SpringDemo项目

一、项目说明

1.项目预览

2.项目的功能点

- 分页
- 数据校验
- ajax异步请求
- 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））

3.项目技术点

- 1.基础框架 ssm(spring+springmvc+mybatis)
- 2.数据库 Oracle
- 3.项目的依赖管理管理 maven
- 4.分页 pagehelper
- 5.log4j2日志

二、基础环境的搭建

1.创建maven工程

- 在idea中创建maven项目，选Apache的webapp

2.引入项目依赖的jar包

在pom.xml文件中导入以下jar包的依赖点击进入maven仓库中央仓库

- spring （Spring JDBC（事务控制）；Spring Aspects（切面编程））
- springmvc（Spring Web MVC）
- mybatis （Mybatis；Mybatis Spring（整合包））
- 数据库连接池（dbcp）,驱动包（Oracle Connect）
- 其他(Jstl Servlet Api junit)

3.引入bootstrap，一个前端的框架

- 1.去bootstrap官网下载
- 2.在webapp下新建一个static文件夹，把bootstrap的文件放进去
- bootstrap的用法，参照官方文档
- 引入bootstrap之前一点要先引入jQuery（在webapp下新建一个js文件夹，去放jQuery）
- 引入的方式

```
<!--Bootstrap和jQuery的引入-->
<link href="${APP_PATH}/static/bootstrap-3.3.7-dist/css/bootstrap.min.css" rel="stylesheet">
<link href="${APP_PATH}/static/bootstrap-3.3.7-dist/css/bootstrap-datetimepicker.min.css" rel="stylesheet">
<!--Bootstrap依赖jQuery,所以要把jQuery放前面,以免Bootstrap在jQuery使用时没法使，有时候模态框一直没法调用就是因为这个问题-->
<script type="text/javascript" src="${APP_PATH}/static/js/jquery-1.12.4.min.js"></script>
<script src="${APP_PATH}/static/bootstrap-3.3.7-dist/js/bootstrap.min.js"></script>
<script src="${APP_PATH}/static/bootstrap-3.3.7-dist/js/bootstrap-datetimepicker.min.js"></script>
<script src="${APP_PATH}/static/bootstrap-3.3.7-dist/js/bootstrap-datetimepicker.zh-CN.js"></script>
```

三、配置文件的编写

1.web.xml的配置

（1）启动spring容器的监听器（项目一起动，spring的容器就启动）

- 配置监听器：项目一启动，就会加载spring的配置配置文件（指定spring配置文件的位置）

xml

```
 <listener>
        <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:spring.xml</param-value>
    </context-param>>
```

- **spring的配置文件：主要配合和业务逻辑有关的**

(2) 配置前端控制器

- 用来拦截所有的请求
- 指定springmvc配置的位置

xml

```
<!--2、SpringMVC的前端控制器，拦截所有请求-->
    <servlet>
        <servlet-name>spring-mvc</servlet-name>
        <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:spring-mvc.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>spring-mvc</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
```

(3) 配置字符编码过滤器（防止乱码）

- 代码

xml

```
<!-- 字符编码过滤，解决请求中文乱码问题 -->
    <filter>
        <filter-name>characterEncodingFilter</filter-name>
        <filter-
class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
        <init-param>
            <param-name>encoding</param-name>
            <param-value>UTF-8</param-value>
        </init-param>
        <init-param>
            <param-name>forceResponseEncoding</param-name>
            <param-value>true</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>characterEncodingFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

(4) 配置支持Rest风格URI的过滤器

- 因为页面无法支持delete、put请求，还需要一个过滤器
- 代码

xml

```
<!--使用Rest风格的URI-->
<!-- HiddenHttpMethodFilter:将普通的get/post请求转化为put/delete请求 -->
<filter>
        <filter-name>HiddenHttpMethodFilter</filter-name>
        <filter-
class>org.springframework.web.filter.HiddenHttpMethodFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>HiddenHttpMethodFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

(5) 配置支持put请求的过滤器

- 因为tomcat在得到put请求的时候不会去处理，

- 配置过滤器会自动的封装前台传递过来的PUT请求的参数，如果不配置HttpPutFormContentFilter过滤器的话，那么tomcat便不会自主的分封装PUT请求的参数。
- ```
  <!-- 解决无法直接发送PUT请求，带着请求体的问题，在修改员工信息的时候使用 -->
      <filter>
          <filter-name>HttpPutFormContentFilter</filter-name>
          <filter-
  class>org.springframework.web.filter.HttpPutFormContentFilter</filter-class>
      </filter>
      <filter-mapping>
          <filter-name>HttpPutFormContentFilter</filter-name>
          <url-pattern>/*</url-pattern>
      </filter-mapping>
  ```

2.spring-mvc.xml的配置

- 主要包含网站跳转逻辑的组件

（1）扫描所有的业务逻辑组件

- 在java.com.wantao包下创建bean、controller、service、dao、utils、test包
- 我们希望只扫描controller（控制器）

xml

```
<!--springmvc的配置文件，包含网站跳转逻辑控制，配置 -->
<!-- 配置扫描controller下边的控制器 -->
<context:component-scan base-package="com.spring.controller">
</context:component-scan>
<!--     启动注解驱动的的spring mvc-->
 <mvc:annotation-driven></mvc:annotation-driven>
```

（2）配置视图解析器

- 方便页面的解析
- 在WEB-INF下建一个view包

xml

```
<!--     解析模型视图,视图解析器-->
    <bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/" />       <!-- 前缀 -->
        <property name="suffix" value=".jsp" />   <!-- 后缀 -->
    </bean>
```
```

(3) 标准配置

- 代码

  xml

```
<!-- 将springmvc不能处理的请求交给tomcat -->
<mvc:default-servlet-handler />
```

3.spring.xml的配置

- 主要是配置和逻辑有关的
- Spring配置文件的核心点：（数据源、与mybatis的整合、事务控制）

(1) 配置数据源（dbcp)

- 代码

  xml

```
 <!--加载jdbc.properties-->
    <context:property-placeholder location="classpath:jdbc.properties">
</context:property-placeholder>
  <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName" value="${driver}"></property>
        <property name="url" value="${url}"></property>
        <property name="username" value="${user}"></property>
        <property name="password" value="${password}"></property>
    </bean>
```

- 也可以把连接信息写在jdbc.properties文件中

(2) 和mybatis的整合配置

- 配置SqlSessionFactoryBean，他能创建SqlSessionFactory

  xml

```
<!--=================配置spring和mabatis的整合=================== -->
<!--创建mybatis的工厂类对象-->
    <bean class="org.mybatis.spring.SqlSessionFactoryBean">
        <!--指定数据源-->
        <property name="dataSource" ref="dataSource"></property>
        <!--加载mybatis的映射文件 在value中可以使用*号通配符-->
        <property name="mapperLocations"
value="classpath:com/spring/mapper/*.xml"></property>
        <!--加载mybatis中的配置文件-->
        <property name="configLocation" value="classpath:mybatis.xml">
</property>
    </bean>
    <!--在spring 的工厂中生成mapper接口的实现类对象 -->
    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <!--指定要扫描哪个包下面所有的mapper接口，加入到IOC容器中-->
        <property name="basePackage" value="com.spring.mapper"></property>
    </bean>
```

```
</bean>
<!--=====================================-->
```

- **注意：在spring和mybatis整合的时候，并不是一定要把mapper的映射文件放到和dao层的文件一个路径，也可以在sqlSessionFactory中配置映射文件存放的位置；在MapperScannerConfiger中配置要扫描的dao层**

(3) 事务控制的配置

- 要能管事务，就是控制住数据源

  xml

```
<!--创建spring的事物管理器-->
    <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
      <!-- 控制数据源 -->
        <property name="dataSource" ref="dataSource"></property>
    </bean>
    <!-第七步，声明以注解的方式配置声明式事物-->
    <tx:annotation-driven transaction-manager="transactionManager" >
</tx:annotation-driven>
```

- 1.开启注解的事务，2.使用xml配置形式的事务（比较重要的都是采用配置式）
  - 本次并没有用到AOP，所以切点的事务控制并未用到

xml

```
 <!--开启注解事务或者使用xml配置形势的事务（主要的都是配置式）-->
 <aop:config>
     <!--切点表达式 -->
     <aop:pointcut expression="execution(* com.wantao.service..*(..))"
id="txPointCut" />
     <!--配置事物增强 -->
     <aop:advisor advice-ref="txAdvice" pointcut-ref="txPointCut" />
 </aop:config>
 <!--配置事物增强,事务如何切入-->
 <tx:advice id="txAdvice" transaction-manager="transactionManager">
     <tx:attributes>
     <!--代表所有方法都是事务方法-->
     <tx:method name="*" />
     <!--以find开头的所有方法-->
     <tx:method name="find*" read-only="true" />
     <tx:method name="get*" read-only="true" />
     </tx:attributes>
 </tx:advice>
```

4.mybatis.xml（全局配置文件+mapper文件）

全局配置文件

- **注意：这个全局配置文件不是必须的，也可以放在放在applicationContext.xml中，以property的形势放在class为SqlSessionFactoryBeanbean中。**
- 搜索mybatis的官方文档（点击进入）

- 找到表头，复制进去

xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-config.dtd">
```

