

Java狂神说SpringBoot员工管理系统

一、技术选型

- 第一阶段（模拟数据阶段，不涉及数据库）：SpringBoot、Thymeleaf等
- 第二阶段：整合MyBatis、Druid、log4j
- 第三阶段：整合Shrio、Shiro整合MyBatis认证授权、Shiro整合Thymeleaf

实现流程

二、第一阶段-模拟数据阶段

2.0使用IDEA创建Maven的SpringBoot工程

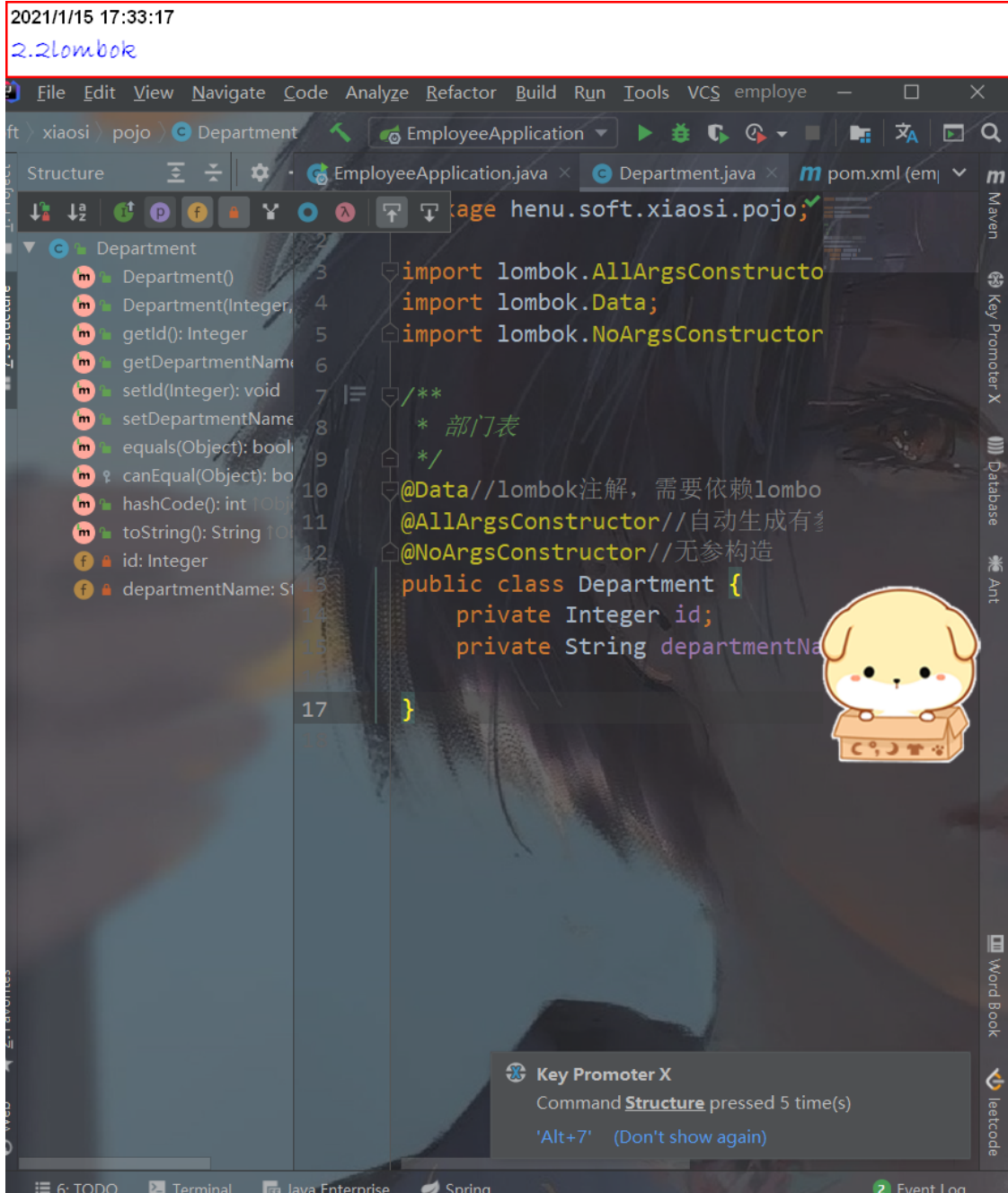
创建时选中web、lombok依赖

2.1 静态资源

- html文件放在 `templates` 目录下
- 样式资源放在 `static` 目录下

2.2pojo

1. Department部门表实体类
 2. Employee员工表实体类
- 使用lombok来简化pojo，添加 `@Data`、`@AllArgsConstructor`、`@NoArgsConstructor` 注解来自动生成构造方法和setter、getter方法
 - 需要lombok依赖和IDEA的Lombok插件
 - Employee类包含成员变量Department,对应数据库的主外键信息。



2.3dao

1. DepartmentDao

2. EmployeeDao

- 模拟数据库数据，使用Map存储数据，然后定义方法实现增删改查
- 对于DepartmentDao，模拟获取全部部门信息、根据 部门id 查询部门信息
- 对于EmployeeDao，模拟获取全部的员工信息、根据 员工id 查询员工信息、增加员工信息（定义 initId++模拟主键自增）、根据员工id删除员工

2.4首页

实现跳转到index.html

- 自定义视图解析器组件：MyMvcConfig，实现WebMvcConfigure接口，重写 `addViewController()` 方法

可以使用@Controller、@RequestMapping实现方法跳转页面，现在自定义MyMvcConfig组件可以方便拓展相关功能，方便路由、同时保留原部分的自动配置（更多参考：<https://www.jb51.net/article/174767.htm>）

2021/1/15 20:27:49

29.1.1 Spring MVC Auto-configuration

Spring Boot provides auto-configuration for Spring MVC that works well with most applications.

