开发人员：玉耀臣

2. 项目起止时间2020.9.29-2020.10.9

3. Git地址：

4. **项目主题**：人事管理

5. **需求分析**：该系统主要由管理员处理对员工数据增删查改，以及对这些数据的分析由前端页面柱形图、饼图、折线图直观分析结果，所以数据库设计应该为角色表，管理员账号表，角色权限表，员工表

6. **技术选型**：敏捷型开发选择**Spring boot**框架，SpringBoot是为了简化Spring应用的创建、运行、调试、部署等而出现的，使用它可以做到专注于Spring应用的开发，而无需过多关注XML的配置，简单来说，springboo提供了一堆依赖打包，并且已经按照习惯解决了依赖问题；

   SpringBoot默认使用tomcat作为服务器，使用logback提供日志记录；

   SpringBoot提供了一系列的依赖包，所以需要构建工具的支持，maven或者gradle；

   分析得出本系统功能逻辑一般，没有太多复杂的SQL语句，所以选择Spring data JPA。

   JPA的出现主要是为了简化持久层开发以及整合ORM技术，结束Hibernate、TopLink、JDO等ORM框架各自为营的局面。JPA是在吸收现有ORM框架的基础上发展而来，易于使用，伸缩性强。总的来说，JPA包括以下3方面的技术：

   - **ORM映射元数据**：支持XML和注解两种元数据的形式，元数据描述对象和表之间的映射关系

   - **API**：操作实体对象来执行CRUD操作

   - **查询语言**：通过面向对象而非面向数据库的查询语言（`JPQL`）查询数据，避免程序的SQL语句紧密耦合

   使用jpa我们可以在做一般的增删查改中可以直接使用JpaRepository中提供的方法，只需要在Service或Controller中调用即可，对于复杂的SQL操作可以选择自定义查询Using @Query或者原生查询Native Queries，原生查询为直接使用当前SQL语言。

   由于决定使用JPA，所以对于选择关系型数据库来说可以选择自己熟悉的，在这里主要使用**Oracle。**

   为了降低数据库压力选择**Redis**作为数据缓存。

   作为MVC架构的经典选择了**JSP**作为前端。

   前端使用的框架主要为**Jquery、Bootstrap、Materialize-UI、layui**，在页面交互中使用Bootstrap，布局和样式选择了Materialize-UI，layui只作为弹窗使用。

   这样子的页面风格比较简约，可以直面主题；

   系统安全选择Spring security做权限验证，可以做到对角色分配资源、页面控制器。

   分页选择NavigationTag对前端的分页排序 显示格式：首页 上一页 1 2 3 4 5下一页 尾页

7. **详细说明**：该项目主要是为人事管理企业员工，主要提供了对员工的CURD以及对员工近期一月入职、离职、总在职人数统计、工资范围分析、当年十二个月的人员变动分析。其中集成了Spring Security权限验证，现在开发后台功能主要为管理员权限所用，若开发员工权限页面只需要在数据库增加员工权限即可。

8. 从零搭建Demo

9. 开发环境为Windows10系统

10. 选择IntelliJ IDEA 2019.3.5 x64版本 https://www.jetbrains.com/idea/download/other.html

11. 先去下载一个Maven3.5 https://archive.apache.org/dist/maven/maven-3/3.5.0/source/ 下载，下载完成后解压到本地C盘即可，修改maven配置文件，找到conf文件中settings.xml打开

```
<localRepository>C:/apache-maven-3.5.0/repository</localRepository>
```

这里改maven 本地仓库地址

```
<mirrors>

    <mirror>
        <id>nexus-aliyun</id>
        <name>Nexus aliyun</name>
        <url>http://maven.aliyun.com/nexus/content/groups/public/</url>
        <mirrorOf>central</mirrorOf>
    </mirror>
</mirrors>
```