(1) 配置时间拦截器和pageHelper分页

- 代码

xml

```
<plugins>
        <!-- 拦截器配置 -->
      <plugin interceptor="com.spring.util.SqlCostInterceptor" />
        <!-- 配置分页插件 -->
        <plugin interceptor="com.github.pagehelper.PageInterceptor">
            <!--pageInfo合理化分页参数-->
            <property name="reasonable" value="true"/>
        </plugin>
    </plugins>
```

(2) 配置log4j2日志

- mybatis配置log4j2的打印方式
- 添加log4j2的xml文件
- 代码

xml

```
<!-- mybatis 文件 -->
<!-- 使用log4j2打印查询语句 -->
    <settings>
        <setting name="logImpl" value="LOG4J2" />
    </settings>

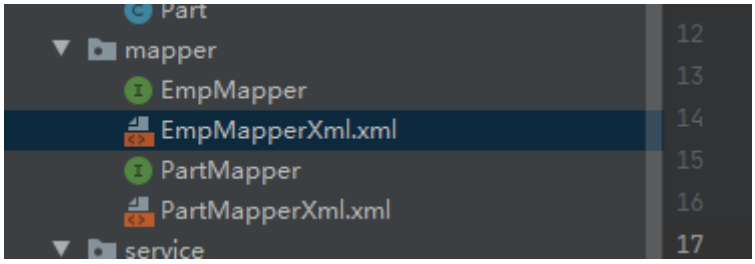
<!-- log4j2.xml 文件-->
    <Configuration status="WARN" monitorInterval="1800">
    <appenders>
        <Console name="consolePrint" target="SYSTEM_OUT">
            <PatternLayout charset="UTF-8" pattern="%d{HH:mm:ss} [%t]
%-5level %logger{36} - %msg%n" />
        </Console>
    </appenders>

    <loggers>
        <!-- 将业务dao接口填写进去,并用控制台输出即可 -->
        <logger name="com.spring.mapper" level="TRACE" additivity="false">
            <appender-ref ref="consolePrint"/>
        </logger>
<!--如果level写成TRACE则控制台会打印sql语句和查询的结果集-->
        <root level="DEBUG">
```

```
            <appender-ref ref="consolePrint" />
        </root>
    </loggers>
```

四、编写所需要的接口和类

(一)、创建mapper接口



mapper接口的方法主要是对应mapper.xml文件中的增删操作的id，代码如下

```
/**
     * 查询所有用户信息
     */
    public List<Emp> findAll();
    /**
     * 根据Id查询
     */
    public Emp findById(int empno);
    /**
     * 根据Id删除员工信息
     */
    public int deleteEmp( int empno);
    /**
     * 方法描述：根据ids集合进行删除
     int deleteByIds(List<Integer> ids);
```

(二)、创建对应的mapper.xml文件

```
 <!-- mapper.xml文件-->
     <select id="findAll" resultType="com.spring.enity.Emp">
        select EMP.*,PART.DEPTNAME from EMP left join PART on EMP.DEPTNO=
PART.DEPTID order by EMP.ID ASC
    </select>
    <select id="findById" resultType="com.spring.enity.Emp">
        select * from emp where id = #{id}
    </select>
    <delete id="deleteEmp" parameterType="int">
        delete from emp where id = #{id}
    </delete>
    <!--批量删除-->
     <delete id="deleteByIds" parameterType="list">
        delete from emp where id IN
```

```
        <foreach collection="list" item="id" open="(" separator="," close=")">
            #{id}
        </foreach>
    </delete>
```

五、完成查询逻辑

0.分页查询

1. 在controller层中传入要查询的页数，默认值为1

- 在controller层中传入要查询的页数，默认值为1

java

```
/**
     * 需要导入json包
     * 分页数据
     * @param pn
     * @return
     */
    @RequestMapping(value = "findemp", method = RequestMethod.GET)
    @ResponseBody
    public Msg getEmpsWithJson(
            @RequestParam(value = "pn", defaultValue = "1") Integer pn) {
        //这是一个分页查询
        //引入PageHelper分页插件
        //在查询之前只需要调用，传入页码，以及每页的大小
        PageHelper.startPage(pn, 5);
        //startPage后面紧跟的这个查询就是一个分页查询
        List<Emp> emp = emps.findAll();
        //使用PageInfo查询包装后的结果，只需要把PageInfo交给页面就行
        //封装了详细的分页信息，包括有我们查询出来的数据,传入连续显示的页数
        PageInfo page = new PageInfo(emp, 5);
        return Msg.success().add("pageInfo", page);
    }
```

2. 引入PageHelper分页插件

- 搜索pagehelper插件（GitHub）-》中文文档

java

```
//获取第1页，10条内容，默认查询总数count
PageHelper.startPage(1, 10);
List<Country> list = countryMapper.selectAll();
//用PageInfo对结果进行包装
PageInfo page = new PageInfo(list);
//测试PageInfo全部属性
```

```
//PageInfo包含了非常全面的分页属性
assertEquals(1, page.getPageNum());//当前页码
assertEquals(10, page.getPageSize());//每页有多少条记录
assertEquals(1, page.getStartRow());//开始的记录
assertEquals(10, page.getEndRow());//结束的记录
assertEquals(183, page.getTotal());//总记录数
assertEquals(19, page.getPages());//总页码数
assertEquals(1, page.getFirstPage());//第一页
assertEquals(8, page.getLastPage());//最后一页
assertEquals(true, page.isFirstPage());//是否第一页
assertEquals(false, page.isLastPage());//是否最后一页
assertEquals(false, page.isHasPreviousPage());//是否有前一页
assertEquals(true, page.isHasNextPage());//是否有后一页
```

- 用PageInfo包装查出来的数据，查询出来的数据写到Model中
- 在pom.xml文件中引入相应的依赖

xml

```
<!--分页插件-->
    <dependency>
      <groupId>com.github.pagehelper</groupId>
      <artifactId>pagehelper</artifactId>
      <version>5.1.4</version>
    </dependency>
```

- 在mybatis全局配置中注册这个插件（上边已经写了）

3.写单元测试（用spring的单元测试），测试利用分页插件能否取到值

- 1.在test包下，创建一个MVCTest的测试类
- 2.spring4测试的时候，需要servlet3.0的支持,导包的时候，需导入3.0以上的

java

```
@RunWith(SpringJUnit4ClassRunner.class)
@WebAppConfiguration
@ContextConfiguration(locations = {"classpath:spring.xml","classpath:spring-mvc.xml"})
public class MvcTest {
    //传入Springmvc的ioc
    @Autowired
    WebApplicationContext context;
    //虚拟mvc请求，获取到处理结果
    MockMvc mockMvc;

    @Before
    public void initMockMvc(){
        mockMvc = MockMvcBuilders.webAppContextSetup(context).build();
    }
    @Test
    public void testPage() throws Exception {
```

```
        //模拟请求拿到返回值
        MvcResult result =
mockMvc.perform(MockMvcRequestBuilders.get("/findemp").param("pn","1"))
                .andReturn();
        //请求成功以后，请求域中会有pageInfo；我们可以取出PageInfo进行验证
        MockHttpServletRequest request = result.getRequest();
        PageInfo pi = (PageInfo) request.getAttribute("pageInfo");
        System.out.println("当前页码："+pi.getPageNum());
        System.out.println("总页码:" + pi.getPages());
        System.out.println("总记录数:" + pi.getTotal());
        System.out.println("页面需要连续显示的页码:");
        int[] nums = pi.getNavigatepageNums();
        for(int i : nums ){
            System.out.println(""+ i);
        }

        //获取员工数据
        List<Emp> list = pi.getList();
        for(Emp emp : list ){
            System.out.println("ID:" +
emp.getId()+"==>Name:"+emp.getEname());
        }
    }
```

3.完成查询界面构建（bootstrap）

- 逻辑：先来到index.sjp首页，首页会直接转到的emps请求，controller收到请求后，查询到所有的数据和分页信息后，然后来到show.jsp页面，所以我们在show.jsp页面展示员工列表就行了。

1.引入bootstrap

- 开头代码

html

```
<%//存入当前项目的名称找
pageContext.setAttribute("APP_PATH",request.getContextPath());
%>
<link rel="stylesheet"
href="${APP_PATH}/static/bootstrap-3.3.7-dist/css/bootstrap.min.css"></link>
<script type="text/javascript"
src="${APP_PATH}/static/bootstrap-3.3.7-dist/js/bootstrap.min.js"></script>
<script type="text/javascript"
src="${APP_PATH}/static/jquery-3.1.1.min.js"></script>
```

2.利用bootstrap搭建页面

- 根据官方文档可以得知：把div的class设为container即可设为栅格系统
- 分成四行，第一行显示标题，第二行显示新增和删除的按钮，第三行显示查询出的数据，第四行显示分页信息
- 每一行的class叫row，每一列的为col-md-（数字1-12）

```
<div class="container">
      <!--标题 -->
      <div class="row">
          <div class="col-md-12">
              <h1>SSM-CRUD</h1>
          </div>
      </div>
      <!--按钮 -->
      <div class="row"></div>
      <!--显示表格数据 -->
      <div class="row"></div>
      <!--显示分页信息-->
      <div class="row"></div>
  </div>
```