The auto-configuration adds the following features on top of Spring's defaults:

- Inclusion of `ContentNegotiatingViewResolver` and `BeanNameViewResolver` beans.
- Support for serving static resources, including support for WebJars (covered later in this document).
- Automatic registration of `Converter`, `GenericConverter`, and `Formatter` beans.
- Support for `HttpMessageConverters` (covered later in this document).
- Automatic registration of `MessageCodesResolver` (covered later in this document).
- Static `index.html` support.
- Custom `Favicon` support (covered later in this document).
- Automatic use of a `ConfigurableWebBindingInitializer` bean (covered later in this document).

自定义MyMvcConfig组件可以方便拓展相关功能同时保留原部分的自动配置

If you want to keep Spring Boot MVC features and you want to add additional MVC configuration (interceptors, formatters, view controllers, and other features), you can add your own `@Configuration` class of type `WebMvcConfigurer` but without `@EnableWebMvc`. If you wish to provide custom instances of `RequestMappingHandlerMapping`, `RequestMappingHandlerAdapter`, or `ExceptionHandlerExceptionResolver`, you can declare a `WebMvcRegistrationsAdapter` instance to provide such components.

If you want to take complete control of Spring MVC, you can add your own `@Configuration` annotated with `@EnableWebMvc`.

2021/1/15 20:33:20

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.View;
import org.springframework.web.servlet.ViewResolver;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

import java.util.Locale;

// 如果，你diy一些定制化的功能，只要写这个组件，然后将它交给springboot，springboot就会帮我们自动装配！
// 扩展 springmvc dispatcherservlet
@Configuration
public class MyMvcConfig implements WebMvcConfigurer {

    //ViewResolver 实现了视图解析器接口的类，我们就可以把它看做视图解析器
    @Bean
    public ViewResolver myViewResolver(){
        return new MyViewResolver();
    }

    //自定义了一个自己的视图解析器MyViewResolver
    public static class MyViewResolver implements ViewResolver {
        @Override
        public View resolveViewName(String viewName, Locale locale) throws Exception {
            return null;
        }
    }
}
```

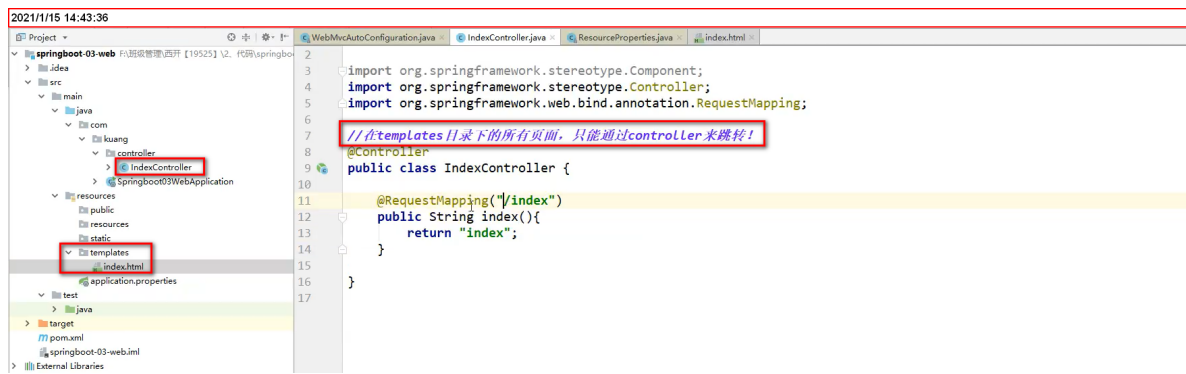
```
@Configuration
public class MyMvcConfig implements WebMvcConfigurer {

    @Override//重写视图跳转控制器
    public void addViewControllers(ViewControllerRegistry registry) {
        registry.addViewController("/").setViewName("index");
        registry.addViewController("/index.html").setViewName("index");
    }
}
```

2.5bug

- 出现不能访问index.html的问题，原因是没有导入 thymeleaf 依赖，而默认templates下的静态资源只能@Controller跳转

```
<!-- thymeleaf依赖-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```



- 静态资源样式无法加载：路径问题，`static/assert`下的css等资源文件可以直接被 `templates`下的html 使用 `/assert/xxx.css` 来引用到.最好换为thymleaf语法风格的引用

2.6国际化

- 在resources创建3个绑定的配置文件，分别对应各种语言
- 将html中的转为thymleaf接管
- 自定义 国际化转换组件 `MyLocaleResolve` 实现 `LocaleResolver`接口，解析 按钮 带来的参数 然后将语言对应起来，最后@Bean将组件装配给 `MyMvcConfig` 组件类

```
public class MyLocaleResolve implements LocaleResolver {

    //解析请求
    @Override
    public Locale resolveLocale(HttpServletRequest httpServletRequest) {
        //获取语言请求参数
        String language = httpServletRequest.getParameter("l");
        //默认
        Locale locale = Locale.getDefault();

        //如果请求的连接携带了国际化语言参数
        if(!StringUtils.isEmpty(language)){
            String[] split = language.split("_");
            //拿出来参数地区-语言
            return new Locale(split[0], split[1]);
        }

        return locale;
    }

    @Override
    public void setLocale(HttpServletRequest httpServletRequest,
        HttpServletResponse httpServletResponse, Locale locale) {

    }
}
```

1. 首页配置:

1. 注意点, 所有页面的静态资源都需要使用thymeleaf接管;
2. url: `@{}`

2. 页面国际化:

