SSM_CRUD

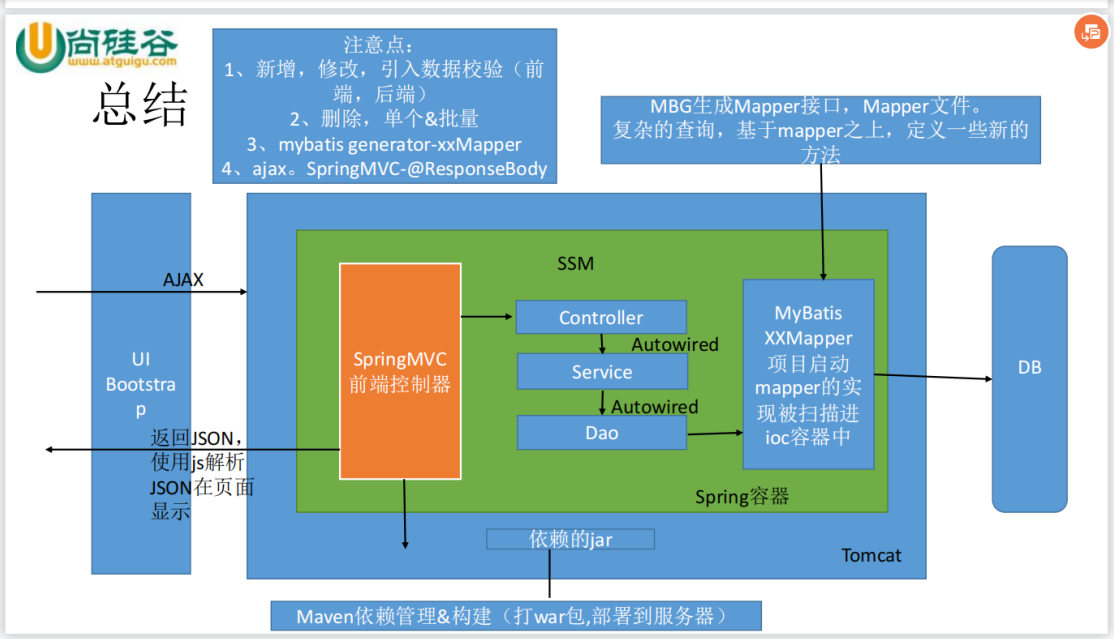
笔记本:	Java_尚硅谷		
创建时间:	2023/2/2 9:08	**更新时间:**	2023/2/5 22:07
作者:	M33		
URL:	about:blank		

功能点

- 1、分页
- 2、数据校验： jquery前端校验+JSR303后端校验
- 3、ajax（采用封装了ajax的axios）
- 4、Rest风格的URI；使用HTTP协议请求方式的动词，来表示对资源的操作（GET（查询），POST（新增），PUT（修改），DELETE（删除））

技术点

- 基础框架-ssm（SpringMVC+Spring+MyBatis）
- 数据库-MySQL
- 前端框架-bootstrap快速搭建简洁美观的界面 vue
- 项目的依赖管理-Maven
- 分页-pagehelper
- 逆向工程-MyBatis Generator



- **创建工程**
- **导入常用依赖**

  spring
  springmvc
  mybatis
  thymeleaf
  spring注解
  junit

数据库连接池，驱动包
插件，如逆向工程、分页插件等

- **引入bootstrap前端框架**(快速搭建前端，官网里下载jar导入到static文件夹中，html的head中引入，引入格式参考官网)：

```
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css"
integrity="sha384-
HSMxcRTRxnN+Bdg0JdbxYKrThecOKuH5zCYotlSAcp1+c8xmyTe9GYg1l9a69psu"
crossorigin="anonymous">
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"
integrity="sha384-
aJ21OjlMXNL5UyIl/XNwTMqvzeRMZH2w8c5cRVpzpU8Y5bApTppSuUkhZXN0VxHd"
crossorigin="anonymous"></script>
```