3.添加删除和新增按钮

- 设置按钮的位置（在列偏移中设置自己占4列，偏移8列）

  html

```
<!--按钮 -->
<div class="row">
         <div class="col-md-4 col-md-offset-8">
             <button>新增</button>
             <button>删除</button>
         </div>
     </div>
```

- 美化按钮（根据官方文档，只需要给相应按钮的class增加值即可

html

```
 <!--按钮 -->
      <div class="row">
          <div class="col-md-4 col-md-offset-8">
              <button class="btn btn-primary">新增</button>
              <button class="btn btn-danger">删除</button>
          </div>
      </div>
```

4.添加显示数据的table

- 设置table的样式

html

```
  <!--  显示表格数据  -->
          <div class="row">
              <div class="col-md-12">
                  <table class="table table-hover" id="emp_table">
                      <thead>
                      <tr>
                          <th><input type="checkbox" id="check_all"/></th>
```

```
                        <th>#</th>
                        <th>员工编号</th>
                        <th>员工姓名</th>
                        <th>职位</th>
                        <th>入职时间</th>
                        <th>基本工资</th>
                        <th>部门名称</th>
                        <th>操作</th>
                    </tr>
                    </thead>
                    <tbody>
                    </tbody>
                </table>
            </div>
```

5.添加分页显示

- 组件-》分页，在里面添加分页信息

  html

```
<!--显示分页信息-->
    <div class="row">
        <!--分页文字信息-->
        <div class="col-md-6" id="page_info_area">
        </div>
        <!--分页条信息-->
        <div class="col-md-6" id="page_nav_area">
        </div>
    </div>
```

现在页面已经搭起来了，后面要做的就是把查出来的数据，放到页面中显示。

六、把数据插到页面中

1.引入标签库

- 员工的数据需要遍历得到，引入相应的标签库

  jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
```

2.遍历员工数据并填充到相应的位置

0.业务逻辑

- 1.index.jsp页面，直接发送ajax请求进行员工的分页查询
- 2.服务器将查出的数据，以json字符串的形式返回给浏览器
- 3.浏览器收到js字符串。可以使用js对json进行解析，使用js通过dom增删改的方式改变页面。
- 4.返回json。实现客户端的无关性。

1.在Empcontroller类中写一个返回json字符串的方法

- 页面需要分页数据，可以直接返回PageInfo对象，用@ResponseBody就能直接把对象转为json字符串
- 要想@ResposeBody正常工作需要导入jackson包
- - 去中央仓库搜索Jackson databind
- - 在pom.xml中引入相应的依赖

xml

```xml
<dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
        <version>2.10.0</version>
    </dependency>
```

2.创建一个专门返回json数据的类

- **处理上面的数据后，成功失败这些状态信息，应该通知给浏览器。所以，创建一个通用的能带有状态信息的返回json数据的类**
- 在bean包下创建一个Msg类，完成上面的功能

java

```java
/**
 * 通用提示的类
 */
public class Msg {
    //状态码 100-成功 200-失败
    private int code;
    //提示信息
    private String msg;

    //用户要返回给浏览器的数据
    //在add方法中有getExtend取得键值
    private Map<String,Object> extend = new HashMap<>();

    public static Msg success(){
        //result为了调用setCode和setMsg，并不是浏览器的数据结果，数据在extend中
        Msg result = new Msg();
        result.setCode(100);
        result.setMsg("处理成功");
        return result;
    }

    public static Msg fail(){
        Msg result = new Msg();
        result.setCode(200);
        result.setMsg("处理失败");
        return result;
    }
```

```
    public Msg add(String key, Object value){
        this.getExtend().put(key, value);
        return this;
    }
    public Map<String, Object> getExtend() { return extend; }
    public void setExtend(Map<String, Object> extend) { this.extend =
extend;}
    public int getCode() { return code; }
    public void setCode(int code) {this.code = code; }
    public String getMsg() { return msg; }
    public void setMsg(String msg) { this.msg = msg;}
}
```

- 然后希望返回给客户端的数据带上状态信息，EmpController类中方法为

java

```
/**
     * 需要导入json包
     * 分页数据
     * @param pn
     * @return
     */
    @RequestMapping(value = "findemp", method = RequestMethod.GET)
    @ResponseBody
    public Msg getEmpsWithJson(
            @RequestParam(value = "pn", defaultValue = "1") Integer pn) {
        //引入PageHelper分页插件
        //在查询之前只需要调用，传入页码，以及每页的大小
        PageHelper.startPage(pn, 5);
        //startPage后面紧跟的这个查询就是一个分页查询
        List<Emp> emp = emps.findAll();
        //使用PageInfo查询包装后的结果，只需要把PageInfo交给页面就行
        //封装了详细的分页信息，包括有我们查询出来的数据,传入连续显示的页数
        PageInfo page = new PageInfo(emp, 5);
        return Msg.success().add("pageInfo", page);
    }
```

- 此时在浏览器输入/emps请求，即可得到json的字符串如下

java

```
{"code":100,"msg":"处理成功","extend":{"pageInfo":{"total":6,"list":
[{"id":22,"empno":"QA-2002002","ename":"狗圣","job":"经理","hiredate":"2020-
07-09 00:00:00.0","sal":10000.0,"deptno":3,"deptname":"市场"},
{"id":23,"empno":"QP-20200101","ename":"王五","job":"测试","hiredate":"2020-
06-01 00:00:00.0","sal":7000.0,"deptno":2,"deptname":"技术"},
{"id":24,"empno":"QA-2203032","ename":"测试","job":"测试","hiredate":"2020-
06-10 00:00:00.0","sal":6000.0,"deptno":2,"deptname":"技术"},
{"id":25,"empno":"TD-20160302","ename":"李璇","job":"工程
师","hiredate":"2016-03-02
00:00:00.0","sal":20000.0,"deptno":2,"deptname":"技术"},
{"id":26,"empno":"MO-229300","ename":"李四","job":"工程师","hiredate":"2016-
03-03
00:00:00.0","sal":8999.0,"deptno":2,"deptname":"术"}],"pageNum":1,"pageSize"
:5,"size":5,"startRow":1,"endRow":5,"pages":2,"prePage":0,"nextPage":2,"isFi
rstPage":true,"isLastPage":false,"hasPreviousPage":false,"hasNextPage":true,
"navigatePages":5,"navigatepageNums":
[1,2],"navigateFirstPage":1,"navigateLastPage":2,"lastPage":2,"firstPage":1}
}}
```


- **以上就是我们需要解析的数据。接下来，我们就从首页出发，发一个ajax请求，拿到这些数据，对这些数据拿json解析出来，使用dom增删改的形式把数据显示出来。**

1.index.jsp页面，直接发送ajax请求进行员工的分页查询

(1) 准备工作

- **原来是要发请求跳转到别的页面，现在直接在index页面中显示**
- 把原来的的index.jsp重命为index1.jsp，然后重新建一个index.jsp页面
- 把show.jsp页面的代码全选复制到index.jsp内
- 把分页的文字信息删掉，分页条删掉，表格数据的显示c:foreach也删掉，以后的分页文字信息、分页条信息、表格的数据显示全是对于json数据的解析。

(2) 从index.jsp发起请求

- 页面加载完成以后，直接发送一个ajax请求，要到一个分页数据

javascript

```
<script type="text/javascript">
function to_page(pn) {
        $.ajax({
            url: "${APP_PATH}/findemp",
            data: "pn=" + pn,
            type: "GET",
            //function中的result代表服务器返回的数据，和Msg类中的result无关
            success: function (result) {
                console.log(result);
                //1、解析显示员工数据
                build_emps_table(result);
                //2、解析并显示分页信息
                build_page_info(result);
                //3、解析显示分页条信息
                build_page_nav(result);
            }
```

```
        });
    }
</script>
```

(3) 解析list里的员工数据

- 把员工的数据和操作的按钮一起添加

  javascript

```
function build_emps_table(result) {
        //清空
       $("#emp_table tbody").empty();
        var emps = result.extend.pageInfo.list;
        $.each(emps, function (index, item) {
            var checkboxId = $("<td><input type='checkbox'
class='check_item'/></td>");
             var idTd = $("<td></td>").append(item.id);
              var empnoTd = $("<td></td>").append(item.empno);
              var enameTd = $("<td></td>").append(item.ename);
              var jobTd = $("<td></td>").append(item.job );
               var hiredateTd = $("<td></td>").append(item.hiredate);
                var sal = $("<td></td>").append(item.sal);
                var deptname = $("<td></td>").append(item.deptname);
             /**
              * <botton class="btn btn-primary btn-sm">
              <span class="glyphicon glyphicon-pencil" aria-hidden="true">
</span>编辑
              </botton>
              * @type {*|jQuery}
              */
             var editBtn = $("<botton></botton>").addClass("btn btn-primary
btn-sm edit_btn")
                 .append($("<span></span>")).addClass("glyphicon glyphicon-
pencil")
                 .append("编辑");
             //为编辑按钮添加属性，来表示当前id
             editBtn.attr("edit-id",item.id);
             var delBtn = $("<botton></botton>").addClass("btn btn-danger
btn-sm delete_btn")
                 .append($("<span></span>")).addClass("glyphicon glyphicon-
trash")
                 .append("删除");
              delBtn.attr("del-id",item.id);
             var btnTd = $("<td></td>").append(editBtn).append("
").append(delBtn);
           $("<tr></tr>").append(checkboxId)
                 .append(idTd)
                 .append(empnoTd)
                 .append(enameTd)
                 .append(jobTd)
                 .append(hiredateTd)
                 .append(sal)
                 .append(deptname)
                 .append(btnTd)
                 .appendTo("#emp_table tbody");
```

