Cookie和Session

- Cookie和Session
  - Cookie
    - 什么是Cookie
    - Cookie的创建
    - 服务器如何获取Cookie
    - Cookie值的修改
    - Cookie生命控制
    - Cookie有效路径Path的设置
    - Cookie练习---免输入用户名登录
  - Session会话
    - 什么是Session会话
    - 如何创建 Session 和获取(id 号,是否为新）
    - Session 域数据的存取
    - Session 生命周期控制
      - Session 默认的超时时长
      - Session 超时的概念介绍
    - 浏览器和 Session之间关联的技术内幕

Cookie

什么是Cookie

1、Cookie 翻译过来是饼干的意思

2、Cookie 是服务器通知客户端保存键值对的一种技术

3、客户端有了 Cookie 后，每次请求都发送给服务器

4、每个Cookie的大小不能超过4kb

Cookie的创建


```java
//Servlet 程序中的代码
protected void createCookie(HttpServletRequest req,
HttpServletResponse resp) throws ServletException,IOException {
    //1 创建 Cookie 对象
    Cookie cookie = new Cookie("key4", "value4");
    //2 通知客户端保存 Cookie
    resp.addCookie(cookie);
    //1 创建 Cookie 对象
    Cookie cookie1 = new Cookie("key5", "value5");
    //2 通知客户端保存 Cookie
    resp.addCookie(cookie1);

    resp.getWriter().write("Cookie 创建成功");
}
```

服务器如何获取Cookie

服务器获取客户端的 Cookie 只需要一行代码：`req.getCookies():Cookie[]`


```java
public class CookieUtils {

    /**
     * 查找指定名称的Cookie对象
     * @param name
     * @param cookies
     * @return
     */
    public static Cookie findCookie(String name, Cookie[] cookies){
        if(name  == null || cookies == null || cookies.length == 0){
            return null;
        }

        for(Cookie cookie : cookies){
            if(name.equals(cookie.getName())){
                return cookie;
            }
        }

        return null;
    }
}
```

```java
protected void getCookie(HttpServletRequest req, HttpServletResponse resp) throws ServletException,IOException {
    Cookie[] cookies = req.getCookies();
```

```
    for (Cookie cookie : cookies) {
        // getName 方法返回 Cookie 的 key（名）
        // getValue 方法返回 Cookie 的 value 值
        resp.getWriter().write("Cookie[" + cookie.getName() +
"=" + cookie.getValue() + "] <br/>");
    }
    Cookie iWantCookie = CookieUtils.findCookie("key1",
cookies);
    // for (Cookie cookie : cookies) {
        // if ("key2".equals(cookie.getName())) {
            // iWantCookie = cookie;
            // break;
        // }
    // }
    // 如果不等于 null，说明赋过值，也就是找到了需要的 Cookie
    if (iWantCookie != null) {
        resp.getWriter().write("找到了需要的 Cookie");
    }
}
```

Cookie值的修改

方案一：

1. 先创建一个要修改的同名（key）的Cookie对象。
2. 在构造器，同时赋予新的Cookie值。
3. 调用response.addCookie(Cookie);

```
// 方案一：
// 1、先创建一个要修改的同名的 Cookie 对象
// 2、在构造器，同时赋于新的 Cookie 值。
    Cookie cookie = new Cookie("key1","newValue1");
// 3、调用 response.addCookie( Cookie ); 通知 客户端 保存修改
    resp.addCookie(cookie);
```

方案二：

1. 先查找到需要修改的Cookie对象
2. 调用setValue()方法赋予新的Cookie值
3. 调用response.addCookie()通知客户端保存修改

```
// 方案二：
// 1、先查找到需要修改的 Cookie 对象
    Cookie cookie = CookieUtils.findCookie("key2", req.getCookies());
    if (cookie != null) {
        // 2、调用 setValue()方法赋于新的 Cookie 值。
        cookie.setValue("newValue2");
        // 3、调用 response.addCookie()通知客户端保存修改
        resp.addCookie(cookie);
    }
```

Cookie生命控制

> *Cookie 的生命控制指的是如何管理 Cookie 什么时候被销毁（删除）*

setMaxAge()

