RBAC

课程目标

- 理解项目要做权限控制的原因。
- 理解如何设计权限控制的功能，熟悉表结构，表之间的关系。
- 掌握通过代码实现权限控制的功能。
    - 掌握部门管理。
    - 掌握权限管理。
    - 掌握角色管理。
    - 掌握员工管理。
    - 掌握登录，登出，登录拦截。
    - 掌握权限拦截。
- 了解统一异常处理。

一、权限控制概述（理解）

1、访问控制目的

在实际的组织中，为了完成组织的业务工作，需要在组织内部设置不同的职位，职位既表示一种业务分工，又表示一种责任与权利。根据业务分工的需要，职位被划分给不同群体，各个群体的人根据其工作任务的需要被赋予不同的职责和权利，每个人有权了解与使用与自己任务相关的信息与资源，对于那些不应该被知道的信息则应该限制他们访问。这就产生了访问控制的需求。

限制主体对资源的访问，限制用户可以访问而且只能访问自己被授权的资源，从而保障数据资源在合法范围内得以有效使用和管理。权限管理几乎出现在任何系统里面，只要有用户和密码的系统。

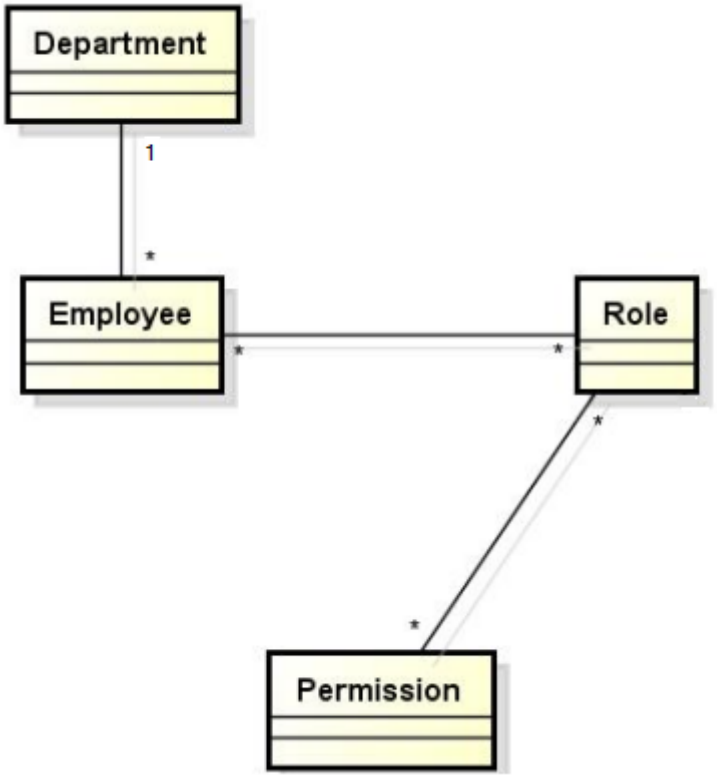
2、访问控制策略

在权限管理中使用最多的还是基于角色访问控制（RBAC：Role Based Access Control）。基于角色的访问控制，是 20 世纪 90 年代研究出来的一种模型。这种模型的基本概念是把许可权（Permission）与角色（Role）联系在一起，用户通过充当合适角色的成员而获得该角色的许可权。

例如，在一间公司中，有老板、经理、行政、人事、财务等不同的职位，在通常情况下，职位所赋予的权利是不变的，但在某个职位上工作的人可以根据需要调整。RBAC 模型对组织内部的这些关系与访问控制要求给出了非常恰当的描述。

3、RBAC 重要对象

- 用户（Employee）：角色施加的主体；用户通过拥有某个或多个角色以得到对应的权限。
- 角色（Role）：表示一组权限的集合。
- 权限（Permission）：一个资源代表一个权限，是否能访问该资源，就是看是否有该权限。