```
        });
    }
```

（5）解析分页信息

- 显示分页信息和分页条信息

  javascript

```
//解析显示分页信息
    function build_page_info(result) {
      $("#page_info_area").empty();
        $("#page_info_area").append(
            "当前第 " + result.extend.pageInfo.pageNum + " 页" + " 总 "
            + result.extend.pageInfo.pages + " 页" + " 总 "
            + result.extend.pageInfo.total + " 条记录");
        totalRecord = result.extend.pageInfo.pages;
        currentPage = result.extend.pageInfo.pageNum;
    }

    //解析显示分页条
    function build_page_nav(result) {
        $("#page_nav_area").empty();
        var ul = $("<ul></ul>").addClass("pagination");
        var firstPageLi = $("<li></li>").append($("<a></a>").append("首
页").attr("href", "#"));
        var prePageLi = $("<li></li>").append($("<a>
</a>").append("&laquo;"));

        if (result.extend.pageInfo.hasPreviousPage == false) {
            firstPageLi.addClass("disabled");
            prePageLi.addClass("disabled");
        }

        firstPageLi.click(function () {
            to_page(1);
        })

        prePageLi.click(function () {
            to_page(result.extend.pageInfo.pageNum - 1);
        })

        var nextPageLi = $("<li></li>").append($("<a>
</a>").append("&raquo;"));
        var lastPageLi = $("<li></li>").append($("<a></a>").append("尾
页").attr("href", "#"));

        nextPageLi.click(function () {
            to_page(result.extend.pageInfo.pageNum + 1);
        })

        lastPageLi.click(function () {
            to_page(result.extend.pageInfo.pages);
        })
        if (result.extend.pageInfo.hasNextPage == false) {
            nextPageLi.addClass("disabled");
            lastPageLi.addClass("disabled");
        }
```

```
        ul.append(firstPageLi).append(prePageLi);
        $.each(result.extend.pageInfo.navigatepageNums, function (index,
item) {
            var numLi = $("<li></li>").append($("<a></a>").append(item));
            if (result.extend.pageInfo.pageNum == item) {
                numLi.addClass("active");
            }

            numLi.click(function () {
                to_page(item);
            });
            ul.append(numLi);
        });
        ul.append(nextPageLi).append(lastPageLi);
        var navEle = $("<nav></nav>").append(ul);
        navEle.appendTo("#page_nav_area");
    }
```

- 需要把从index页面发送请求的方式改一下，改成跳转页面的方法

  javascript

```
function to_page(pn) {
       $.ajax({
        url:"${APP_PATH}/emp",
           data:"pn="+pn,
           type:"get",
           success:function(result){//请求成功的回调函数,result就是服务器响应给浏
览器的数据
               //console.log(result);
               //1.解析并显示员工数据
               bulid_emps_table(result);
               //2.解析并显示分页信息
               //分页文字信息
               build_page_info(result);
               //分页条
               build_emps_nav(result);
           }
       });
    };
```

这样用户的信息就显示完全了，下面用ajax来完成剩下的功能

剩余项目

- **剩余的项目都用ajax请求完成**
- 用rest风格的URI

REST风格

- URI*

路径	请求类型	需求
/em/{id}	GET	查询员工
/emp	POST	保存员工
/em/{id}	PUT	修改员工
/em/{id}	DELETE	删除员工

七、用户新增

0.逻辑

- 1.在index.jsp页面点击“新增”
- 2.弹出新增对话框（模态框）
- 3.去数据库查询部门列表显示在对话框中
- 4.用户输入数据，并进行校验，完成保存

1.点击新增，弹出模态框

（1）在页面中引入模态框

bootstarp中寻找模态框，然后复制代码，根据需要改成自己的

html

```
<!-- 用户新增模态框 -->
<div class="modal fade" id="myModal" tabindex="-1" role="dialog" aria-
labelledby="myModalLabel">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <button type="button" class="close" data-dismiss="modal" aria-
label="Close"><span aria-hidden="true">&times;</span></button>
        <h4 class="modal-title" id="myModalLabel">Modal title</h4>
      </div>
      <div class="modal-body">
        ...
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-default" data-
dismiss="modal">Close</button>
        <button type="button" class="btn btn-primary">Save changes</button>
      </div>
    </div>
  </div>
</div>
```

- 为新增按钮添加id

html

```
<button class=" btn btn-danger">
    <span class="glyphicon glyphicon-plus" aria-hidden="true"
id="emp_add_model_btn">新增</span>
</button>
```

(2) 点击新增，弹出模态框

- 为新增按钮绑事件：结合文档中模态框的用法

`

```
//点击新增按钮弹出模态框
    $("#emp_add_modal_btn").click(function () {
        //表单重置（完整重置）
        reset_form("#myModal form")
        $("#myModal").modal({
            backdrop: "static"
        });
        getDepts("#myModal select");
    });
```

(3) 把模态框需要的样子定义出来

- 对标题、按钮修改，添加表单（水平表单，在bootstrap中找）

html

```
<!-- 新增Modal -->
<div class="modal fade" id="myModal" tabindex="-1" role="dialog" aria-
labelledby="myModalLabel">
    <div class="modal-dialog" role="document">
        <div class="modal-content">
            <div class="modal-header">
                <button type="button" class="close" data-dismiss="modal"
aria-label="Close"><span aria-hidden="true">&times;</span>
                </button>
                <h4 class="modal-title" id="myModalLabel">员工添加</h4>
            </div>
            <div class="modal-body">
                <form class="form-horizontal">
                    <div class="form-group">
                        <label for="ename_add_input" class="col-sm-2
control-label">员工姓名</label>
                        <div class="col-sm-10">
                            <input type="text" name="ename" class="form-
control" id="ename_add_input"
                                   placeholder="姓名">
                            <span class="help-block"></span>
                        </div>
                    </div>
                    <div class="form-group">
                        <label for="empno_add_input" class="col-sm-2
control-label">员工编号</label>
                         <div class="col-sm-10">
```

```
                            <input type="text" name="empno" class="form-
control" id="empno_add_input">
                            <span class="help-block"></span>
                        </div>
                    </div>
                    <div class="form-group">
                        <label for="emjob_add_input" class="col-sm-2
control-label">职位</label>
                        <div class="col-sm-10">
                            <input type="text" name="job" class="form-
control" id="emjob_add_input"
                                   placeholder="经理/工程师">
                            <span class="help-block"></span>
                        </div>
                    </div>
                    <div class="form-group">
                        <label for="sal_add_input" class="col-sm-2 control-
label">基本工资</label>
                        <div class="col-sm-10">
                            <input type="text" name="sal" class="form-
control" id="sal_add_input">
                            <span class="help-block"></span>
                        </div>
                    </div>
                    <div class="form-group">
                        <label for="hiredate_add_input" class="col-sm-2
control-label">入职时间</label>
                        <div class='input-group date' id='datetimepicker3'>
                            <input type='text' name="hiredate" class="form-
control" id="hiredate_add_input" >
                            <span class="input-group-addon">
                               <span class="glyphicon glyphicon-calendar">
</span>
                            </span>
                        </div>
                    </div>
                    <div class="form-group">
                        <label class="col-sm-2 control-label">部门名称
</label>
                        <div class="col-sm-4">
                            <!--部门提交Id即可-->
                            <select class="form-control" name="deptno">
                            </select>
                        </div>
                    </div>
                </form>
            </div>
            <div class="modal-footer">
                <button type="button" class="btn btn-default" data-
dismiss="modal">关闭</button>
                <button type="button" class="btn btn-primary"
id="emp_save_btn">保存</button>
            </div>
        </div>
    </div>
</div>
```

2.填充模态框中的部门信息

- 需求：希望部门信息是从数据库中查到的
- 分析：弹出模态框之前，发ajax请求，查出所有的部门信息，把部门信息显示到下拉列表中。
- 在index.jsp页面写一个查询部门信息的ajax请求的方法

javascript

```
// 获取部门信息
    function getDepts(ele) {
        $(ele).empty();
        $.ajax({
            url: "/depts",
            type: "GET",
            success: function (result) {
                console.log(result)
                // $("#dept_add_select").append("")
                $.each(result.extend.depts, function () {
                    var optionEle = $("<option>
</option>").append(this.deptname).attr("value", this.deptid);
                    optionEle.appendTo(ele)
                })
            }
        })
    }
```

- 控制器中需要有一个拦截他的方法，在controller包下创建一个PartController.java,用来处理和部门有关的请求

java

