第3天 权限控制

今日目标

- 理解权限角色模型
- 掌握文件权限控制
- 掌握注解权限控制
- 掌握精选题目审核

1. 权限控制概述

1.1 权限控制定义

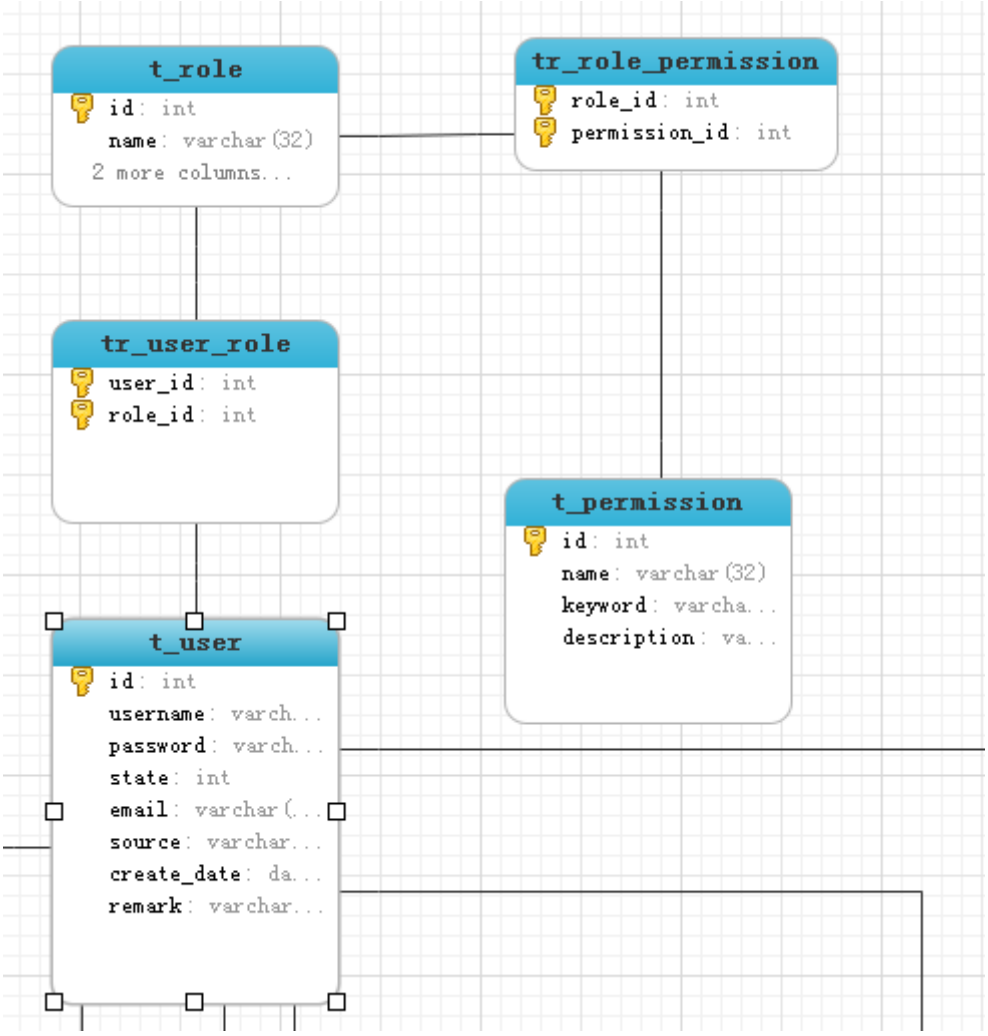
当前系统要求不同的用户针对不同的菜单及操作应该有不同的权限。比如管理员可以操作精选题库列表及相应的审核操作，如果是普通录入人员，只能操作基础题目菜单及题库录入的接口。所以精选题库及题目审核必须具有该权限的用户才可以操作，如果登录用户没有这个权限，系统应提示用户。

权限具体是什么，权限在系统中就是一个字符串标识，标识系统的每个功能动作，比如题目新增（QUESTION_ADD）、题目删除、题目修改、题目列表获取(QUESTION_LIST)，题目审核等。每个用户的权限，是通过角色来汇集，一个角色可以包含多个权限，一个用户可以有多个角色，比如定义了管理员角色（ROLE_ADMIN）或题目录入角色（ROLE_QUESTION_RECORDER）。这套基于权限角色的控制系统，实际是权限控制模型RBAC模型。

1.2 权限控制模型

基于角色的权限访问控制（Role-Based Access Control），在RBAC中权限与角色相关联，用户通过成为适当角色的成员而得到这些角色的权限。这就极大地简化了权限的管理。在一个组织中，角色是为了完成各种工作而创造，用户则依据它的责任和资格来被指派相应的角色，用户可以很容易地从一个角色被指派到另一个角色。角色可依新的需求和系统的合并而赋予新的权限，而权限也可根据需要而从某角色中回收。角色与角色的关系可以建立起来以囊括更广泛的客观情况。

实现最终的权限控制，需要有一套表结构支撑，最基本的表包含用户表t_user、权限表t_permission、角色表t_role、用户角色关系表t_user_role、角色权限关系表t_role_permission五张表。有的系统还需要菜单及菜单关系表，当前系统的菜单相对稳定，故暂时不需要动态实现。



