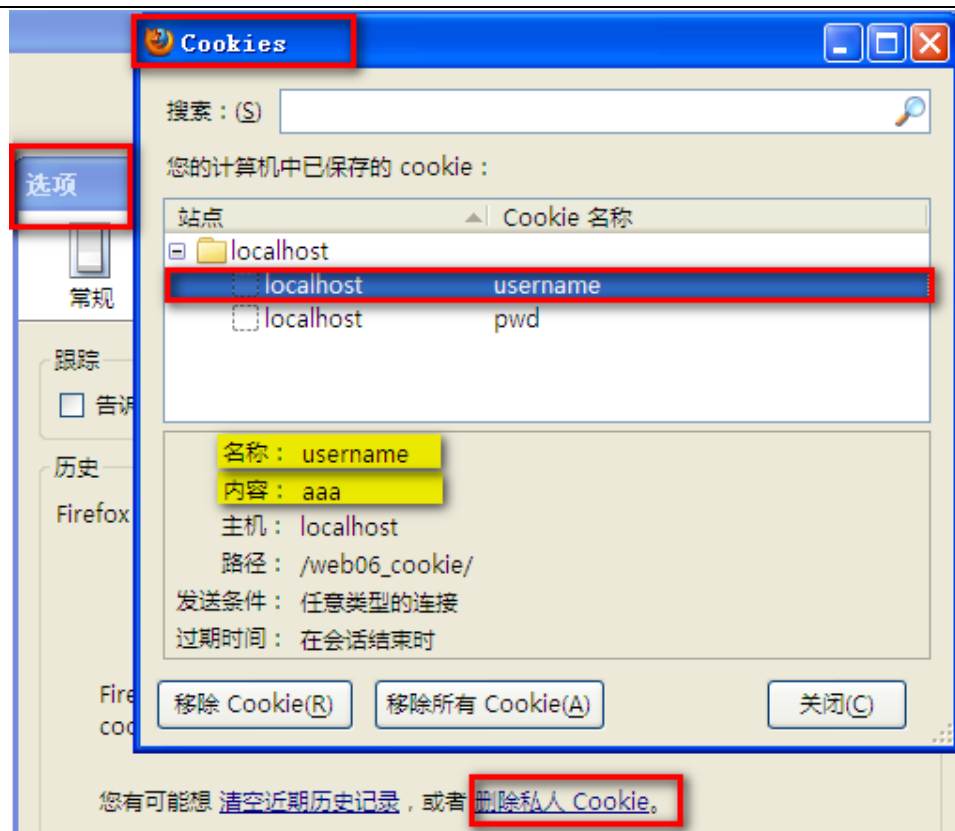


6) 访问 [http://localhost:8888/web06\\_cookie/addCookie](http://localhost:8888/web06_cookie/addCookie)

a. 查看 TCP/IP Monitor

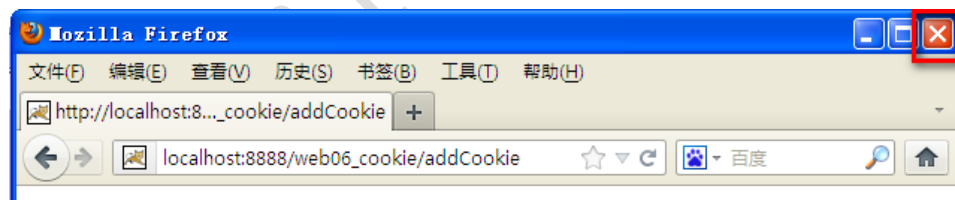


b. 使用 FireFo 浏览器查看 Cookie

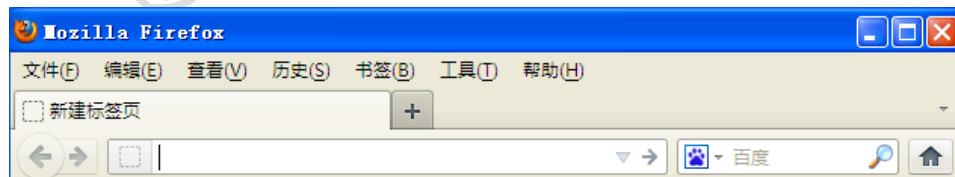


## 7) 关闭 Firefox 浏览器&&重新启动&&查看 Cookie

### a. 关闭 FF



### b. 重新启动 FF

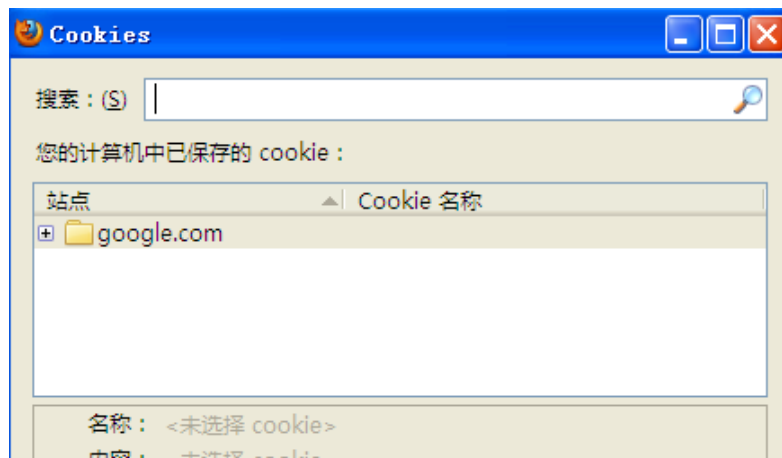


### c. 查看 Cookie

#### 保存的 Cookie "localhost"消失

说明采用如下方式生成的 Cookie 保存在内存中

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

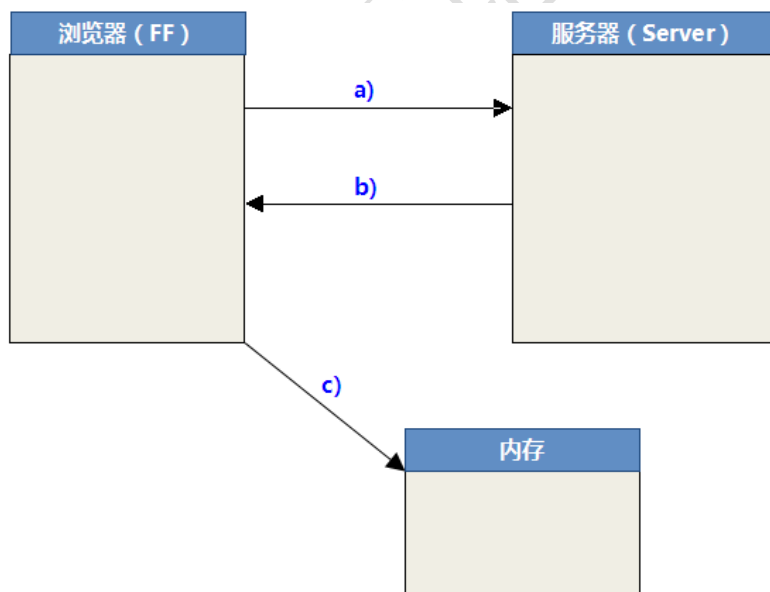


#### 图示说明

启动 FireFox 后，操作系统会为浏览器开辟一块内存空间

- 浏览器 ( FireFox ) 访问服务器
- 服务器生成消息头 setCookie 发回给浏览器
- 浏览器将服务器生成的消息头保存到内存区域

当浏览器关闭，系统回收为浏览器分配的内存，Cookie 也相应消失



## 2.1.2. 查找 Cookie \*\*

### 演示 2

#### 8) 新建 FindCookieServlet

```
FindCookieServlet.java X
13 public class FindCookieServlet extends HttpServlet
14 {
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         Cookie[] cookies = request.getCookies();
23         if(cookies !=null){
24             for(int i=0;i<cookies.length;i++){
25                 Cookie currCookie = cookies[i];
26                 String name = currCookie.getName();
27                 String value = currCookie.getValue();
28                 out.println("<h1>name:" + name
29                     + " value:"
30                     + value+"</h1>");
31             }
32         }else{
33             out.println("<h1>没有cookie</h1>");
34         }
35         out.close();
36     }
37 }
```

#### 9) web.xml

```

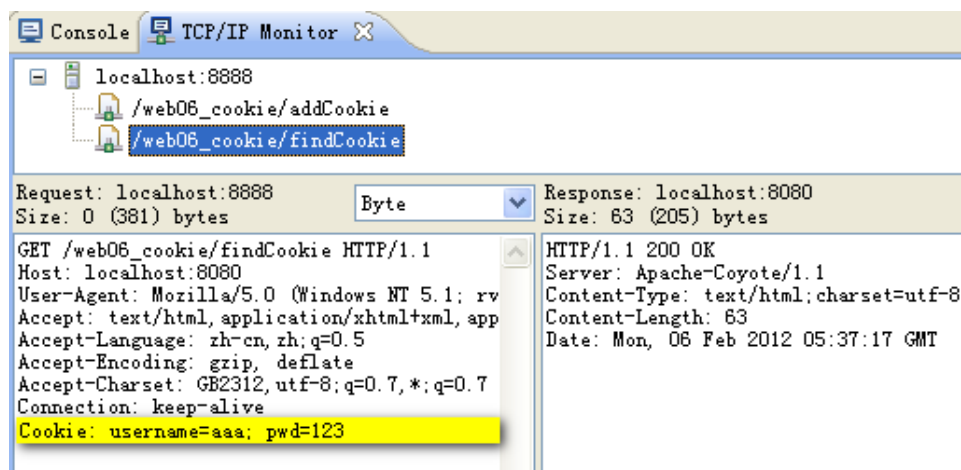
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app version="2.4"
3     xmlns="http://java.sun.com/xml/ns/j2ee"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-in:
5     xsi:schemaLocation="http://java.sun.com/xml/ns,
6     http://java.sun.com/xml/ns/j2ee/web-app_2_4.xs
7 <servlet>
8     <servlet-name>AddCookieServlet</servlet-name>
9     <servlet-class>web.AddCookieServlet</servlet-cl
10 </servlet>
11 <servlet>
12     <servlet-name>FindCookieServlet</servlet-name>
13     <servlet-class>web.FindCookieServlet</servlet-c
14 </servlet>
15 <servlet-mapping>
16     <servlet-name>AddCookieServlet</servlet-name>
17     <url-pattern>/addCookie</url-pattern>
18 </servlet-mapping>
19 <servlet-mapping>
20     <servlet-name>FindCookieServlet</servlet-name>
21     <url-pattern>/findCookie</url-pattern>
22 </servlet-mapping>
23 </web-app>

```

10) 访问 [http://localhost:8888/web06\\_cookie/findCookie](http://localhost:8888/web06_cookie/findCookie)