```
/**
     * 返回所有的部门信息
   * 返回JSON
     */
    @RequestMapping("/depts")
    @ResponseBody
    public Msg getDepts() {
        List<Part> list = partService.findAllPart();
        return Msg.success().add("depts", list);
    }
```

- 在service包下创建PartService接口和PartServiceImpl实现类

java

```
    /**
     * 查询所有部门信息
     */
    public List<Part> findAllPart();


     @Override
    public List<Part> findAllPart() {
        List<Part> list =partMapper.findAllPart();
        return list;
    }
```

3.点击保存按钮保存数据

- 点击保存，发送ajax请求，保存数据
- 为保存按钮添加id
  `id="emp_save_btn"`
- 为保存按钮绑事件

javascript

```
//保存
    $("#emp_save_btn").click(function () {
            $.ajax({
                    url: "${APP_PATH}/insertorupdate",
                    type: "POST",
                    data: $("#myModal form").serialize(),
                    success: function (result) {
                        if(result==1){
                            $("#myModal").modal("hide");
                            to_page(totalRecord);}
                    }
                })
    });
```

- 在EmpContorller中添加保存的方法

java

```
 /**
     * 员工保存
     * @param emp
     * @return
     */
    @RequestMapping(value = "/emp/{id}", method = RequestMethod.PUT)
    @ResponseBody
    public Msg saveEmp(Emp emp) {
        emps.updateEmp(emp);
        return Msg.success();
    }
```

- 在EmpService和EmpServiceImpl中创建员工保存的方法

java

```
    /**
     * 增加员工信息
     */
    public int addEmp(Emp emp);


    public int addEmp(Emp emp) {
        return  empMapper.addEmp(emp);
    }
}
```

4.员工编号校验

- 新增和修改时，员工编号是唯一的值，不能重复，所以通过后端逻辑进行数据校验。

在mapper接口和xml中添加查询方法，查找验证数据库是否存在当前员工编号

```
    public Integer checkEmpno(@Param("empno") String empno);


     <select id="checkEmpno" resultType="Integer">
        select count(*)from EMP where EMPNO like '%${empno}%'
    </select>
```

- 在service和serviceImpl中添加方法
-

```
/**
     * 检测重复
     * @param empno
     * @return
     */
   public boolean checkEmpno(String empno);

@Override
    public boolean checkEmpno(String empno) {
        System.out.printf(empno);
        long count = empMapper.checkEmpno(empno);
```

```
        System.out.printf("d"+count);
        if (count == 0) {
            return true;
        } else
            return false;
    }
```

- 在controller中添加接口

```
@ResponseBody
    @RequestMapping("/checkEmpno")
    public Msg check(@RequestParam("empno")String empno){
        boolean b = emps.checkEmpno(empno);
        if (b){
            return Msg.success();
        }else {
            return Msg.fail().add("va_msg","员工编号重复！");
        }
    }
```

- 前端添加ajax的请求和验证

```
//校验员工编号是否可用
            $("#empno_add_input").change(function () {
                //发送ajax请求校验用户名是否可用
                var empno = this.value;//this.value是输入框的值
                $.ajax({
                    url: "${APP_PATH}/checkEmpno",
                    data: "empno=" + empno,
                    type: "post",
                    success: function (result) {
                        if (result.code == 100) {
                            show_validate_msg("#empno_add_input", "success",
"员工编号可用");
                            $("#emp_save_btn").attr("ajax-va", "success");
                        } else {
                            show_validate_msg("#empno_add_input", "error",
result.extend.va_msg);
                            $("#emp_save_btn").attr("ajax-va", "error");

                        }
                    }
                });
            });

        function show_validate_msg(ele, status, msg) {
        //清除当前元素的校验状态
        $(ele).parent().removeClass("has-success has-error");
        $(ele).next("span").text("");
        if ("success" == status) {
            $(ele).parent().addClass("has-success");
            $(ele).next("span").text(msg);
        } else if ("error" == status) {
            $(ele).parent().addClass("has-error");
```

```
            $(ele).next("span").text(msg);
        }
    }


    //保存
    $("#emp_save_btn").click(function () {
       // 添加保存时候的判断，
            if($(this).attr("ajax-va")=="error"){
                return false;
            }
            $.ajax({
                    url: "${APP_PATH}/insertorupdate",
                    type: "POST",
                    data: $("#myModal form").serialize(),
                    success: function (result) {
                        if(result==1){
                            $("#myModal").modal("hide");
                            to_page(totalRecord);}
                    }
                })
    });
```

八、修改

0.逻辑

- 1.点击编辑
- 2.弹出用户修改的模态框（显示用户信息）
- 3.点击更新，完成用户修改

1.创建员工修改的模态框

- 把员工新增的模态框复制一份，用于员工修改的模态框

html

```
<!--员工修改的模态框-->
<div class="modal fade " id="empUpdateModal" tabindex="-1" role="dialog"
aria-labelledby="myModalLabel">
    <div class="modal-dialog" role="document">
        <div class="modal-content" style="height: 550px">
            <div class="modal-header">
                <button type="button" class="close" data-dismiss="modal"
aria-label="Close"><span aria-hidden="true">&times;</span>
                </button>
                <h4 class="modal-title">员工修改</h4>
            </div>
            <div class="modal-body" style="height:400px">
                <form class="form-horizontal">
                    <div class="form-group">
```

```
                        <label for="empName_add_input" class="col-sm-2
control-label">员工姓名</label>
                        <div class="col-sm-10">
                            <p class="form-control-static"
id="empname_update_static"></p>
                        </div>
                    </div>
                    <div class="form-group">
                        <label for="empno_add_input" class="col-sm-2
control-label">员工编号</label>
                        <div class="col-sm-10">
                            <input type="text" name="empno" class="form-
control" id="empno_update_input">
                            <span class="help-block"></span>
                        </div>
                    </div>
                    <div class="form-group">
                        <label for="emjob_add_input" class="col-sm-2
control-label">职位</label>
                        <div class="col-sm-10">
                            <input type="text" name="job" class="form-
control" id="emjob_update_input"
                                   placeholder="经理/工程师">
                            <span class="help-block"></span>
                        </div>
                    </div>
                    <div class="form-group">
                        <label for="sal_add_input" class="col-sm-2 control-
label">基本工资</label>
                        <div class="col-sm-10">
                            <input type="text" name="sal" class="form-
control" id="sal_update_input">
                            <span class="help-block"></span>
                        </div>
                    </div>
                    <div class="form-group">
                        <label for="hiredate_add_input" class="col-sm-2
control-label">入职时间</label>
                        <div class="col-sm-10">
                          <p class="form-control-static"
id="hiredate_update_input"></p>
                        </div>
                    </div>
                    <div class="form-group">
                        <label class="col-sm-2 control-label">部门名称
</label>
                        <div class="col-sm-4">
                            <!--部门提交Id即可-->
                            <select class="form-control" name="deptno">
                            </select>
                        </div>
                    </div>
                </form>
                <div class="modal-footer" >
                    <button type="button" class="btn btn-default" data-
dismiss="modal">关闭</button>
                    <button type="button" class="btn btn-primary"
id="emp_update_btn">更新</button>
```

```
                </div>
            </div>
        </div>
    </div>
</div>
```

- 因为编辑按钮是通过ajax请求查出数据之后创建的，所以对他用加载完页面直接帮事件的方法行不通
- 为编辑绑事件的方法：1.创建完编辑按钮之后，为按钮绑事件；2.通过on方法（这里使用第二种）
- 之前查出的部门信息放到了员工添加的列表中，只需要把要添加的元素传进来即可
- 点击编辑，弹出模态框

javascript

```
//这里绑定事件要注意,按钮是之后添上去的标签，所以用on绑定单击事件
    $(document).on("click",".edit_btn",function () {
        //alert("fsf")
        //1、查出部门信息，显示部门列表
        getDepts("#empUpdateModal select");
        //3、把员工id传递给更新按钮
        $("#emp_update_btn").attr("edit-id",$(this).attr("edit-id"));
        //2、查出员工信息，显示员工信息
        getEmp($(this).attr("edit-id"));
        $("#empUpdateModal").modal({
            backdrop:"static"
        })
    });
```

2.点击编辑，显示员工员工信息

- index中查询员工息的方法

  javascript

```
function getEmp(id) {
        $.ajax({
          url:"${APP_PATH}/emp/" + id,
            type:"GET",
            success:function (result) {
                // console.log(result)
                var empData = result.extend.emp;
                $("#empname_update_static").text(empData.ename);
                //.val() 能够取到 针对text，hidden可输入的文本框的value值。
                // 而 .attr('value') 可以取到html元素中所设置的属性 value的值，不能获取动态的如input type="text" 的文本框手动输入的值
                $("#empno_update_input").val(empData.empno);
                $("#emjob_update_input").val(empData.job);
                $("#sal_update_input").val(empData.sal);
                $("#hiredate_update_input").text(empData.hiredate);
                $("#empUpdateModal select").val([empData.deptno]);
            }
        })
    }
```

- 在EmpController中创建相应的方法

java

