# 上传文件

## 上传页面不刷新

### 前端

原理方式：注意：ajax是不支持文件上传的

1. 使用iframe标签关联form表单：iframe标签用name属性值和form标签的target属性值一一对应
2. 设置iframe的框为不显示：style="display: none;"
3. Form进行上传文件该有的还要设置 如：enctype="multipart/form-data"和 <input type="file">

这3步就实现了无刷新页面效果

<body>

<iframe style="display: none;" name="abc"></iframe> 页面刷新其实就是这个页面框，设置了隐藏所以看不到刷新效果

<form target="abc" method="post" action="xxx" enctype="multipart/form-data">

<input type="file" name="upload" />

<input type="submit" name="upload" />

</form>

</body>

使用jquery插件：OCUpload进行操作：它就是跟原理方式效果实现一样的

jquery.ocupload-1.1.2.js就这一个js文件

使用步骤

第一步 HTML只要定义一个：<input id="eee" type="button" value="上传 " />即可，没有特殊的要求

第二步方法：$("input[name=ddd]")..upload({

name: 'file',//上传组件的name属性，即<input type='file' name='file'/>

action: '',//向服务器请求的路径

enctype: 'multipart/form-data',//mime类型，默认即可

params: {},//请求时额外传递的参数，默认为空

autoSubmit: true,//是否自动提交，即当选择了文件，自动关闭了选择窗口后，是否自动提交请求。

onSubmit: function() {},//提交表单之前触发事件

onComplete: function() {},//提交表单完成后触发的事件

onSelect: function() {}//当用户选择了一个文件后触发事件

});

EasyUI方法

$('#ff').form('submit')

# 下载

Web技术19天销售绑定下载

public class SaleListServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

//1.调用Service中查询销售榜单的方法

OrderService service = BasicFactory.getFactory().getService(OrderService.class);

List<SaleInfo> list = service.saleList();

//2.将销售榜单信息组织称csv格式的数据

StringBuffer buffer = new StringBuffer();

buffer.append("商品编号,商品名称,销售数量\r\n");

for(SaleInfo si : list){

buffer.append(si.getProd\_id()+","+si.getProd\_name()+","+si.getSale\_num()+"\r\n");

}

String data = buffer.toString();

//3.提供下载

String filename = "Estore销售榜单\_"+new Date().toLocaleString()+".csv"; //csv格式文件和excel差不多(可以普通文本打开，**逗号分隔值**)

response.setHeader("Content-Disposition", "attachment;filename="+URLEncoder.encode(filename,"utf-8"));//设置下载头

response.setContentType(this.getServletContext().getMimeType(filename));//配合用，设置下载的默认文件名

response.getWriter().write(data);//

}

# 权限管理

## 表设计和entity类

### 表设计

三张表：权限表、用户表、角色表，两个多对多关系表

权限表和角色表是多对多关系：一个角色会有很多权限，一个权限可以有很多角色使用

用户表和角色表是多对多关系:一个用户不一定只有一个角色，一个角色可以很多人用

权限表：

`id` varbinary(32) NOT NULL COMMENT '主键',

`name` varchar(255) DEFAULT NULL,

`code` varchar(255) DEFAULT NULL COMMENT '此表的唯一标识，因为主键很长繁杂',

`desc` varchar(255) DEFAULT NULL COMMENT '描述',

`url\_path` varchar(255) DEFAULT NULL COMMENT '页面路径',

`menubaar` varchar(255) DEFAULT NULL COMMENT '是否生成菜单',

`zindex` int(11) DEFAULT NULL COMMENT '优先级',

`pid` varchar(32) DEFAULT NULL COMMENT '父权限编号和本表的id对应',

角色表：其实这个表不是必须的，它是每个权限的集合，

引入角色表：是为了简化（方便）授权：假设系统有1000个角色，创建一个新用户就要一点点的为这个用户授权，太麻烦了，引入角色，每一个角色有相应的权限，那么新建的用户就直接赋予这个用户一个角色，就不用了一点一点的去点击授权了，就好比把这个权限分类