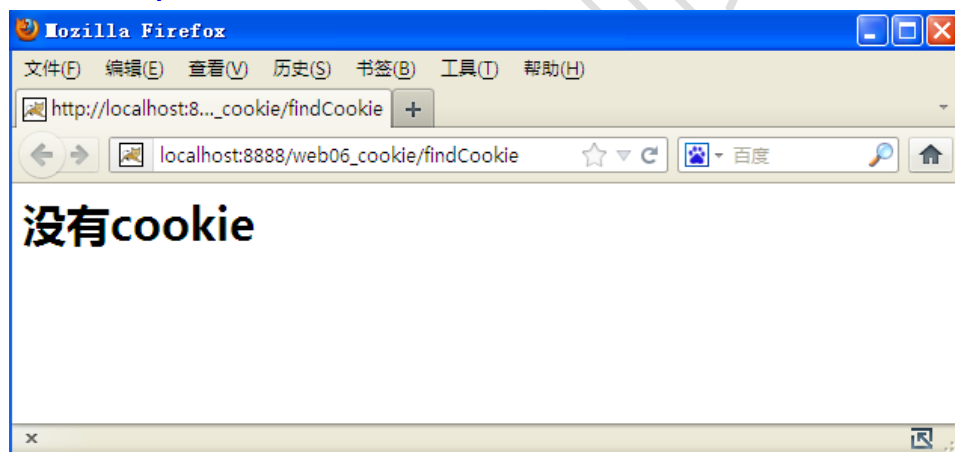


11) 查看 TCP/IP Monitor



## 12) 重启浏览器

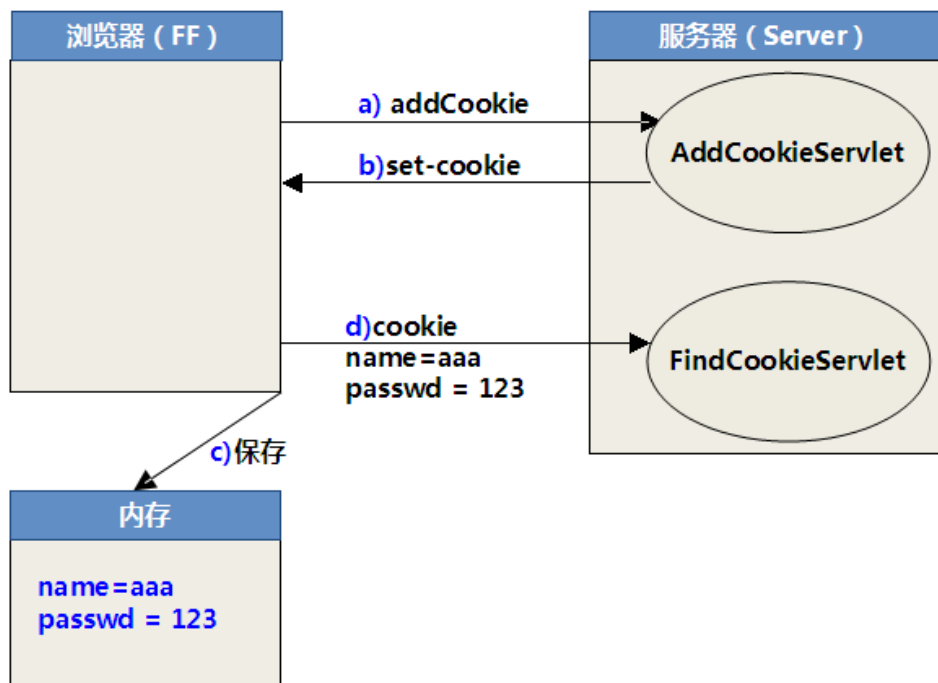
## 13) 访问 [http://localhost:8888/web06\\_cookie/findCookie](http://localhost:8888/web06_cookie/findCookie)



## 图示说明

### 保存和查询 Cookie 流程

- 浏览器向服务器发送 addCookie 请求  
服务器中的 AddCookieServlet 创建了两个 Cookie : cookie 和 cookie2
- 服务器端执行语句 `response.addCookie(cookie);` 生成消息头 "set-cookie" ,  
并将两个 Cookie 以键值对的方式 ( "name=aaa" 、 "passwd=123" ) 存放在消息头中发送给浏览器
- 浏览器将 Cookie 信息保存到本地内存中
- 浏览器继续向服务器发送请求 ( 带着消息头 cookie )  
服务器端的 FindCookieServlet 找到 Cookie 信息 , 并显示给浏览器



### 2.1.3. cookie 保存时的编码问题 \*\*

cookie 的值只能是 ascii 字符，如果是中文，需要将中文转换成 ascii 字符形式。可以使用 `URLEncoder.encode()` 方法和 `URLDecoder.decode()` 方法来进行这种转换。

#### 演示 3

##### 14) 修改 `AddCookieServlet`



```

AddCookieServlet.java
3 import java.io.IOException;
12
13 public class AddCookieServlet extends HttpServlet
14 {
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         //encode方法：先对"张三丰"进行utf-8编码。然后
23         //将编码之后的那个字节数组转换成一个ascii字符串。
24         String username = URLEncoder.encode("张三丰",
25             "utf-8");
26         Cookie cookie = new Cookie("username", username);
27         Cookie cookie2 = new Cookie("pwd", "123");
28         response.addCookie(cookie);
29         response.addCookie(cookie2);
30         out.close();
31     }

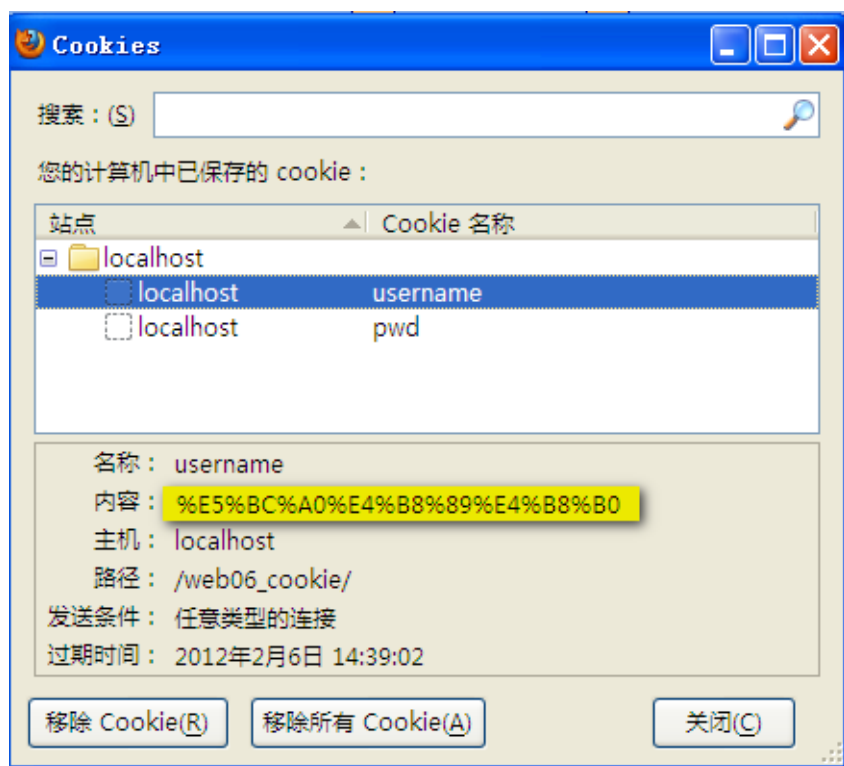
```

#### 15) 修改 FindCookieServlet

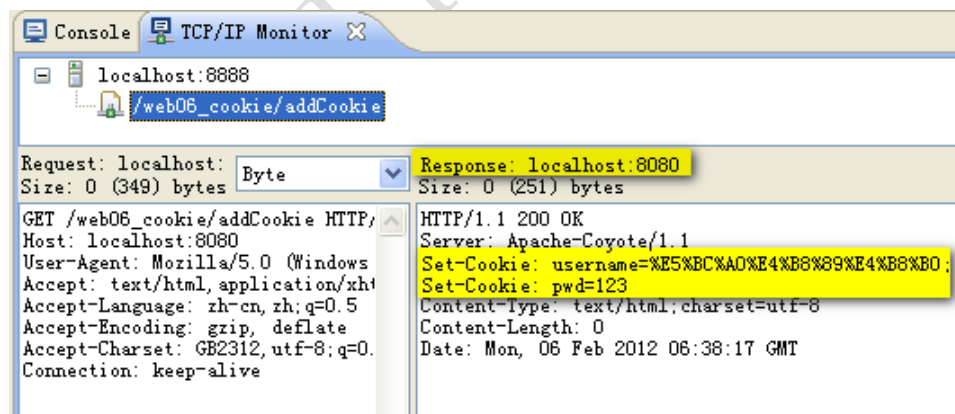
```
FindCookieServlet.java X
14
15 public void service(
16     HttpServletRequest request,
17     HttpServletResponse response)
18     throws ServletException, IOException {
19     response.setContentType(
20         "text/html;charset=utf-8");
21     PrintWriter out = response.getWriter();
22     Cookie[] cookies = request.getCookies();
23     if (cookies != null) {
24         for (int i = 0; i < cookies.length; i++) {
25             Cookie currCookie = cookies[i];
26             String name = currCookie.getName();
27             String value = currCookie.getValue();
28             out.println("<h1>name:" + name
29                 + " value:"
30                 + URLDecoder.decode(
31                     value, "utf-8") + "</h1>");
32         }
33     } else {
34         out.println("<h1>没有cookie</h1>");
35     }
36     out.close();
37 }
38 }
```

16) 访问 [http://localhost:8888/web06\\_cookie/addCookie](http://localhost:8888/web06_cookie/addCookie)

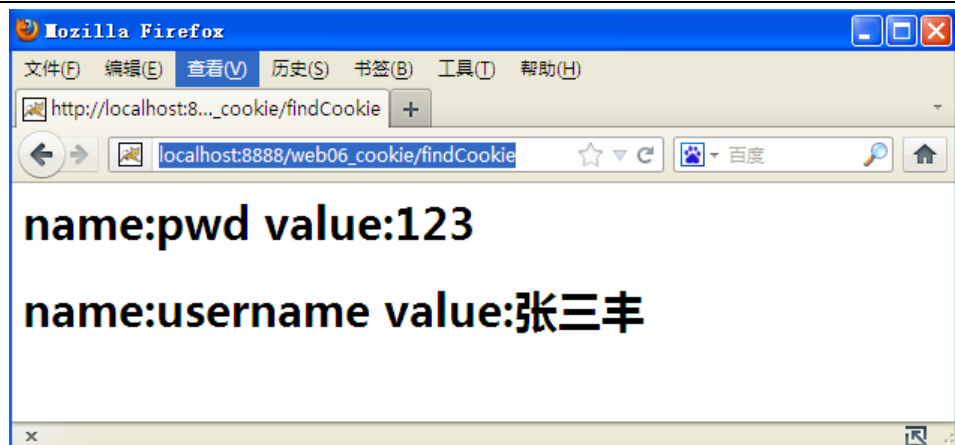
17) 使用浏览器查看



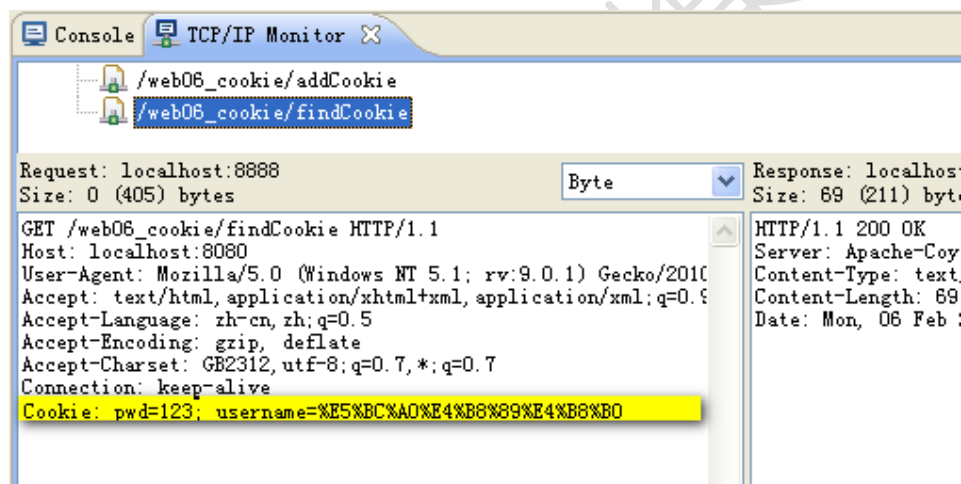
## 18) 查看 TCP/IP Monitor



## 19) 访问 http://localhost:8888/web06\_cookie/findCookie



## 20) 查看 TCP/IP Monitor



### 2.1.4. cookie 的保存时间 \*

`cookie.setMaxAge(int seconds);`

**seconds > 0**

浏览器会将 cookie 以文件的方式保存在硬盘上。在超过指定的时间以后，会删除该文件。

**seconds < 0**

**默认值**，浏览器会将 cookie 保存在内存里面。只有当浏览器关闭之后，才会删除。

**seconds = 0**

立即删除该 Cookie

#### 演示 4

##### 21) 修改 AddCookieServlet

```

AddCookieServlet.java
13 public class AddCookieServlet extends HttpServlet
14 {
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         //encode方法：先对"张三丰"进行utf-8编码。然后
23         //将编码之后的那个字节数组转换成一个ascii字符串。
24         String username = URLEncoder.encode("张三丰",
25             "utf-8");
26         Cookie cookie = new Cookie("username",username);
27         cookie.setMaxAge(45);
28         Cookie cookie2 = new Cookie("pwd", "123");
29         response.addCookie(cookie);
30         response.addCookie(cookie2);
31         out.close();
32     }
}