4、RBAC 流程与模型分析

管理员可为用户分配角色和权限（用户、角色和权限数据还有其之间关系数据都得保存起来）。

> 动手画流程图分析以及模型图。

二、项目搭建（掌握）

1、项目准备

使用 Maven 构建项目，使用之前练习 Spring Boot 的项目进行改造即可。

> 学习新的东西一定要搞清楚，有什么用？为什么要用？如何使用？

2、Bootstrap（了解）

Bootstrap 是美国 Twitter 公司的设计师 Mark Otto 和 Jacob Thornton 合作基于 HTML、CSS、JavaScript 开发的简洁、直观、强悍的前端开发框架，使得 Web 开发更加快捷。 Bootstrap 提供了优雅的 HTML 和 CSS 规范，它即是由动态 CSS 语言 Less 写成。（重点是响应式，能适应各种各种设备）

2.1、学习方法

阅读官方文档，找到想要的案例，代码调整修改。

2.2、HelloWorld

- 项目中添加 Bootstrap 文件资源；
- 页面引入相关的样式和 JS；

```
<!-- Bootstrap 核心 CSS 文件 -->
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@3.3.7/dist/css/bootstrap.min.cs
s" rel="stylesheet">
<!-- jQuery (Bootstrap 的所有 JavaScript 插件都依赖 jQuery，所以必须放在前边) -->
  <script src="https://cdn.jsdelivr.net/npm/jquery@1.12.4/dist/jquery.min.js
"></script>
<!-- Bootstrap 核心 JavaScript 文件 -->
<script src="https://cdn.jsdelivr.net/npm/bootstrap@3.3.7/dist/js/bootstrap.
min.js"></script>
```

- 添加文档案例查看是否有效。

2.3、常用组件

面板，栅格系统，表格，按钮，表单，字体图标，模态框，列表等。

3、AdminLTE（了解）

AdminLTE 是受欢迎的开源的管理仪表盘和控制面板的 WebApp 模板。它是基于 Bootstrap 的 CSS 框架，反应灵敏的 HTML 模板。利用所有 Bootstrap 的组件对大部分使用插件进行设计和调整风格，创建出可以用作后端应用程序的用户界面一致性设计。AdminLTE 是基于模块化设计，很容易在其之上定制和重制。

项目的页面是基于 AdminLTE 的模板来改造的，而 AdminLTE 又是基于 Bootstrap 框架，所以后续如果有新功能新样式要加，可以直接在 AdminLTE 官网找案例，或者 Bootstrap 官网找案例，都可以直接使用。

4、装饰页面

使用授课资料中提供的静态资源文件，拷贝到当前项目中来，结构如下：

- resources
  - static
    - css
    - js
    - img
    - login.html

测试访问 /static/login.html，查看页面效果。

三、部门管理（掌握）

1、部门查询

1.1、后端 PageHelper 分页插件

PageHelper 是一个 MyBatis 的分页插件，负责将已经写好的 SQL 语句，进行分页加工。无需你自己去封装以及关心 SQL 分页等问题，使用很方便。

> 要掌握如何在一个新项目中加入该分页插件，以及如何实现分页功能。

1.2.1、添加依赖

```
<dependency>
    <groupId>com.github.pagehelper</groupId>
    <artifactId>pagehelper-spring-boot-starter</artifactId>
    <version>1.3.0</version>
</dependency>
```

1.2.2、配置分页插件

修改 application.properties，如下配置：

```
pagehelper.reasonable=true
pagehelper.page-size-zero=true
```

1.2.3、实现分页

- 删除 Mapper.xml 文件中查询数量的代码。
- 删除 Mapper.xml 文件的分页查询的 SQL 中的 Limit 子句，QueryObject 中也不需要提供 getStart 方法了。
- 使用分页插件提供的 PageInfo 类进行封装，不需要我们自己再定义 PageResult 类了。

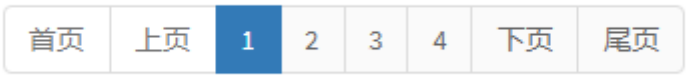
```
public PageInfo<Department> query(QueryObject qo) {
    // 使用分页插件，传入当前页，每页显示数量
    PageHelper.startPage(qo.getCurrentPage(), qo.getPageSize());
    List<Department> departments = departmentMapper.selectForList(qo);
    return new PageInfo(departments);
}
```

- 修改页面获取分页信息的取值（这个步骤留到下面前端分页插件的使用的时候进行修改），因为现在我们使用 com.github.pagehelper.PageInfo 来封装分页查询的数据的。