1.3 权限控制分析

- 认证授权

权限控制的基本实现思路是先通过用户名和密码对用户进行认证，认证就是判断当前用户是否是系统用户，然后再根据用户ID或用户名获取权限信息即授权信息。

授权信息是用户角色及权限标识的集合，把用户的角色、权限标识集合与用户关联，最终放入当前会话中。

- 权限控制

刚才的步骤是获取用户的权限，那权限控制是什么？

是需要提前配置哪些资源，比如//pages/questionClassicList.html可以让哪些角色访问，配置哪些资源，比如/question/findClassicListByPage可以让哪些权限访问。

- 认证流程

1. 当前端发请求到后端服务器时，服务器拦截客户请求
2. 分析当前的请求是否有权限限制，所谓权限限制就是看是否被配置了需要哪些角色或哪些权限可以访问，
    1. 如果没有配置，直接放行；
    2. 如果配置了，就需要查询当前用户的权限标识列表与当前配置的权限标识是否一致，如果一致可以放行，如果不一致提示权限不足。

1.4 实现思路

通过权限控制分析，需要完成如下动作：

1. 配置需要权限控制的资源

   资源配置有两种方式，一种是在文件中配置，另外一种是在类的方法上通过注解来配置。

   - 文件配置

     文件配置可采用xml或properties方式，通常稍微复杂的配置会采用xml格式，通常xml配置文件，会先定义数据规范，比如定义dtd或schema，然后根据规范书写正式的xml文件，简单的可以直接写xml文件，只要符合xml规范即可。

     文件配置的内容就同使用简单xml标签定义哪些资源需要哪些权限，哪些资源需要哪些角色。比如如下：

     ```
     <security pattern="/pages/questionBasicList.html"
               has_role="ROLE_ADMIN,ROLE_QUESTION_RECORDER"/>
     <security pattern="/pages/questionClassicList.html"
               has_role="ROLE_ADMIN"/>
     ```

     以上配置表示，当访问/pages/questionBasicList.html时，需要当前用户具有有ROLE_ADMIN或ROLE_QUESTION_RECORDER的角色。

   - 注解配置

     注解配置主要是通过自定义注解来配置类中哪些方法需要哪些权限或哪些角色来访问，比如如下：

     ```
     @HmAuthority("QUESTION_LIST")
     @HmRequestMapping("/question/findClassicListByPage")
     public void questionClassicList (HttpServletRequest
     request,HttpServletResponse response) throws ServletException,
     IOException {
        log.debug("questionList pageBean:{}",pageBean);
     }
     ```

     以上通过@HmAuthority（自定义注解）来表示，当访问questionClassicList方法时（客户端是通过HmRequestMapping定义的 “/question/findClassicListByPage” 访问路径来访问此方法）当前用户需要具有“QUESTION_LIST”这个权限标识。

2. 加载并解析需要权限控制的资源

   无论是文件配置还是类注解配置，目的都是为表示哪过访问资源需要哪些角色或权限可以访问，所以需要提前加载并解析出这种映射关系，并保存到到应用上下文(内存)中。

   文件配置将采用sax/dom4j的方式进行xml解析。

   类注解配置将通过反射机制获取对象上的注解配置信息。

3. 拦截请求资源，匹配权限控制

   Web的拦截机制将采用过滤器技术来完成，即在当前Web应用中配置一个安全过滤器，可以在过滤器init()方法中加载资源配置（文件配置、注解配置），在doFilter()方法中完成请求拦截与权限验证。

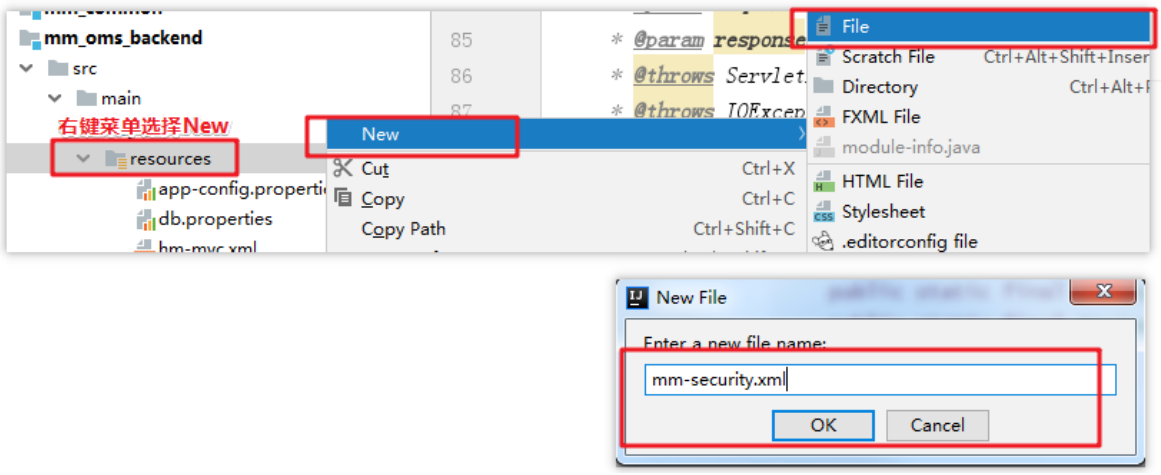
   权限资源配置有文件配置和注解配置两种方式，接下来将分别来实现这两种方式。

2.文件配置权限控制

2.1 定义权限配置文件

在resources下创建一个xml文件，名称自定义，在这里起名为mm-security.xml，注解扫描包后续章节会用，暂时加上。

创建过程，如图所示：



代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>
    <!--授权列表-->
    <security pattern="/pages/index.html"
has_role="ROLE_ADMIN,ROLE_QUESTION_RECORDER"/>
    <security pattern="/pages/questionBasicList.html"
has_role="ROLE_ADMIN,ROLE_QUESTION_RECORDER"/>
    <security pattern="/pages/questionClassicList.html" has_role="ROLE_ADMIN"/>
    <security pattern="/pages/userList.html" has_role="ROLE_ADMIN"/>