```

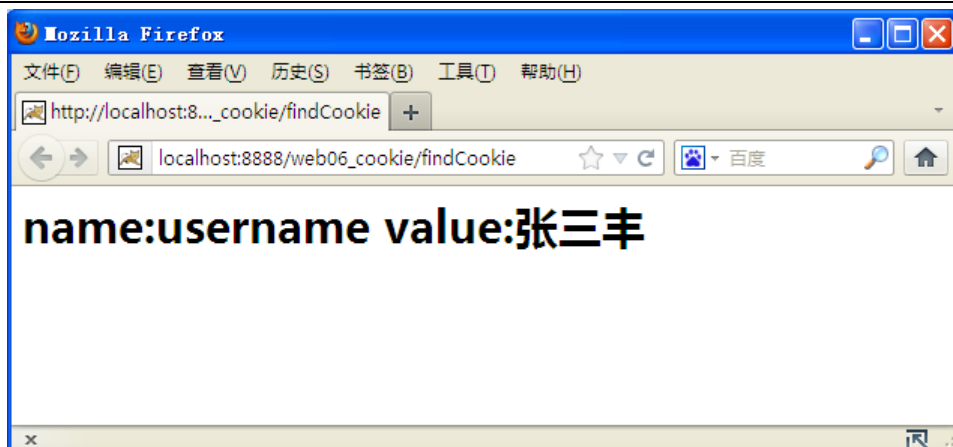
##### 22) 重新部署

23) 访问 [http://localhost:8888/web06\\_cookie/addCookie](http://localhost:8888/web06_cookie/addCookie)

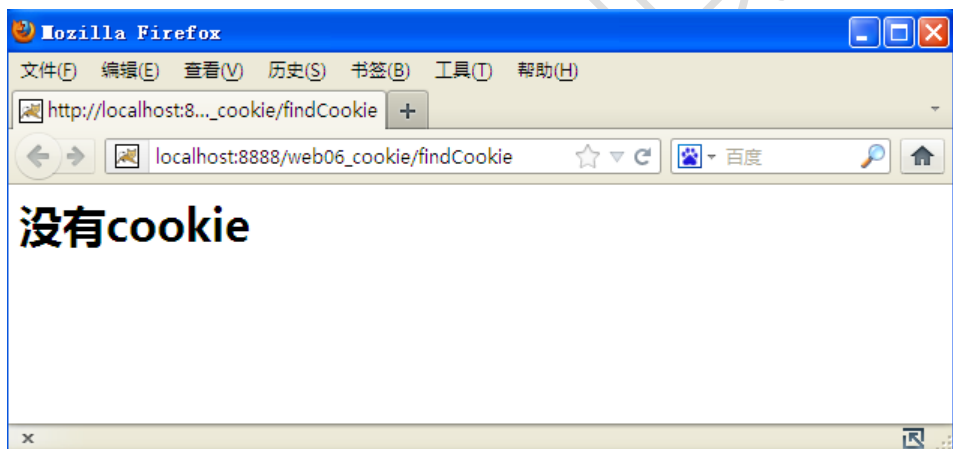
24) 在 45 秒之内访问 [http://localhost:8888/web06\\_cookie/findCookie](http://localhost:8888/web06_cookie/findCookie)

name:pwd value:123 的 cookie 失效

name:username value:张三丰的 cookie 仍然有效



25) 45 秒后访问 [http://localhost:8888/web06\\_cookie/findCookie](http://localhost:8888/web06_cookie/findCookie)  
name:username value:张三丰的 cookie 失效



### 2.1.5. 删除 cookie \*

#### 演示 5

#### 删除 name 为 "username" 的 cookie 步骤

第 1 步：创建一个同名并且内容为空的 cookie

```
Cookie c = new Cookie("username", "");
```

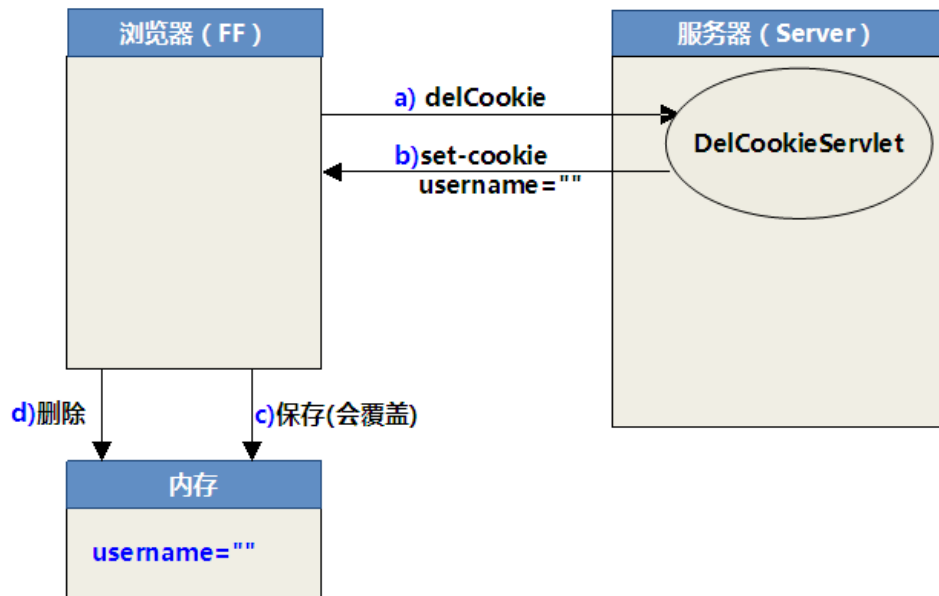
第 2 步：调用方法，参数为 0

```
c.setMaxAge(0);
```

第 3 步：调用方法

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

图示

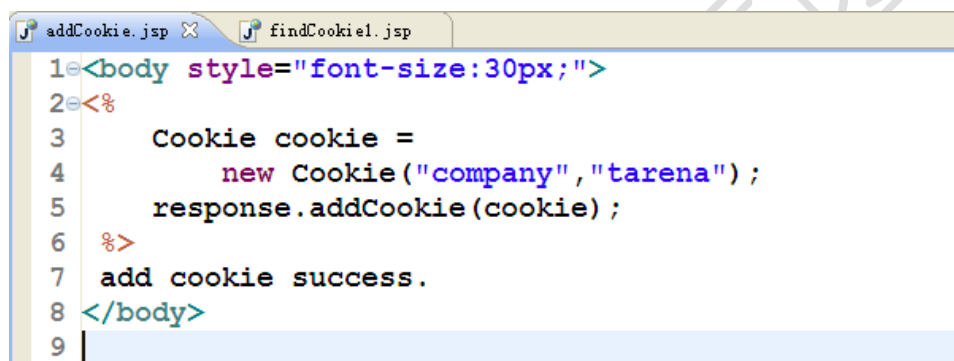
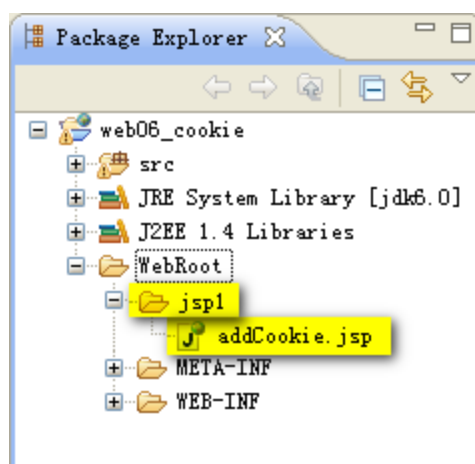


### 2.1.6. cookie 的路径问题 \*\*

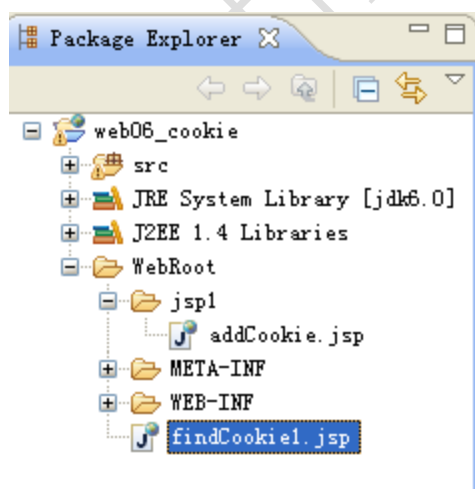
浏览器在向服务器上的某个地址发送请求时，  
会先比较 cookie 的路径与向访问的路径(地址)是否匹配，只有匹配的 cookie，才会发送。  
cookie 的路径可以通过 `cookie.setPath(String path)` 方法来设置。  
如果没有设置，则有一个缺省的路径，缺省的路径是生成该 cookie 的组件的路径。  
比如: `/appname/addCookie` 保存了一个 cookie，  
则该 cookie 的路径就是 `/appname/addCookie`。

演示 6

26) 新建文件夹 `jsp1`，并在之下创建 `addCookie.jsp`



## 27) 新建 findCookie1.jsp





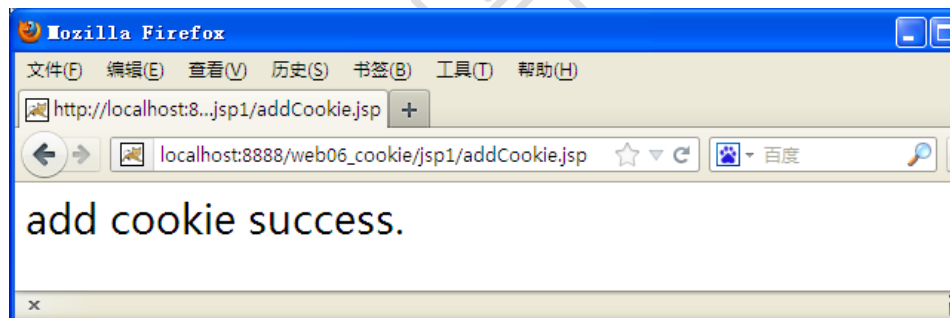
```

1 <body style="font-size:30px;">
2 <%
3     Cookie[] cookies = request.getCookies();
4     if(cookies !=null){
5         for(int i=0;i<cookies.length;i++){
6             Cookie currCookie = cookies[i];
7             String name = currCookie.getName();
8             String value = currCookie.getValue();
9             out.println("<h1>name:" + name
10                + " value:" + value + "</h1>");
11         }
12     }else{
13         out.println("<h1>no cookie</h1>");
14     }
15 %>
16 </body>
17

```

## 28) 部署项目

29) 访问 [http://localhost:8888/web06\\_cookie/jsp1/addCookie.jsp](http://localhost:8888/web06_cookie/jsp1/addCookie.jsp)



