# 安全模块

## 用户密码加密存储

用户密码加密算法：对用户的密码进行sha-1算法加密。迭代1024次，并将salt放在前16位中。

SystemService中

/\*\*

\* 生成安全的密码，生成随机的16位salt并经过1024次 sha-1 hash

\*/

**public** **static** String entryptPassword(String plainPassword) {

**byte**[] salt = Digests.*generateSalt*(*SALT\_SIZE*);//16

**byte**[] hashPassword = Digests.*sha1*(plainPassword.getBytes(), salt, *HASH\_INTERATIONS*);

**return** Encodes.*encodeHex*(salt)+Encodes.*encodeHex*(hashPassword);

}

解密同步配置：因为用户密码在存储的时候是用的加密存储，所以对应的解密一定要一至细节如下。

SystemAuthorizingRealm中

@PostConstruct

**public** **void** initCredentialsMatcher() {

HashedCredentialsMatcher matcher = **new** HashedCredentialsMatcher(SystemService.*HASH\_ALGORITHM*);//SHA-1

matcher.setHashIterations(SystemService.*HASH\_INTERATIONS*);//1024

setCredentialsMatcher(matcher);

}

## 安全码验证

当同一个用户请超过3次。用户登陆页面（sysLogin.jsp）向ValidateCodeServlet请求验证图片

ValidateCodeServlet生成验证图片并存入Session中

系统扩展了SystemAuthorizingRealm ，FormAuthenticationFilter 和 UsernamePasswordToken两个类

在Shiro验证的时候FormAuthenticationFilter会提取出validateCode并存入UsernamePasswordToken中

然后在SystemAuthorizingRealm中先判断是否需要验证validateCode

如果需要则从session中获取先前存入的session code并与UsernamePasswordToken中的code进行比对

# 主题更换

在head.jsp中通过查询cookie.theme.value值来替换bootstrap的css文件，默认是'cerulean',细节如下：

<link href="${ctxStatic}/bootstrap/2.3.1/css\_${not empty cookie.theme.value ? cookie.theme.value:'cerulean'}/bootstrap.min.css" type="text/css" rel="stylesheet" />

更改主题的接口为：

LoginControler中

@RequestMapping(value = "/theme/{theme}")

**public** String getThemeInCookie(@PathVariable String theme, HttpServletRequest request, HttpServletResponse response){

**if** (StringUtils.*isN*otBlank(theme)){

CookieUtils.*setCookie*(response, "theme", theme);

}**else**{

theme = CookieUtils.getCookie(request, "theme");

}

**return** "redirect:"+request.getParameter("url");

}

# 缓存设计

对每一个用户所需要用到的资源都用Map做了缓存。如果用户不存在则是创建一个Map<String, Object>对象，如果存在，则取principal中的Map<String, Object>对象做为缓存，因为principal会随着用户的logout自动释放，每一个用户都有自己的缓存。这样每个用户的缓存是独立的。获取逻辑如下

**public** **static** Map<String, Object> getCacheMap(){

Map<String, Object> map = Maps.*newHashMap*();

**try**{

Subject subject = SecurityUtils.*getSubject*();

Principal principal = (Principal)subject.getPrincipal();

**return** principal!=**null**?principal.getCacheMap():map;

}**catch** (UnavailableSecurityManagerException e) {

}**catch** (InvalidSessionException e){

}

**return** map;

}

放入Shiro管理的principal是自定义的如下：

SystemAuthorizingRealm中的子类

public static class Principal implements Serializable

# 用户信息模块

与用户相关的接口全部在UserController中。

所有接口先统一通过下面代码获取用户信息。如果当前用户没有给定id则认为是新用户。创建一个空用户  
 @ModelAttribute

**public** User get(@RequestParam(required=**false**) String id) {

**if** (StringUtils.*isNotBlank*(id)){

**return** systemService.getUser(id);

}**else**{

**return** **new** User();

}

}

## 创建用户接口

/sys/user/form

从下面代码可以看出，谁创建的就以谁的公司和办公室为默认值

**if** (user.getCompany() == **null** || user.getCompany().getId() == **null**) {

user.setCompany(UserUtils.*getUser*().getCompany());//用当前用户所在的公司创建

}

**if** (user.getOffice() == **null** || user.getOffice().getId() == **null**) {

user.setOffice(UserUtils.*getUser*().getOffice());//用当前用户所在办公室创建

}

## 默认根路径跳转

<!-- 定义无Controller的path<->view直接映射 -->

<mvc:view-controller path=*"/"* view-name=*"redirect:${web.view.index}"*/>

# 翻页处理

com.thinkgem.jeesite.common.persistence.Page<T>

的toString()方法中实现BootStrap显示细节，因为数据在Page之中。在Jsp页面中使用<div class="pagination">${page}</div>

# roleForm.jsp中树形目录选择

## office弹出对话框式树型选择

用tags:treeselect标签将界面操作逻辑封装

在tags:treeselect中用jBox来调用/tag/treeselect转向tagTreeselect.jsp页面，并传入相关参数，其中重要的参数为url，即要展示的JSON格式数据来源

tagTreeselect.jsp：中只负责数据展示，与选择等相关操作。（用zNode tree来展示）

当发生“确定”与双击操作时都是调用jBox的ok键

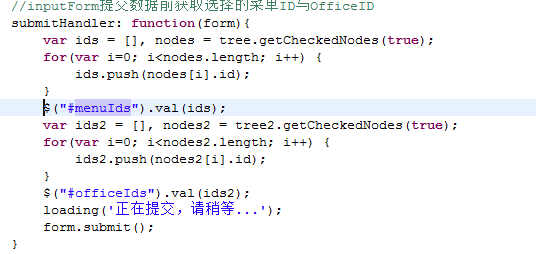
而ok键事件发生后：jBox会拿到tree结点对像，并从对像上获取id和name并写入input中

## 角色授权树型选择操作

先通过后台传过来的数据构建zNode tree

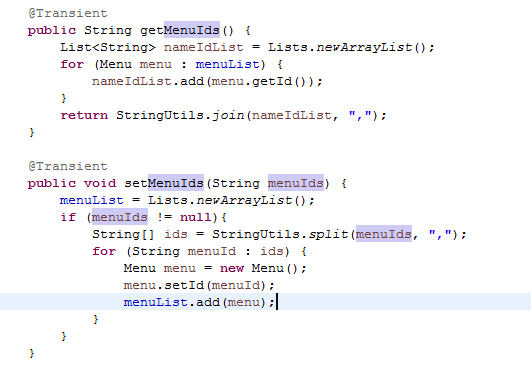
由zNode tree来管理数据的选择

在表单提交时（submitHandler）获取选择数据并添加到相应的input中。然后提交。如下图



这里用了一个小技巧。springMVC 进行前后台数据绑定的时候其实是调用Model的set与get方法。（**所以只要有这两个方法即可，不用有成员变员也行**）

给Role模型添加了两个方法，并用Transient来标记不写入数据库。如下



这样就可以自动把数据写回到Role中