ssm练习第五天

第一章：用户模块

第一节：获取用户名

1. 服务器端后台对象封装的分析
    1. 对象的封装，认证通过后会返回User对象，该对象中包含用户名等信息。
    2. principal（主角）就是User对象。
    3. 框架会把principal封装到Authentication（认证）对象中
    4. Authentication对象会封装到SecurityContext（Security上下文对象）中
    5. 最后会把SecurityContext绑定到当前的线程中
2. 服务器可以编写如下代码

```
        // 先获取到SecurityContext对象
        SecurityContext context = SecurityContextHolder.getContext();
        // 获取到认证的对象
        Authentication authentication = context.getAuthentication();
        // 获取到登录的用户信息
        User user = (User) authentication.getPrincipal();
        System.out.println(user.getUsername());
```

3. 会把SecurityContext对象存入到HttpSession对象中
4. 在JSP的页面上可以使用2种方式来获取用户名
    1. 使用EL表达式方式获取

```
${ sessionScope.SPRING_SECURITY_CONTEXT.authentication.principal.username }
```

    2. 使用security框架提供方式获取

```
<%@ taglib prefix="security" uri="http://www.springframework.org/security/tags" %>
<security:authentication property="principal.username"/>
```

第二节：给用户分配角色功能

1. 需求：给用户分配角色的功能
2. 跳转到用户分配角色页面的功能
    1. 服务器端的代码

```
    UserController的代码
    /**
     * 跳转到添加角色的页面
     * @param id
     * @return
```

```
     */
    @RequestMapping("/initAddRole")
    public ModelAndView initAddRole(Long id) {
        SysUser user = userService.findOne(id);

        // 用户所拥有的角色查询出来
        List<Role> userRoles = user.getRoles();
        StringBuffer sb = new StringBuffer();
        // 拼接角色名称的字符串
        for (Role role : userRoles) {
            sb.append(role.getRoleName());
            sb.append(",");
        }
        String roleStr = sb.toString();
        // 查询所有的角色
        List<Role> rolelist = roleService.findAll();

        ModelAndView mv = new ModelAndView();
        mv.addObject("user", user);
        mv.addObject("roleStr", roleStr);
        mv.addObject("rolelist", rolelist);
        // 再查询出角色集合
        mv.setViewName("user-role-add");
        return mv;
    }
```

2. JSP页面的代码

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
<c:forEach items="${rolelist}" var="role">
<tr>
<td>
<input name="ids" type="checkbox" value="${role.id}" <c:if
test="${fn:contains(roleStr,role.roleName)}">checked</c:if> >
</td>
<td>${role.id}</td>
<td>${role.roleName }</td>
<td>${role.roleDesc}</td>

</tr>
</c:forEach>
</tbody>
```

3. 给用户分配角色的功能

1. 服务器端的代码

```
    Controller的代码
    /**
     * 给用户设置角色功能
     * @return
     * @throws Exception
```

```
     */
    @RequestMapping("/addRoleToUser")
    public String addRoleToUser(Long userId,Long [] ids) throws Exception {
        // 给用户分配角色
        userService.addRoleToUser(userId,ids);
        return "redirect:/user/findAll";
    }


    UserServiceImpl的代码
    /**
     * 给用户分配角色功能
     */
    public void addRoleToUser(Long userId, Long[] ids) throws Exception {
        // 先从中间表删除数据
        userDao.delete(userId);
        int a = 10/0;
        // 再添加数据，遍历添加数据
        if(ids != null) {
            for (Long roleId : ids) {
                // 添加数据
                userDao.addRoleToUser(userId,roleId);
            }
        }
    }

    UserDao接口的代码
    /**
     * 从中间表删除和用户有关的角色信息
     * @param userId
     */
    @Delete("delete from sys_user_role where userId = #{userId}")
    void delete(Long userId);

    /**
     * 给用户分配角色
     * @param userId
     * @param roleId
     */
    @Insert("insert into sys_user_role (userId,roleId) values (#{userId},#{roleId})")
    void addRoleToUser(@Param("userId") Long userId, @Param("roleId") Long roleId);
```

第二章：授权功能

第一节：在JSP页面控制菜单权限

1. 在JSP页面中使用security:authorize标签，可以控制菜单是否显示。
    1. security:authorize标签的access="hasAnyRole('ROLE_USER','ROLE_ADMIN')"
2. 强调：因为需要编写表达式，那么需要把表达式设置成true

```
<security:http auto-config="true" use-expressions="true">
        <!-- 配置拦截的请求地址，任何请求地址都必须有ROLE_USER的权限 -->
        <security:intercept-url pattern="/**"
access="hasAnyRole('ROLE_USER','ROLE_ADMIN')"/>
        <!-- 配置具体的页面跳转 -->
        <security:form-login
            login-page="/login.jsp"
            login-processing-url="/login"
            default-target-url="/index.jsp"
            authentication-failure-url="/failer.jsp"
        />

        <!-- 关闭跨越请求 -->
        <security:csrf disabled="true"/>

        <security:access-denied-handler error-page="/403.jsp"/>

        <!-- 退出 -->
        <security:logout invalidate-session="true" logout-url="/logout" logout-success-
url="/login.jsp"/>

    </security:http>
```

3. 在aside.jsp中添加标签进行菜单的控制

```
<%@ taglib prefix="security" uri="http://www.springframework.org/security/tags" %>
<security:authorize access="hasAnyRole('ROLE_USER','ROLE_ADMIN')">
  <li id="system-setting">
  <a href="${pageContext.request.contextPath}/product/findByPageHelper">
  <i class="fa fa-circle-o"></i> 产品管理
  </a>
  </li>
</security:authorize>

