JSP/Servlet

讲义2

第4章 ServletContext和ServletConfig

ServletContext对象

- ServletContext对象是一个web应用的全局对象
- 这个对象会在一个web应用加载的时候被创建，在这个web项目中关闭的时候被销毁
- ServletContext对象和web应用的关系一对一的，不同的web应用中的ServletContext对象是不同的
- 在一个web应用中一个ServletContext对象是唯一的。任何一个资源都可以访问此对象的数据

获得ServletContext的两种方法

- **this**.getServletContext();
- **this**.getServletConfig().getServletContext();

ServeltContext对象和初始化参数相关的方法

- 可以在web.xml文件通过<context-param>标记配置键值对，然后通过方法取出值
- 取出属性值方法，如果属性不存在返回null
  - String getInitParameter(String name)

ServeltContext对象和初始化参数相关的方法示例

- 在web.xml文件中定义ServletContext初始化参数

```
<web-app>
……
    <context-param>
        <param-name>sitename</param-name>
        <param-value>http://www.xxxsite.com.cn</param-value>
    </context-param>
    <context-param>
        <param-name>username</param-name>
        <param-value>tom</param-value>
    </context-param>
……
</web-app>
```

- 取出初始化参数值

```
ServletContext context = this.getServletContext();//返回ServletContext对象
String sitename = context.getInitParameter(“sitename”);//得到sitename值
String username = context.getInitParameter(“username”);//得到username值
```

ServletContext对象和范围相关方法

- ServletContext对象也是一个范围对象，有着类似request对象在request范围内存储数据的方法
- 和范围相关的方法有setAttribute,getAttribute,removeAttribute等
- 在ServletContext范围内设置的属性值可以被此web应用中的任何程序访问到

ServletContext对象其他方法

- 根据传入的相对路径得到其在文件系统的中的物理路径
  - String getRealPath(String path)
- 例如

  String url = this.getServletContext().getRealPath("a.txt");

  out.println(url);
- 打印结果
  - D:\Tomcat 6.0\webapps\TestJSP\a.txt

ServletConfig对象

- 是一个Servlet配置对象，每一个Servlet都有一个对应ServletConfig对象，ServletConfig对象和Servlet是一对一的关系
- 获得ServletConfig对象的方法
  - ServletConfig config = **this**.getServletConfig();

ServletConfig对象和初始化参数相关的方法

- 可以在web.xml文件中，<servlet>标记中配置<init-param>标记来设置Servlet初始化参数，然后通过此对象的getInitParameter方法获得属性值

ServletConfig对象和初始化参数相关的方法示例

- web.xml文件配置

```
<servlet>
        <servlet-name>FirstServlet</servlet-name>
        <servlet-class>com.china.soft.FirstServlet</servlet-class>
        <init-param>
                <param-name>txt</param-name>
                <param-value>hello,world</param-value>
        </init-param>
</servlet>
```

- 获得属性值

```
ServletConfig config = this.getServletConfig();
String txt = config.getInitParameter("txt");
out.println(txt);
```

第5章 会话跟踪

会话跟踪

- http协议是无状态的，浏览器和服务器并不是持久连接的，而是请求完数据之后就断开连接，这样可以降低资源的浪费，缺点就是无法维护通信的状态
- 例如在一个网页登陆之后，在另外一个网页无法得知这个用户是登陆了还是没有登陆
- 我们必须用一些技术手段来保留会话中的一些数据，这种技术手法称为“会话跟踪”，常用的会话跟踪手法有四种：
  - **url重写** 即在地址栏的url地址后面追加get形式的参数：?a=xx&b=xx
  - **隐藏域** 在表单里设置隐藏域存储数据，然后提交的时候一起提交过去
  - **cookie对象** 将数据设置在客户端cookie文件中，然后在另一个页面读取
  - **HttpSession对象** 将数据存储在session对象中，然后在另一个页面读取

Cookie对象

- cookie是一个纯文本文件，它以键值对的形式将信息记录下来，存放在客户端
- 创建Cookie对象示例

  Cookie username = new Cookie("username","tianshui");

  Cookie password = new Cookie("password","123456");
- Cookie对象的常用方法
  - setMaxAge 设置Cookie过期时间，以秒计。如果不设置该值，则Cookie仅在当前会话有效，即用户关闭浏览器之前有效，不会保存到Cookie文件中
  - getName 获得Cookie的名字
  - getValue 获得Cookie的值

写入Cookie示例代码

```
//创建cookie对象
Cookie username = new Cookie("username","tianshui");
Cookie password = new Cookie("password","123456");
//给用户名和密码的cookie设置失效时间，默认按秒算
//如果不设置，默认当前会话有效
username.setMaxAge(60 * 60 * 24 * 365);//一年有效
password.setMaxAge(60 * 60 * 24 * 365);//一年有效
//写入cookie对象到客户端
response.addCookie(username);
response.addCookie(password);
```