1. 我们需要配置 `i18n` 文件
2. 我们如果需要在项目中进行按钮自动切换, 我们需要自定义一个组件 `LocaleResolver`
3. 记得将自己写的组件配置到 `spring` 容器 `@Bean`
4. `#{}`

thymeleaf当配置文件时用`#{}`

2.7 登录

- 表单action交给thymeleaf接管, 设置提交到LoginController, 然后SpringMVC的老一套取出来用户名和密码, 实现逻辑判断
 - 密码正确: 重定向到指定虚拟映射页面 (因为get的方式会将参数暴露在链接中, 因此需要在自定义跳转组件MyMvcConfig中创建虚拟映射)
 - 密码错误: 在html中设置p标签, 内容交给thymeleaf, 将Controller的msg密码错误信息取出来, 然后转发到index.html, 注意提示信息需要配合逻辑判断使用

```
@Controller
public class LoginController {
    @RequestMapping("/user/login")
    public String login(
        @RequestParam("username") String username,
        @RequestParam("password") String password,
        Model model){
        //具体的业务
        if(!StringUtils.isEmpty(username) && "123456".equals(password)){
            return "redirect:/main.html";//重定向
        }
        else{
            model.addAttribute("msg", "用户名或者密码错误!");
            return "index";//转发
        }
    }
}
```

2.8 登录拦截器

登录之后才能进入后台: 设置拦截器组件 LoginConfig 实现 HandlerInterceptor 接口, 重写第一个放行方法, 接受登录之后的Session实现放行, 否则转发到首页

- 需要注意要对index、user/login等登录页面过滤不拦截

- 需要对static/assert/下的静态资源过滤不拦截

```

**
 * 自定义拦截器组件
 */

public class MyLoginHandlerInterceptor implements HandlerInterceptor {

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse
response, Object handler) throws Exception {
        //从session中取数据，判断是否成功登录
        Object loginUser = request.getSession().getAttribute("loginUser");
        if(loginUser == null){
            //没有登录
            request.setAttribute("msg", "请登录后再操作！");
            request.getRequestDispatcher("/").forward(request, response);
            return false;
        }
        else{

            return true;
        }
    }
}

```

```

//自定义拦截器组件，装配组件
@Override
public void addInterceptors(InterceptorRegistry registry) {
    registry.addInterceptor(new MyLoginHandlerInterceptor())
        .addPathPatterns("/**")

    .excludePathPatterns("index.html", "/", "/user/login", "/assert/**");
}

```

2.9查

- 提取前端公共页面：前端公共代码组件的抽取，list.html和dashboard.html页面的侧边栏和顶部栏都是相同的，可以抽取出来
- 按钮样式高亮处理
- 展示员工信息数据

4. 员工列表展示

1. 提取公共页面

1. `th:fragment="sidebar"`2. `th:replace="~{commons/commons::topbar}"`

3. 如果要传递参数, 可以直接使用 () 传参, 接收判断即可!

2. 列表循环展示

```

<div class="table-responsive">
    <table class="table table-striped table-sm">
        <thead>
            <tr>

                <th>id</th>
                <th>lastName</th>
                <th>email</th>
                <th>gender</th>
                <th>department</th>
                <th>date</th>
                <th>操作</th>

            </tr>
        </thead>
        <tbody>
            <tr th:each="emp:${emps}">
                <td th:text="${emp.getId()}"></td>
                <td th:text="${emp.getLastName()}"></td>
                <td th:text="${emp.getEmail()}"></td>
                <td th:text="${emp.getGender()==0?'女':'男'}">

            </td>

                <td
                    th:text="${emp.getDepartment().getDepartmentName()}"></td>
                <td
                    th:text="${#dates.format(emp.getBirth(),'yyyy-MM-dd HH:mm:ss')}"></td>
                <td>
                    <a class="btn btn-sm btn-primary">编辑</a>
                    <a class="btn btn-sm btn-danger">删除</a>
                </td>
            </tr>

        </tbody>
    </table>
</div>

```

2.10增

点击新增表单跳转逻辑

- html链接跳转---》add对应的Controller---》Controller查询信息回显存储在model中----》thymeleaf遍历信息
- 数据回显：部门的信息需要在Controller中存入model中，等会遍历显示在html表单中

```
<!--其中th:value用来表示部门的id,我们实际传入的值为id-->
<option th:each="department:${departments}"
th:text="${department.getDepartmentName()}" th:value="${department.getId()}">
</option>
```

点击确定新增

- 接受前端表单来的Bean，直接将employee添加到模拟数据集合，然后重定向

```
//转到添加的表单
@GetMapping("/add")
public String toAddPage(Model model){

    //部门信息数据回显
    Collection<Department> departments = departmentDao.getDepartments();
    model.addAttribute("departments",departments);
    return "emp/add";
}

//添加员工
@PostMapping("/add")
public String adding(Employee employee){
    employeeDao.save(employee);

    return "redirect:/emps";
}
```

- 对于日期的添加的格式，可以自定义日期格式组件来完成设置或者是直接在配置文件中指定

2021/1/16 15:29:37

配置文件指定日期格式

```
add.html | WebMvcAutoConfiguration.java | application.properties | WebMvcProperties.java | EmployeeController.java
1 # 关闭默认图标
2 spring.mvc.favicon.enabled=false
3 # 关闭模板引擎的缓存
4 spring.thymeleaf.cache=false
5 server.servlet.context-path=/kuang
6 # 我们的配置文件的真实位置
7 spring.messages.basename=i18n.login
8
9 # 时间日期格式化!
10 spring.mvc.date-format=yyyy-MM-dd
```