CREATE TABLE `role` (

`id` varchar(32) NOT NULL COMMENT '主键UUID',

`name` varchar(255) DEFAULT NULL COMMENT '角色名称',

`code` varchar(255) DEFAULT NULL COMMENT '此表的唯一标识，id太长不好处理',

`desc` varchar(255) DEFAULT NULL COMMENT '描述',

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

User用户表

CREATE TABLE `user` (

`id` bigint(20) NOT NULL,

`username` varchar(32) DEFAULT NULL,

`password` varchar(32) DEFAULT NULL,

`birthday` date DEFAULT NULL,

`gender` varchar(10) DEFAULT NULL COMMENT '性别',

`phone` varchar(20) DEFAULT NULL COMMENT '手机',

`remark` varchar(255) DEFAULT NULL COMMENT '备注',

PRIMARY KEY (`id`),

KEY `FK\_Reference\_1` (`password`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

关系表

角色和用户表

CREATE TABLE `user\_role` (

`user\_id` varchar(32) NOT NULL,

`role\_id` varchar(32) NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

角色和权限表

CREATE TABLE `role\_authority` (

`role\_id` varchar(32) NOT NULL,

`function\_id` varchar(32) NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

### Entity类

Role

public class Role implements java.io.Serializable {

private String id;

private String name;

private String code;

private String description;

private Set functions = new HashSet(0);//当前角色对应的多个权限

private Set users = new HashSet(0);//当前角色对应的多个用户

权限表

private String id;

private Function parentFunction;//当前权限的上级权限

private String name;

private String code;

private String description;

private String page;

private String generatemenu;//是否生成菜单，1：是 0：否

private Integer zindex;

private Set roles = new HashSet(0);//当前权限对应的多个角色

private Set children = new HashSet(0);//当前权限的下级权限

用户表

private String id;

private String username;

private String password;

private Date birthday;

private String gender;

private String station;

private String phone;

private String remark;

private Set noticebills = new HashSet(0);

private Set roles = new HashSet(0);

## 原始方法

注意：此原始和表设计没有关系

简单介绍 (web技术视频第19天) 注意：虽然low但是给你一个配置权限方向感

1使用filter(进行权限管理)

2使用txt文本每一行记录uri最后的地址：如 localhost:8080/wwl/user.jsp,这个user.jsp就是要记录的内容

3使用字符缓冲流一行一行读取文本txt内容封装到list中

使用步骤

第一步 创建filter过滤器

public class PrivilegeFilter implements Filter {

//封装从txt文件中读取的内容：每一个list代表不同权限文件中的内容

private List<String> admin\_list = new ArrayList<String>();//admin权限

private List<String> user\_list = new ArrayList<String>();//普通用户权限能访问的页面

public void destroy() {}

public void doFilter(ServletRequest request, ServletResponse response,

FilterChain chain) throws IOException, ServletException {

HttpServletRequest req = (HttpServletRequest) request;

HttpServletResponse resp = (HttpServletResponse) response;

String uri = req.getRequestURI();//获取用户访问页面的路径uri

if(admin\_list.contains(uri) || user\_list.contains(uri)){

//说明当前资源需要权限：查看用户有无登录

if(req.getSession(false)==null || req.getSession().getAttribute("user")=="null"){

response.getWriter().write("当前资源需要权限,请先登录!");

return;

}

User user = (User) req.getSession().getAttribute("user");

//判断用户请求的页面是否有权限：contains方法是判断是否有此内容

if(admin\_list.contains(uri) && "admin".equals(user.getRole())){

chain.doFilter(request, response);//放行

}else if(user\_list.contains(uri) && "user".equals(user.getRole())){

chain.doFilter(request, response);

}else{

throw new RuntimeException("您不具有对应的权限!!!");

}

}else{

//不需要权限：一般就是登录页面和注册页面

chain.doFilter(request, response);

}

}

//过滤器在启动web容器的时候就进行加载权限配置文件：放入list中(内存中)

public void init(FilterConfig filterConfig) throws ServletException {

ServletContext context = filterConfig.getServletContext();

try {

//加载admin.txt

BufferedReader adminReader = new BufferedReader(new FileReader(context.getRealPath("WEB-INF/admin.txt")));

String line = null;

while((line=adminReader.readLine())!=null){

admin\_list.add(line);

}

//加载user.txt

BufferedReader userReader = new BufferedReader(new FileReader(context.getRealPath("WEB-INF/user.txt")));

line = null;

while((line=userReader.readLine())!=null){

user\_list.add(line);