在mirrors中使用阿里云的镜像，相比于原仓库来说，在国内下载依赖速度飞快，如果不配置这个而是用IDEA的默认maven时 maven打包速度太慢

12. 再去下载一个jdk1.8.0_191 版本为1.8以上即可

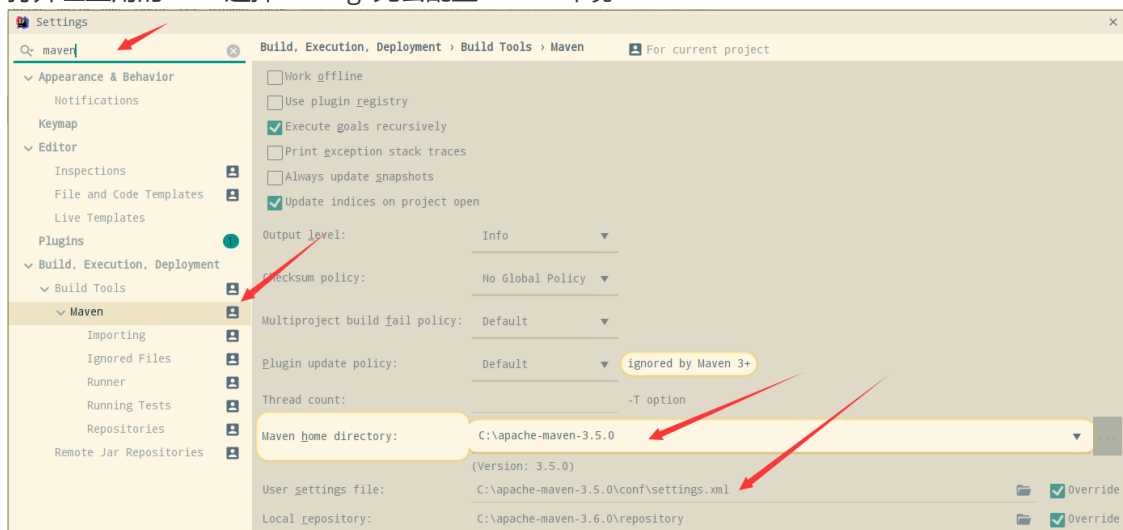13. 下载 Redis window版本 百度网盘下载地址 https://pan.baidu.com/s/1z1_OdNVbtgyEjiktqgB83g 密码：kdfq 解压完成后打开cmd指令窗口
3.输入你刚才解压的文件路径
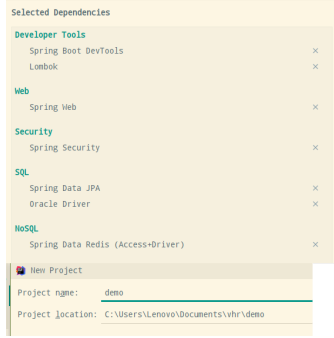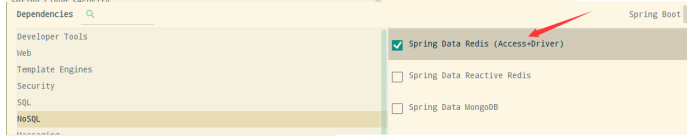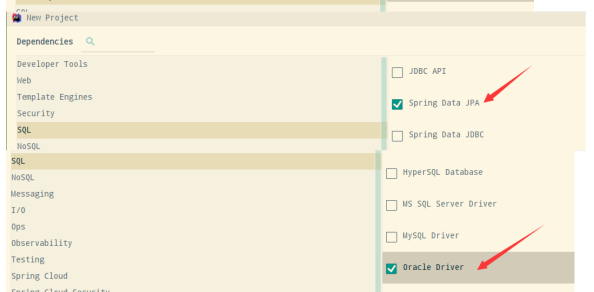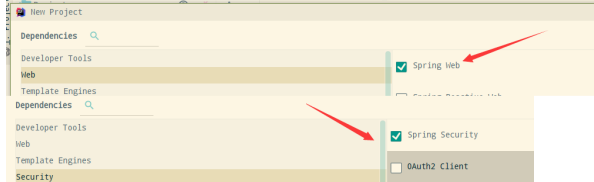4.然后输入redis-server redis.windows.conf 命令 5随后,进入右击此电脑–管理–服务和应用程序–服务 启动服务Redis
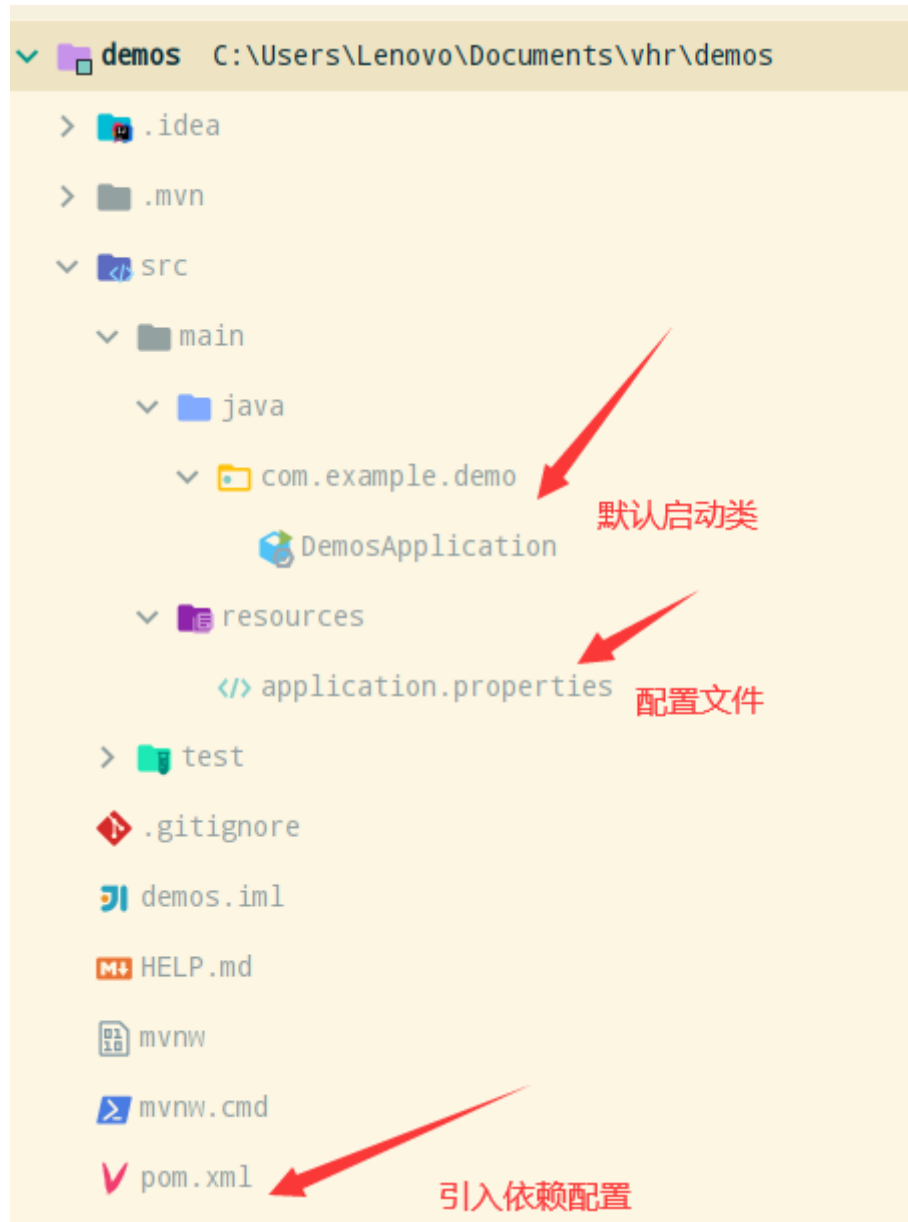
14. oracle数据库按照百度经验进行下载安装即可