2.11改

- 和增类似，点击按钮到Controller，回显数据到model,然后在表单中回显，提交表单和增共用一个方法（区别是改携带id）
- 需要注意的是，这里需要设置隐藏的id框，通过参数是否携带id区分是修改而不是新增
- 关于修改用户的id通过Restful风格传递

```
<a class="btn btn-sm btn-primary" th:href="@{/edit/{id}(id=${emp.getId()})}">编辑</a>
```

```
@RequestMapping("/edit/{id}")
public String edit(@PathVariable("id") int id,Model model){
    //数据回显
    Employee employee = employeeDao.getEmployeeById(id);
    model.addAttribute("employee",employee);
    Department department = departmentDao.getDepartmentById(id);
    model.addAttribute("department",department);

    Collection<Department> departments = departmentDao.getDepartments();
    model.addAttribute("departments",departments);
    return "/emp/edit";
}
```

2.12删

删就比较好实现了，点击删除按钮，携带用户id到Controller，完成删除

```
<a class="btn btn-sm btn-success" th:href="@{/delete/{id}(id=${emp.getId()})}">删除</a>
```

```
    }
    //删除
    @RequestMapping("/delete/{id}")
    public String delete(@PathVariable("id") Integer id){

        employeeDao.delete(id);
        return "redirect:/emps";

    }
}
```

2.13404错误页面

直接将错误页面放到templates下，SpringBooo会自动帮我们去找

2.14注销

就是删除session,在Logincontroller中设置

```
@RequestMapping("/user/logout")
public String logout(HttpSession session){
    session.invalidate();
    return "redirect:/index.html";//重定向
}
```

三、第二阶段-整合MyBatis、Druid、log4j阶段

2.15逻辑步骤

整合MyBatis、整合Druid

- 导依赖
- 在application.yml设置数据源信息
- 编写mapper下的接口，声明dao层的操作方法
- 编写resources/mapper下的XxxMapper.xml文件
- 编写service下的XxxService，实现业务逻辑方法
- 编写controller下的XxxController，实现Model和View之间二段路由跳转
- 编写config下的DruidConfig配置类

需要注意的地方：

1. 需要在application.yml指定XxxMapper.xml中书写SQL时 pojo 的别名
2. 使用druid需要配置数据源相关参数，并在DruidConfig加上 @Bean
3. 需要导MySQL、JDBC、Druid、MyBatis的依赖

```
<!--      JDBC依赖-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
</dependency>

<!-- https://mvnrepository.com/artifact/com.alibaba/druid依赖 -->
<dependency>
    <groupId>com.alibaba</groupId>
```

```

        <artifactId>druid</artifactId>
        <version>1.1.21</version>
    </dependency>
<!--      MyBatis依赖-->
    <dependency>
        <groupId>org.mybatis.spring.boot</groupId>
        <artifactId>mybatis-spring-boot-starter</artifactId>
        <version>2.1.3</version>

    </dependency>

```

◦ DruidConfig

```

package henu.soft.xiaosi.config;

import com.alibaba.druid.pool.DruidDataSource;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import javax.sql.DataSource;

@Configuration
public class DruidConfig {

    /**
     * 将自定义的 Druid数据源添加到容器中，不再让 Spring Boot 自动创建
     * 绑定全局配置文件中的 druid 数据源属性到
     * com.alibaba.druid.pool.DruidDataSource从而让它们生效
     * @ConfigurationProperties(prefix = "spring.datasource")：作用就是将 全局配
     * 置文件中
     * 前缀为 spring.datasource的属性值注入到
     * com.alibaba.druid.pool.DruidDataSource 的同名参数中
     */
    @ConfigurationProperties(prefix = "spring.datasource")
    @Bean
    public DataSource druidDataSource() {
        return new DruidDataSource();
    }
}

```

◦ application.yml

```

spring:
  datasource:
    username: root
    password: 720720
    # ?serverTimezone=UTC解决时区的报错
    url: jdbc:mysql:///springboot?
serverTimezone=UTC&useUnicode=true&characterEncoding=utf-8
    driver-class-name: com.mysql.cj.jdbc.Driver
    type: com.alibaba.druid.pool.DruidDataSource

```

```

#Spring Boot 默认是不注入这些属性值的，需要自己绑定
#druid 数据源专有配置
initialSize: 5
minIdle: 5
maxActive: 20
maxWait: 60000
timeBetweenEvictionRunsMillis: 60000
minEvictableIdleTimeMillis: 300000
validationQuery: SELECT 1 FROM DUAL
testWhileIdle: true
testOnBorrow: false
testOnReturn: false
poolPreparedStatements: true

#配置监控统计拦截的filters，stat:监控统计、log4j: 日志记录、wall: 防御sql注入
#如果允许时报错 java.lang.ClassNotFoundException:
org.apache.log4j.Priority
#则导入 log4j 依赖即可，Maven 地址：
https://mvnrepository.com/artifact/log4j/log4j
filters: stat,wall,log4j
maxPoolPreparedStatementPerConnectionSize: 20
useGlobalDataSourceStat: true
connectionProperties:
druid.stat.mergeSql=true;druid.stat.slowSqlMillis=500

#MyBatis
mybatis:
# 起别名
type-aliases-package: henu.soft.xiaosi.pojo
mapper-locations: classpath:mybatis/mapper/*.xml