    <!--注解扫描包-->
    <scan package="com.itheima.mm.controller" />
</beans>
```

2.2 定义权限常量类

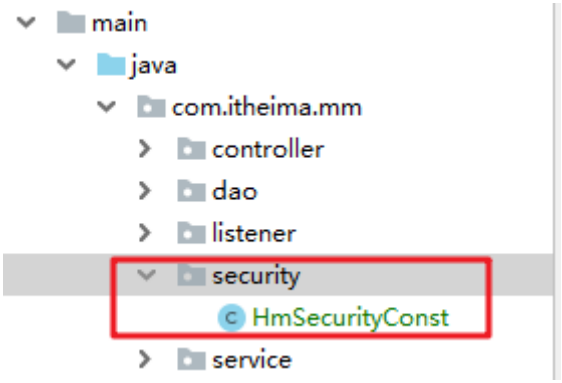
在权限资源定义、解析、匹配过程中，会经常使用一些共同的标识，故需要定义一个常量类。

在com.itheima.mm包中，创建一个security包，后续与权限有关的类都放在此包中。

在security包中，创建一个HmSecurityConst的类，代码如下：

```
/**
 * @author : seanyang
 * @date : Created in 2019/8/12
 * @description : 权限常量类
 * @version: 1.0
 */
public class HmSecurityConst {
    public static final String TAG_SECURITY = "security";
    public static final String TAG_SECURITY_ATTR_PATTERN = "pattern";
    public static final String TAG_SECURITY_ATTR_HAS_ROLE = "has_role";
    public static final String CONFIG_SECURITYCONFIGLOCATION =
"SecurityConfigLocation";
    public static final String TAG_SCAN = "scan";
    public static final String TAG_SCAN__PACKAGE = "package";
}
```

如图所示：



2.3 定义权限过滤器

- 定义过滤器

在security包中，创建一个HmSecurityFilter的类，该类通过监听器配置参数获取权限配置文件名称，然后在init方法中读取该参数，并调用解析匹配方法，当前先构建代码结构，类中定义Map类型成员属性accessPathAuthMaps，该成员属性存储解析过的资源配置信息，这个作为权限验证容器。其中key是资源路径，value是角色或权限标识。

代码如下：f

```
/**
 * @author : seanyang
 * @date : Created in 2019/8/12
 * @description :
 * @version: 1.0
 */
@Slf4j
public class HmSecurityFilter implements Filter {
    // 存储访问路径映射权限列表
    private Map<String, String> accessPathAuthMaps = new HashMap<>();
    // 注解解析扫描类根包
    private String basePackage = "";
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        // 初始化配置
        log.info("security init....");
        // 读取配置文件的参数，调用配置文件解析
        String filePath =
filterConfig.getInitParameter(CONFIG_SECURITYCONFIGLOCATION);
        parseConfig(filePath);
        // 读取类中的方法注解权限配置
        log.info("basePackage:{},accessPathAuthMaps:
{}",basePackage,accessPathAuthMaps);
    }
    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain) throws IOException, ServletException {
        // 权限验证
        log.info("security doFilter....");
        filterChain.doFilter(servletRequest,servletResponse);

    }
    @Override
```

```
    public void destroy() {
        log.info("security destroy....");
    }
    /**
     * 读取解析配置文件
     * @param filePath
     */
    public void parseConfig(String filePath){
        log.info("security parseConfig....{}",filePath);
    }
}
```

- 注册过滤器

  打开web.xml文件，追加如下配置，其中参数SecurityConfigLocation是固定的，该名称在常量类中已定义，mm-security.xml文件建议放在resources目录下，配置信息如下：

```
  <!--编码过滤器-->
  <filter>
    <filter-name>CharacterEncodingFilter</filter-name>
    <filter-class>com.itheima.framework.mvc.CharacterEncodingFilter</filter-
class>
  </filter>
  <!--配置权限过滤器，配置文件默认从resources目录读取-->
  <filter>
    <filter-name>MmSecurityFilter</filter-name>
    <filter-class>com.itheima.mm.security.HmSecurityFilter</filter-class>
    <init-param>
      <param-name>SecurityConfigLocation</param-name>
      <param-value>mm-security.xml</param-value>
    </init-param>
  </filter>
  <filter-mapping>
    <filter-name>CharacterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
  <filter-mapping>
    <filter-name>MmSecurityFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
```

  切记MmSecurityFilter，一定要在CharacterEncodingFilter字符集过滤之后，否则返回的数据会有乱码。

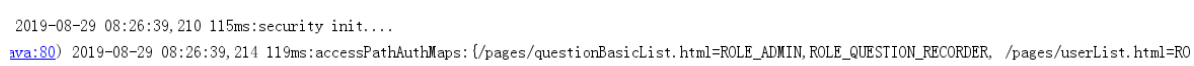
2.4 解析权限配置

解析xml,需要使用dom4j库，工程已导入该依赖库，具体实现如下：