15. 开发环境跟开发工具都已准备好可以打开IDEA

打开左上角的file---选择Settings先去配置maven环境

配置完成后按照图片新建项目

New Project

- Java
- Java Enterprise
- JBoss
- Clouds
- Spring
- Java FX
- Android
- IntelliJ Platform Plugin
- Spring Initializr
- Maven
- Gradle
- Groovy
- Grails
- Application Forge
- Kotlin
- Static Web
- Node.js and NPM
- Flash
- Empty Project

Project SDK: 1.8 (java version "1.8.0_191") 本地java版本

Choose Initializr Service URL.

Default: https://start.spring.io

Custom:

Make sure your network connection is active before continuing.

PREVIOUS NEXT CANCEL HELP

New Project

Project Metadata

Group: com.example

Artifact: demo

Type: Maven Project (Generate a Maven based project archive.) 默认选择为maven project

Language: Java 默认java语言

Packaging: Jar

Java Version: 8 选择java版本为8

Version: 0.0.1-SNAPSHOT

Name: demo

Description: Demo project for Spring Boot

Package: com.example.demo

PREVIOUS NEXT CANCEL HELP

New Project

Dependencies

- Developer Tools
  - Web
  - Template Engines
  - Security
  - SQL
  - NoSQL

选择热部署和Lombok 简化实体类Set Get方法

- Spring Boot DevTools
- Lombok
- Spring Configuration Processor

New Project

Dependencies

- Developer Tools
- Web
- Template Engines

- Spring Web

Dependencies

- Developer Tools
- Web
- Template Engines
- Security

- Spring Security
- OAuth2 Client

New Project

Dependencies

- Developer Tools
- Web
- Template Engines
- Security
- SQL
- NoSQL

SQL
- NoSQL
- Messaging
- I/O
- Ops
- Observability
- Testing
- Spring Cloud

- JDBC API
- Spring Data JPA
- Spring Data JDBC
- HyperSQL Database
- MS SQL Server Driver
- MySQL Driver
- Oracle Driver