```

3. 书写多表查询的SQL的时候，需要注意插入的参数和pojo、model中的数据对应

```

<mapper namespace="henu.soft.xiaosi.mapper.EmployeeMapper">
  <resultMap id="EmployeeMap" type="Employee">
    <id property="id" column="id"/>
    <result property="lastName" column="lastName"/>
    <result property="email" column="email"/>
    <result property="gender" column="gender" />
    <result property="birth" column="birth"/>
    <!--关联表，需要对应，department实体pojo和employee实体pojo的成员变量-->
    <!-- department 的id 对应着 employee表的外键 departmentId-->
    <association property="department" javaType="Department">
      <id property="id" column="id"/>
      <result property="departmentName" column="departmentName"/>
    </association>
  </resultMap>

  <!-- 查询所有的员工和部门-->
  <select id="getEmployees" resultMap="EmployeeMap">
    SELECT e.id,e.lastName,e.email,e.gender,d.departmentName,e.birth
    FROM employee e,department d
    WHERE e.departmentId = d.id;

```

```
</select>
```

整合log4j阶段

- 导依赖（需要去除Springboot的默认日志记录）
- 创建resources/log4j.properties配置文件
- 配置log4j.properties

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
  <exclusions>
    <exclusion>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-logging</artifactId>
    </exclusion>
  </exclusions>
</dependency>

<dependency>
  <groupId>log4j</groupId>
  <artifactId>log4j</artifactId>
  <version>1.2.17</version>
</dependency>
```

```
# LOG4J配置
log4j.rootCategory=DEBUG, stdout

# 控制台输出
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS} %5p
%c{1}:%L - %m%n
```

2.16bug1:

```
attempted to return null from a method with a primitive return type (int))
```

报错原因：涉及自动装箱拆箱的过程，Integer是int的包装类，int的初值为0，Integer的初值为null；
但是虽然页面报500但是数据库数据确实插入、删除成功了，

- 插入的时候：
 - mapper接口方法返回值为int报错，service层的为int报错
 - mapper接口方法返回值为Integer报错，service层的为Integer不在报错
- 删除的时候：
 - mapper接口方法返回值为Integer,service层的为int，发生空指针异常
 - mapper接口方法返回值为Integer,service层的为Integer，不在报错
 - mapper接口方法返回值为int,service层的为int，不在报错

总之，还是尽量保持一致，网上查阅MyBatis的insert、update、delete实际是返回受影响的行数，这里不知道为什么插入成功但是报错

2.17bug2:

修改时下拉框数据回显异常

- 报错原因：

MyBatis的多表查询，Employee表关联Department

employee表中外键departmentId 和 实体中的department成员变量并不是直接对应的

使用一般的查询springboot并不能完美封装字段，查询某员工信息之后department成员变量为null，因此在SpringEL中出现空指针异常报错

- 解决办法

需要二次封装，再次定义 根据员工id获取部门id和部门名称，查询将department实体封装到employee内部

- 注意的问题
 - XxxMapper.xml文件的编写

```
<?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="henu.soft.xiaosi.mapper.EmployeeMapper">
  <resultMap id="EmployeeMap" type="Employee">
    <id property="id" column="id"/>
    <result property="lastName" column="lastName"/>
    <result property="email" column="email"/>
    <result property="gender" column="gender" />
    <result property="birth" column="birth"/>
  </resultMap>
</mapper>
```

```

        <!--关联表,department为映射的成员变量, Department为另一个pojo-->
        <association property="department" javaType="Department">
            <!--为Department表的字段-->
            <id property="id" column="id"/>
            <result property="departmentName" column="departmentName"/>
        </association>

    </resultMap>

    <!--      查询所有的员工和部门-->
    <select id="getEmployees" resultMap="EmployeeMap">
        SELECT e.id,e.lastName,e.email,e.gender,d.departmentName,e.birth
        FROM employee e,department d
        WHERE e.departmentId = d.id;

    </select>
    <!--      新增一个员工-->
    <select id="save" parameterType="Employee" >
        insert into employee(lastName,email,gender,departmentId,birth)
        values(#{lastName},#{email},#{gender},#{department.id},#{birth});
    </select>
    <!--      查询一个员工-->
    <select id="getEmployeeById" resultType="Employee" parameterType="Integer">
        select * from employee where id = #{id}
    </select>
    <select id="delete" parameterType="int" >
        delete from employee where id = #{id}
    </select>

</mapper>

```

四、第三阶段-整合Shiro、MyBatis、Thymeleaf阶段

2.18逻辑步骤

1. 导入Shiro整合依赖
2. 编写ShiroConfig核心配置类
 - 认证信息：内置过滤器，设置页面权限为登录访问，可以跳转到登录页面（自己写的，不像SpringSecurity内置），然后整合Mybatis从数据库获取信息认证登录功能
 - 授权信息：内置过滤器，设置页面权限为指定参数，可以跳转 未授权提示页面
3. 编写UserRealm 授权、认证 实现逻辑配置类
 - 实现核心授权功能
 - 实现核心认证功能
4. 编写LoginController
 - 开启登录认证,封装登录信息到subject、用户注销功能

```

    ◦   @RequestMapping("/login")
        public String login(String username, String password, Model model){

            //获取当前的用户
            Subject subject = SecurityUtils.getSubject();
            //封装用户登录数据

```

```

//在UserRealm内实现验证
UsernamePasswordToken token = new
UsernamePasswordToken(username, password);
try{
    subject.login(token);
    return "redirect:/main.html";

} catch(UnknownAccountException e){
    model.addAttribute("msg","用户名错误! ");
    return "index";
} catch(IncorrectCredentialsException e){
    model.addAttribute("msg","密码错误! ");
    return "index";
}

```

- 实现登录认证、授权页面的跳转

5. Shiro整合MyBatis、Druid

- 导入依赖
- 配置数据源信息、MyBatis的配置信息
- 认证：从数据库查询数据，使用Shiro完成
- 授权：授予不同登录用户不同的登录权限

6. Thymeleaf整合Shiro

- 实现权限不同的用户显示不同的信息
- 实现登录成功之后不再显示登录按钮

2.19认证、授权、注销

编写ShiroConfig核心配置类、UserRealm用户认证授权核心类、LoginController

- ShiroConfig：固定的套路 + 编写认证、授权的页面