## 30) 查看 Cookie

JSESSIONID 在后续知识讲解，暂时忽略

### a. 浏览器查看

存在名为 `company` 的 cookie



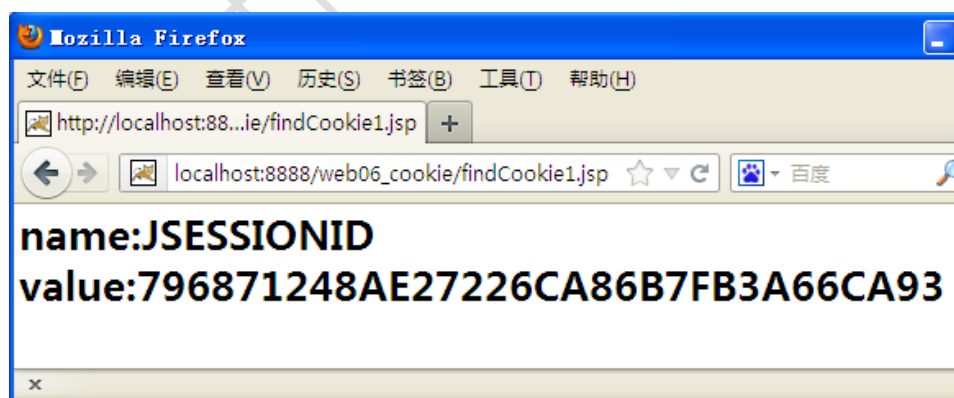
#### b. 访问查看 Cookie

没有名为 company 的信息

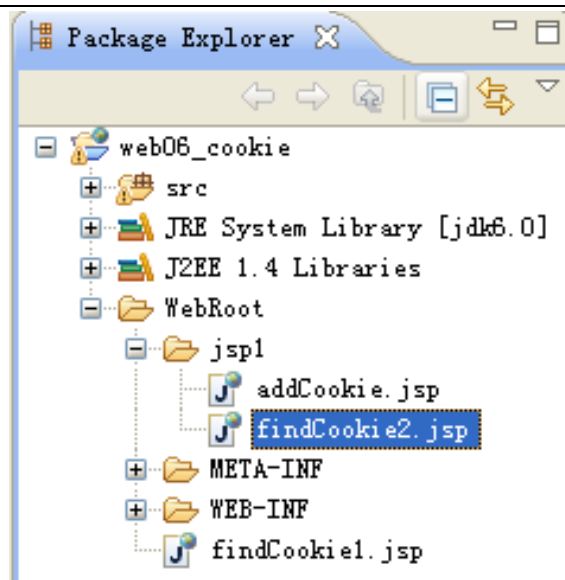
原因是路径问题：

Cookie 的路径是：**/web06\_cookie/jsp1/**

访问的地址：**http://localhost:8888/web06\_cookie/findCookie1.jsp**



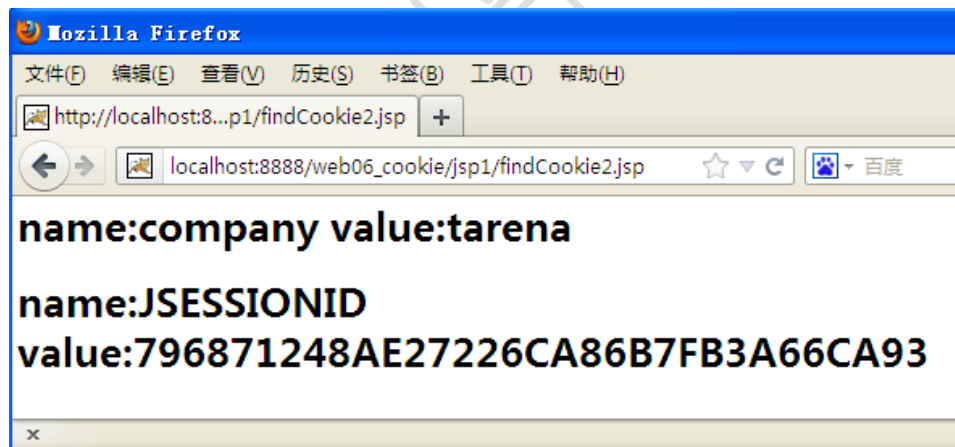
31) 复制 findCookie1.jsp 到 jsp1 目录下，改名为 findCookie2.jsp



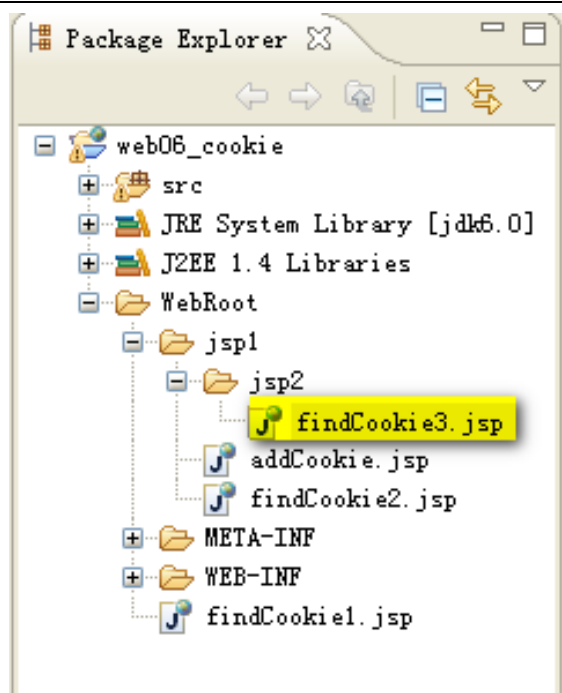
32) 部署项目

33) 访问 [http://localhost:8888/web06\\_cookie/jsp1/findCookie2.jsp](http://localhost:8888/web06_cookie/jsp1/findCookie2.jsp)

可以查询到名为 company 的 cookie



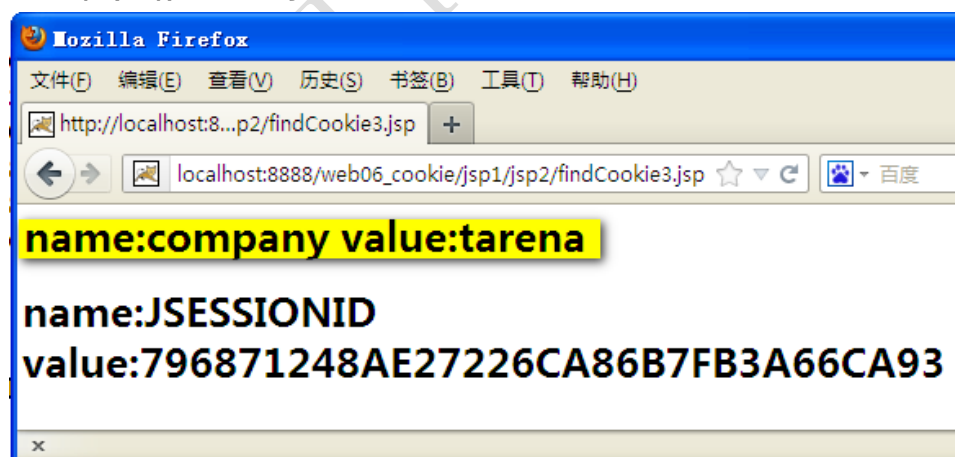
34) 新建目录 jsp2 , 复制 findCookie1.jsp 为 findCookie3.jsp



### 35) 部署项目

36) 访问 [http://localhost:8888/web06\\_cookie/jsp1/jsp2/findCookie3.jsp](http://localhost:8888/web06_cookie/jsp1/jsp2/findCookie3.jsp)

在子路径中同样也能访问到



规则:

cookie 的路径必须是要访问的路径的上层目录或者是与要访问的路径相等，浏览器才会将 cookie 发送给服务器。

一般可以设置 `setPath("/appname")`, 表示访问该应用下的所有地址，均会发送 cookie。

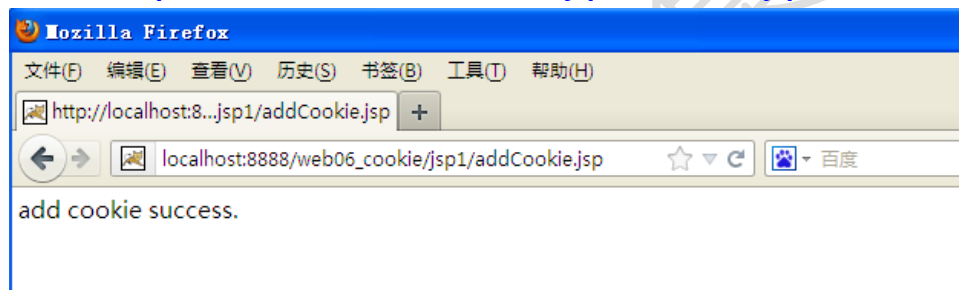
### 37) 解决办法

#### a. 修改 addCookie.jsp

```
addCookie.jsp
1<body style="font-size:30px;">
2<%
3    Cookie cookie =
4        new Cookie("company","tarena");
5    cookie.setPath("/web06_cookie");
6    response.addCookie(cookie);
7    %>
8    add cookie success.
9</body>
10
```

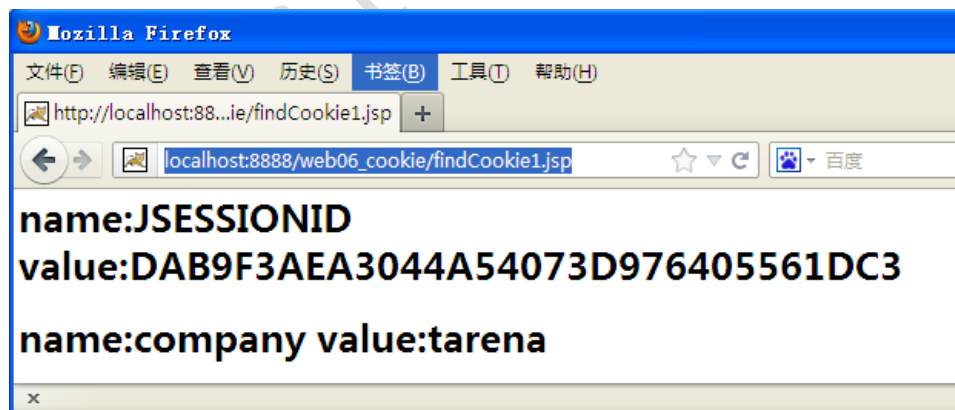
#### b. 部署项目

#### c. 访问 http://localhost:8888/web06\_cookie/jsp1/addCookie.jsp

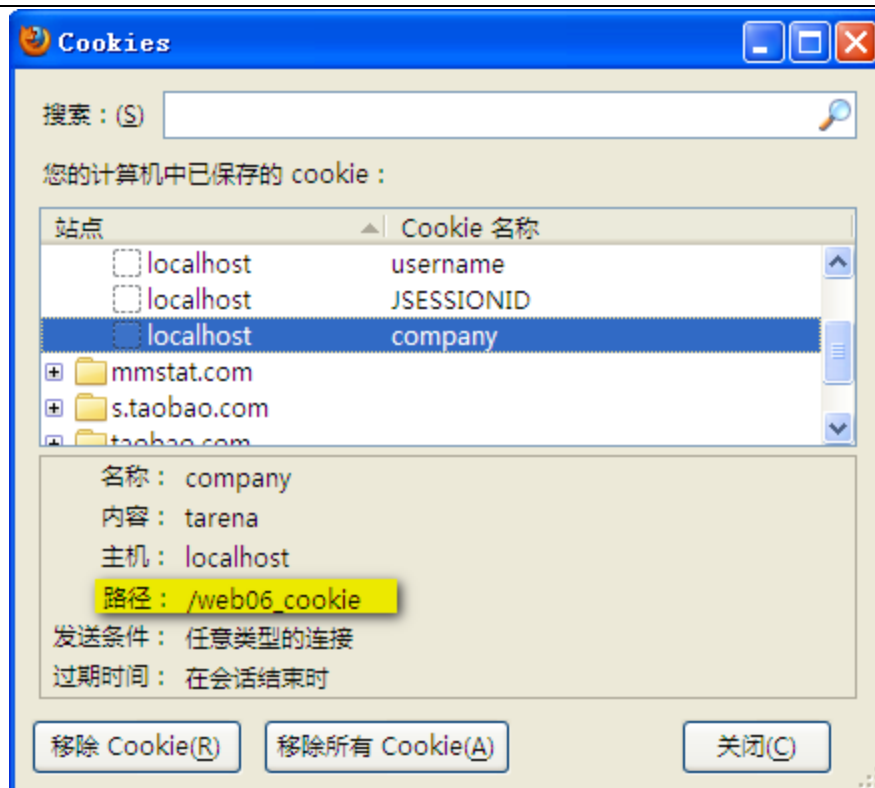


#### d. 访问 http://localhost:8888/web06\_cookie/findCookie1.jsp

可以访问到了



#### e. 对比步骤 30)，路径发生了变化



### 2.1.7. cookie 的限制 \*

- ✓ cookie 可以禁止
- ✓ cookie 的大小有限制(4k 左右)
- ✓ cookie 的数量也有限制(浏览器大约能保存 300 个)
- ✓ cookie 的值只能是字符串，要考虑编码问题。
- ✓ cookie 不安全