Dependencies

- Developer Tools
- Web
- Template Engines
- Security
- SQL
- NoSQL

Spring Boot

- Spring Data Redis (Access+Driver)
- Spring Data Reactive Redis
- Spring Data MongoDB

Selected Dependencies

Developer Tools
- Spring Boot DevTools
- Lombok

Web
- Spring Web

Security
- Spring Security

SQL
- Spring Data JPA
- Oracle Driver

NoSQL
- Spring Data Redis (Access+Driver)

New Project

Project name: demo

Project location: C:\Users\Lenovo\Documents\vhr\demo

最终生成的项目结果如下

现在先去配置**POM.XML** 引入我们另外需要的依赖，这里需要特别注意的是maven官方库中没有oracle jdbc驱动的问题解决，先到oracle官网下载ojdbc6驱动jar，下载好后，运行maven命令安装jar到本地仓库，在需要引入的maven工程的pom.xml内增加引入即可，具体步骤的可以百度搜索。

```xml
<dependencies>
        <!--  JPA-->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <!-- redis-->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-redis</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.session</groupId>
        <artifactId>spring-session-data-redis</artifactId>
    </dependency>
    <!-- web-->
    <dependency>
```

```xml
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <!-- LOG-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-logging</artifactId>
            <version>2.3.4.RELEASE</version>
            <scope>compile</scope>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-actuator</artifactId>
        </dependency>
<!--            热部署-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>true</scope>
            <optional>true</optional>
        </dependency>

        <!-- oracle odbc -->
        <dependency>
            <groupId>com.oracle</groupId>
            <artifactId>ojdbc6</artifactId>
            <version>11.2.0.3</version>
        </dependency>

        <dependency>
            <groupId>com.jayway.jsonpath</groupId>
            <artifactId>json-path</artifactId>
        </dependency>
        <!-- 添加spring-boot-starter-security 依赖 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-taglibs</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-test</artifactId>
        </dependency>
        <!-- 添加 servlet 依赖. -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <scope>provided</scope>
        </dependency>

        <!-- 添加 JSTL（JSP Standard Tag Library，JSP标准标签库） -->
        <dependency>
            <groupId>javax.servlet</groupId>
```

```xml
            <artifactId>jstl</artifactId>
        </dependency>

        <!--添加 tomcat 的支持.-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-tomcat</artifactId>
            <scope>provided</scope>
        </dependency>

        <!-- Jasper是tomcat中使用的JSP引擎，运用tomcat-embed-jasper可以将项目与tomcat
分开 -->
        <dependency>
            <groupId>org.apache.tomcat.embed</groupId>
            <artifactId>tomcat-embed-jasper</artifactId>
            <scope>provided</scope>
        </dependency>
<!--        string转long类型与 字符串截取-->
        <dependency>
            <groupId>com.google.guava</groupId>
            <artifactId>guava</artifactId>
            <version>21.0</version>
        </dependency>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.2.5</version>
        </dependency>
<!--        简化实体类的 get set方法-->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>



<!--        测试-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
            <exclusions>
                <exclusion>
                    <groupId>org.junit.vintage</groupId>
                    <artifactId>junit-vintage-engine</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
```

```
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <configuration>
                <!-- 如果没有该项配置，devtools不会起作用，即应用不会restart -->
                <fork>true</fork>
            </configuration>
        </plugin>
    </plugins>
</build>
```

现在去配置**application.properties**，SpringBoot可以识别两种格式的配置文件，分别是yml文件与properties文件，可以将application.properties文件换成application.yml，application.properties默认放在：src/main/resource目录下，SpringBoot会自动加载

application.properties

配置端口号和内嵌Tomcat表单序列化size

```
#配置端口号 和 内嵌tomcat表单序列化size
debug=true
server.port=8081
server.tomcat.max-http-form-post-size=-1
```