- **编写ssm整合的关键配置文件**（从模板中复制，再微调）

web.xml

```
<!--配置Spring的编码过滤器CharacterEncodingFilter-->
    解决获取请求参数的乱码问题，可以使用SpringMVC提供的编码过滤器
CharacterEncodingFilter
<!--配置处理请求方式的过滤器HiddenHttpMethodFilter-->
    常用的请求方式有get，post，put，delete
    但是目前浏览器只支持get和post，若在form表单提交时，为method设置了其他请求方式的字符
串（put或delete），则按照默认的请求方式get处理
    若要发送put和delete请求，则需要通过spring提供的过滤器HiddenHttpMethodFilter-->
<!--配置SpringMVC的前端控制器DispatcherServlet-->
    设置SpringMVC配置文件自定义的位置和名称
    将DispatcherServlet的初始化时间提前到服务器启动时
    设置springMVC的核心控制器所能处理的请求的请求路径
<!--配置Spring的监听器ContextLoaderListener，在服务器启动时就加载Spring的配置文件-->
    设置Spring配置文件自定义的位置和名称
```

spring.xml

```
<!--扫描组件component-scan（除控制层）-->
    才能识别@Service、@Repository等的注解
<!--引入jdbc.properties,配置数据源-->
    取出jdbc中的数据，用来连接数据库
<!--配置SqlSessionFactoryBean-->
    才可以直接在Spring的IOC中获取SqlSessionFactory
<!--配置mapper接口的扫描-->
    可以将指定包下所有的mapper接口,通过SqlSession创建代理实现类对象，并将这些对象交给
IOC容器管理
```

springmvc.xml

```
<!--扫描组件component-scan（除控制层）-->
    才能识别@controller的注解
<!--配置视图解析器ThymeleafViewResolver-->
<!--配置默认的servlet处理静态资源-->
<!--开启MVC注解驱动启动-->
    先用DispatcherServlet处理，处理不了的用默认servlet
<!--配置视图控制器，设置启动后跳转到index-->

Spring MVC 的工作流程：
1）客户端请求提交到调度器DispatcherServlet;
2）由DispatcherServlet调度器解析URL，调用URL相对应的映射处理器HandlerMapping
3）DispatcherServlet将请求提交到控制器Controller;
4）Controller调用业务逻辑（service层和dao层）处理后，向DispatcherServlet返回
```

mybatis-config.xml

```
<!--将下划线映射为驼峰-->
    若数据库字段名和实体类中的属性名不一致，但是字段名符合数据库的规则（使用_），实体类中
的属性名符合Java的规则（使用驼峰）则可设定转化
<!--配置分页插件-->
```

- **使用mybatis的逆向工程生成对应的bean以及mapper：**

  1.pom.xml中添加依赖和插件(如果之前没添加的话)
  2.创建逆向工程的配置文件（直接复制模板就行）　文件名必须是：**generatorConfig.xml**
  3. 执行MBG插件的generate目标
  出现问题：逆向工程产生很多.java.class文件，解决方法：多加一行

```
<!-- 数据库的连接信息 -->
<jdbcConnection driverClass="com.mysql.cj.jdbc.Driver"
                connectionURL="jdbc:mysql://localhost:3306/ssm_crud?serverTimezone=UTC"
                userId="root"
                password="756954">
    <property name="nullCatalogMeansCurrent" value="true"/>
</jdbcConnection>
```

- **生成逆向工程后，修改一些sql语句，采用联表查询，因为部门名字也要显示出来**

  在**Empt.class**里面加一个属性为Dept
  在**EmpMapper.class**里面加两个方法：

```
List<Emp> selectByExampleWithDept(EmpExample example);
Emp selectByPrimaryKeyWithDept(Integer empId);
```