```
/**
     * 按照员工id查询员工
     * @param id
     * @return
     */
    //rest风格的url
    @RequestMapping(value = "/emp/{id}", method = RequestMethod.GET)
    @ResponseBody
    public Msg getEmp(@PathVariable("id") Integer id) {
        Emp emp = emps.findById(id);
        return Msg.success().add("emp", emp);
    }
```

- 在EmplService和EmplServiceImpl中创建相应的方法

java

```
    /**
     * 根据Id查询
     */
    public Emp findById(int id);


    public Emp findById(int id) {
        return empMapper.findById(id);
    }
```

3.点击修改，更新员工

- 为更新按钮绑定单击事件

javascript

```
    //点击更新，更新员工信息
    $("#emp_update_btn").click(function () {
        $.ajax({
            url:"${APP_PATH}/emp/"+$(this).attr("edit-id"),
            type:"PUT",
            data:$("#empUpdateModal form").serialize(),
            success:function (result) {
                //alert(result.msg);
                $("#empUpdateModal").modal("hide");
                  //2.回到本页面
                    to_page(currentPage)
            }
        })
```

```
    })
```

- 因为发送的是ajax请求，要写请求的处理事件
- 在EmployeeController中添加相应的保存方法

java

```
/**保存员工

     * @param
     * @return Message
     * @description:
     *    解决方法: 1.发送post方法,通过HiddenHttpMethodFilter
     *        2.发送put请求,通过HttpPutFormContentFilter(通过web.xml配置)
     */
    @RequestMapping(value = "/emp/{id}", method = RequestMethod.PUT)
    @ResponseBody
    public Msg saveEmp(Emp emp) {
        emps.updateEmp(emp);
        return Msg.success();
    }
```

- 在service和Impl中中添加相应的保存方法
- 员工更新 这里ajax请求直接发put请求而不是post请求,那么所有的参数都会获取不到,因为tomcat只会封装post的数据。也就是说request.getParameter("empId")为空,springmvc也无法封装Bean
- 因为tomcat不封装put请求发送的数据，spring提供了支持（HttpPutFormContentFilter）
  - 在web.xml中，把这个过滤器配置上

    xml

```
<filter>
        <filter-name>HttpPutFormContentFilter</filter-name>
    <filter-
class>org.springframework.web.filter.HttpPutFormContentFilter</filter-
class>
    </filter>
  <filter-mapping>
        <filter-name>HttpPutFormContentFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

九、删除

1.单个删除

URI：/emp/{id} DELETE形式的请求

（1）单个删除的步骤

- 在EmpController中拦截发送的请求

java

```
/**
     * @param
     * @return Message
     * @description:单个删除1  批量删除:1-2-3
     */
    @DeleteMapping(value = "/emp/{ids}")
    @ResponseBody
    public Msg deleteEmployee(@PathVariable("ids") String ids) {
        if (ids.contains("-")) {
        String[] str_ids = ids.split("-");//分割成数组
        List<Integer> del_ids = new ArrayList<>();
            for (String id : str_ids) {
                del_ids.add(Integer.parseInt(id));
            }
            emps.deleteByIds(del_ids);
        }else {
            Integer id = Integer.parseInt(ids);
            emps.deleteEmp(id);
        }
        return Msg.success();
    }
```

- 在Service和Impl中中创建相应的方法

java

```
    public int deleteByIds(List<Integer> ids);

        @Override
    public int deleteByIds(List<Integer> ids) {
        return empMapper.deleteByIds(ids);
    }
```

- 用on为删除按钮绑定单击事件
- 弹出是否确认删除的对话框（拿出员工的员工的名字）
- 删除成功后，返回当前页码

javascript

```
// 单个删除
      $(document).on("click",".delete_btn",function(){
          //弹出是否确认删除对话框(找到当前的祖先节点的tr，tr下的第二个td的文本值)
          //alert($(this).parents("tr").find("td:eq(1)").text());
          var empName = $(this).parents("tr").find("td:eq(3)").text();
          var ids = $(this).attr("del-id");//要删除的员工id
          if(confirm("确认删除【"+empName+"】吗？")){
              //点击确认发送ajax请求，删除即可
              $.ajax({
```

```
                url:"${APP_PATH}/emp/"+ids,
                 type:"DELETE",
                 success:function(result){
                    alert(result.msg);
                    //回到本页
                     to_page(currentPage);
                 }
             });
         }
     });
```

1. 构造删除按钮的时候给按钮添加一个自定义的属性来表示员工id

```
  var delBtn = $("<botton></botton>").addClass("btn btn-danger btn-sm
delete_btn")
                 .append($("<span></span>")).addClass("glyphicon glyphicon-
trash")
                 .append("删除");
              delBtn.attr("del-id",item.id);
```

2. *这样就完成了单个删除**

2.批量删除

（1）添加多选框

```
<!--  显示表格数据  -->
    <div class="row">
        <div class="col-md-12">
            <table class="table table-hover" id="emp_t
                <thead>
                <tr>
                    <th><input type="checkbox" id="check_all"/></th>
                    <th>#</th>
                    <th>员工编号</th>
                    <th>员工姓名</th>
                    <th>职位</th>
                    <th>入职时间</th>
                    <th>基本工资</th>
                    <th>部门名称</th>
                    <th>操作</th>
```

1. 解析数据的时候也需要checkbox

```
        //清空
        $("#emp_table tbody").empty();
        var emps = result.extend.pageInfo.list;
        $.each(emps, function (index, item) {
            var checkboxId = $("<td><input type='checkbox' class='check_item'/></td>");
            var idTd = $("<td></td>").append(item.id);
            var empnoTd = $("<td></td>").append(item.empno);
            var enameTd = $("<td></td>").append(item.ename);
            var jobTd = $("<td></td>").append(item.job );
             var hiredateTd = $("<td></td>").append(item.hiredate);
              var sal = $("<td></td>").append(item.sal);
```

2. 把CheckBox添加到遍历的元素中

```
        delBtn.attr("del-id",item.id);
      var btnTd = $("<td></td>").append(editBtn).append(" ").append(delBtn);
     $("<tr></tr>").append(checkboxId)
            .append(idTd)
            .append(empnoTd)
            .append(enameTd)
            .append(jobTd)
            .append(hiredateTd)
            .append(sal)
            .append(deptname)
            .append(btnTd)
            .appendTo("#emp_table tbody");
  });
```

(2) 完成全选全不选功能

- 点击最上面的按钮，让下面的按钮与最上面的按钮的checked的值保持同步

javascript

```
//完成全选，全不选功能
    $("#check_all").click(function(){
        //attr获取checked是undefined;
        //我们这些dom原生的属性：attr获取自定义属性的值，用prop修改和读取dom原生属性的
值;
        $(this).prop("checked");
        //让下面的选择框和第一个的值相同
        $(".check_item").prop("checked",$(this).prop("checked"));
    });
```

- 当下面的选择框全部点满后，上面的选择框自动的选中
- 为下面的选择框添加单击事件
- 用checked选择器，查看被选中的个数

code

```
//给check_item绑定单击事件
    $(document).on("click",".check_item",function(){
        //判断当前选中的元素是否全选
        var flag = ($(".check_item:checked").length ==
$(".check_item").length);
         $("#check_all").prop("checked",flag);
    });
```

(3) 批量删除

- 给全部删除的按钮添加id
- 给删除按钮绑定单击事件
- 遍历被选中的员工，并提示

javascript

```
//点击全部删除，就批量删除
         $("#emp_del_all_btn").click(function () {
            //遍历每一个被选中的，查出名字
             var empNames = "";
             var del_idstr = "";
            $.each($(".check_item:checked"),function(){
              empNames += $(this).parents("tr").find("td:eq(3)").text()+",";
              //组装员工id字符串
                del_idstr +=
$(this).parents("tr").find("td:eq(1)").text()+"-";
            });
            //去除多余的（最后一个）逗号：从0开始截取字符串，到导数第二个
            empNames = empNames.substring(0,empNames.length-1);
             //去除多余的（最后一个）-：从0开始截取字符串，到导数第二个
            del_idstr = del_idstr.substring(0,del_idstr.length-1);
            if(confirm("确认删除【"+empNames+"】员工吗？")){
                //发送ajax请求删除
                $.ajax({
                   url:"${APP_PATH}/emp/"+del_idstr,
                   type:"delete",
                   success:function(result){
                       alert(result.msg);
                       //回到当前页面
                       to_page(currentPage);
                   }
                });
            }

         });
```

controller中的方法，根据请求的值来选择是单个删除还是批量删除。

java

```
/**
 * @param
 * @return Message
 * @description:单个批量删除 单个删除:1 批量删除:1-2-3
 */
@DeleteMapping(value = "/emp/{ids}")
@ResponseBody
public Message deleteEmployee(@PathVariable("ids") String ids) {
    if (ids.contains("-")) {//包含-就是批量删除，调用方法如下
        String[] str_ids = ids.split("-");//分割成数组
        List<Integer> del_ids = new ArrayList<>();
        for (String id : str_ids) {
            del_ids.add(Integer.parseInt(id));
        }
        employeeService.deleteBatch(del_ids);
    } else {//不包含-就是单个删除，调用方法如下：
        Integer id = Integer.parseInt(ids);
        employeeService.deleteEmployee(id);
    }
    return Message.success();
}
```