```
/**
 * 读取解析配置文件
 * @param filePath
 */
public void parseConfig(String filePath){
   //加载配置文件(约定配置文件存放在类加载路径下)
   if(!filePath.startsWith("/")){
      filePath = "/".concat(filePath);
   }
```

```
    InputStream resourceAsStream =
 HmSecurityFilter.class.getResourceAsStream(filePath);
    try{
        // 解析XML，获取资源访问权限路径
        SAXReader reader = new SAXReader();
        Document document = reader.read(resourceAsStream);
        // 获取security节点，通过XPath表达式语法获取元素
        List<Node> nodeList =
document.selectNodes("//"+HmSecurityConst.TAG_SECURITY);
        // 读取当前标签pattern has_role 属性
        for (Node node:nodeList){
            Element element = (Element)node;
            String accessPath =
element.attribute(HmSecurityConst.TAG_SECURITY_ATTR_PATTERN).getStringValue();
            String accessRole =
element.attribute(HmSecurityConst.TAG_SECURITY_ATTR_HAS_ROLE).getStringValue();
            accessPathAuthMaps.put(accessPath,accessRole);
        }
        // 读取scan节点
        Node nodeScan = document.selectSingleNode("//"+HmSecurityConst.TAG_SCAN);
        if(nodeScan!=null){
            Element element = (Element)nodeScan;
            basePackage =
element.attribute(HmSecurityConst.TAG_SCAN__PACKAGE).getStringValue();
        }
        log.debug("basePackage:{}","accessPathAuthMaps:
{}",basePackage,accessPathAuthMaps);
    }catch(Exception e){
        e.printStackTrace();
    }
}
```

以上完成后，可先编译运行，查看控制器台是否打印读取的配置信息，如图如下：



2.5 验证权限控制

验证权限的过程如下：

1. 获取客户端访问的URI，比如/mm/pages/xxx.html
2. 获取当前Web上下文路径，比如/mm
3. 把客户端访问路径中的上下路径去掉，比如/pages/xxx.html
4. 把访问路径中有.do结尾的，去掉.do后缀
5. 匹配当前访问路径是否在权限容器中
    1. 不在容器，直接放行
    2. 在容器，获取这个路径配置的授权信息(权限、角色)
6. 获取当前Session会话信息
    1. Session会话信息为null，直接重定向到登录页面
    2. Session会话信息不为null，获取当前用户的权限配置信息
7. 匹配当前用户授权信息是否包含在当前资源的授权列表中
    1. 如果在，直接放行
    2. 如果不在，提示用户权限不足的信息

通过以上验证权限流程，需要当前用户在登录时从数据库读取角色权限信息并加载到内存。

2.5.1 新增角色权限Dao及映射文件

- RoleDao接口及映射文件

```
/**
 * @author ： seanyang
 * @date ： Created in 2019/8/12
 * @description ： 角色Dao
 * @version: 1.0
 */
public interface RoleDao {
    /**
     * 根据用户ID，获取角色列表
     * @param userId
     * @return
     */
    Set<Role> selectRoleByUserId(Integer userId);
}
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.itheima.mm.dao.RoleDao">
    <select id="selectRoleByUserId" resultType="com.itheima.mm.pojo.Role">
        SELECT * FROM t_role
        WHERE id in (SELECT role_id FROM tr_user_role WHERE user_id = #
{userId})
    </select>
</mapper>
```

- PermissionDao接口及映射文件

```
/**
 * @author ： seanyang
 * @date ： Created in 2019/8/12
 * @description ： 权限Dao
 * @version: 1.0
 */
public interface PermissionDao {
    /**
     * 根据角色ID，获取权限列表
     * @param roleId
     * @return
     */
    Set<Permission> selectPermissionByRoleId(Integer roleId);
}
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.itheima.mm.dao.PermissionDao">
    <select id="selectPermissionByRoleId"
resultType="com.itheima.mm.pojo.Permission">
        SELECT * FROM t_permission
        WHERE id IN (SELECT permission_id FROM tr_role_permission WHERE
role_id = #{roleId})
    </select>
</mapper>
```

2.5.2 更新UserService接口及实现类

增加一个基于用户ID，获取授权信息(角色标识、权限标识)列表的方法

```
/**
 * 根据用户Id,获取用户授权列表
 * @param userId
 * @return
 */
List<String> findAuthorityByUserId(Integer userId);
```

```
@Override
public List<String> findAuthorityByUserId(Integer userId) {
    List<String> authorityList = new ArrayList<>();
    // 获取角色列表
    SqlSession sqlSession = getSession();
    RoleDao roleDao = getDao(sqlSession,RoleDao.class);
    Set<Role> roles = roleDao.selectRoleByUserId(userId);
    // 获取权限列表
    PermissionDao permissionDao = getDao(sqlSession,PermissionDao.class);
    // 把角色、权限关键封装到authorityList
    for (Role role:roles){
        authorityList.add(role.getKeyword());
        Set<Permission> permissions =
permissionDao.selectPermissionByRoleId(role.getId());
        for (Permission permission:permissions){
            authorityList.add(permission.getKeyword());
        }
    }
    sqlSession.close();
    return authorityList;
}
```

2.5.3 更新UserController及Service调用

在UserController中找到login方法，在用户登录成功后，调用业务方法setAuthorityList获取用户授权列表，并设置给当前用户。

代码如下：