#### 演示 7

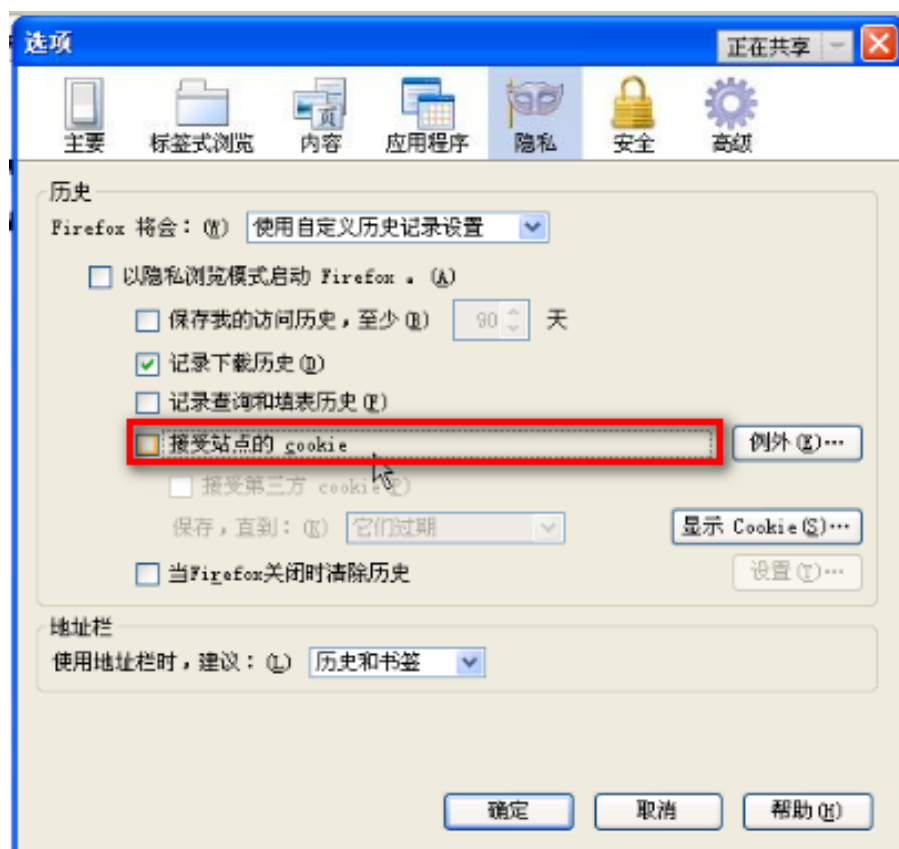
#### 38) 修改浏览器设置

##### cookie 可以禁止

Firefox 不同版本的设置会略有不同，如下列举两个版本的

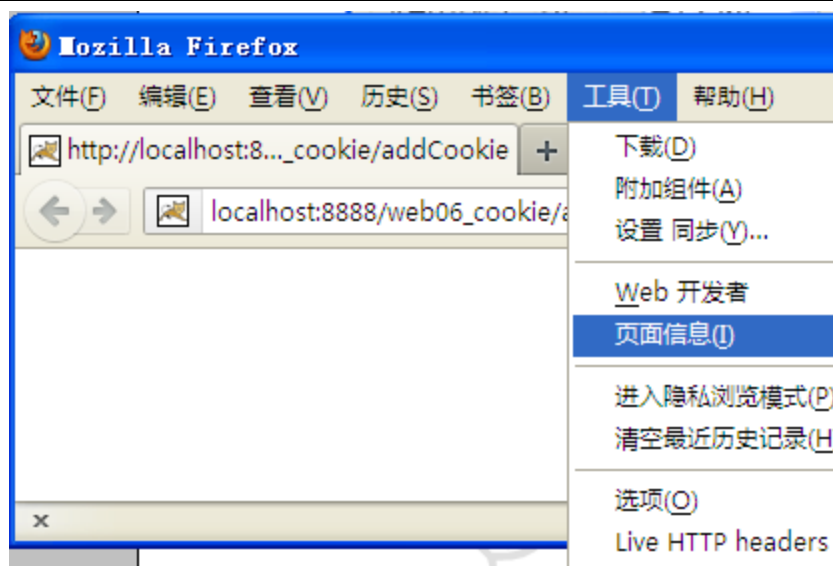
版本未知

将对勾取消

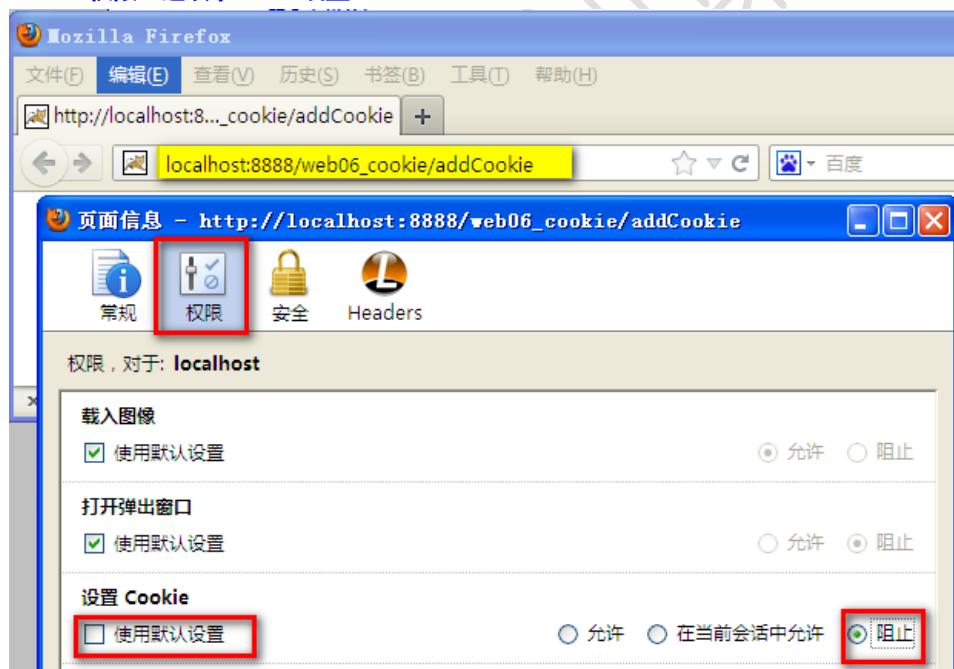


版本 9.0.1

a. “工具” -- “页面信息”



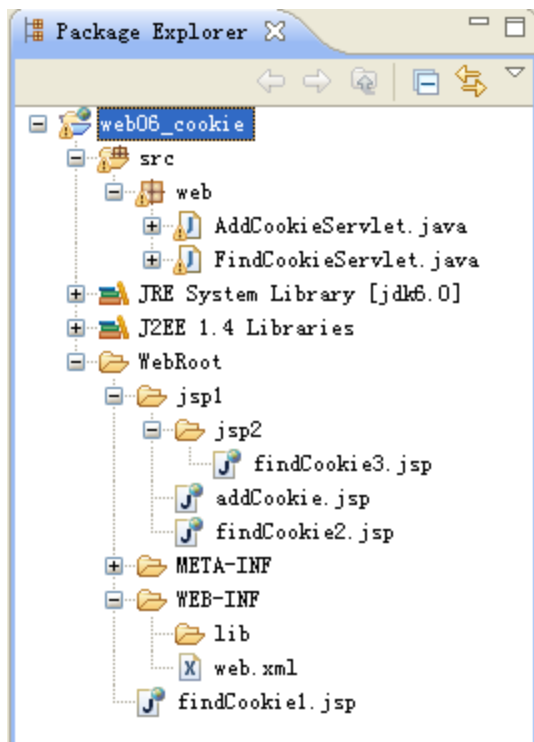
b. “权限”选项卡 -- “设置 Cookie”



【案例 1】Cookie01 \*\*



## 1) 项目结构



## 2) AddCookieServlet.java

```
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 AddCookieServlet extends HttpServlet {
    public void service(
        HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
```

```
response.setContentType(
    "text/html;charset=utf-8");
PrintWriter out = response.getWriter();
//encode 方法：先对"张三丰"进行 utf-8 编码。然后
//将编码之后的那个字节数组转换成一个 ascii 字符串。
String username = URLEncoder.encode("张三丰",
    "utf-8");
Cookie cookie = new Cookie("username",username);
cookie.setMaxAge(45);
Cookie cookie2 = new Cookie("pwd","123");
response.addCookie(cookie);
response.addCookie(cookie2);
out.close();
}
}
```

### 3) FindCookieServlet.java

```
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 FindCookieServlet 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(int i=0;i<cookies.length;i++){
                Cookie currCookie = cookies[i];
                String name = currCookie.getName();
                String value = currCookie.getValue();
                out.println("<h1>name:" + name
                    + " value:"
                    + URLDecoder.decode(
                        value,"utf-8")+ "</h1>");
            }
        }else{
            out.println("<h1>没有 cookie</h1>");
        }
        out.close();
    }
}

```

#### 4) web.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
    xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
        http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    <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-mapping>
    <servlet-name>FindCookieServlet</servlet-name>
    <url-pattern>/findCookie</url-pattern>
</servlet-mapping>
</web-app>
```

## 5) addCookie.jsp

```
<body style="font-size:30px;">
<%
    Cookie cookie =
        new Cookie("company", "tarena");
    cookie.setPath("/web06_cookie");
    response.addCookie(cookie);
%>
add cookie success.
</body>
```

## 6) findCookie1.jsp, findCookie2.jsp, findCookie3.jsp

```
<body style="font-size:30px;">
<%
    Cookie[] cookies = request.getCookies();
    if(cookies != null){
        for(int i=0;i<cookies.length;i++){
            Cookie currCookie = cookies[i];
            String name = currCookie.getName();
            String value = currCookie.getValue();
            out.println("<h1>name:" + name
                + " value:" + value + "</h1>");
        }
    }else{
        out.println("<h1>no cookie</h1>");
    }
%>
</body>
```

## 2.1.8. 课堂练习 \*\*

### 【案例 2】Cookie02 \*\*

#### 练习描述

写一个 Add\_FindCookieServlet, 该 servlet 先查询有没有一个名叫 name 的 cookie , 如果有, 则显示该 cookie 的值 , 如果没有, 则创建该 cookie(cookie 的名字:name, cookie 的值:zs)。

#### 参考代码

##### 1) Add\_FindCookieServlet

```
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 Add_FindCookieServlet 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){
            boolean flag = false;
            for(int i=0;i<cookies.length;i++){
```

```

        Cookie curr = cookies[i];
        if(curr.getName().equals("name")){
            //找到了，则显示 cookie 的值
            out.println(curr.getValue());
            //将标志位这是为 true
            flag = true;
        }
    }
    //没有找到，则创建 cookie
    if(!flag){
        Cookie cookie =
            new Cookie("name","zs");
        response.addCookie(cookie);
    }
    }else{
        //创建 cookie
        Cookie cookie = new Cookie("name","zs");
        response.addCookie(cookie);
    }
    out.close();
}
}

```

### 常见的错误写法

在写程序时因为逻辑不清犯的错误，在 for 循环查找没有结束前，就创建 Cookie