```

package henu.soft.xiaosi.config;

import org.apache.shiro.spring.web.ShiroFilterFactoryBean;
import org.apache.shiro.web.mgt.DefaultWebSecurityManager;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.util.LinkedHashMap;
import java.util.Map;

@Configuration
public class ShiroConfig {

    //3.用户
    @Bean

```



```

public ShiroFilterFactoryBean
getShiroFilterFactoryBean(@Qualifier("securityManager")
DefaultWebSecurityManager defaultWebSecurityManager){
    ShiroFilterFactoryBean bean = new ShiroFilterFactoryBean();
    //设置安全管理器
    bean.setSecurityManager(defaultWebSecurityManager);

    //添加内置过滤器

    Map<String,String> filterMap = new LinkedHashMap<>();

    //设置未登录认证拦截器
    filterMap.put("/emps","authc");
    filterMap.put("/add","authc");
    filterMap.put("/edit/*","authc");
    filterMap.put("/main.html","authc");
    filterMap.put("/getDepartmentById","authc");
    filterMap.put("/getDepartments/*","authc");

    //设置权限拦截器
    filterMap.put("/add","perms[user:add]");
    filterMap.put("/add","perms[user:all]");

    filterMap.put("/add/*","perms[user:add]");
    filterMap.put("/add/*","perms[user:all]");

    filterMap.put("/edit","perms[user:add]");
    filterMap.put("/edit","perms[user:all]");

    filterMap.put("/edit/*","perms[user:add]");
    filterMap.put("/edit/*","perms[user:all]");

    filterMap.put("/delete","perms[user:all]");
    filterMap.put("/delete/*","perms[user:all]");

    //设置登录的请求,转到自己写的认证页面
    bean.setLoginUrl("/toLogin");

    //未授权访问跳转页面, 点击有权限的页面, 会自动指定UserRealm的方法
    bean.setUnauthorizedUrl("/limit");

    bean.setFilterChainDefinitionMap(filterMap);

    return bean;
}

```

//2. 管理

```

@Bean(name = "securityManager")
public DefaultWebSecurityManager
getDefaultWebSecurityManager(@Qualifier("userRealm") UserRealm userRealm){
    DefaultWebSecurityManager securityManager = new
DefaultWebSecurityManager();
    //关联Realm对象
    securityManager.setRealm(userRealm);
}

```

```

        return securityManager;
    }

    //1. 创建Realm对象，用于处理数据
    @Bean(name = "userRealm")
    public UserRealm userRealm(){
        return new UserRealm();
    }
}

```

- UserRealm：认证、授权的核心逻辑实现

```

package henu.soft.xiaosi.config;

import henu.soft.xiaosi.mapper.UserMapper;
import henu.soft.xiaosi.pojo.User;
import org.apache.shiro.SecurityUtils;
import org.apache.shiro.authc.*;
import org.apache.shiro.authz.AuthorizationInfo;
import org.apache.shiro.authz.SimpleAuthorizationInfo;
import org.apache.shiro.realm.AuthorizingRealm;
import org.apache.shiro.subject.PrincipalCollection;
import org.apache.shiro.subject.Subject;
import org.springframework.beans.factory.annotation.Autowired;

public class UserRealm extends AuthorizingRealm {

    @Autowired
    UserMapper userMapper;
    //授权
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection
principalCollection) {
        SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();

        //拿到当前登录的对象
        Subject subject = SecurityUtils.getSubject();
        User currentUser = (User)subject.getPrincipal();
        //System.out.println("debug==>" + currentUser.getLimits());

        //拿出来user表中limits字段
        info.addStringPermission(currentUser.getLimits());
        return info;
    }

    //认证
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken
authenticationToken) throws AuthenticationException {
        //      String username = "sc1";
        //      String password = "123456";
    }
}

```

```

        //刚才的token是全局的，可以在这里取出认证
        UsernamePasswordToken token = (UsernamePasswordToken)
authenticationToken;

        User user = userMapper.findLoginUserByName(token.getUsername());

        if(user == null){
            //用户名认证，用户名不同会自动抛出用户不存在异常
            return null;
        }

        System.out.println("认证方法执行了debug==>" + user.getLimits());
        //密码认证Shiro自己做，如果还需要设置权限授予，需要将user参数传递给
SimpleAuthenticationInfo
        return new SimpleAuthenticationInfo(user,user.getPassword(),"");

    }
}

```

- LoginController: 配合Shiro实现用户数据封装登录、注销功能

```

package henu.soft.xiaosi.controller;

import javax.servlet.http.HttpSession;

import org.apache.shiro.SecurityUtils;
import org.apache.shiro.authc.IncorrectCredentialsException;
import org.apache.shiro.authc.UnknownAccountException;
import org.apache.shiro.authc.UsernamePasswordToken;
import org.apache.shiro.subject.Subject;
import org.springframework.boot.Banner;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;

import org.thymeleaf.util.StringUtils;

@Controller
public class LoginController {
    @RequestMapping("/toLogin")
    public String toLogin(){
        return "index";
    }

    @RequestMapping("/login")
    public String login(String username, String password, Model model){

        //获取当前的用户
        Subject subject = SecurityUtils.getSubject();
    }
}

```

```

        //封装用户登录数据
        //在UserRelam内实现验证
        UsernamePasswordToken token = new UsernamePasswordToken(username,
password);
        try{
            subject.login(token);
            return "redirect:/main.html";