```
@HmRequestMapping("/user/login")
public void login(HttpServletRequest request, HttpServletResponse response)
throws ServerException, IOException {
```

```
    log.debug("UserController save...");
    User loginForm = parseJSON2Object(request, User.class);
    log.debug("loginForm:{}",loginForm);
    User user = userService.findByUsername(loginForm.getUsername());
    if(user == null){
        printResult(response,new Result(false,"用户名不正确"));
        return;
    }
    if (user.getPassword().equals(loginForm.getPassword())){
        // 通过业务方法获取当前用户权限列表，并放入当前用户对象
        user.setAuthorityList(userService.findAuthorityByUserId(user.getId()));
        log.debug("user:{}",user);
        // 把用户对象放入session中
        request.getSession(true).setAttribute(SESSION_KEY_USER,user);
        printResult(response,new Result(true,"登录成功"));
    }else{
        printResult(response,new Result(false,"密码错误"));
    }
}
```

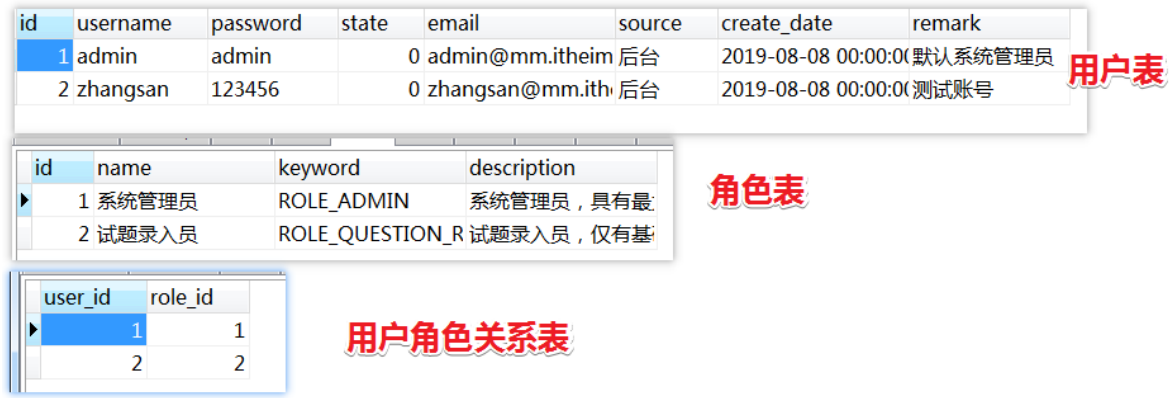
2.5.4 更新HmSecurityFilter，验证授权

```
@Override
public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain) throws IOException, ServletException {
    log.debug("自定义权限过滤器 doFilter ");
    HttpServletRequest request = (HttpServletRequest)servletRequest;
    HttpServletResponse response = (HttpServletResponse)servletResponse;
    String requestURI = request.getRequestURI();
    String contextPath = request.getContextPath();
    String accessPath = requestURI.substring(contextPath.length());
    // 根据访问路径，判断是否在权限容器
    String authString = accessPathAuthMaps.get(accessPath);
    log.debug("获取访问URI:{},contextPath:{},accessPath:{},authString:
{}",requestURI,contextPath,accessPath,authString);
    if(authString == null){
        // 不在权限容器
        log.debug("不在权限容器,继续执行");
        filterChain.doFilter(servletRequest,servletResponse);
        return;
    }
    log.debug("在权限容器，继续判断是否用户有权限访问");
    HttpSession session = request.getSession(false);
    if(session == null){
        log.debug("在权限容器，当前会话已失效，需要重新登录");
        response.sendRedirect("http://localhost:8080/mm/login.html");
        return;
    }
    User user = (User) session.getAttribute(GlobalConst.SESSION_KEY_USER);
    log.debug("在权限容器，会话有效，用户存在，匹配资源");
    // 获取用户的权限列表
    List<String> userAuthorityList = user.getAuthorityList();
    log.debug("userAuthorityList:{} authString:
{}",userAuthorityList,authString);
    String [] authStringArray = authString.split(",");
    boolean isAuth =false;
    for (String auth:authStringArray){
```

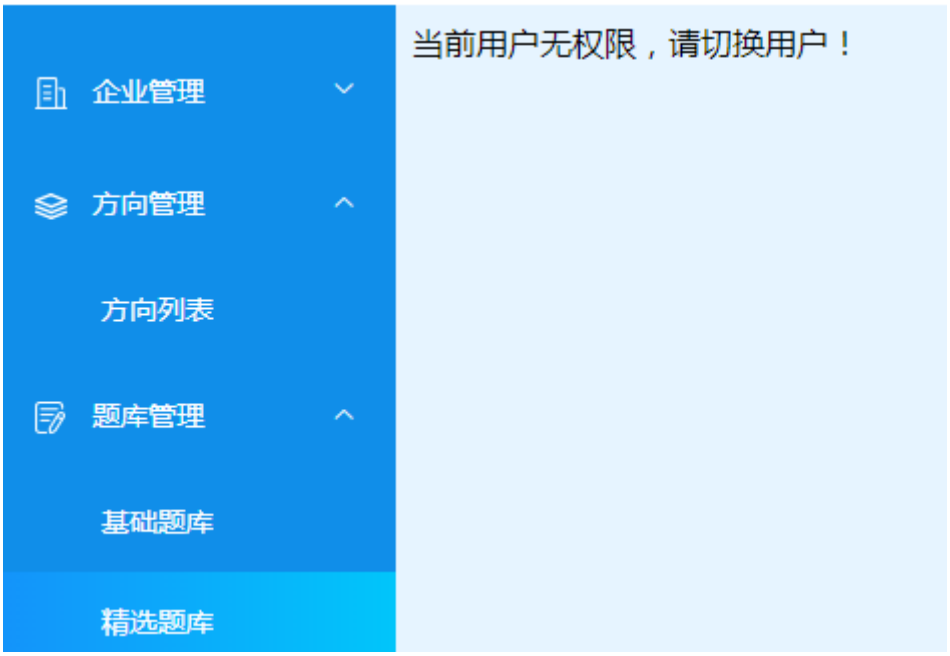
```
            if(userAuthorityList.contains(auth)){
                isAuth = true;
                break;
            }
        }
        if (isAuth){
            log.debug("在权限容器，会话有效，用户权限可以匹配资源，放行");
            filterChain.doFilter(servletRequest,servletResponse);
        }else{
            log.debug("在权限容器，会话有效，用户权限不可以匹配资源，权限不足");
            response.getWriter().print("当前用户权限不足，请切换用户！");
        }
    }
```

2.5.5 测试权限效果

当前数据库t_user表，初始化了admin、zhangsan两个用户，查询角色信息如下：



zhangsan角色是试题录入员，admin是系统管理员，在管理后台，采用不同的用户登录系统，其中zhangsan登录后，如果访问精选题目列表，应该如下提示：



3.注解配置权限控制

文件权限配置，可以实现全部资源的授权配置，为了更加灵活方便，也可以采用方法注解的方式实现。注解配置需要先做注解定义，然后解析注解，然后在需要配置的资源使用注解。

3.1 定义方法注解

在security包中，创建注解权限类HmAuthority

