# 1. 项目的开发

项目开发过程简介：

（1）需求分析，完成《需求规格说明书》。亲身体会，一定要舍得在前期需求和开发上下功夫，这样虽然前期成本较高，但以后维护会轻松很多，否则前期无法把握好的话，后期维护将不堪重负，甚至项目夭折。

（2）《概要设计说明书》

（3）《详细设计说明书》

（4）开发过程：团队协作。

（5）《测试报告》

项目一般会采用如下技术：

（1）页面一般用JSP等技术展现；

（2）数据的传递一般使用MVC；

（3）使用三层开发模型。

下面讲两个解决实际问题的点。

# 2. 解决表单的重复提交

在用表单提交数据时，很可能会遇到表单重复提交的情况，以下两种情况就会造成表单重复提交：

（1）用户点击了提交按钮，但服务器处理较慢，还未响应页面，此时用户又点击了提交按钮；

（2）用户点击了提交按钮，服务器成功处理并跳转（不是重定向）到了其他页面，此时用户刷新了这个新的页面，会造成表单重复提交。

读者可以对上述情况进行验证（对于情况1，可以使用Thread.sleep(2000)进行模拟），会发现无论是GET还是POST请求，都有重复提交的情况。如果不解决表单重复提交的问题，则可能导致用户响应结果不一致，或数据重复添加。

如何解决表单重复提交问题呢？对于情况1，可以在用户点击按钮后马上禁用按钮，对于情况2，可使用重定向到新页面（即Redirect After Post）。

这里再提供一种在服务器端防止表单重复提交的手段，从源头禁止重复提交的发生。我们使用令牌（token）机制。具体如下：

（1）JSP中产生一个唯一的token，并将此token保存在session中，这样只要用户访问到JSP就会产生token；

（2）JSP的表单中使用一个input隐藏域，值就是这个token，这样表单提交时会带着这个token提交；

（3）服务器端拿到表单上传数据中的token，与session中的token对比，如果一致说明是首次提交，然后进行数据处理并将session中的token删除；如果不一致，说明不是首次提交，不予处理。

示例代码如下：

JSP中：

|  |
| --- |
| <%@ **page import**="**br.bean.Student**" %> <%@ **page import**="**java.util.UUID**" %> <%@ **page contentType**="**text/html;charset=UTF-8**" **language**="**java**" %> <**html**> <**head**>  <**title**>提交</**title**> </**head**> <**body**>  **<%** *// 用UUID生成token* String token = UUID.*randomUUID*().toString();  *// 保存token到session中* session.setAttribute(**"token"**, token);  **%>** <**form action="${**pageContext.request.contextPath**}/resubmitServlet" method="POST"**>  *<%-- 表单隐藏域中包含token --%>* <**input type="hidden" name="token" value="<%=**token**%>"**>  姓名：<**input type="text" name="name"**>  <**input type="submit"**>  </**form**> </**body**> </**html**> |

Servlet：

|  |
| --- |
| **package** br.action;  **import** javax.servlet.ServletException; **import** javax.servlet.annotation.WebServlet; **import** javax.servlet.http.HttpServlet; **import** javax.servlet.http.HttpServletRequest; **import** javax.servlet.http.HttpServletResponse; **import** javax.servlet.http.HttpSession; **import** java.io.IOException;  @WebServlet(**"/resubmitServlet"**) **public class** ResubmitServlet **extends** HttpServlet {  @Override  **protected void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  HttpSession session = req.getSession();  String sessionToken = (String) session.getAttribute(**"token"**); *// session中的token* **if** (req.getParameter(**"token"**).equals(sessionToken)) { *// 对比  // 相等说明是首次提交* System.***out***.println(**"这是首次提交，name为："** + req.getParameter(**"name"**));  req.getRequestDispatcher(**"/2.jsp"**).forward(req, resp);  }  *// 最后删除session中的token即可* session.removeAttribute(**"token"**);  *// 如果不是首次提交，什么也不做* }   @Override  **protected void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  doGet(req, resp);  } } |

为了避免在JSP中使用代码，我们写一个自定义标签：

|  |
| --- |
| **package** br.el;  **import** javax.servlet.http.HttpSession; **import** javax.servlet.jsp.JspException; **import** javax.servlet.jsp.PageContext; **import** javax.servlet.jsp.tagext.SimpleTagSupport; **import** java.io.IOException; **import** java.util.UUID;  **public class** TokenTag **extends** SimpleTagSupport {  @Override  **public void** doTag() **throws** JspException, IOException {  String token = UUID.*randomUUID*().toString(); *// 产生token* PageContext pc = (PageContext) getJspContext();  HttpSession session = pc.getSession(); *// 获得session* session.setAttribute(**"token"**, token);  *// 向页面输出一个隐藏域* pc.getOut().write(**"<input type='hidden' name='token' value='"** + token + **"'/>"**);  } } |

token.tld文件为：

|  |
| --- |
| *<?***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-jsptaglibrary\_2\_1.xsd"  version="2.1"**>  <**tlib-version**>1.0</**tlib-version**>  <**short-name**>token</**short-name**>  <**uri**>http://www.demo.com/token</**uri**>  <**tag**>  <**name**>setToken</**name**>  <**tag-class**>br.el.TokenTag</**tag-class**>  <**body-content**>empty</**body-content**>  </**tag**> </**taglib**> |