        } catch(UnknownAccountException e){
            model.addAttribute("msg", "用户名错误! ");
            return "index";
        } catch(IncorrectCredentialsException e){
            model.addAttribute("msg", "密码错误! ");
            return "index";
        }
    }

}

/*
@RequestMapping("/user/login")
public String login(
    @RequestParam("username") String username,
    @RequestParam("password") String password,
    Model model, HttpSession session){
    //具体的业务
    if(!StringUtils.isEmpty(username) && "123456".equals(password)){

        //设置登录成功的Session
        session.setAttribute("loginUser",username);
        return "redirect:/main.html";
    }
    else{
        model.addAttribute("msg", "用户名或者密码错误!");
        return "index";
    }
}

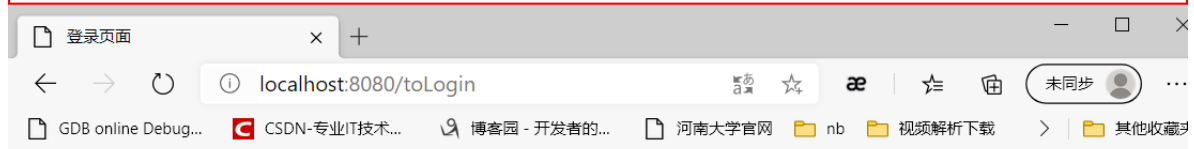
}
*/
@RequestMapping("/logout")
public String logout(){
    //注销功能
    Subject subject = SecurityUtils.getSubject();
    subject.logout();
    return "redirect:/index.html";
}
}

```

2021/1/18 10:34:36

认证

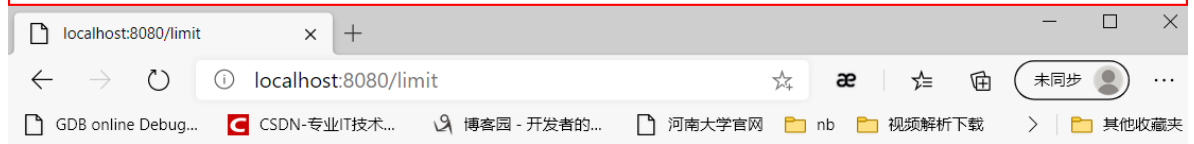
未登录的情况在地址栏输入被拦截跳转到登录页面



2021/1/18 10:33:02

Shiro授权

用户无权限跳转指定页面

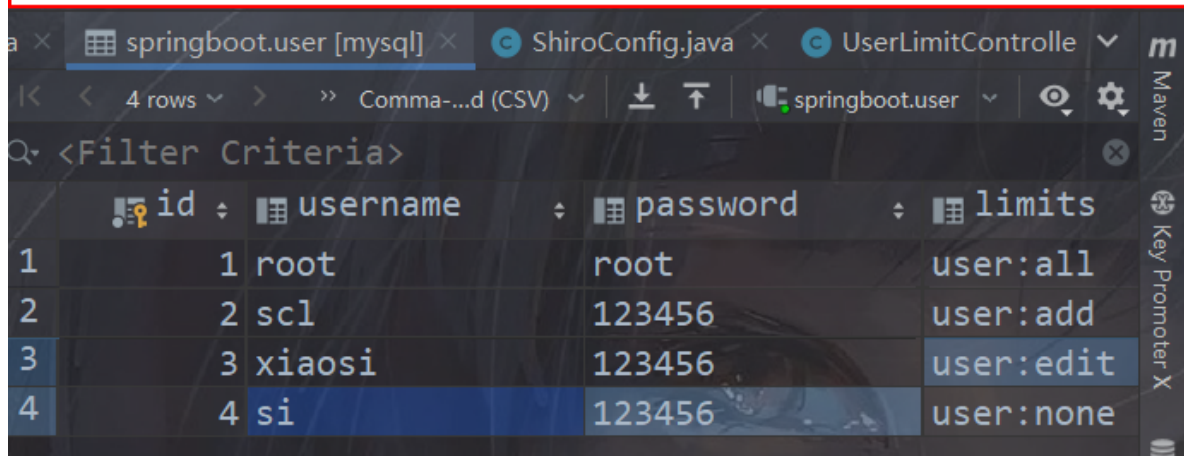


权限不足，请联系管理员获取权限！

2021/1/18 10:33:32

授权

不同用户的权限不同



The screenshot shows a MySQL database table with the following data:

| | id | username | password | limits |
|---|----|----------|----------|-----------|
| 1 | 1 | root | root | user:all |
| 2 | 2 | scl | 123456 | user:add |
| 3 | 3 | xiaosi | 123456 | user:edit |
| 4 | 4 | si | 123456 | user:none |

2.20不同权限不同显示

逻辑步骤

- 导依赖
- ShiroConfig创建ShiroDialect方法, 加上 @Bean (注册组件)
- 测试
 - 可以直接取出当前用户显示用户名

```
shiro,你好,<shiro:principal property="username"/>
```

- 可以根据用户的权限和角色来显示不同的资源

```
<!-- 验证当前用户是否拥有 update 权限 -->
<a shiro:hasPermission="update" href="createUser.html">添加用户</a>
<!-- 验证当前用户是否拥有以下任意一个权限 -->
<p shiro:hasAnyPermissions="add, update, delete">
    You can see or delete users.
</p>
```

实现配置

- 依赖