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

第二步 配置文件内容 其实老师建议放入数据库中

Admin.txt下的内容：以下内容只有admin管理员可以访问

/addProd.jsp

/AddprodServlet

/SaleListServlet

user.txt内容 以下页面普通用户可以访问

/addOrder.jsp

/cart.jsp

/confirm.jsp

/orderList.jsp

/pay.jsp

/AddCartServlet

/ChangeCartServlet

/ClearCartServlet

/DelCartServlet

/DelOrderServlet

/OrderListServlet

/PayServlet

## Shiro

https://blog.csdn.net/jinnianshilongnian/article/details/84528037 详细的shiro使用

### (认证)使用步骤

第一步 导包

除了基本的以外，我们还需要shiro-web和shiro-spring的的架包，下面是所需要的所有shiro架包，至于其他的架包，像缓存的架包，Spring和SpringMVC的架包还是和我们以前使用的架包一样的。

核心包：shiro-core-1.2.2.jar、commons-beanutils-1.8.3.jar

LOG包： slf4j-api-1.6.4.jar、commons-logging-1.1.3.jar

web工程包: shiro-web

spring整合包: shiro-spring

Maven

<!-- 日志包：commons-logging -->

<dependency>

<groupId>commons-logging</groupId>

<artifactId>commons-logging</artifactId>

<version>1.1.3</version>

</dependency>

<!-- 核心包：有三个文件： shiro-core、commons-beanutils、slf4j-api、 -->

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-core</artifactId>

<version>1.3.2</version>

</dependency>

<!-- web工程包 ：shiro-web-->

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-web</artifactId>

<version>1.2.3</version>

</dependency>

<!-- shiro和spring整合包 -->

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-spring</artifactId>

<version>1.2.3</version>

</dependency>

</dependencies>

或者一个包就搞定所有：但是官方不推荐这样做，网上也不推荐，但是bos系统老师第10天就是只用下面的包

<!-- 引入shiro框架的依赖 shiro-all不止配置上面的还配置了很多 -->

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-all</artifactId>

<version>1.2.2</version>

</dependency>

第二步 web.xml配置

这里要注意：过滤器的名称一定要和spring配置文件注入的shiro过滤器工厂中的bean属性id一致

<!-- 配置spring提供的用于整合 shiro框架L注意建议一定要先进入过滤器：放在其它框架过滤器前面 -->

<filter>

<!-- 名字要和你在applicationContext.xml注册的bean要一致，不然报错：找不到你所起的名字类-->

<filter-name>shiroFilter</filter-name>

<filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>

</filter>

<filter-mapping>

<filter-name>shiroFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

第三步 applicationContext.xml中配置shiro

注意细节：配置shiro过滤器工厂类，需要注入属性有三大类：1、安全管理器对象、2、验证后需要跳转的页面、3、拦截规则配置

<!—1、配置shiro过滤器工厂框架过滤器：id名称 web.xml中配置的过滤器需要用到 -->

<bean id="shiroFilter" class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">

<!-- 注入安全管理器对象 -->

<property name="securityManager" ref="securityManager"></property>

<!-- 注入相关页面访问URL -->

<!—认证不通过跳转页面，没有认证会直接跳转到此登录页面 -->

<property name="loginUrl" value="/login.jsp"></property>

<!-- 登录成功后跳转的页面:想springmvc都有这个功能，此url可以不配置 -->

<property name="successUrl" value="/index.jsp" ></property>

<!-- 权限不足，跳转的页面 -->

<property name="unauthorizedUrl" value="/unauthorized.jsp"></property>

<!-- 注入URL拦截的规则 -->

<property name="filterChainDefinitions">

<!-- -->

<value>

<!--anon是匿名认证期：说白点就是不用进行认证就通过 -->

/css/\*\* =anon

/js/\*\*=anon

/images/\*\* = anon

<!-- 这里是bos系统设置的验证码，不拦截验证码，加一个\*是因为它的验证码jsp下还有参数随机码(页面缓存机制不刷新)-->

/validatecode.jsp\* =anon

<!-- 权限认证 perms简称 ,参数值自定义-->

/page\_base\_staff.action= perms["staff-list"]

<!-- authc过滤器简称：此过滤器检查当前用户是否已经认证过，没有拦截一切请求包括css、js等文件，通俗讲就是：是否已经完成登录 -->

/\*=authc <!-- 上面没有匹配到的来到这里，进行认证 -->

</value>

</property>

</bean>

<!-- 2、 注册安全管理器对象 -->

<bean id="securityManager" class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">

