1. **简介**

依赖环境:

JDK7.0+, Spring3.0+,mybatis3.0

1. **身份认证**

配置文件如下:

第一步:自定义认证逻辑

创建一个处理认证逻辑的类(AuthService),集成基础类(BaseAuthService),实现perfectUserBean以及validateAccToken方法:

perfectUserBean方法:通过token令牌初始化用户信息,返回UserBean(此类为个抽象类,应用自定义用户信息结构继承即可)

validateAccToken方法:此方法用来自定义验证访问的令牌是否合法的逻辑.

代码片段如下:

@Service("authService")  
@Transactional  
public class AuthService extends BaseAuthService {  
 protected static Logger *logger* = LoggerFactory.*getLogger*(AuthService.class);  
 @Autowired  
 private SysUserMapper sysUserMapper;  
 @Autowired  
 private SysTokenMapper sysTokenMapper;  
 @Override  
 public UserBean perfectUserBean(String accToken) {  
 VSysuser vSysuser = null;  
 if (accToken != null) {  
 vSysuser = new VSysuser();  
 SysUser sysUser = (SysUser) Springkit.*getRequest*().getSession().getAttribute(BizConstants.*SESSION\_USER*);  
 vSysuser.setPersonName(sysUser.getRealName());  
 vSysuser.setUserId(sysUser.getUserCode());  
 vSysuser.setSchoolCode(sysUser.getSchoolCode());  
 vSysuser.setSchoolName(sysUser.getSchoolName());  
 vSysuser.setUserNo(sysUser.getUserNo());  
 vSysuser.setUserName(sysUser.getUserName());  
 vSysuser.setUserHead(sysUser.getImgUrl());  
 } else {  
 return null;  
 }  
 return vSysuser;  
 }  
  
 @Override  
 public String validateAccToken(AccTokenBean accTokenBean) throws BizException {  
 String newToken = null;  
 //访问的token信息  
 String accToken = accTokenBean.getAccToken();  
 newToken = accToken;  
 *logger*.debug("input.token="+ newToken);  
 String accAddress = accTokenBean.getAccAddress();  
 SysToken sysToken = (SysToken) Springkit.*getRequest*().getSession().getAttribute(BizConstants.*SYS\_TOKEN*);  
 if (sysToken != null) {//合法  
 *logger*.debug("input.accAddress="+ accAddress);  
 *logger*.debug("token.accAddress="+ sysToken.getAddress());  
 Date sysTime = sysToken.getTokenTime();  
 //过期刷新accToken  
 Date curTime = new Date();  
 int loseTime = Integer.*valueOf*(MemoryCache.*getSysConfigKey*("ACCTOKEN\_LOSE\_TIME"));  
 boolean timelose = (curTime.getTime() - loseTime - sysTime.getTime()) > 0;  
 *logger*.debug("token.timelose="+timelose);  
 if (timelose) {  
 *logger*.debug("token过期");  
 throw new AuthorErrorException();  
 }  
 } else {  
 *logger*.debug("token不合法!");  
 throw new AuthorErrorException();  
 }  
 return newToken;  
 }

第二步:配置拦截器(com.roroclaw.base.handler.interceptor.AuthInterceptor4All)

*<!—用户认证拦截器-->*<mvc:interceptor>  
 <mvc:mapping path="/\*\*"/>  
 <bean class="com.roroclaw.base.handler.interceptor.AuthInterceptor4All">  
 <property name="baseAuthService" ref="authService"/>  
 <property name="filters">  
 <list>  
 <value>login</value>  
 <value>doLogin</value>  
 <value>opener</value>  
 <value>404</value>  
 <value>error</value>  
 <value>wopi</value>  
 </list>  
 </property>  
 </bean>  
</mvc:interceptor>

1. **统一Restful结构封装**

封装了统一的接口返回:

**配置方式:**

第一步,配置AOP:

*<!--api检查-->*<bean id="infcContollerAOP" class="com.roroclaw.base.handler.aop.InfcContollerAOP"></bean>

第二步,配置切面:

<aop:aspect ref="infcContollerAOP">  
 <aop:around method="around"  
 pointcut="execution (\* com.biz.controllers..\*(..))" />  
</aop:aspect>

切面controller的代码示例:

/\*\*  
 \* 备课本获取教材版本  
 \*  
 \* @return  
 \* @throws BizException  
 \*/  
@RequestMapping(value = "/testinfc.infc")  
@ResponseBody  
public Object testinfc() throws Exception {  
 boolean bol = true;  
 return bol;

}

Controller方法返回值可以为任意的object，框架可自动转换为json格式：

返回正常格式为:

{

"status": "1",

"describe": "success",

"object": 实际数据对象(controller中的返回值)

}

异常时返回json格式:

{

"status": 异常编码（系统定义 0.失败 1.成功 2,用户非法,认证失败）,

"describe": 异常描述,

"object": null

}

1. **统一异常处理**

配置com.roroclaw.base.handler.ExceptionHandler,示例如下:

*<!-- 全局异常配置 start -->*<bean id="exceptionResolver" class="com.roroclaw.base.handler.ExceptionHandler">  
 <property name="loginUrl" value="user/redirectLogin.do"/>  
</bean>

loginUrl属性:捕捉到认证异常（com.roroclaw.base.handler.AuthorErrorException）,默认跳转位置

异常处理分为两类:根据请求的request头的type属性区分,

1.默认判定为页面跳转请求,将执行重定向跳转 loginurl地址

2.如果为application/json,将返回json异常统一格式.

1. **常量缓存配置:**

**再spring配置文件中初始化缓存bean（com.roroclaw.base.bean.MemoryCache）,示例如下:**

*<!-- 内存初始化 -->*<bean id="projectCache" class="com.roroclaw.base.bean.MemoryCache">  
 *<!—这里配置读取工程目录下的sysconfig.properties -->* <property name="sysConfigLocation" value="classpath:sysconfig.properties" />  
</bean>

**代码中使用常量方式:**

这里取sysconfig.properties文件中key值为fileUpload.dir的值.代码如下:

MemoryCache.getSysConfigKey("fileUpload.dir");