```

Add_FindCookieServlet.java
19  PrintWriter out = response.getWriter();
20  Cookie[] cookies = request.getCookies();
21  if(cookies != null){
22      for(int i=0;i<cookies.length;i++){
23          Cookie curr = cookies[i];
24          if(curr.getName().equals("name")){
25              //找到了, 则显示cookie的值
26              out.println(curr.getValue());
27          }else{
28              Cookie cookie =
29                  new Cookie("name", "zs");
30              response.addCookie(cookie);
31          }
32      }
33  }
34  }else{
35      //创建cookie
36      Cookie cookie = new Cookie("name", "zs");
37      response.addCookie(cookie);
38  }
39  out.close();
40  }

```

## 2) web.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <servlet>
    <servlet-name>Add_FindCookieServlet</servlet-name>
    <servlet-class>web.Add_FindCookieServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>Add_FindCookieServlet</servlet-name>
    <url-pattern>/add_findCookie</url-pattern>
  </servlet-mapping>
</web-app>

```

### 3) 测试之前

注意如果之前禁用过 Cookie，请还原

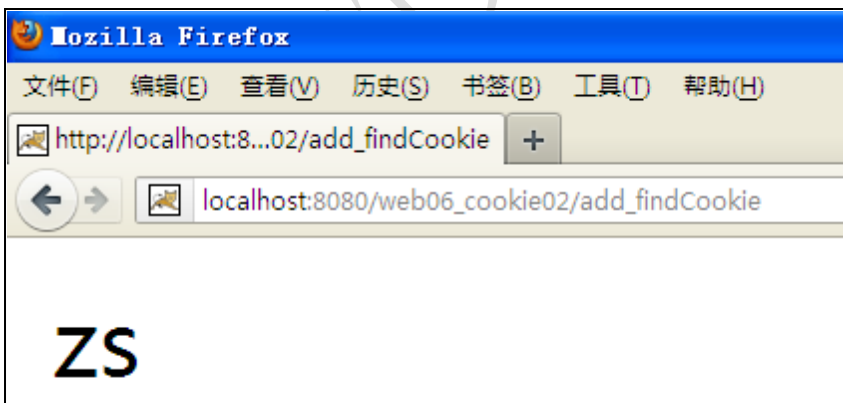


### 4) 访问 [http://localhost:8080/web06\\_cookie02/add\\_findCookie](http://localhost:8080/web06_cookie02/add_findCookie)

添加 Cookie

### 5) 刷新

查询到已经有 Cookie，输出到了页面



## • 优化

### 6) 新建 Add\_FindCookieServlet2

```
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 Add_FindCookieServlet2
    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(int i=0;i<cookies.length;i++){
                Cookie curr = cookies[i];
                if(curr.getName().equals("name")){
                    out.println(curr.getValue());
                    out.close();
                    return;
                }
            }
        }
        Cookie cookie = new Cookie("name", "zs");
        response.addCookie(cookie);
        out.close();
    }
}
```

- 继续优化

## 7) 新建 CookieUtil.java

添加 Cookie 时调用工具即可

请练习并掌握该类，后续在讲购物车时，我们会用到

```
package util;

import java.io.UnsupportedEncodingException;
import java.net.URLDecoder;
import java.net.URLEncoder;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * cookie 工具类
 * @author teacher
 *
 */
public class CookieUtil {
    //缺省的应用名
    private static String default_path =
        "/web06_cookie";
    //缺省的生存时间
    private static int default_age = 365 * 24 * 3600;

    /**
     * 添加一个 cookie
     * @param name
     * @param value
     * @param response
     * @param age
     * @throws UnsupportedEncodingException
     */
    public static void addCookie(String name,
        String value, HttpServletResponse response,
        int age) throws UnsupportedEncodingException{
        Cookie cookie =
            new Cookie(
                name, URLEncoder.encode(value, "utf-8"));
        cookie.setMaxAge(age);
        cookie.setPath(default_path);
    }
}
```

```

        response.addCookie(cookie);
    }

    public static void addCookie(String name,
                                String value,HttpServletResponse response)
                                throws UnsupportedEncodingException{
        addCookie(name,value,response,default_age);
    }
    /**
     * 依据 cookie 的名字，查找 cookie 的值，如果找不到，返回 null。
     * @param name
     * @param request
     * @return
     * @throws UnsupportedEncodingException
     */
    public static String findCookie(String name,
                                    HttpServletRequest request)
                                    throws UnsupportedEncodingException{
        String value = null;
        Cookie[] cookies = request.getCookies();
        if(cookies !=null){
            for(int i=0;i<cookies.length;i++){
                Cookie curr = cookies[i];
                if(curr.getName().equals(name)){
                    value =
                        URLDecoder.decode(
                            curr.getValue(),"utf-8");
                }
            }
        }
        return value;
    }

    /**
     * 删除 cookie
     * @param name
     * @param response
     */

```

```
public static void deleteCookie(String name,
    HttpServletResponse response){
    //请尝试自己添加
}
}
```

## 2.2. Session \*\*

### 2.2.1. 什么是 session \*

浏览器访问服务器时，服务器会创建一个 session 对象(该对象有一个唯一的 id, 一般称为 sessionId)。服务器在缺省情况下，会将 sessionId 以 cookie 机制发送给浏览器。当浏览器再次访问服务器时，会将 sessionId 发送给服务器。服务器依据 sessionId 就可以找到对应的 session 对象。通过这种方式，就可以管理用户的状态。

### 2.2.2. 如何获得 session 对象 \*

方式一：

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

**当 flag = true:**

服务器会先查看请求中是否包含 sessionId,

如果没有，则创建一个 session 对象。

如果有，则依据 sessionId 去查找对应的 session 对象，如果找到，则返回。

如果找不到，则创建一个新的 session 对象。

**当 flag = false:**

服务器会先查看请求中是否包含 sessionId,

如果没有,返回 null。

如果有，则依据 sessionId 去查找对应的 session 对象，如果找到，则返回。

如果找不到，返回 null。

方式二：

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

与 request.getSession(true)等价。

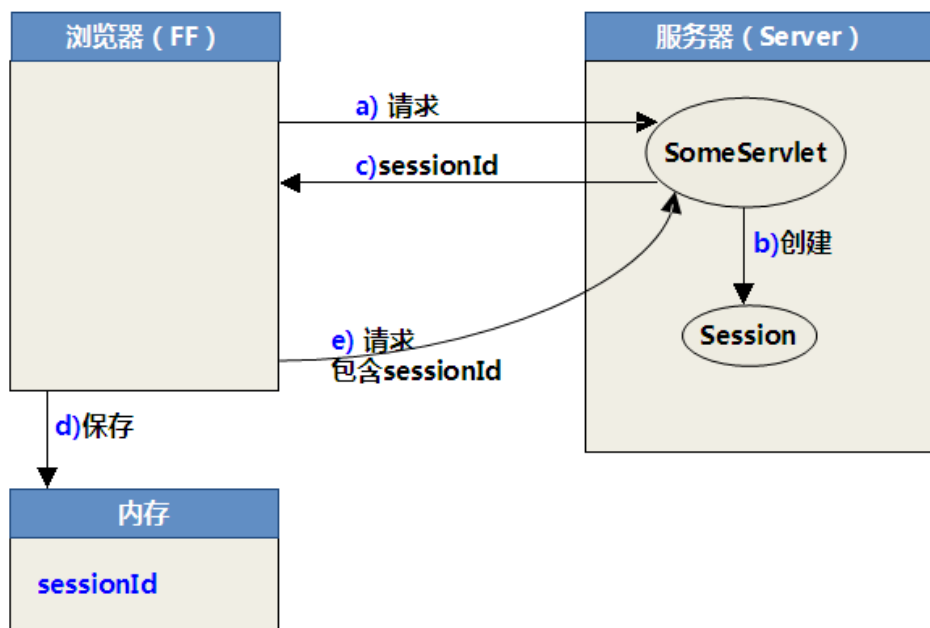
### 图示演示

- 1) 浏览器向服务器发请求，访问某一个 Servlet 或 JSP

服务器先查看包含于请求的 Cookie 信息中 (消息头是 cookie ) 是否有 sessionId ( 第一次访

问时是没有的)

- 2) 浏览器第一次访问，服务器会创建一个 Session 对象，SomeServlet 和该 Session 对象之间存在引用关系（即 Servlet 可以访问该 Session 对象了）
- 3) SomeServlet 以 Cookie 的方式（消息头为 set-cookie）将 SessionId 响应给浏览器
- 4) 浏览器将 SessionId 保存到内存中
- 5) 当浏览器再次访问服务器时，请求中的 Cookie 信息中包含 sessionId，SomeServlet 会根据 sessionId 找到对应的 Session 对象



### 2.2.3. HttpSession 接口提供的一些方法 \*\*

- 1) 获得 sessionId。  
`String session.getId();`
- 2) 绑定数据  
`session.setAttribute(String name, Object obj);`  
服务器在对 session 进行持久化操作时，比如钝化、激活，会使用序列化协议，obj 最好实现 Serializable 接口  
`Object session.getAttribute(String name);`
- 3) 如果 name 对应的值不存在，返回 null。  
`session.removeAttribute(String name);`

**服务器在对 session 进行持久化操作时，为什么 obj 最好实现 Serializable 接口？**

obj 进行序列化的原因是可以写到硬盘上。

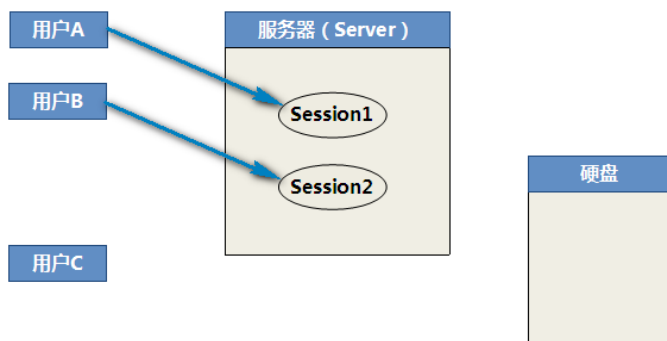
服务器上一般会保存中许多的 Session 对象 里面存放了大量用户状态的数据( 比如网上购物信息 ), 通常情况下内存是有限的, 对于海量的访问, 这些 Session 是不够的, 所以一般情况下, 我们配备一些数据库 ( 也可以文件的方式保存到硬盘 ), 用于保存 Session 中的数据。

Session 中某个用户的数据进行序列化保存到数据库 ( 或硬盘 ) 时, 该 Session 就可以空闲出来用于新用户的连接。

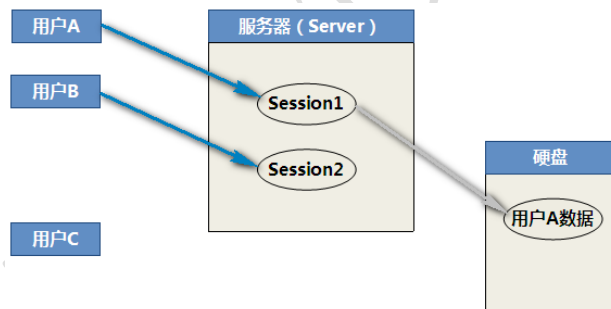
#### 图示如下

服务器共有 2 个 Session 对象

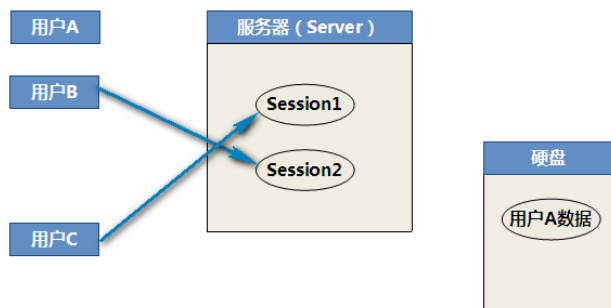
- 1) 当用户 A 连接时, 服务器分配 Session1 给用户 A
- 2) 当用户 B 连接时, 服务器分配 Session2 给用户 B