```
/**
 * @author : seanyang
 * @date : Created in 2019/8/12
 * @description : 方法注解，声明方法具体权限内容
 * @version: 1.0
 */
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface HmAuthority {
   String value();
}
```

@Target：

表示该注解可以用于什么地方，可能的ElementType参数有：`CONSTRUCTOR`：构造器的声明`FIELD`：域声明（包括`enum`实例）`LOCAL_VARIABLE`：局部变量声明`METHOD`：方法声明`PACKAGE`：包声明`PARAMETER`：参数声明`TYPE`：类、接口（包括注解类型）或`enum`声明

@Retention

表示需要在什么级别保存该注解信息。可选的RetentionPolicy参数包括：`SOURCE`：注解将被编译器丢弃`CLASS`：注解在class文件中可用，但会被VM丢弃`RUNTIME`：VM将在运行期间保留注解，因此可以通过反射机制读取注解的信息

3.2 解析方法注解

修订HmSecurityFilter类，在init方法时加载注解解析，以下先实现注解解析：

```
/**
 * 从注解实例中，获取注解的权限配置
 */
public void parseAnnotation(){
   List<Class<?>> classsFromPackage =
HmClassScanner.getClasssFromPackage(basePackage);
   //遍历类，检查是否使用的@Component注解
   if(null == classsFromPackage || classsFromPackage.size() == 0){
      log.debug("parseBeans... basePackage:{} is null.",basePackage);
      return;
   }
   for (Class<?> aClass : classsFromPackage) {
      Method[] methods = aClass.getMethods();
      for (Method method:methods){
         // 读取bean的HmRequestMapping注解
         // 读取bean的MmAuthority注解
         if(method.isAnnotationPresent(HmAuthority.class)){
            String accessPath =
method.getAnnotation(HmRequestMapping.class).value();
            String authority = method.getAnnotation(HmAuthority.class).value();
            accessPathAuthMaps.put(accessPath,authority);
```

```
            }
        }
    }
    log.debug("accessPathAuthMaps:{}",accessPathAuthMaps);
}
```

3.3 在init方法中解析方法注解

```
@Override
public void init(FilterConfig filterConfig) throws ServletException {
    // 初始化配置
    log.debug("security init....");
    // 读取配置文件的参数，调用配置文件解析
    String filePath =
filterConfig.getInitParameter(CONFIG_SECURITYCONFIGLOCATION);
    parseConfig(filePath);
    // 读取类中的方法注解权限配置
    parseAnnotation();
    log.debug("basePackage:{},accessPathAuthMaps:
{}",basePackage,accessPathAuthMaps);
}
```

3.4 在Controller方法中使用注解

可以在任何需要权限的方法都可以加入权限注解，在这里选精选题目列表方法questionClassicList，在方法上加入注解，代码如下：