<security:authorize access="hasAnyRole('ROLE_USER','ROLE_ADMIN')">
  <li id="system-setting"><a
  href="${pageContext.request.contextPath}/order/findAll">
  <i class="fa fa-circle-o"></i> 订单管理
  </a>
  </li>
</security:authorize>
```

第二节：在服务器端控制权限

1. 演示问题：菜单没有显示，如果直接访问地址栏，那么也会进入到具体的方法中
2. 在讲各种注解之前，一定一定需要先配置AOP注解的支持，而且一定需要在springmvc.xml配置文件中配置

```
<!-- 开启AOP的支持 -->
<aop:aspectj-autoproxy proxy-target-class="true"/>
```

3. JSR-250注解方式权限拦截

1. 在pom.xml文件中引入坐标

```
    <dependency>
            <groupId>javax.annotation</groupId>
            <artifactId>jsr250-api</artifactId>
            <version>1.0</version>
        </dependency>
```

2. 在spring-security.xml配置文件中开启JSR-250的注解支持

```
<security:global-method-security jsr250-annotations="enabled"/>
```

3. 在Controller的类或者方法上添加注解

```
@RolesAllowed("ROLE_ADMIN")
public class RoleController {
```

4. 配置没有权限的提示页面

```
<security:access-denied-handler error-page="/403.jsp"/>
```

4. security注解方式权限拦截
    1. 在spring-security.xml配置文件中开启注解支持

```
<security:global-method-security secured-annotations="enabled"/>
```

    2. 在Controller类或者方法添加注解

```
@Secured("ROLE_ADMIN")
public class RoleController
```

5. Spring表达式的方式
    1. 在spring-security.xml配置文件中开启注解支持

```
<security:global-method-security pre-post-annotations="enabled"/>
```

    2. 在Controller类或者方法添加注解

```
@PreAuthorize("hasAuthority('ROLE_ADMIN')")
public class RoleController {
```

第三章：系统日志功能

第一节：搭建日志模块开发环境

1.sql语句

```
CREATE TABLE sys_log(
    id number PRIMARY KEY ,
    visitTime TIMESTAMP,
    username VARCHAR2 (50),
    ip VARCHAR2(30),
    method VARCHAR2(200)
)
```

2.实体类

```
public class SysLog {

    private Long id;
    private Date visitTime;
    private String visitTimeStr;
    private String username;
    private String ip;
    private String method;
  }
```

第二节：编写切面类的代码

1. 非常非常重要的步骤，开启AOP的注解，一定在springmvc.xml配置文件中开启

   ```
   <aop:aspectj-autoproxy proxy-target-class="true"/>
   ```

2. 配置监听request对象的监听器

```
   <!-- 配置监听器，监听request域对象的创建和销毁的 -->
    <listener>
        <listener-
class>org.springframework.web.context.request.RequestContextListener</listener-class>
    </listener>
```

3. 切面类的具体的代码如下

   ```
   package cn.itcast.aop;

   import java.util.Date;

   import javax.servlet.http.HttpServletRequest;

   import org.aspectj.lang.JoinPoint;
   import org.aspectj.lang.annotation.After;
   import org.aspectj.lang.annotation.Aspect;
   import org.aspectj.lang.annotation.Before;
   import org.springframework.beans.factory.annotation.Autowired;

   import org.springframework.security.core.Authentication;
   ```

```java
import org.springframework.security.core.context.SecurityContext;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.User;
import org.springframework.stereotype.Component;

import cn.itcast.domain.SysLog;
import cn.itcast.service.SysLogService;

/**
 * 日志的切面类
 * @author Administrator
 */
@Component
@Aspect
public class LogAop {

    // 当前正在访问的方法名称
    private String methodName;

    @Autowired
    private HttpServletRequest request;

    @Autowired
    private SysLogService sysLogService;

    private Class clazz;

    /**
     * 方法执行前增强
     */
    @Before("execution(* cn.itcast.controller.*.*(..))")
    public void logBefore(JoinPoint jp) throws Exception {
        // 获取到目标对象
        clazz = jp.getTarget().getClass();
        // 获取类名
        String className = clazz.getSimpleName();
        // 获取到正在执行的方法名称
        methodName = jp.getSignature().getName();
        methodName = className+"."+methodName;
        System.out.println(jp.getSignature().toShortString());
        System.out.println(jp.getSignature().toShortString());

    }

    /**
     * 方法执行后增强
     */
    @After("execution(* cn.itcast.controller.*.*(..))")
    public void logAfter() {
        SysLog log = new SysLog();
        // 获取ip
        log.setIp(request.getRemoteAddr());

        log.setMethod(methodName);
```

```
        SecurityContext context = SecurityContextHolder.getContext();
        // 获取到认证的对象
        Authentication authentication = context.getAuthentication();
        // 获取到登录的用户信息
        User user = (User) authentication.getPrincipal();
        log.setUsername(user.getUsername());
        log.setVisitTime(new Date());
        // 保存日志
        sysLogService.save(log);
    }

}
```

4. service和dao的代码如下

```
package cn.itcast.service.impl;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import cn.itcast.dao.SysLogDao;
import cn.itcast.domain.SysLog;
import cn.itcast.service.SysLogService;

@Service
public class SysLogServiceImpl implements SysLogService {

    @Autowired
    private SysLogDao sysLogDao;

    /**
     * 保存
     */
    public void save(SysLog log) {
        sysLogDao.save(log);
    }

}

 package cn.itcast.dao;
   import org.apache.ibatis.annotations.Insert;
   import org.springframework.stereotype.Repository;
   import cn.itcast.domain.SysLog;

   @Repository
   public interface SysLogDao {

    @Insert("insert into sys_log(visitTime,username,ip,method) values(#{visitTime},#
{username},#{ip},#{method})")
    void save(SysLog log);
   }
```