- 3) 当用户 C 连接时, 服务器已经没有空闲的 Session 对象可用, 它会使用一些特定的算法 ( 比如最近最少使用算法 LRU ) 选择出一个 Session, 比如 Session1 ( 用户 A 的操作在空闲中 ), 服务器将 Session1 中用户 A 的数据序列化到硬盘上

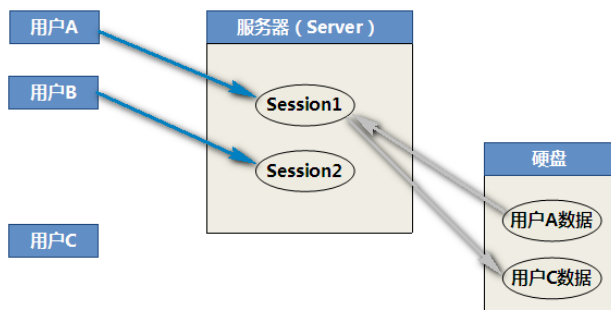


- 4) 之后, 服务器会将 Session1 的数据清空、分配给用户 C 使用



- 5) 当 A 用户重新获得连接、开始操作时，服务器继续根据特定的算法，挑中一个 Session，假如仍然是 Session1，则服务器会将用户 C 的数据序列化到硬盘，将用户 A 的数据从硬盘返回到 Session1 中。

将 Session 中的数据保存到硬盘，或由硬盘读入到 Session 中的过程叫做 Session 的“钝化”与“激活”。



### 【案例 3】Session 实现访问计数 \*\*

#### 1) 新建 CountServlet

```
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();
    }
}
```

```

HttpSession session = request.getSession();
System.out.println(session.getId());

Integer count =
    (Integer) session.getAttribute("count");
if(count == null){
    count = 1;
}else{
    count++;
}
session.setAttribute("count", count);
out.println("你是第 " + count + " 次访问");
out.close();
}
}

```

## 2) web.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
    xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    <servlet>
        <servlet-name>CountServlet</servlet-name>
        <servlet-class>web.CountServlet</servlet-class>
    </servlet>

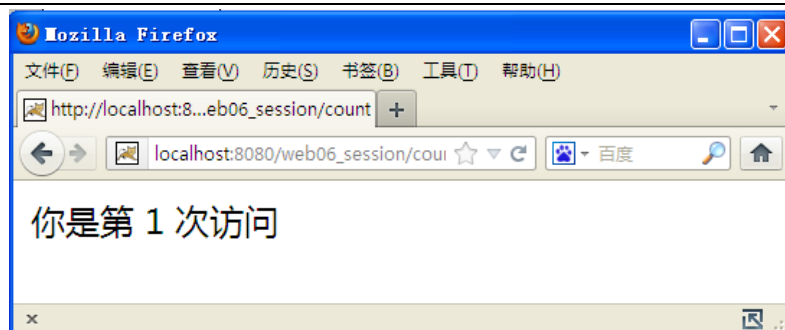
    <servlet-mapping>
        <servlet-name>CountServlet</servlet-name>
        <url-pattern>/count</url-pattern>
    </servlet-mapping>
</web-app>

```

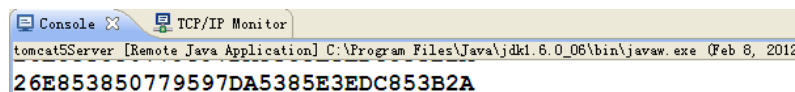
## 3) 部署项目

## 4) 访问 [http://localhost:8080/web06\\_session/count](http://localhost:8080/web06_session/count)

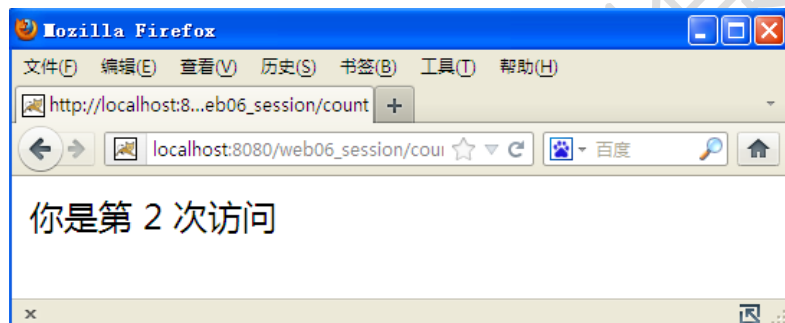




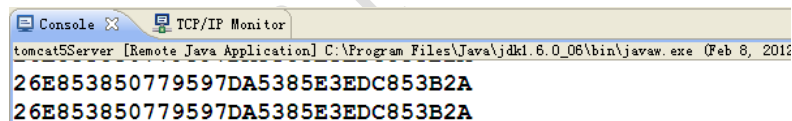
控制台打印的 sessionId



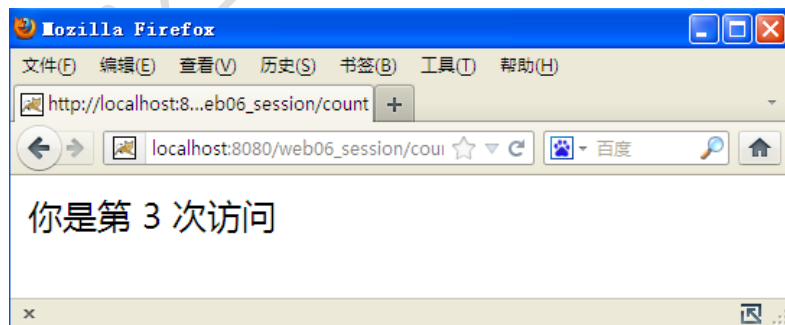
#### 5) 刷新



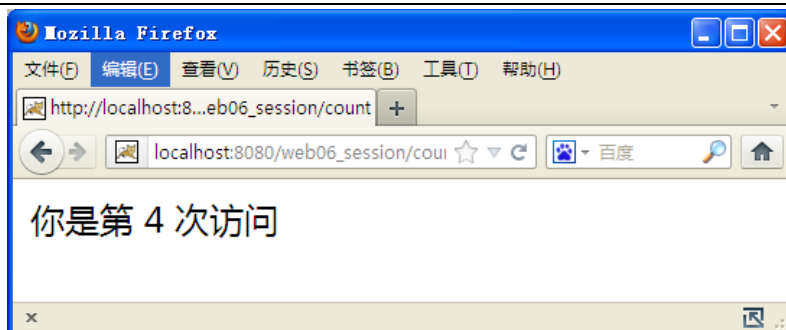
控制台打印的 sessionId (相同的 sessionId)



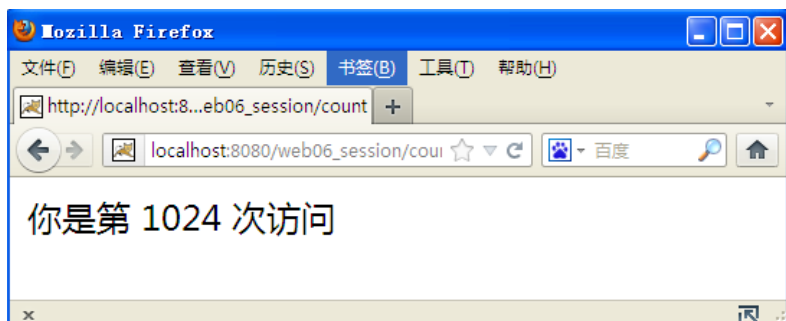
#### 6) 继续刷新



#### 7) 还在刷新

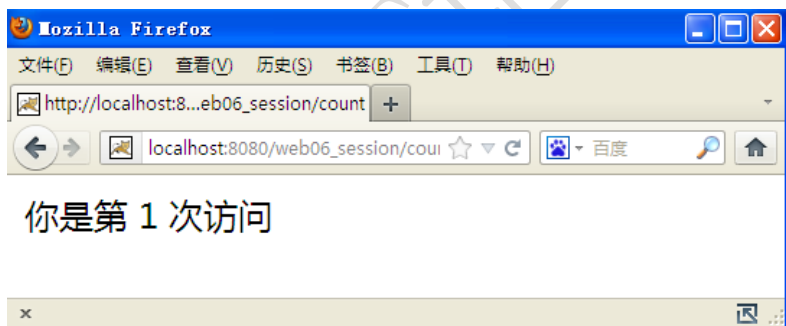


#### 8) 没完没了的刷新

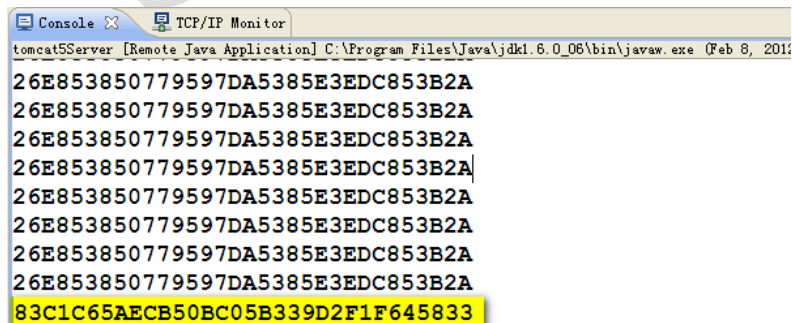


#### 9) 重启浏览器，访问 [http://localhost:8080/web06\\_session/count](http://localhost:8080/web06_session/count)

关闭浏览器后，Session 将失效



#### 控制台打印的 sessionId (不同的 sessionId)



#### 2.2.4. session 超时 \*

服务器会将超过指定时间的 session 对象删除(在指定的时间内, 该 session 对象没有使用)。

**方式一：**

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

**方式二：**

服务器有一个缺省的超时限制, 可以通过相应的配置文件来重新设置。

比如可以修改 tomcat 的 web.xml(tomcat\_home/conf 下面), 这样对所有应用都起作用。

```
<session-config>
    <session-timeout> 30</session-timeout>
</session-config>
```

另外, 也可以只修改某个应用的 web.xml。

**演示**

**session 超时**

- **方式一**

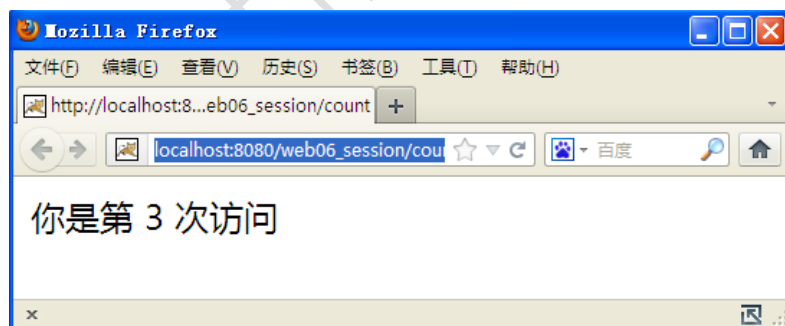
加入代码 `session.setMaxInactiveInterval(int seconds);`

1) **修改 CountServlet**

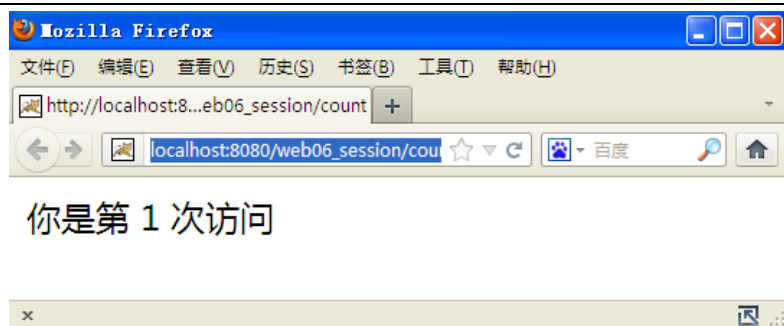
加入代码, 设置 40 秒后删除 session