读取Cookie示例

```
//声明字符串变量
String username = "";
String password = "";
//获得cookie数组
Cookie[] cookies = request.getCookies();
//如果没有任何cookie，数组对象为null，防止出现空指针异常
if (cookies != null)
//遍历cookie，找到用户名和密码
for (Cookie cookie : cookies){
    if (cookie.getName().equals("username"))
        username = cookie.getValue();
    if (cookie.getName().equals("password"))
        password = cookie.getValue();
}
//输出用户名和密码
response.getWriter().println(username+"/"+password);
```

HttpSession对象

- javax.servlet.http.HttpSession接口对象是目前用的最广泛的会话跟踪技术，俗称session
- 当客户端向服务器发出请求之后，服务器会给这个请求开辟一块单独的内存空间，然后产生一个对于当前服务器端内存来说不重复的id，也就是sessionID，在响应的时候把这个sessionID发送给客户端，当用户下次请求的时候需要把这个sessionID回传给服务器，服务器通过这个ID找到对应的缓冲区。
- 不同的session的ID不同，所以能够区分不同的客户端，这个sessionID是一个128位的二进制数, 在客户端被保存浏览器进程中

获得session对象

- 可以通过如下两种方式获得session对象
  - HttpSession session = request.getSession();
  - HttpSession session = request.getSession(false);
- 第一种方法获得session对象的特点是：如果当前这个会话的客户端的session对象已经创建过，则返回这个sesssion对象，如果这个客户端是第一次请求则创建一个新的session对象并返回这个新的session对象
- 第二种方法获得session对象的特点是：如果当前这个会话的客户端的session对象已经创建过，则返回这个sesssion对象，如果这个客户端是第一次请求，以前的session对象不存在则不创建session对象，返回一个null

Session对象的范围相关的方法

- 同request,ServletContext对象一样，session也是一个和范围相关的对象，可以在一个会话的范围内存取数据
- 和范围相关的方法 setAttribute,getAttribute,removeAttribute
- 如果由一个浏览器派生出另外一个浏览器窗口，那么这两个浏览器窗口属于一个浏览器进程，共享一个session对象

禁用客户端cookie的问题

- 系统自动会把sessionID存储到客户端的会话级的cookie中，这个cookie的名字为jsessionid，值就是sessionID
- 如果客户端禁用了cookie，那么就无法把session放到cookie中，那么session就会失效
- 解决的方法是把sessionID以jsessionid=sessionid的形式放到URL中，普通链接URL用response.encodeURL方法，重定向的时候用encodeRedirect方法

判断用户是否开启cookie

```
<script>
    function CookieEnable() {
        var result = false;
        if (navigator.cookiesEnabled)
            return true;
        document.cookie = "testcookie=yes;";
        var cookieSet = document.cookie;
        if (cookieSet.indexOf("testcookie=yes") > -1)
            result = true;
        document.cookie = "";
        return result;
    }
    if (!CookieEnable()) {
        alert("对不起，您的浏览器的Cookie功能被禁用，请开启");
    }
</script>
```

session的生命周期

- 客户端从最后一次请求服务器开始，如果在一定时间内没有对服务器发出任何请求，那么服务器就会销毁这个会话的session对象，这一段时间叫做session的超时时长
- 获得最后一次请求的时间
  - **long** time = session.getLastAccessedTime();
- 获得超时时长（按秒计）
  - **long** time = session.getMaxInactiveInterval();
- 设置超时时长（按秒计）
  - session.setMaxInactiveInterval(60);
- 在web.xml文件中设置超时时长（按分钟计）

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

session的其他方法

- 显式回收当前的session对象
  - void invalidate()
- 返回sessionID
  - String getId()
- 取得session的创建时间
  - long getCreationTime()
- 判断当前session对象是否是新创建的
  - boolean isNew()

Servlet线程的安全性

- Servlet容器对每一个Servlet，只创建一个实例，如果由多个客户端请求同时访问一个Servlet，则每个客户端单独启动一个线程，所以Servlet的访问形式是“单实例多线程”的
- 如果是Servlet内的局部变量，由于声明在方法内部，所以是线程安全的
- 如果是Servlet内的实例变量，由于是属于同一个实例，所以是线程不安全的
- 可以让Servlet实现SingleThreadModel接口，保证Servlet同一时刻只为一个客户端服务，但是不推荐使用
- 关于范围相关的对象，request对象是线程安全的，session对象和ServletContext对象是线程不安全的