十、项目预览

1.列表显示和分页操作

SSM-CRUD

新增 全部删除

#	员工编号	员工姓名	职位	入职时间	基本工资	部门名称	操作
22	QA-2002002	狗圣	经理	2020-07-09 00:00:00.0	10000	市场	编辑 删除
23	QP-20200101	王五	测试	2020-06-01 00:00:00.0	7000	技术	编辑 删除
24	QA-2203032	测试	测试	2020-06-10 00:00:00.0	6000	技术	编辑 删除
25	TD-20160302	李璇	工程师	2016-03-02 00:00:00.0	20000	技术	编辑 删除
26	MO-229300	李四	工程师	2016-03-03 00:00:00.0	8999	技术	编辑 删除

当前第 1 页 总 2 页 总 8 条记录

首页 « 1 2 » 尾页

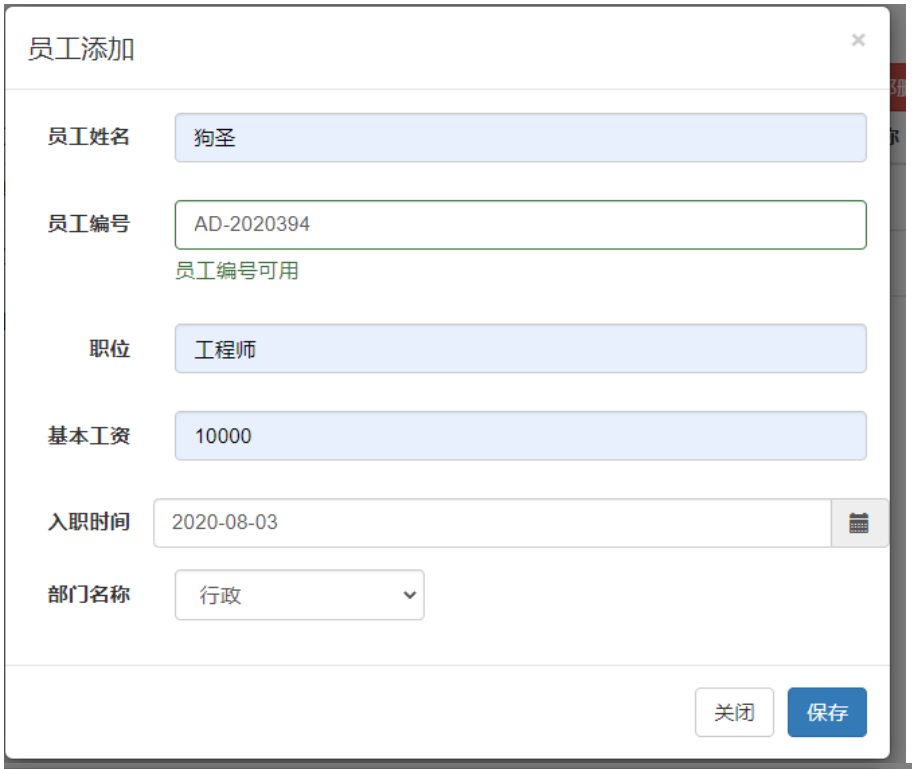
新增 全部删除

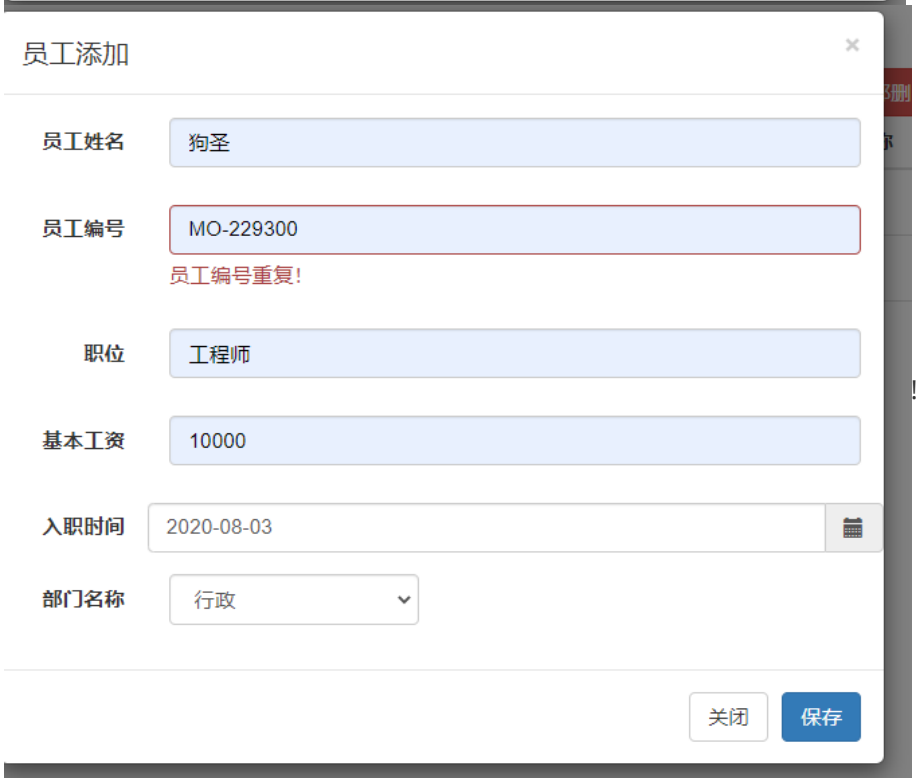
#	员工编号	员工姓名	职位	入职时间	基本工资	部门名称	操作
27	MMD-23323	张思	研发	2016-02-29 00:00:00.0	7000	行政	编辑 删除
29	MO-229307	李宣	工程师	2020-08-05 00:00:00.0	8999	行政	编辑 删除
30	MO-2293001	李四	工程师	2020-07-27 00:00:00.0	8999	行政	编辑 删除

当前第 2 页 总 2 页 总 8 条记录

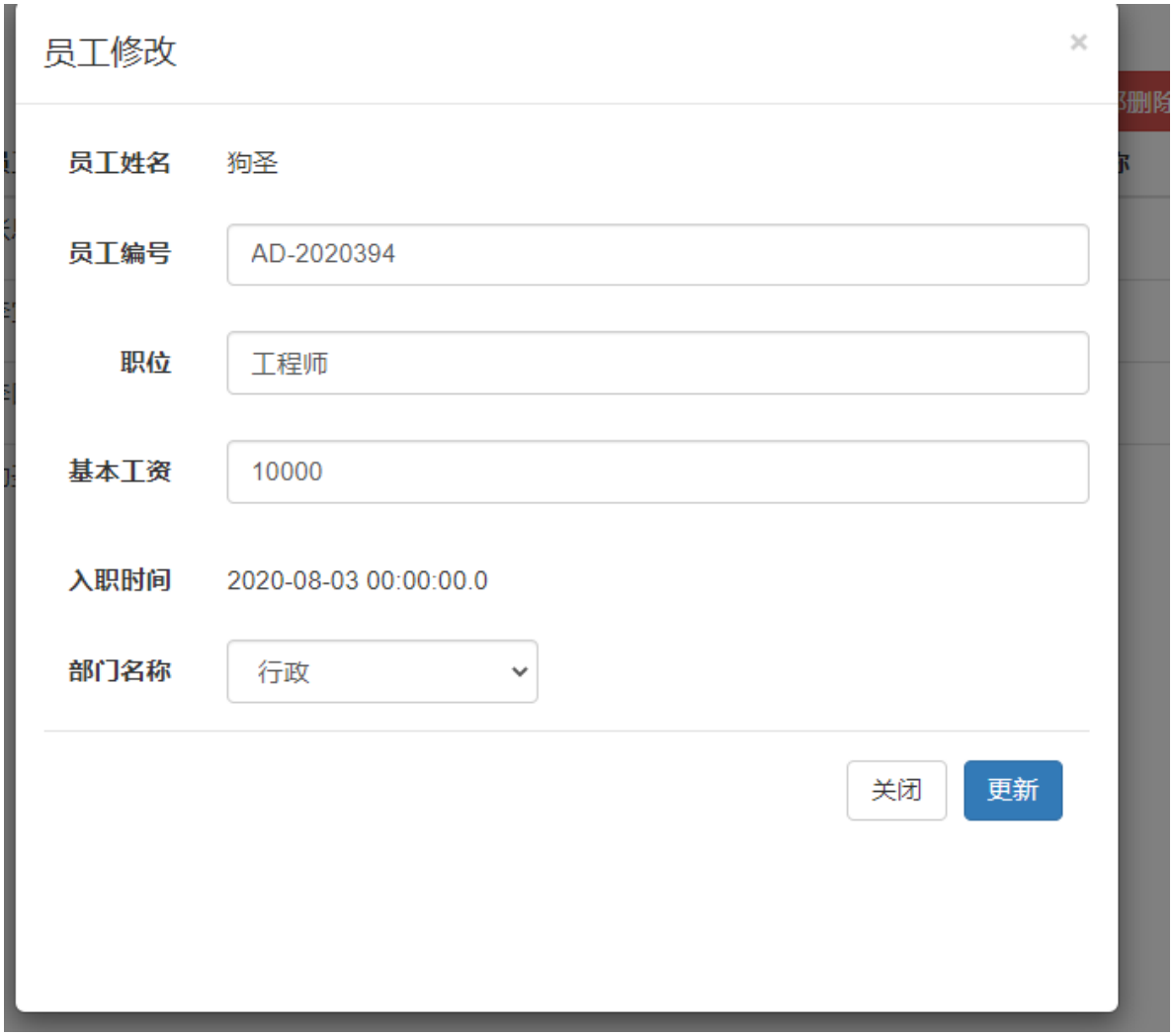
首页 « 1 2 » 尾页

2.新增和员工编号验证


员工添加
员工姓名
狗圣
员工编号
AD-2020394
员工编号可用
职位
工程师
基本工资
10000
入职时间
2020-08-03
部门名称
行政
关闭
保存