1.1、前端 twbs-pagination 分页插件

twbs-pagination 是一个简单的自适应 Bootstrap 样式的分页插件，用于前端绘制分页相关的样式效果。

> 理解参数的作用，以后如果参数名有变化，会改即可，不作过多要求。




```
$(function(){
    var totalPages = /*[[${pageInfo.pages}]]*/ 1;
    var startPage = /*[[${pageInfo.pageNum}]]*/ 1;
    $('#pagination').twbsPagination({
        totalPages: totalPages,
        startPage: startPage,
        first:'首页',
        prev:'上一页',
        next:'下一页',
        last:'尾页',
        visiblePages: 5,
        onPageClick: function (event, page) {
            $('#currentPage').val(page);
```

```
            $('#searchForm').submit();
        }
    });
});
```

2、部门新增

2.1、Bootstrap 模态框

因为部门保存修改的字段少，很适合使用模态框来改，看着更舒服些，也可以少写一个 input 页面，控制器少写 input 方法，少执行一条 SQL 语句。

2.2、加入模态框

- 参考 Bootstrap 官方文档拷贝模态框的元素，放在页面 body 元素的直接子元素中，把之前 input 页面的表单拷贝过来放在 modal-body 中。
- 修改添加按钮，按钮的 href 属性不再需要指定 input 的地址了，因为我们要使用到点击事件。
- 给按钮绑定点击事件，点击之后弹出模态框。

```
$('#模态框的id').modal('show'); //官方文档中表示通过该方法即可弹出模态框
```

3、部门修改

编辑回显，当用户点击编辑按钮的时候，在模态框中回显要编辑的数据。

> 先自行思考有哪些方式可以实现？用已学过的知识点。

- 直接从页面上获取被编辑数据，回显到模态框中，这是比较方便的。
- 在对应的实体类中添加一个 get 方法，用于获取 JSON 字符串数据。
- 在编辑按钮上使用 data-json 属性绑定当前的属性数据。
- 点击编辑按钮后，在事件中获取该按钮 data-json 属性中的值，并使用 DOM 操作回显在模态框中。

> 实现功能时，可先参考流程图实现，注意编辑和新增共用模态框的缓存问题。

4、部门删除

4.1、需求

点击删除之前先提示确认框，用户确认之后才进行删除操作。

目前用户删除操作是没有什么提示，即使有提示默认提示框也是很丑的，风格搭配与 Bootstrap 不一致。另外一个问题是点击删除就可以直接删除数据，可能会出现误操作的情况。

4.2、SweetAlert2（了解）

SweetAlert2 是一个美观，响应，可定制，替代 JavaScript 的弹出框。

4.2.1、引入插件

```
<link rel="stylesheet"
href="/static/js/plugins/sweetalert2/sweetalert2.min.css">
<script src="/static/js/plugins/sweetalert2/sweetalert2.min.js"></script>
```

4.2.2、使用插件

查看官方文档配置案例，拷贝过来修改即可。

```
Swal.fire({
    title: 'Are you sure?',
    text: "You won't be able to revert this!",
    icon: 'warning',
    showCancelButton: true,
    confirmButtonColor: '#3085d6',
    cancelButtonColor: '#d33',
    confirmButtonText: 'Yes, delete it!',
    cancelButtonText: 'No, cancel!'
}).then((result) => {
    if(result.value) {
        // 点了确定做什么，由开发者决定
    }
});
```

四、权限管理（掌握）

1、权限分析

1.1、权限表设计

- 权限名称是给分配权限的人看的，用中文，见名知意。
- 权限表达式是给程序做判断的时候用的，用英文，并规定格式。

1.2、权限来源

要做资源的访问权限限制，其实就是对系统中的动态资源或者说控制器中的处理方法做限制，因为处理方法包含对数据库的 CRUD 操作。换句话说，控制器中的处理方法就是一个一个权限，即数据库中权限表的数据来源于所有控制器的一个一个处理方法。权限表达式的值需要具有唯一性，那么我们就约定权限表达式组成：控制器类名首字母小写除去 Controller 加上方法名，例如：`department:list`，这样就可以唯一了。