```
CountServlet.java
13 public void service(
14     HttpServletRequest request,
15     HttpServletResponse response)
16     throws ServletException, IOException {
17
18     response.setContentType(
19         "text/html;charset=utf-8");
20     PrintWriter out = response.getWriter();
21     HttpSession session = request.getSession();
22     session.setMaxInactiveInterval(40);
23     System.out.println(session.getId());
24
25     Integer count =
26         (Integer) session.getAttribute("count");
27     if(count == null){
28         count = 1;
29     }else{
30         count++;
31     }
32     session.setAttribute("count", count);
33     out.println("你是第 " + count + " 次访问");
34     out.close();
35 }
36 }
```

- 2) 重新部署
- 3) 重启浏览器
- 4) 访问 [http://localhost:8080/web06\\_session/count](http://localhost:8080/web06_session/count)，刷新



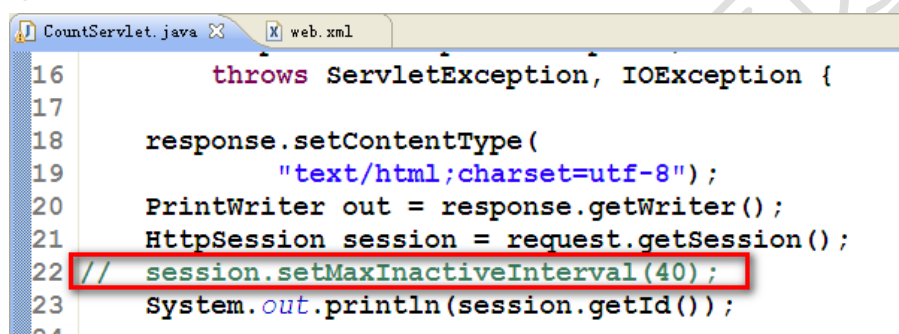
- 5) 空闲 40 秒后刷新，Session 失效



- 方式 2 (一般情况下不修改, 会影响所有应用)

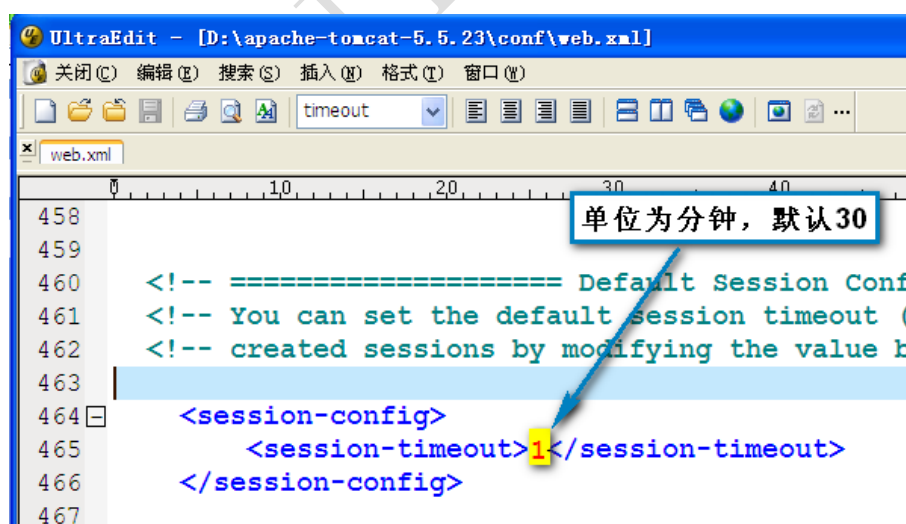
修改 tomcat 的 web.xml

#### 6) 注释 Servlet 中修改的代码



#### 7) 修改 tomcat 主目录下的 conf\web.xml

localhost:8080/web06\_session/count\



#### 8) 重启 tomcat 服务器

使修改的 d:/tomcat/conf/web.xml 生效

#### 9) 重新部署项目

10) 访问 [http://localhost:8080/web06\\_session/count](http://localhost:8080/web06_session/count)

11) 空闲 1 分钟后, Session 失效


### ● 方式 3

修改本应用的 web.xml

12) 将 d:/tomcat/conf/web.xml 改回默认值

13) 重启 tomcat 服务器

14) 修改本应用的 web.xml



```

6      http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd
7      <servlet>
8          <servlet-name>CountServlet</servlet-name>
9          <servlet-class>web.CountServlet</servlet-class>
10     </servlet>
11
12     <servlet-mapping>
13         <servlet-name>CountServlet</servlet-name>
14         <url-pattern>/count</url-pattern>
15     </servlet-mapping>
16     <session-config>
17         <session-timeout>1</session-timeout>
18     </session-config>
19 </web-app>
20

```

15) 访问 [http://localhost:8080/web06\\_session/count](http://localhost:8080/web06_session/count)

16) 空闲 1 分钟后, Session 失效

### 2.2.5. 删除 session \*

调用 `session.invalidate();` 方法将立即删除 Session 对象。

## 3. 【案例 4】session 验证 \*\*\*

Session 验证可以防止非登录用户通过地址栏输入地址直接访问受保护的页面。

### step1

在登录成功之后, 在 session 上绑定一些数据。绑定的数据是自定义的, 一般情况下我们绑定用户信息, 比如: `session.setAttribute("user", user);`

### step2

在访问需要保护的页面或者资源时, 执行 `Object obj = session.getAttribute("user");` ;  
如果 obj 为 null, 说明没有登录, 一般重定向到登录页面。

Obj 不为 null 时，用户可以访问页面。

### 要求

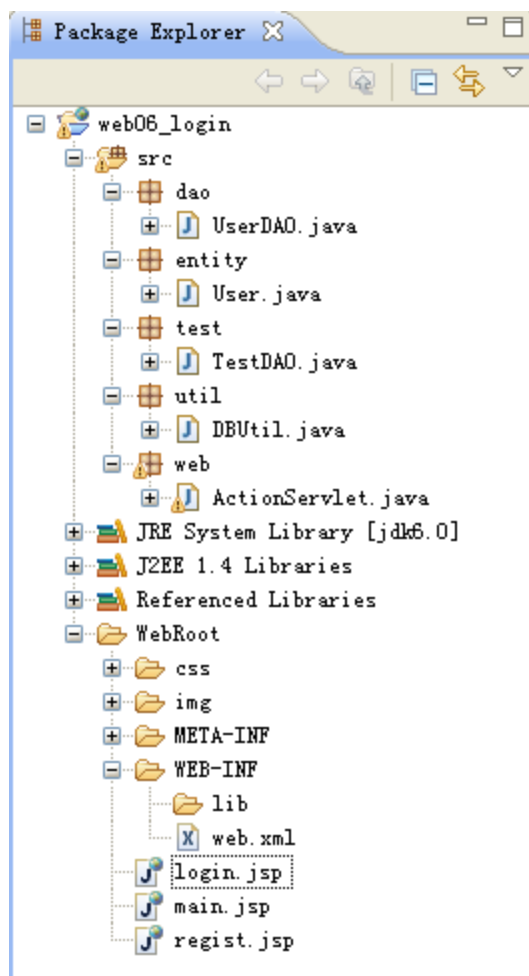
修改登录项目，添加 session 验证

请下载 web05\_login.zip

### 参考代码

1) 新建 web06\_login 项目，将 web05\_login 代码拷贝过来

2) web06\_login 项目结构



### • step1

在 session 上绑定一些数据

3) 修改 ActionServlet

```

ActionServlet.java
54 }else if(path.equals("/login")){
55     String username =
56         request.getParameter("username");
57     String pwd = request.getParameter("pwd");
58     UserDao dao = new UserDao();
59     try {
60         User user =
61             dao.findByUsername(username);
62         if(user!=null &&
63             user.getPwd().equals(pwd)){
64             //登录成功
65             //向session中绑定数据
66             HttpSession session =
67                 request.getSession();
68             System.out.println("sessionId:"
69                 + session.getId());
70             session.setAttribute("user", user);
71
72             response.sendRedirect("main.jsp");
73         }else{
74             //登录失败
75             request.setAttribute("login_error",

```

- step2

在访问需要保护的页面执行 `Object obj = session.getAttribute("user");` 语句

#### 4) 修改 main.jsp

注意：默认情况下，在jsp页面中是可以直接使用 session 的

```

main.jsp
1 <%@page pageEncoding="utf-8"
2 contentType="text/html; charset=utf-8" %>
3 <%
4     //session验证
5     Object obj = session.getAttribute("user");
6     if(obj == null){
7         response.sendRedirect("login.jsp");
8         return;
9     }
10 %>
11 <h1>主功能页面</h1>
12

```

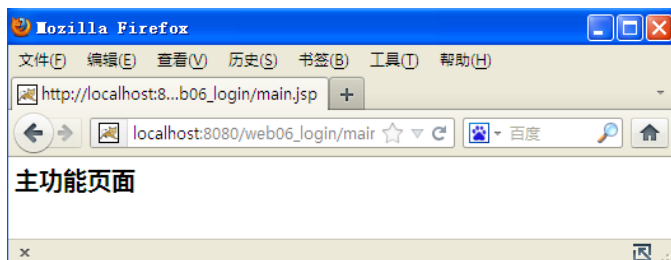
#### 5) 部署项目

#### 6) 访问 localhost:8080/web06\_login/main.jsp



不能访问，将重定向到 login.jsp

## 7) 使用用户 zs 密码 test 登录成功



## 8) 访问其他网站，比如 bbs.tarena.com.cn

## 9) 直接在地址栏输入 localhost:8080/web06\_login/main.jsp

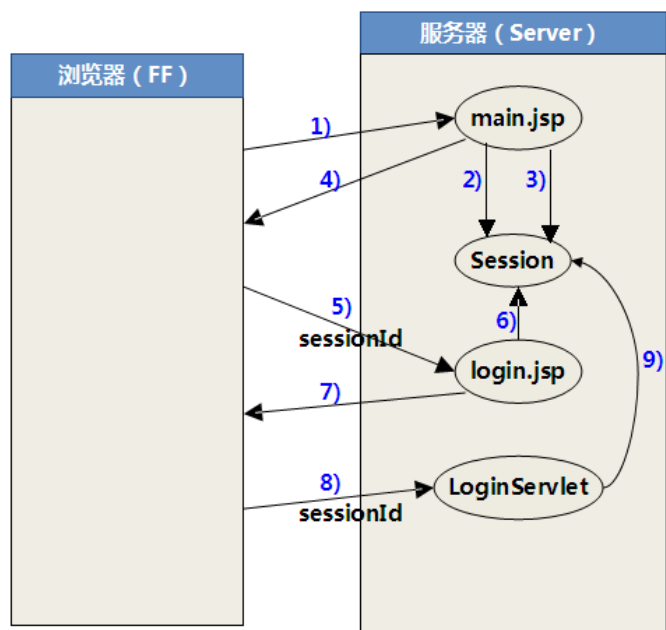
可以访问

session 验证成功。

### 图示说明

#### 图示 1

- 1) Browser 从地址栏访问 main.jsp
- 2) 服务器创建一个 Session 对象
- 3) main.jsp 执行 session.getAttribute("user"); 代码进行 session 验证，找不到 user
- 4) 返回 302 和重定向地址
- 5) 重定向到 login.jsp
- 6) login.jsp 使用之前的 session  
因为任意的 jsp 都会执行 request.getSession() 语句
- 7) login.jsp 生成登录页面并发送给浏览器
- 8) 用户输入 "zs" "test" 调用了 login.do 向 LoginServlet 发请求
- 9) LoginServlet 使用 Session 对象，向其中绑定数据 user
- 10) 登录成功，重定向到 main.jsp
- 11) 当用户再次在地址栏直接访问 main.jsp 时，因为 Session 对象中已经有了验证数据 user，所以直接可以访问



图示 2

### Session 验证正确

- 1) 用户通过 login.do 访问 LoginServlet
- 2) **step1** LoginServlet 将 user 对象绑定到 Session 对象上
- 3) 服务器发送 302 状态码 (包含 sessionId)
- 4) 浏览器将 sessionId 保存到内存
- 5) 浏览器重定向到 main.jsp (包含 sessionId)
- 6) **step2** main.jsp 从之前的 Session 对象中取到 user 对象 (验证成功)
- 7) main.jsp 将生成的页面返回给浏览器