- 正数，表示在指定的秒数后过期
- 负数，表示浏览器一关，Cookie 就会被删除（默认值是-1）
- 零，表示马上删除 Cookie

```
/**
* 设置存活 1 个小时的 Cooie
* @param req
* @param resp
* @throws ServletException
* @throws IOException
*/
protected void life3600(HttpServletRequest req, HttpServletResponse resp) throws ServletException,IOException {
    Cookie cookie = new Cookie("life3600", "life3600");
    cookie.setMaxAge(60 * 60); // 设置 Cookie 一小时之后被删除。无效
    resp.addCookie(cookie);
```

```
12      resp.getWriter().write("已经创建了一个存活一小时的 Cookie");
13  }
14
15  /**
16  * 马上删除一个 Cookie
17  * @param req
18  * @param resp
19  * @throws ServletException
20  * @throws IOException
21  */
22  protected void deleteNow(HttpServletRequest req,
    HttpServletResponse resp) throws ServletException,IOException {
23      // 先找到你要删除的 Cookie 对象
24      Cookie cookie = CookieUtils.findCookie("key4",
    req.getCookies());
25      if (cookie != null) {
26          // 调用 setMaxAge(0);
27          cookie.setMaxAge(0); // 表示马上删除，都不需要等待浏览器关闭
28          // 调用 response.addCookie(cookie);
29          resp.addCookie(cookie);
30          resp.getWriter().write("key4 的 Cookie 已经被删除");
31      }
32  }
33
34  /**
35  * 默认的会话级别的 Cookie
36  * @param req
37  * @param resp
38  * @throws ServletException
39  * @throws IOException
40  */
41  protected void defaultLife(HttpServletRequest req,
    HttpServletResponse resp) throws ServletException,IOException {
42      Cookie cookie = new Cookie("defalutLife","defaultLife");
43      cookie.setMaxAge(-1);//设置存活时间
44      resp.addCookie(cookie);
45  }
```