1.3、数据录入

如何把控制器中每个处理方法怎么转化成权限表中的数据？

- 一条条手动添加太麻烦，需要用代码来批量添加。
- 我们自定义一个注解，在控制器的处理方法上贴该注解，并直接在注解上设置权限名称和权限表达式，自己再定义第三方程序扫描这些贴了注解的方法，用程序录入数据。而且使用注解实现的好处：
  - 一是可直接在注解上设置权限名称和权限表达式数据。
  - 二是可标识哪些方法（资源）需要进行权限控制，**区分一个处理方法是否要做权限限制**，就看是否有贴注解，贴了代表要限制，反之不要，代表所有人能访问。

2、权限查询

- 普通分页功能。

3、权限加载

> 需要画图分析流程。

需求：程序根据代码自动生成权限数据，无须用户一条一条手动添加，实现步骤：

- 页面提供“权限加载”的按钮：

```
<a href="javascript:;" class="btn btn-success btn-reload" style="margin:
10px;">
    <span class="glyphicon glyphicon-repeat"></span> 重新加载
</a>
```

- 给按钮绑定点击事件。
- 点击后使用确认框提示用户 `加载时间可能较长，是否确认进行加载？`，用户确认后则发送请求到后台。
- 后台接收到请求后，则进行加载逻辑。代码逻辑思路如下：
  - 获取 RequestMappingHandlerMapping 对象(直接用 @Autowired 注入)，从该对象中获取所有的 HandlerMethod 对象（getHandlerMethods），判断方法是否有贴权限注解（getMethodAnnotation）。或者通过获取 Spring 容器对象，再从中获取控制器对象，获取控制对象的字节码对象，反射获取所有控制器中的处理方法。
  - 若贴有自定义注解，则从注解中获取权限名称和权限表达式，还要判断这个方法的权限表达式是否已经存在数据库中，若该权限表达式不存在数据库，则创建 Permission 对象，封装数据并存入数据库中，若已存在，可直接跳过。

> Spring MVC 应用启动时，会利用 RequestMappingHandlerMapping 对象，搜集并分析每个控制器中每个带 @RequestMapping 注解的处理方法，通过 HandlerMethod 来封装和表示该方法，并与注解中的映射路径一一绑定。HandlerMethod 封装了很多属性，可以方便的访问到请求方法、方法参数、方法上的注解、所属类等信息。

五、角色管理（掌握）

> 添加与编辑记得要处理角色与权限的关系，操作与下面添加编辑员工处理与角色一样。

1、角色查询

完成角色分页查询。

2、角色新增

- 左边权限下拉框中显示权限数据。
- 完成权限的左移右移功能。
- 角色保存时全选右边权限下拉框的选项。
    - 问题：下拉框只会提交选中的数据，若没有选中的数据是不会提交的，这样会导致右边没有被选中就提交不了。
    - 思路：修改按钮为普通按钮，绑定点击事件处理函数，在事件函数中把右边的 select 元素中的 option 设置为选中后，再提交表单。
- 角色添加时处理权限。
    - 后台在保存角色时，还需要往角色权限中间表插入数据，保存角色和权限的关系。

3、角色修改

- 角色基本数据的回显。
- 右边权限下拉框回显角色对应权限。
    - 注意：若要回显当前巨角色所拥有的权限，须进行连表查询或额外 SQL 查询，并且把结果封装到对象的属性中，才有数据可回显。至于 SQL 怎么写以及如何封装结果，不记得请回顾 MyBatis 课程。
- 权限去重。
    - 问题：在权限回显的时候，发现左右两边的权限有重复，其实右边显示的权限，就是角色已经拥有的，不应该在左边出现，这样看起来会更清晰。
    - 思路：页面加载完后，拿两边的 option 对比，遍历左边拿到每个权限 id，若已经存在右边，则删除，若不存在，则保留。
- 角色编辑时处理权限。
    - 更新角色信息时，也需要更新角色权限中间表，一般使用的方式是先删除旧有的关系，再加新的关系。

六、员工管理（掌握）

1、员工查询

- 实现分页查询，参考部门页面实现。
- 实现过滤查询，关键字以及部门下拉框，注意查询条件回显。

2、员工删除

- 抽取删除的 JS 代码，注意后台还要从员工角色中间表清除关系数据。

3、员工新增

- 部门下拉框显示部门数据。
- 左边角色下拉框中显示角色数据。
- 实现角色左移右移功能。
- 超管隐藏角色编辑。
    - 需求：超级管理员具备所有的权限，没有必要给其分配角色。
    - 思路：若勾选了超级管理员，则删除角色配置的相关元素；若取消勾选，则还原被删除的元素。
- 员工保存时全选右边角色下拉框的选项。