  在**EmpMapper.xml**里面新增**结果集**：

```
<resultMap id="WithDeptResultMap" type="bean.Emp" >
  <id column="emp_id" property="empId" jdbcType="INTEGER" />
  <result column="emp_name" property="empName" jdbcType="VARCHAR" />
  <result column="gender" property="gender" jdbcType="CHAR" />
  <result column="email" property="email" jdbcType="VARCHAR" />
  <result column="dept_id" property="deptId" jdbcType="INTEGER" />
<!--多对一映射处理，使用association处理映射关系-->
  <association property="dept" javaType="Dept">
    <id column="dept_id" property="deptId"></id>
    <result column="dept_name" property="deptName"></result>
  </association>
</resultMap>
```

  在**EmpMapper.xml**里面**新增查询列表**：

```
<sql id="WithDept_Column_List" >
  e.emp_id, e.emp_name, e.gender, e.email, e.dept_id, d.dept_id, d.dept_name
</sql>
```

  在**EmpMapper.xml**里面**新增查询方法**：

```
<select id="selectByExampleWithDept" resultMap="WithDeptResultMap"
parameterType="bean.EmpExample" >
  select
  <if test="distinct" >
    distinct
  </if>
  <include refid="WithDept_Column_List" />
  FROM t_emp e
  LEFT JOIN t_dept d
  ON e.dept_id=d.`dept_id`
  <if test="_parameter != null" >
    <include refid="Example_Where_Clause" />
  </if>
  <if test="orderByClause != null" >
    order by ${orderByClause}
  </if>
</select>
```

```
<select id="selectByPrimaryKeyWithDept" resultMap="WithDeptResultMap"
parameterType="java.lang.Integer" >
  select
  <include refid="WithDept_Column_List" />
  FROM t_emp e
  LEFT JOIN t_dept d
  ON e.dept_id=d.`dept_id`
  where e.emp_id = #{empId,jdbcType=INTEGER}
</select>
```

- **测试增删改查功能**

  导入依赖，注意各个版本号要一致，不然会报错

```
test中
//*1、@RunWith(SpringJUnit4ClassRunner.class)导入SpringTest模块
// *2、@ContextConfiguration指定Spring配置文件的位置
// *3、@autowired要使用的组件即可 自动装配
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:spring.xml")
public class TestCrud {
    @Autowired
    private DeptMapper deptMapper;
    @Test
    public void TestCrud1(){
        测试增删改查
    }
}
```

- **增加批量删除的功能（最后发现根本没用这个方法）**

  EmpMapper接口中定义方法，在EmpMapper.xml中写方法

```
int deleteMultiByPrimaryKey(@Param("empIds") String empIds);//可以通过@Param注解标识mapper接口中的方法参数 此时，会将这些参数放在map集合中，以@Param注解的value属性值为键，以参数为值；
```

```
<delete id="deleteMultiByPrimaryKey" parameterType="java.lang.Integer" >
  delete from t_emp
  where emp_id in (${empIds}) //MyBatis获取参数值的两种方式：${}和#{}，${}的本质就是字符串拼接，#{}的本质就是占位符赋值
</delete>
```

- **增加查询所有员工的功能**

  EmpMapper接口中定义方法，在EmpMapper.xml中写方法

```
List<Emp> getAllEmployee();
```

```
<select id="getAllEmployee" resultMap="WithDeptResultMap"
parameterType="java.lang.Integer" >
  select
  <include refid="WithDept_Column_List" />
  FROM t_emp e
  LEFT JOIN t_dept d
  ON e.dept_id=d.`dept_id`
</select>
```