Cookie有效路径Path的设置

> *Cookie 的path 属性可以有效的过滤哪些Cookie 可以发送给服务器。哪些不发。path 属性是通过请求的地址来进行有效的过滤。*

CookieA **path=/工程路径**

CookieB **path=/工程路径/abc**

请求地址如下：

http://ip:port/工程路径/a.html

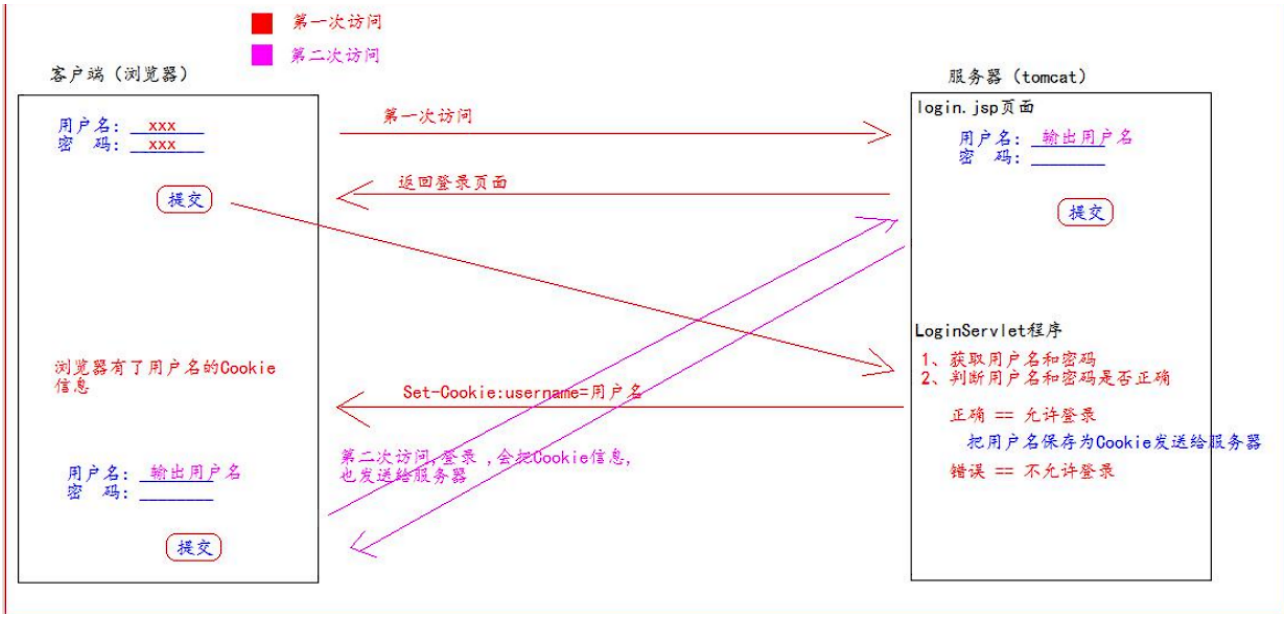
- CookieA 发送
- CookieB 不发送

http://ip:port/工程路径/abc/a.html

- CookieA发送
- CookieB发送

```
protected void testPath(HttpServletRequest req,
HttpServletResponse resp) throws ServletException,IOException {
    Cookie cookie = new Cookie("path1", "path1");
    // getContextPath() ---> 得到工程路径
    cookie.setPath( req.getContextPath() + "/abc" ); // ---> /工程
路径/abc
    resp.addCookie(cookie);
    resp.getWriter().write("创建了一个带有 Path 路径的 Cookie");
}
```

Cookie练习---免输入用户名登录



login.jsp页面

```
<form
action="http://localhost:8080/13_cookie_session/loginServlet"
method="get">
    用户名：<input type="text" name="username"
value="${cookie.username.value}"> <br>
    密码：<input type="password" name="password"> <br>
    <input type="submit" value="登录">
</form>
```

LoginServlet程序

```
@Override
protected void doGet(HttpServletRequest req, HttpServletResponse
resp) throws ServletException,IOException {
    String username = req.getParameter("username");
    String password = req.getParameter("password");
    if ("wzg168".equals(username) && "123456".equals(password))
{
        //登录 成功
        Cookie cookie = new Cookie("username", username);
        cookie.setMaxAge(60 * 60 * 24 * 7);//当前 Cookie 一周内有效
        resp.addCookie(cookie);
        System.out.println("登录 成功");
```

```
    } else {
        // 登录 失败
        System.out.println("登录 失败");
    }
}
```

Session会话

什么是Session会话

1. Session 就一个接口（HttpSession）。
2. Session 就是会话。它是用来维护一个客户端和服务器之间关联的一种技术。
3. 每个客户端都有自己的一个 Session 会话。
4. Session 会话中，我们经常用来保存用户登录之后的信息

如何创建 Session 和获取(id 号,是否为新）

如何创建和获取 Session。它们的 API 是一样的。

```
request.getSession()
    //第一次调用是：创建 Session 会话
    //之后调用都是：获取前面创建好的 Session 会话对象。
```

```
isNew();
    //判断到底是不是刚创建出来的（新的）
    //true 表示刚创建
    //false 表示获取之前的创建
```

每个会话都有一个身份证号。也就是 ID 值。而且这个 ID 是唯一的。

```
getId()
    //得到 Session
```

```
protected void createOrGetSession(HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException{
    //创建和获取Session会话对象
    HttpSession session = req.getSession();
    //判断当前Session会话，是否是新创建出来的
    boolean isNew = session.isNew();
    //获取Session会话的唯一标识id
    String id = session.getId();
    resp.getWriter().writer("得到的Session，id为：" + id + "
<br>");
    resp.getWriter().writer("这个Session是否是新创建的：：" + isNew
+ "<br>");
    //记得在web.xml中配置
}
```

Session 域数据的存取

```
/**
* 往 Session 中保存数据
* @param req
* @param resp
* @throws ServletException
* @throws IOException
*/
protected void setAttribute(HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException {
    req.getSession().setAttribute("key1", "value1");
    resp.getWriter().write("已经往 Session 中保存了数据");
}
/**
* 获取 Session 域中的数据
* @param req
* @param resp
* @throws ServletException
* @throws IOException
*/
protected void getAttribute(HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException {
```

```
20      Object attribute = req.getSession().getAttribute("key1");
21      resp.getWriter().write("从 Session 中获取出 key1 的数据是：" +
    attribute);
22  }
```

Session 生命周期控制

```
1  public void setMaxInactiveInterval(int interval)
2      //设置 Session 的超时时间（以秒为单位），超过指定的时长，Session就会被
    销毁
3      //值为正数的时候，设定 Session 的超时时长。
4      //负数表示永不超时（极少使用）
5
6  public int getMaxInactiveInterval()
7      //获取Session的超时时间
8
9  public void invalidate()
10     //让当前Session会话马上超时无效
11     //会使此会话无效，然后取消对任何绑定到它的对象的绑定
```