    - 问题：下拉框只会提交选中的数据，若没有选中的数据是不会提交的，这样会导致右边没有被选中就提交不了。
    - 思路：修改按钮为普通按钮，绑定点击事件处理函数，在事件函数中把右边的 select 元素中的 option 设置为选中后，再提交表单。
- 员工添加时处理角色。
    - 后台在保存员工时，还需要往员工角色中间表插入数据，保存员工和角色的关系。

4、员工修改

- 员工基本数据的回显。
- 部门单选下拉框的回显。
- 右边角色下拉框回显员工对应的角色。
    - 注意：若要回显当前员工的所属部门和所拥有的角色，须进行连表查询或额外 SQL 查询，并且把结果封装到对象的属性中，才有数据可回显。
- 角色去重。
    - 问题：在角色回显的时候，发现左右两边的角色有重复，其实右边显示的角色，就是用户已经拥有的，不应该在左边出现，这样看起来会更清晰。
    - 思路：页面加载完后，拿两边的 option 对比，遍历左边拿到每个角色 id，若已经存在右边，则删除，若不存在，则保留。
- 细节完善。
    - 用户名作为登录账户，一般不能编辑，并且不进行验证，但要注意后端 update 的 SQL 中不能去更新用户名。（使用 disabled）。
    - 密码框隐藏。
        - 需求：管理员编辑员工信息是不需要提供修改密码的功能，密码修改功能应该在个人设置自行修改。
        - 思路：使用判断，如果是编辑则不显示密码框，但要注意后端 update 的 SQL 中不能去更新密码。
    - 超管隐藏角色编辑。
        - 需求：前面做了点击超管，删除角色相关元素，但那是点击才会触发，如果是直接通过代码回显，不会触发点击事件，则无法做到这个效果了。
        - 思路：回显时也要判断若是超级管理员，需要删除角色相关元素。
- 员工编辑时处理角色。
    - 更新员工信息时，也需要更新员工角色中间表，一般使用的方式是先删除旧有的关系，再加新的关系。

5、员工模块完善

5.1、Bootstrap-validator

Bootstrap-validator 是一个表单验证插件，在做 Web 项目的时候，表单数据验证是再常见不过的需求了，友好的错误提示能增加用户体验，提高程序稳定性。

5.2、引入插件

```
<!--引入验证插件的样式文件-->
<link rel="stylesheet" href="/js/plugins/bootstrap-
validator/css/bootstrapValidator.min.css"/>
<!--引入验证插件的 JS 文件-->
<script type="text/javascript" src="/js/plugins/bootstrap-
validator/js/bootstrapValidator.min.js"></script>
<!--中文语言库-->
<script type="text/javascript" src="/js/plugins/bootstrap-
validator/js/language/zh_CN.js"></script>
```

5.3、使用插件

注意：记得把按钮的 type 改成 submit，把之前用 JS 提交表单的代码删除掉。

参考：插件\bootstrap-validator\demo\index.html 案例文件，拷贝并进行修改。

```
$("#editForm").bootstrapValidator({
    feedbackIcons: { // 图标
        valid: 'glyphicon glyphicon-ok',
        invalid: 'glyphicon glyphicon-remove',
        validating: 'glyphicon glyphicon-refresh'
    },
    fields:{ // 配置要验证的字段
        username:{
            validators:{ // 验证的规则
                notEmpty:{ // 不能为空
                    message:"用户名必填" // 错误时的提示信息
                },
                stringLength: { // 字符串的长度范围
                    min: 1,
                    max: 5
                }
            }
        },
        name:{
            validators:{ // 验证的规则
                notEmpty:{ // 不能为空
                    message:"姓名必填" // 错误时的提示信息
                },
                stringLength: { // 字符串的长度范围
                    min: 1,
                    max: 5
                }
            }
        },
        password:{
            validators:{
                notEmpty:{ // 不能为空
                    message:"密码必填" // 错误时的提示信息
                },
            }
        },
        repassword:{
            validators:{
                notEmpty:{ // 不能为空
                    message:"密码必填" // 错误时的提示信息
                },
```

```
                identical: {// 两个字段的值必须相同
                    field: 'password',
                    message: '两次输入的密码必须相同'
                },
            }
        },
        email: {
            validators: {
                emailAddress: {} // 邮箱格式
            }
        },
        age:{
            validators: {
                between: { // 数字的范围
                    min: 18,
                    max: 60
                }
            }
        }
    }
}).on('success.form.bv', function(e) {
    $('.selfRoles > option').prop('selected', 'true');
    // TODO 这里可以改成用异步的方式提交表单
});
```