配置本地Oracle数据库

```
#配置oracle驱动以及数据库用例
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521/deptyyc
spring.datasource.username=YYC
spring.datasource.password=YYC
```

配置日志信息

```
#logging configuration
#格式化，只能输出日期和内容
logging.level.org.springframework.boot.autoconfigure=ERROR
logging.pattern.console="%d - %msg%n"
#配置日志输出位置
logging.file.path=D:/
#配置日志输出文件 ，指定文件名
logging.file.name=D:/test.log
logging.file.max-size=10MB
```

配置Redis

```
#redis
# Redis数据库索引（默认为0）
spring.redis.database=0
# Redis服务器地址
spring.redis.host=127.0.0.1
# Redis服务器连接端口
spring.redis.port=6379
```

```
# Redis服务器连接密码（默认为空）
spring.redis.password=
# 连接池最大连接数（使用负值表示没有限制）
spring.redis.pool.max-active=8
# 连接池最大阻塞等待时间（使用负值表示没有限制）
spring.redis.pool.max-wait=-1
# 连接池中的最大空闲连接
spring.redis.pool.max-idle=8
# 连接池中的最小空闲连接
spring.redis.pool.min-idle=0
# 连接超时时间（毫秒）
spring.redis.timeout=0
```

配置JPA

```
#配置Spring Data JPA
#数据库类型
spring.jpa.database=oracle
#是否展示sql语句
spring.jpa.show-sql=false
#项目启动时，create为清空数据库并重新建表，update为更新
spring.jpa.hibernate.ddl-auto=update
spring.jpa.hibernate.naming-strategy=org.hibernate.cfg.ImprovedNamingStrategy
```

配置JSP servlet

```
#配置  MVC JSP servlet
spring.mvc.view.prefix=/
spring.mvc.view.suffix=.jsp
server.servlet.jsp.init-parameters.development=true
```

现在去引入静态资源JS,CSS,FONTS,IMG等，在resources下新建static文件夹放入各自的js，css，fonts，img文件夹即可

引入bootstrap，layui跟MaterializeUI跟对应的JQ即可
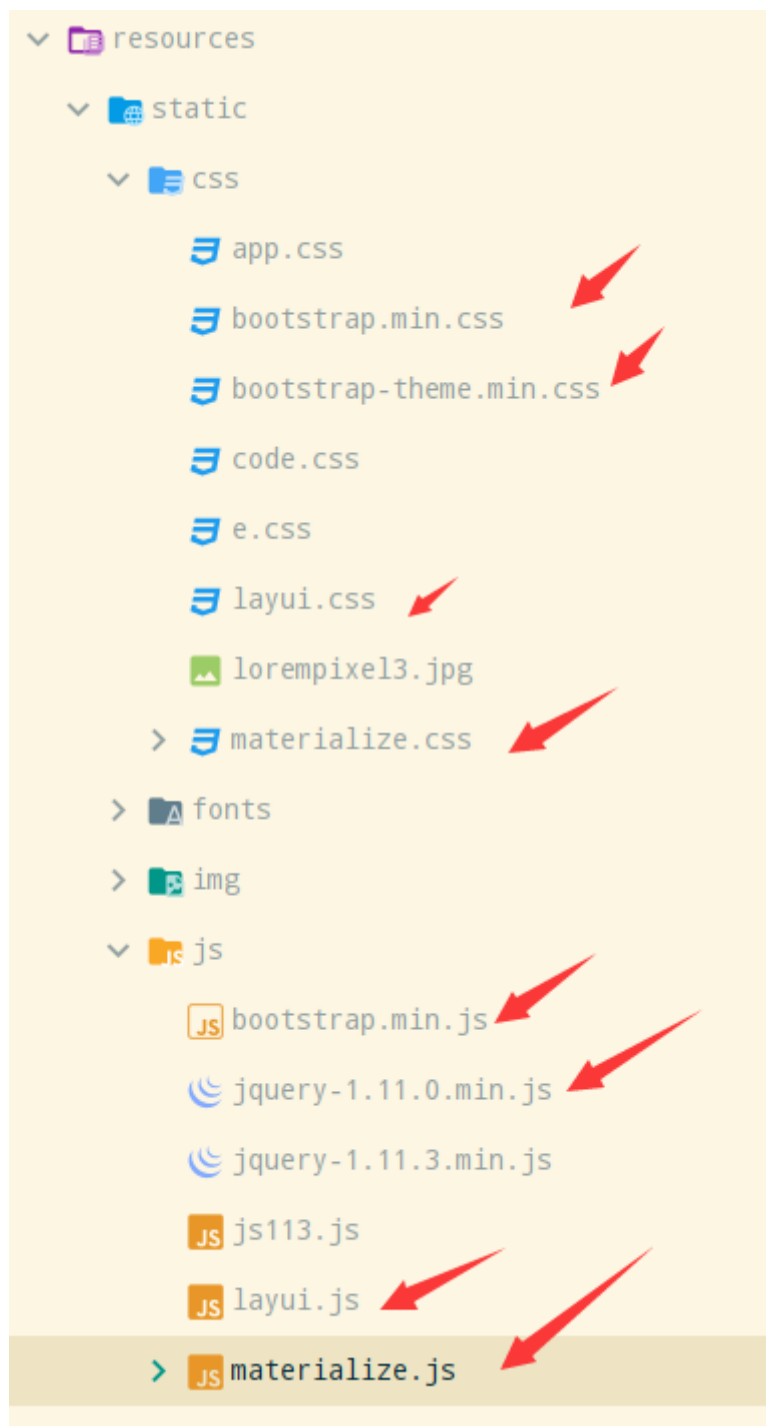
现在去配置分页标签commons.tld，NavigationTag，Page