```
<dependency>

    <groupId>com.github.theborakompanioni</groupId>

    <artifactId>thymeleaf-extras-shiro</artifactId>

    <version>2.0.0</version>

</dependency>
```

- 命名空间

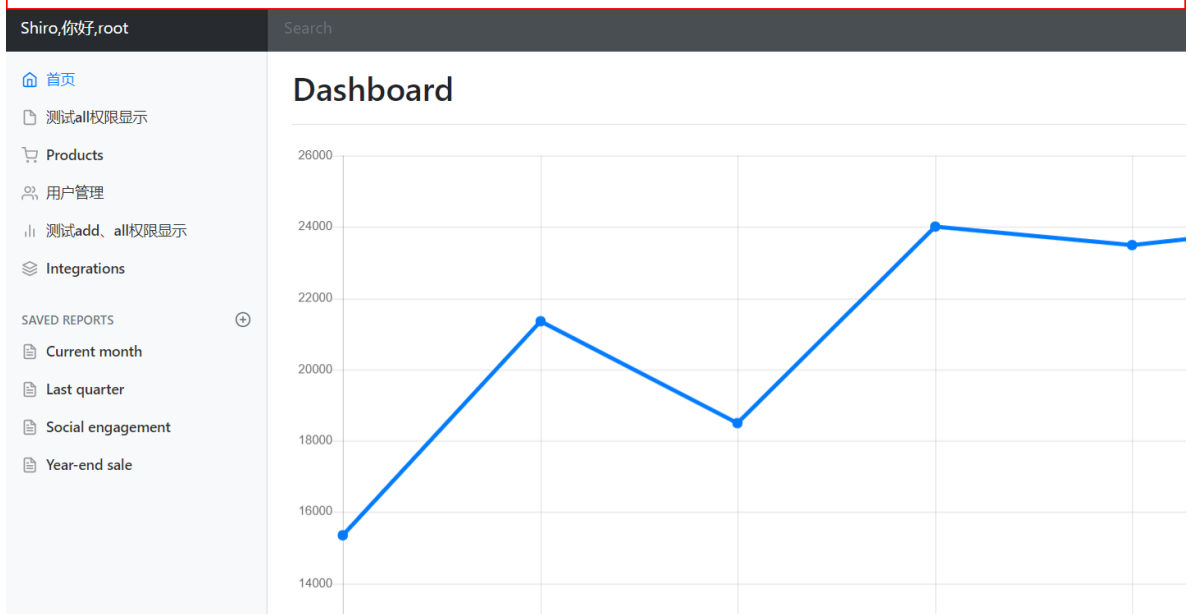
```
xmlns:shiro="http://www.pollix.at/thymeleaf/shiro">
```

- ShiroConfig下的ShiroDialect组件

```
/**
 * Shiro整合Thymeleaf
 *
 * 配置 ShiroDialect (Shiro 方言) 对象
 *
 */
@Bean
public ShiroDialect getShiroDialect(){
    return new ShiroDialect();
}
```

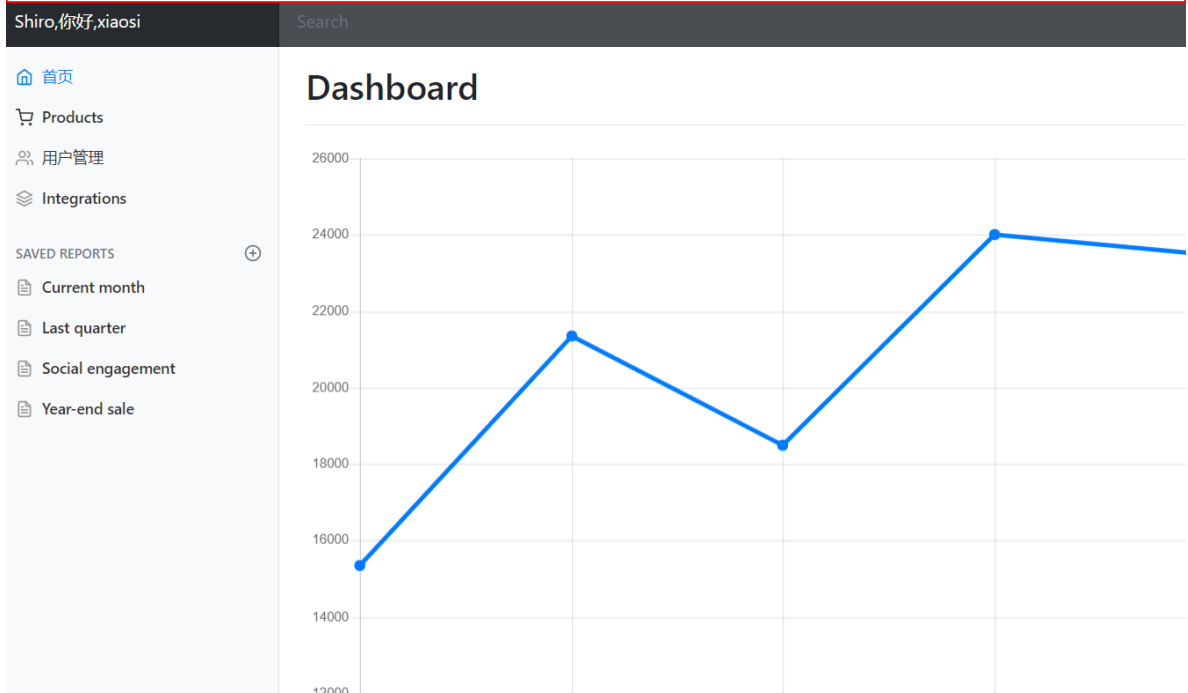
效果

2021/1/18 12:07:34
Thymeleaf整合Shiro
不权限的用户显示不同资源



2021/1/18 12:08:34

不同用户显示不同的资源



2021/1/18 12:08:10

不同用户显示不同的资源