5.4、验证用户是否存在

5.4.1、前端代码

```
name: {
    validators: {
        remote: { // 远程验证
            type: 'get', // 请求方式
            url: '/employee/checkUserName', // 请求地址
            message: '用户名已经存在', // 验证不通过时的提示信息
            delay: 1000, // 输入内容 1 秒后发请求
            /* data: function() {  自定义提交的参数，默认只会提交当前用户名 input 的参数
                return {
                    username: $('[name="username"]').val()
                };
            } */
        }
    }
},
```

5.4.2、后端代码

思路：后台接收到前端传来的用户名后，查询数据库是否已存在该账号，再返回不同的结果给前端。注意：插件要求返回结果需要为键值对形式 key 为 valid，value 为 boolean 类型（使用 Map 封装即可），示例如下：

- valid: true 代表验证通过（该用户名不存在）
- valid: false 代表验证不通过（用户名已经存在）

七、登录拦截（掌握）

思考： 拦截的目的是什么？如何实现？

1、员工登录

1.1、登录需求

因为要判断员工是否有权限访问，首先得知道现在操作的人是谁，所以必须先实现登录功能。

1.2、实现步骤

> 实现前先画流程图分析。

- 提供登录页面，可输入用户名与密码信息，并添加执行登录的按钮。
- 给按钮绑定点击事件
- 事件中发送登录请求，使用 AJAX 方式提交。（使用 AJAX 原因：用户体验更好，既可保留用户刚输入的用户名和密码，失败之后也不需要做请求转发共享错误信息，直接返回 JSON 数据效率更高。）
- 后端接收参数后，查询数据库验证是否正确，若登录失败则返回错误信息给前端，若登录成功则把员工存到 session 中，方便下次获取。
- 页面菜单栏显示登录员工的名字，可以从 session 获取获取。

> 请求传的参数：可使用 jQuery 序列化的方法，对表单参数进行序列化，完成参数拼接。

```
$('表单id').serialize() // 返回结果类似 username=admin&password=1
```

2、用户退出

后台逻辑即从对应 session 中移除之前登录时存入的数据，或者销毁对应 session。

3、登录拦截

思考： 过滤器能做，拦截器也能做，区别是什么？

实现步骤如下：

- 创建登录拦截器类，实现前置拦截方法。
- 判断 session 中是否有员工对象，若有则代表有登录，直接放行，若没有，则重定向到登录页面，不放行。
- 在配置类中注册拦截器（注意需要排除某些资源，就是不需要登录都能访问的资源）。

八、权限拦截（掌握）

思考：拦截的目的是什么？如何实现？

> 实现前先画流程图分析。

1、实现步骤

- 创建权限拦截器类，实现前置拦截方法。
- 编写执行拦截代码（流程图）。
- 在配置类中注册拦截器（注意需要排除某些资源，就是不需要权限都能访问的资源）。

2、问题分析

- 问题 1、权限拦截的逻辑中，我们如何得知用户访问的哪个控制器的哪个处理方法，还有被拦截的方法是否有贴注解？
- 问题 2：要判断用户是否有权限需要获取当前员工的所拥有的权限，才能比较，但是这样会导致频繁读取数据库，如何解决？

> 思路：Spring MVC 框架提供了该功能，在拦截器前置拦截的方法上注入了 HandlerMethod 类型的对象，这个对象就是当前被拦截的处理方法，这跟之前权限加载时所用到的 HandlerMethod 是相同类型的。

> 思路：每次判断权限都要查询数据库，效率太低。而权限数据不会经常变动，可以在登录时就把用户的权限存入 session 中，判断时再从 session 中取出，这样从内存读取数据更快，减轻数据库查询压力。