```
/**
 * 获取题目列表
 * @param request
 * @param response
 * @throws ServletException
 * @throws IOException
 */
@HmAuthority("QUESTION_LIST")
@HmRequestMapping("/question/findListByPage")
public void questionList (HttpServletRequest request,HttpServletResponse
response) throws ServletException, IOException {
    QueryPageBean pageBean = parseJSON2Object(request, QueryPageBean.class);
    if (pageBean == null){
        pageBean = new QueryPageBean();
        pageBean.setCurrentPage(1);
        pageBean.setPageSize(10);
    }
    log.debug("questionList pageBean:{}",pageBean);
    printResult(response,new Result(true,"获取题目列表成
功",questionService.findByPage(pageBean)));
}
```

3.5 测试权限效果

目前admin、zhangsan都有/question/findListByPage都有访问权限，可以测试非登录用户，直接访问API接口，比如在非登录或退出登录的情况下，直接在浏览器输入如下地址：

http://localhost:8080/mm/question/findListByPage.do

查看是否会跳转到登录页面。

4. 精选题目审核

在精选题目列表中，点审核按钮，进入审核预览界面，然后在最下面有审核通过与审核不通过两个操作按钮。要求有权限的用户才可以操作，根据上一节定义的权限框架，为审核操作添加权限注解。

审核通过与审核不通过，实际是为当前题目增加了一条审核日志。设计的表是t_review_log，在添加了审核记录之后，还需要更新主表中的审核状态，保持与审核记录状态的一致性。

4.1 更新接口及映射类

- 更新ReviewLogDao接口及映射类

```
/**
 * 添加审核记录
 * @param reviewLog
 */
void add(ReviewLog reviewLog);
```

```
<insert id="add">
    INSERT INTO t_review_log (comments, status, question_id, user_id,
create_date)
    VALUES (#{comments},#{status},#{questionId},#{userId},#{createDate})
</insert>
```

- 更新QuestionDao接口及映射类

```
/**
 * 更新题目审核状态
 * @param id 题目ID
 * @param reviewStatus 审核状态
 */
void updateReviewStatus(@Param("id") Integer id, @Param("reviewStatus") Integer
reviewStatus);

/**
 * 更新题目状态
 * @param id 题目ID
 * @param status 审核状态
 */
void updateStatus(@Param("id") Integer id, @Param("status") Integer status);
```

```
<update id="updateReviewStatus">
    UPDATE  t_question
    set review_status = #{reviewStatus}
    where id = #{id}
</update>
<update id="updateStatus">
    UPDATE t_question
    SET  status = #{status}
     where id = #{id}
</update>
```

4.2 新增ReviewLogService接口及实现类

```
/**
 * @author : seanyang
 * @date : Created in 2019/8/16
 * @description : 审核业务类接口
 * @version: 1.0
 */
public interface ReviewLogService {
   /**
    * 添加审核记录
    * @param reviewLog
    */
   void addReviewLog(ReviewLog reviewLog);
}
```

```
/**
 * @author : seanyang
 * @date : Created in 2019/8/16
 * @description :
 * @version: 1.0
 */
@HmComponent("reviewLogService")
@Slf4j
public class ReviewLogServiceImpl extends BaseService implements
ReviewLogService {
    @Override
    public void addReviewLog(ReviewLog reviewLog) {
        SqlSession sqlSession = getSession();
        log.info("reviewLog:{}",reviewLog);
        ReviewLogDao reviewLogDao = getDao(sqlSession,ReviewLogDao.class);
        QuestionDao questionDao = getDao(sqlSession,QuestionDao.class);
        // 添加 审核记录
        reviewLog.setCreateDate(DateUtils.parseDate2String(new Date()));
        reviewLogDao.add(reviewLog);
        // 更新 题目状态
        if(reviewLog.getStatus() ==
QuestionConst.ReviewStatus.REVIEWED.ordinal()){
            // 审核通过
            questionDao.updateStatus(reviewLog.getQuestionId(),
 QuestionConst.Status.PUBLISHED.ordinal());
        }else if (reviewLog.getStatus() ==
QuestionConst.ReviewStatus.REJECT_REVIEW.ordinal()
            || reviewLog.getStatus() ==
QuestionConst.ReviewStatus.PRE_REVIEW.ordinal() ){
            // 审核拒绝,改为待审核状态
            questionDao.updateStatus(reviewLog.getQuestionId(),
 QuestionConst.Status.PRE_PUBLISH.ordinal());
        }

questionDao.updateReviewStatus(reviewLog.getQuestionId(),reviewLog.getStatus());
        commitAndCloseSession(sqlSession);
    }
}
```

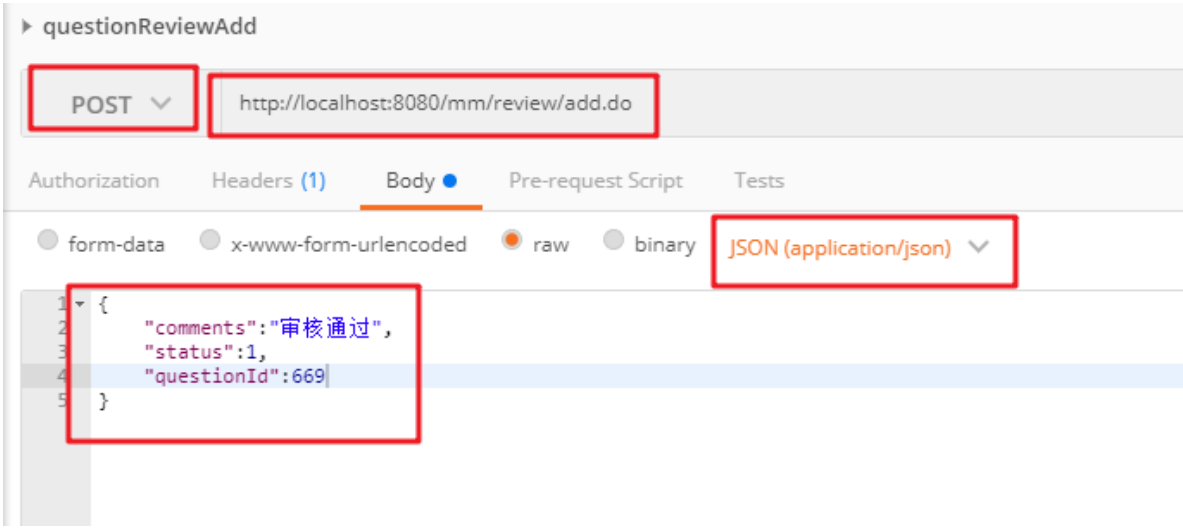
4.3 新增ReviewLogController类及Service调用