正式开始：前后端分离改造版本vue+axios（axios实现了对ajax的封装）

（即前端与后端通过json传递数据，后端给前端json数据后前端渲染出来，而不是后端渲染完返回渲染好的页面）

- **bootstrap快速搭建前端**
- **设置分页插件，获得分页数据，展示分页信息**

通过getPageInfo(pageNum)获取分页信息，保存到vue.pageInfo中，然后页面中通过{{}}调用

默认一启动打开第一页

- **分页栏制作**

利用vue.pageInfo中的信息制作

首页：getPageInfo(1) 如果当前页为首页则禁止按

上一页：存在上一页则允许按

导航栏页码：当前页突出显示

下一页：存在下一页则允许按

尾页：getPageInfo(pages)，如果当前页为尾页则禁止按

```
PageInfo{
pageNum=8, pageSize=4, size=2, startRow=29, endRow=30, total=30, pages=8,
list=Page{count=true, pageNum=8, pageSize=4, startRow=28, endRow=32, total=30,
pages=8, reasonable=false, pageSizeZero=false},
prePage=7, nextPage=0, isFirstPage=false, isLastPage=true, hasPreviousPage=true,
hasNextPage=false, navigatePages=5, navigateFirstPage4, navigateLastPage8,
navigatepageNums=[4, 5, 6, 7, 8]
}
pageNum：当前页的页码
pageSize：每页显示的条数
size：当前页显示的真实条数
total：总记录数
pages：总页数
prePage：上一页的页码
nextPage：下一页的页码
isFirstPage/isLastPage：是否为第一页/最后一页
hasPreviousPage/hasNextPage：是否存在上一页/下一页
navigatePages：导航分页的页码数
navigatepageNums：导航分页的页码，[1,2,3,4,5]
```

- **成功展示员工信息后，完成增加员工的模块框**

1.其中部门是下拉选项，要从数据库中查询部门列表，json发给前端，再渲染到页面

2.**addModal新增按钮的逻辑：清空原本新增模态框中的数据及提示信息—打开模态框**

3.vue中用emp来存放新增模态框中的数据，要与模态框中各位置双向绑定好

4.输入完数据后，要进行**新增信息的校验JSR303**：

名字和邮箱是否长度、内容有效?

名字要在数据库中搜索，有无重复?

5.采用**前端校验+后端程序校验**（一般还会有**数据库约束**）

前端校验：判断符不符合各字段长度等要求

前端校验validate_form逻辑：取出需要校验的输入框的内容，校验，显示提示信息

后端校验名字checkEmpName逻辑：取出名字，发送axios请求，在后端判断名字符不符合字段要求以及有无重复用户名，返回结果，标记提示信息，并给给保存按钮加属性（方便后续判断）

保存新增的员工信息saveEmp逻辑：前端校验是否通过？后端检验名字是否通过？不通过不保存数据到后台；

发送axios请求，后端传回 保存数据成功与否 的信息（后端JSR303检验：在bean上标注规则，在方法中@Valid检验，结果在BindingResult中）

保存成功则关闭模态框，跳到最后一页，保存失败则回显错误提示信息

- **完成员工信息修改的模态框**

编辑按键逻辑：传入点击的empId，发送axios请求，获得该emp的数据，打开模态框显示数据（数据绑定好）

信息修改模态框：名字锁定不被更改，其它可以改

信息修改模态框中保存按键：邮箱前端校验是否通过？将数据封装好，传入后端，后端JSR303校验，校验通过则关闭模态框，跳转到当前页，失败则回显错误信息

- **完成单选框，全选框功能**

单选框逻辑：点击后判断选中的单选框数量等不等于单选框数量，等于则全选框被选中，不等于则全选框不能被选中

全选框逻辑：点击后判断当前全选框是否为选中状态，是则所有单选框要被选中

- **完成单条信息的删除功能**

删除键deleteEmp逻辑：点击后获得empName，提示确认是否删除empName，传到后台，删除后提示成功并跳转到当前页

- **完成多条信息的删除功能**

删除键deleteEmpBatch逻辑：点击后获得被选中的员工，名字拼接在一起变成字符串empName，每个员工名字逗号分隔开，提示确认是否删除empName，传到后台。

后台根据句号分割开每个名字，存到List中，创建删除的条件，调用empMapper.deleteByExample(empExample);完成删除。

删除后清除单选框状态，提示成功并跳转到当前页