3、抽取 UserContext 工具类（了解）

抽取这个工具对现有代码进行优化。

3.1、问题分析

- 问题 1：重复操作 session 的代码太多，session 中的 key 也需要经常使用，而且 key 多了容易混乱写错，增加维护难度。
- 问题 2：如何在工具类中获取 session 呢？作为参数传到工具方法内部，但是这样很麻烦，代码也很多余。

3.2、问题解决

> 思路：抽出一个 UserContext 工具类，工具类中分别提供当前登录用户的 get/set 方法和当前用户权限集合的 get/set 方法，方便外部调用，并把所有 Key 抽成常量来使用。

> 思路：使用 RequestContextHolder 工具类，它是 Spring MVC 提供的工具类，提供了一个静态方法，可以在代码任意的地方获取 request 对象，response 对象，sessison 对象，不需要每次自己把对象传进去，但是这个工具类获取 session 的代码得写在好几个地方，所以我们可以直接再封装为一个用于获取 session 的方法，提供给其他方法调用。

```
public static HttpSession getSession() {
    ServletRequestAttributes attrs = (ServletRequestAttributes)
RequestContextHolder.getRequestAttributes();
    return attrs.getRequest().getSession();
}
```

九、统一异常处理（了解）

1、引出问题

在 Java EE 项目的开发中，不管是对底层的数据库操作过程，还是业务层的处理过程，还是控制层的处理过程，都难以避免遇到各种异常需要处理的问题。若不对异常进行处理的话，给用户看到异常信息是不好的，对不懂程序的用户来说还以为你的网站出问题了；对懂程序的人来说，看到你的异常信息，会显得很 low，特别是 SQL 出错，甚至还会暴露你的数据库字段。

2、如何解决

- 手动 try

    - 弊端是到处是重复代码，系统的代码耦合度高，工作量大且不好统一，维护的工作量也很大。
- 利用 Spring MVC 的方式
    - Spring MVC 为 Controller 处理方法执行出现异常提供了全局统一处理，可以使用 `@ExceptionHandler` 配合 `@ControllerAdvice` 注解实现异常处理，可减少代码量，提高拓展性和可维护性。
    - 添加处理控制器异常处理类，确保 Spring 配置中要能扫描到这个类。
    - 针对不同异常进行不同处理，针对不同处理方法响应的内容，需要进行不同处理，比如原来方法响应 HTML 依然响应 HTML，若原来方法响应 JSON 依然响应 JSON。

```
/**
* 对控制器进行增强处理
*/
@ControllerAdvice
public class RuntimeExceptionHandler {
    /**
    * 该方法是用于捕获并处理某种异常
    * e：现在出现的异常对象
    * method：现在出现异常的那个处理方法
    */
    @ExceptionHandler(RuntimeException.class)
    public String exceptionHandler(RuntimeException e, HandlerMethod method,
Model model, HttpServletResponse response) throws Exception{
        e.printStackTrace(); // 方便开发的时候找 bug
        // 若原本控制器的方法是返回 JSON，现在出异常也应该返回 JSON
        // 获取当前出现异常的方法，判断是否有 ResponseBody 注解，有就代表需要返回 JSON
        if(method.hasMethodAnnotation(ResponseBody.class)){
            response.setContentType("application/json;charset=UTF-8");
            response.getWriter()
                .print(JSON.toJSONString(new JsonResult(false, "系统异常，请联系管
理员")));
            return null;
        }
        model.addAttribute("errorMsg", "系统异常，请联系管理员");
        // 若原本控制器的方法是返回 HTML，现在也应该返回 HTML
        return "common/error";
    }
}
```

3、自定义异常

在开发中还可以根据自己业务的异常情况来自定义业务逻辑异常类，一般继承于 `java.lang.RuntimeException`。

```
public class LogicException extends RuntimeException {
    public LogicException(String errorMsg){
        super(errorMsg);
    }
}
```

比如虽然登录出错，也响应 JSON 数据，但其需要提示的消息更详细，所以通过自定义异常，再利用 Spring MVC 全局处理异常方式，针对这个异常进行专门处理。

练习

- 完成部门管理，包含部门分页查询，删除，新增和修改。
- 完成权限管理，包含权限加载，权限分页查询。
- 完成角色管理，包含角色分页查询，新增和修改。
- 完成员工管理，包含角色分页过滤查询，删除，新增和修改。
- 完成登录，登出和登录拦截功能。
- 完成权限拦截功能。