```
/**
 * @author : seanyang
 * @date : Created in 2019/8/16
 * @description : 题目审核控制器
 * @version: 1.0
 */
@HmComponent
@Slf4j
public class ReviewLogController extends BaseController {

    @HmSetter("reviewLogService")
    private ReviewLogService reviewLogService;

    /**
     * 增加审核
     * @param request
     * @param response
     * @throws ServletException
     * @throws IOException
     */
    @HmRequestMapping("/review/add")
    public void addReview (HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        try{
            ReviewLog reviewLog = parseJSON2Object(request, ReviewLog.class);
            log.info("addReview reviewLog:{}",reviewLog);
            // 获取用户信息
            User user = getSessionUser(request, GlobalConst.SESSION_KEY_USER);
            // 从上下文获取用户ID，调试默认为1
            reviewLog.setUserId(user!=null?user.getId():1);
            reviewLogService.addReviewLog(reviewLog);
            printResult(response,new Result(true,"操作成功"));
        }catch( RuntimeException e){
            log.error("addReview",e);
            printResult(response,new Result(false,"操作失败"));
        }
    }
}
```

4.4 测试接口



4.4 编写前端代码

在questionPreview.html中，找到reviewItem方法，去掉模拟调用，加入axios请求。

```
// 审核试题
reviewItem(status) {
    let t = this;

    // 必传参数
    let params = {
        questionId: this.questionId,
        status: status,
        comments: ''
    };
    //this.$message.success("审核通过");
    //window.location.href = "questionClassicList.html";
    axios.post(app_path + "/review/add.do", params).then((response) => {
        if (response.data.flag) {
            this.$message.success(response.data.message);
            setTimeout(function () {
                window.location.href = "questionClassicList.html";
            }, 1000);
        } else {
            this.$message.error(response.data.message);
        }
    })
}
```

4.5 使用注解权限

去掉配置文件中，针对/pages/questionClassicList.html的权限配置，使用注解来实现权限控制。

4.5.1 注释文件权限配置

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>

    <!--授权列表-->
    <security pattern="/pages/index.html"
has_role="ROLE_ADMIN,ROLE_QUESTION_RECORDER"/>
```

```
    <security pattern="/pages/questionBasicList.html"
has_role="ROLE_ADMIN,ROLE_QUESTION_RECORDER"/>
    <!--
    <security pattern="/pages/questionClassicList.html" has_role="ROLE_ADMIN"/>
    -->
    <security pattern="/pages/userList.html" has_role="ROLE_ADMIN"/>

    <!--注解扫描包-->
    <scan package="com.itheima.mm.controller" />

</beans>
```

4.5.2 加入注解配置权限

为ReviewLogController类的addReview方法，加入注解权限

```
/**
 * 增加审核
 * @param request
 * @param response
 * @throws ServletException
 * @throws IOException
 */
@HmAuthority("QUESTION_REVIEW_UPDATE")
@HmRequestMapping("/review/add")
public void addReview (HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    try{
        ReviewLog reviewLog = parseJSON2Object(request, ReviewLog.class);
        log.info("addReview reviewLog:{}",reviewLog);
        // 获取用户信息
        User user = getSessionUser(request, GlobalConst.SESSION_KEY_USER);
        // 从上下文获取用户ID，调试默认为1
        reviewLog.setUserId(user!=null?user.getId():1);
        reviewLogService.addReviewLog(reviewLog);
        printResult(response,new Result(true,"操作成功"));
    }catch( RuntimeException e){
        log.error("addReview",e);
        printResult(response,new Result(false,"操作失败"));
    }
}
```

4.5.3 修订前端代码

权限验证失败时，服务端返回的不是JSON格式，故需要修改前端代码，代码如下：

```
//this.$message.success("审核通过");
//window.location.href = "questionClassicList.html";
axios.post(app_path + "/review/add.do", params).then((response) => {
    if (response.data.flag) {
        this.$message.success(response.data.message);
        setTimeout(function () {
            window.location.href = "questionClassicList.html";
        }, 1000);
    } else if(response.data.message){
        this.$message.error(response.data.message);
```

```
    }else{
        this.$message.error(response.data);
    }

})
```

验证，当使用zhangsan登录系统，操作审核按钮时，会弹出如下提示：