则我们的JSP中只要在form表单中使用该标签即可：

|  |
| --- |
| <%@ **page contentType**="**text/html;charset=UTF-8**" **language**="**java**" %> <%@**taglib prefix**="**token**" **uri**="**http://www.demo.com/token**" %> <**html**> <**head**>  <**title**>提交</**title**> </**head**> <**body**>  <**form action="${**pageContext.request.contextPath**}/resubmitServlet" method="POST"**>  *<%-- setToken需要在表单中写，在其他地方写没有作用 --%>* <**token:setToken**/>  姓名：<**input type="text" name="name"**>  <**input type="submit"**>  </**form**> </**body**> </**html**> |

为了方便Servlet处理，可以写这样一个过滤器：

|  |
| --- |
| **package** br.filter;  **import** javax.servlet.\*; **import** javax.servlet.annotation.WebFilter; **import** javax.servlet.http.HttpServletRequest; **import** javax.servlet.http.HttpServletResponse; **import** javax.servlet.http.HttpSession; **import** java.io.IOException;  @WebFilter(urlPatterns = {**"/\*"**}) **public class** TokenFilter **implements** Filter {  @Override  **public void** init(FilterConfig filterConfig) **throws** ServletException { }   @Override  **public void** doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) **throws** IOException, ServletException {  HttpServletRequest request;  HttpServletResponse response;  **try** {  request = (HttpServletRequest) req;  response = (HttpServletResponse) resp;  } **catch** (ClassCastException e) {  **throw new** ServletException(**"non-HTTP request or response"**);  }  HttpSession session = request.getSession();  String sToken = (String) session.getAttribute(**"token"**);  String fToken = request.getParameter(**"token"**);  **if**(fToken != **null**){  **if**(!fToken.equals(sToken)){  response.getWriter().write(**"Please do not resbumit"**);  **return**;  }  session.removeAttribute(**"token"**); *// 删除* }  chain.doFilter(request, response); *// 放行* }   @Override  **public void** destroy() { } } |

后续可以再研究下AJAX的重复提交。

# 3. 项目中的权限控制

在实际项目中，基本都需要权限的控制。权限控制就是管理不同用户所拥有的不同功能，例如管理员可登录商城后台添加商品、分类等信息，而普通用户可以浏览和购买商品，而不允许管理商品。

权限控制有粗粒度和细粒度两种管理方式。粗粒度就是根据网站的URL地址设置权限，拥有该权限相当于可访问应用中的该URL网址，否则无法进入。细粒度就是细到每一个按钮的操作都要进行权限控制，即在每次调用方法之前就要进行权限验证。

现在讲粗粒度的权限控制中的一些概念和其中大致是如何实现的。

粗粒度权限模型中一般有功能、权限和用户三个基本概念，作用如下：

|  |  |
| --- | --- |
| 功能 | 每个功能对应一个权限地址URL |
| 角色 | 方便给用户批量授权 |
| 用户 | 用于身份认证 |

注意的是，角色的设立就是为了方便给用户批量授权，如果没有角色的话，为每位用户设置权限会比较麻烦。有了角色，可以为用户设立角色，用户有了某角色就是拥有了该角色的所有功能，例如某用户是管理员角色，则该用户拥有管理员的所有权限。

通过分析功能、角色和用户的关系，不难发现，功能和角色是多对多的关系，角色和用户也是多对多的关系。如果要设置粗粒度权限模型的数据库表，则需要建5张表，分别是功能（Functions）、角色（Roles）和用户（Users）三张实体表，外加“角色功能表（Role\_Function）”和“用户角色表（User\_Role）”两张关系表。

下面是表结构示例：

（1）功能表：

|  |  |
| --- | --- |
| ID（PK） | 主键 |
| NAME | 功能名称，非空，唯一 |
| URL | 对应的地址 |
| DESCRIPTION | 描述 |

（2）角色表：

|  |  |
| --- | --- |
| ID（PK） | 主键 |
| NAME | 角色名称，非空，唯一 |
| DESCRIPTION | 描述 |

（3）用户表：

|  |  |
| --- | --- |
| ID（PK） | 主键 |
| USERNAME | 用户名，非空，唯一 |
| PASSWORD | 密码 |
| DESCRIPTION | 描述 |

（4）角色功能表：

|  |  |
| --- | --- |
| ID（PK） | 主键 |
| F\_ID（FK） | 外键，引用功能表主键 |
| R\_ID（FK） | 外键，引用角色表主键 |

（5）用户角色表：

|  |  |
| --- | --- |
| ID（PK） | 主键 |
| R\_ID（FK） | 外键，引用角色表主键 |
| U\_ID（FK） | 外键，引用用户表主键 |