<property name="realm" ref="bosRealm"/> //注入属性

</bean>

<!—3、注册realm类：自定义类实现此接口或继承内置已经实现此接口的类 -->

<bean id="bosRealm" class="com.itheima.bos.realm.BOSRealm"></bean>

</bean>

演示

老师在这一步配置完后进行演示了：登录操作

在没有登录情况下：在url地址栏敲上没有在拦截规则上配置的路径，会直接返回到登录页面

第四步 看第三步需要自定类realm类去实现Realm接口类或者继承

public class BOSRealm extends AuthorizingRealm{ //老师是继承此抽象类，这有两个方法一个是认证，一个授权

@Autowired

private IUserDao userDao;

//认证方法

protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws AuthenticationException {

System.out.println("自定义的realm中认证方法执行了。。。。");

//强转Token类

UsernamePasswordToken passwordToken = (UsernamePasswordToken)token;

//获得页面输入的用户名

String username = passwordToken.getUsername();

//根据用户名查询数据库中的密码：shiro框架要求这样做，按它规则走

User user = userDao.findUserByUsername(username);//自己的类：通过用户名查询数据库中的密码

if(user == null){

//返回空，证明认证失败，就会抛一个异常，账号不存在等异常

return null;

}

//简单认证信息对象 ： 1Object在web层中需要用到：把对象放入session中 2查询出来的密码，框架用于比对 3、老师说此参数为所谓，

AuthenticationInfo info = new SimpleAuthenticationInfo(user, user.getPassword(), this.getName());

//框架负责比对数据库中的密码和页面输入的密码是否一致，不一致抛异常

return info;

}

//授权方法

protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {

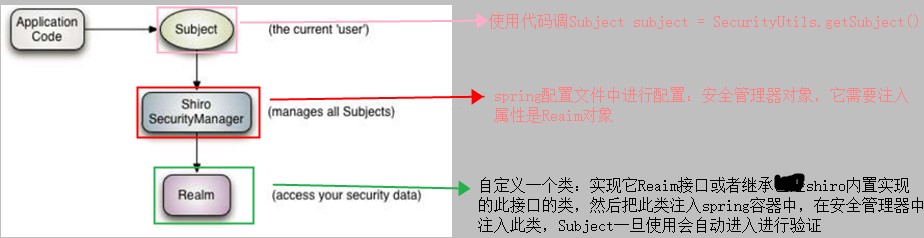
return null;

}

}

第五步 在web层把登录成功后的认证操作关联shiro框架

这里要明白流程：如何交给shiro认证：创建Subject对象把账号密码传进去生成token，框架内容自动调用安全管理器，然后安全管理器调Reaim(需要自定义类实现接口并注入bean中)，在Reaim进行认证自己判断：若返回null验证失败，



Application Code：应用程序代码，由开发人员负责开发的

Subject：框架提供的接口，代表当前用户对象

SecurityManager：框架提供的接口，代表安全管理器对象

Realm：可以开发人员编写，框架也提供一些，类似于ＤＡＯ，用于访问权限数据(可以认为最终保存的数据当成一个数据库)

//用户登录,使用shiro框架提供的方式进行认证操作

public String login(){

//从Session中获取生成的验证码

String validatecode = (String) ServletActionContext.getRequest().getSession().getAttribute("key");

//校验验证码是否输入正确

if(StringUtils.isNotBlank(checkcode) && checkcode.equals(validatecode)){

//使用shiro框架提供的方式进行认证操作开始：

Subject subject = SecurityUtils.getSubject();//获得当前用户对象,状态为“未认证”

AuthenticationToken token = new UsernamePasswordToken(model.getUsername(),MD5Utils.md5(model.getPassword()));//创建用户名密码令牌对象

try{

//看原理 内部就会自动调用你第三步和第四步配置的realm类，返回null失败

subject.login(token);

}catch(Exception e){

e.printStackTrace();

return LOGIN; //异常跳转页面到登录页面

}

User user = (User) subject.getPrincipal();//此方法是取出第四步的最后一步：简单对象验证中放入的第一个参数

ServletActionContext.getRequest().getSession().setAttribute("loginUser", user);//放入session

return HOME;

}else{

//输入的验证码错误,设置提示信息，跳转到登录页面

this.addActionError("输入的验证码错误！");

return LOGIN;

}

}