Session 默认的超时时长

Session 默认的超时时间长为 30 分

```
1  protected void defaultLife(HttpServletRequest req,
    HttpServletResponse resp) throws ServletException, IOException{
2      //获取Session的默认超时时长
3      int maxInactiveInterval =
    req.getSession().getMaxInactiveInterval();
4      resp.getWriter.write("Session的默认超时时长为：" +
    maxInactiveInterval + "秒");
5  }
```

因为在 Tomcat 服务器的配置文件 web.xml中默认有以下的配置，它就表示配置了当前 Tomcat 服务器下所有的 Session 超时配置默认时长为：30 分钟

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

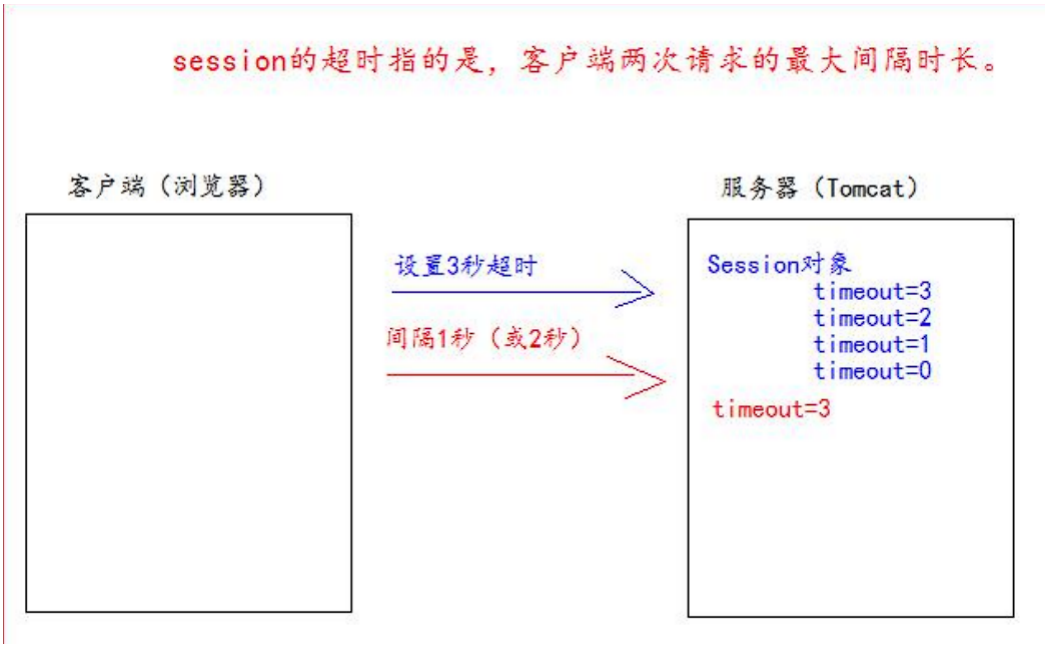
如果说，你希望你的 web 工程，默认的 Session 的超时时长为其他时长。你可以在你自己的 web.xml 配置文件中做 以上相同的配置。就可以修改你的 web 工程所有 Session 的默认超时时长。

```
<!--表示当前 web 工程。创建出来 的所有 Session 默认是 20 分钟 超时时长-->
<session-config>
    <session-timeout>20</session-timeout>
</session-config>
```

如果你想只修改个别 Session 的超时时长。就可以使用上面的 API。

```
setMaxInactiveInterval(int interval)
    //来进行单独的设置。
session.setMaxInactiveInterval(int interval)
    //单独设置超时时长
```

Session 超时的概念介绍



示例代码

```java
protected void life3(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    // 先获取 Session 对象
    HttpSession session = req.getSession();
    // 设置当前 Session3 秒后超时
   session.setMaxInactiveInterval(3);
    resp.getWriter().write("当前 Session 已经设置为 3 秒后超时");
}
```

Session超时示例

```java
protected void deleteNow(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    // 先获取 Session 对象
    HttpSession session = req.getSession();
    // 让 Session 会话马上超时
    session.invalidate();
    resp.getWriter().write("Session 已经设置为超时（无效）");
}
```

浏览器和 **Session**之间关联的技术内幕

> *Session 技术，底层其实是基于 Cookie 技术来实现的。*

返回文首