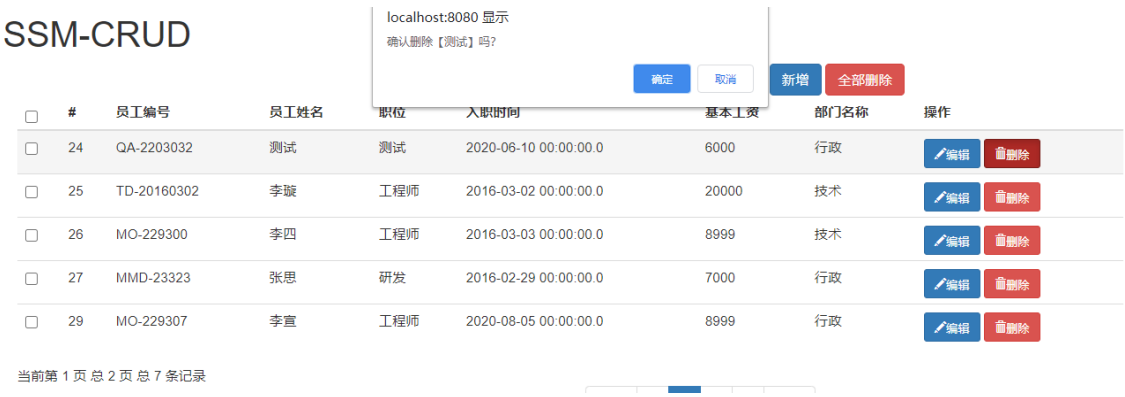

员工添加
员工姓名
狗圣
员工编号
MO-229300
员工编号重复!
职位
工程师
基本工资
10000
入职时间
2020-08-03
部门名称
行政
关闭
保存

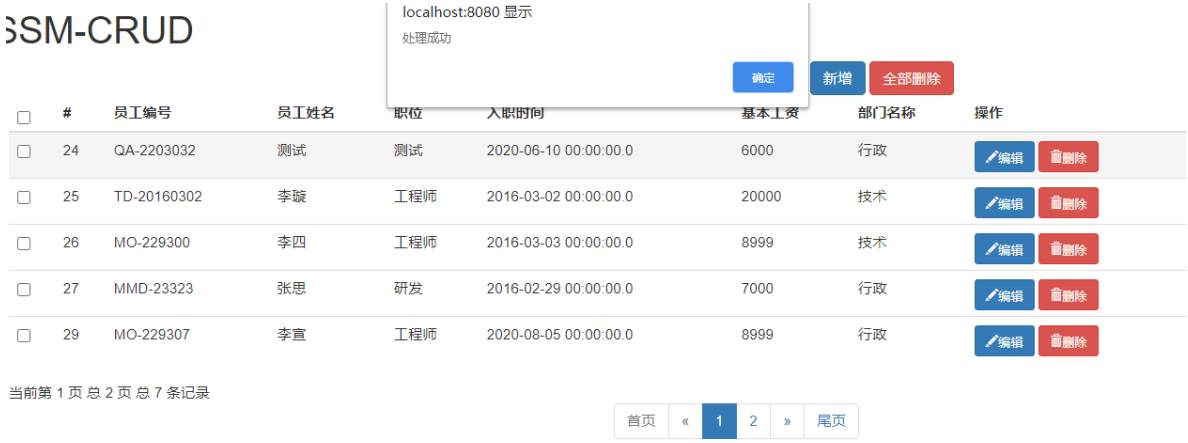
3.编辑



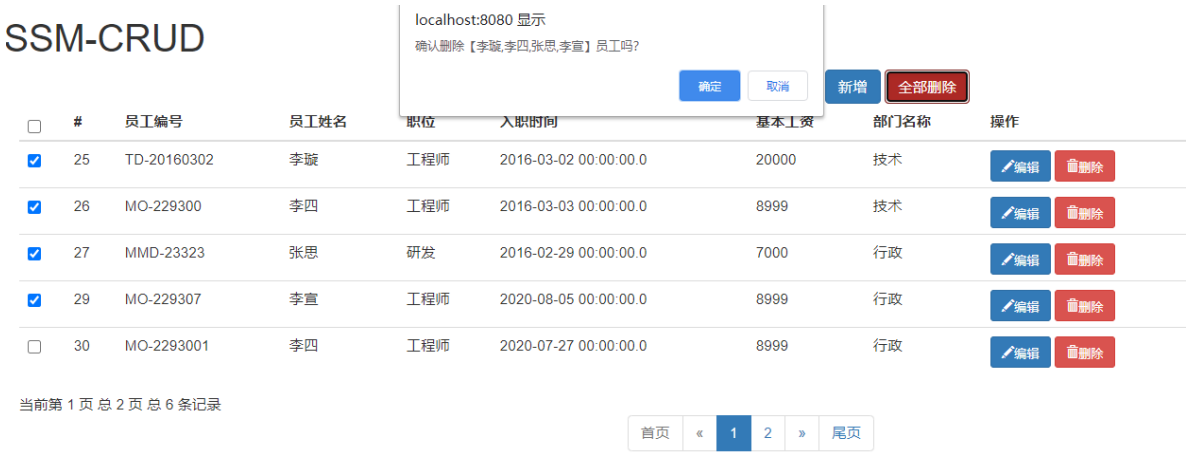
4.删除和批量删除

(1) 删除




SSM-CRUD
localhost:8080 显示
处理成功
确定
新增
全部删除
员工编号 员工姓名 职位 入职时间 基本工资 部门名称 操作
24 QA-2203032 测试 测试 2020-06-10 00:00:00.0 6000 行政 编辑 删除
25 TD-20160302 李璇 工程师 2016-03-02 00:00:00.0 20000 技术 编辑 删除
26 MO-229300 李四 工程师 2016-03-03 00:00:00.0 8999 技术 编辑 删除
27 MMD-23323 张思 研发 2016-02-29 00:00:00.0 7000 行政 编辑 删除
29 MO-229307 李宣 工程师 2020-08-05 00:00:00.0 8999 行政 编辑 删除
当前第 1 页 总 2 页 总 7 条记录
首页 « 1 2 » 尾页

(2) 批量删除


SSM-CRUD
localhost:8080 显示
确认删除【李璇,李四,张思,李宣】员工吗?
确定
取消
新增
全部删除
员工编号 员工姓名 职位 入职时间 基本工资 部门名称 操作
25 TD-20160302 李璇 工程师 2016-03-02 00:00:00.0 20000 技术 编辑 删除
26 MO-229300 李四 工程师 2016-03-03 00:00:00.0 8999 技术 编辑 删除
27 MMD-23323 张思 研发 2016-02-29 00:00:00.0 7000 行政 编辑 删除
29 MO-229307 李宣 工程师 2020-08-05 00:00:00.0 8999 行政 编辑 删除
30 MO-2293001 李四 工程师 2020-07-27 00:00:00.0 8999 行政 编辑 删除
当前第 1 页 总 2 页 总 6 条记录
首页 « 1 2 » 尾页


SSM-CRUD
处理成功
确定
新增
全部删除
员工编号 员工姓名 职位 入职时间 基本工资 部门名称 操作
25 TD-20160302 李璇 工程师 2016-03-02 00:00:00.0 20000 技术 编辑 删除
26 MO-229300 李四 工程师 2016-03-03 00:00:00.0 8999 技术 编辑 删除
27 MMD-23323 张思 研发 2016-02-29 00:00:00.0 7000 行政 编辑 删除
29 MO-229307 李宣 工程师 2020-08-05 00:00:00.0 8999 行政 编辑 删除
30 MO-2293001 李四 工程师 2020-07-27 00:00:00.0 8999 行政 编辑 删除
当前第 1 页 总 2 页 总 6 条记录
首页 « 1 2 » 尾页


SSM-CRUD
新增
全部删除
员工编号 员工姓名 职位 入职时间 基本工资 部门名称 操作
30 MO-2293001 李四 工程师 2020-07-27 00:00:00.0 8999 行政 编辑 删除
31 AD-2020394 狗圣 工程师 2020-08-03 00:00:00.0 10000 行政 编辑 删除
当前第 1 页 总 1 页 总 2 条记录
首页 « 1 » 尾页