在main目录下新建webapp文件夹，webapp目录下新建文件夹WEB-INF放置web.xml,WEB-INF目录下新建文件夹tld放置标签库配置文件commons.tld
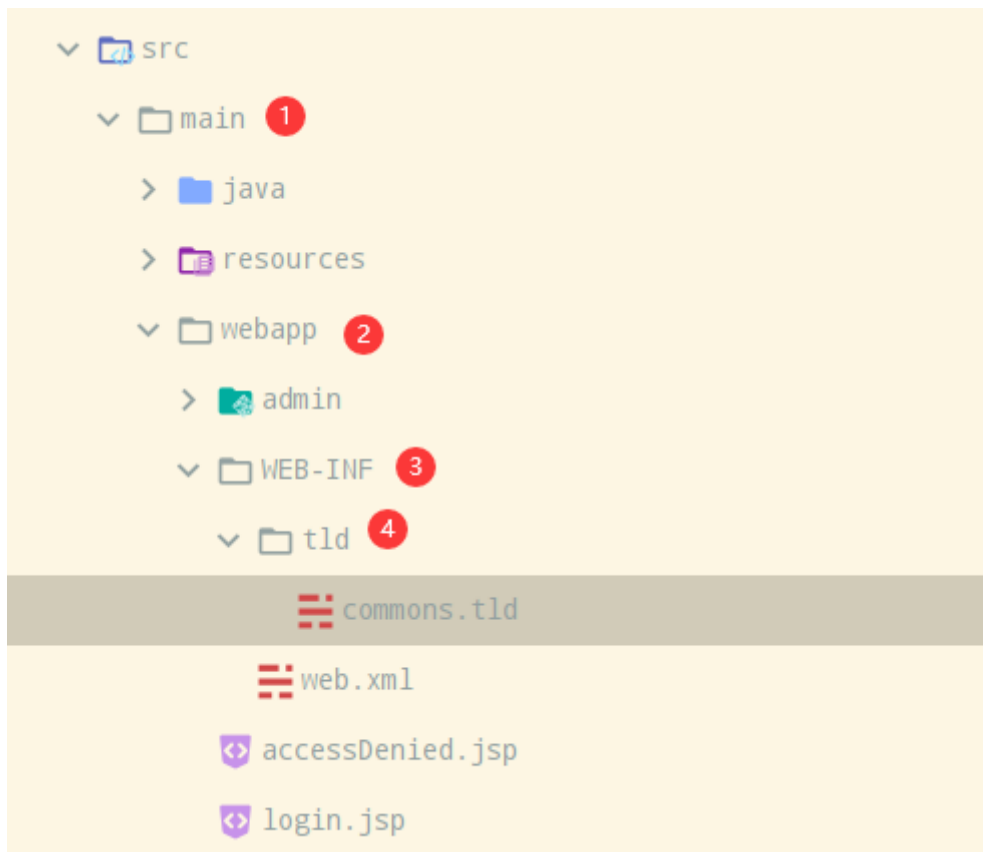
web.xml

```xml
<web-app>
    <display-name>Archetype Created Web Application</display-name>
    <welcome-file-list>
            <welcome-file>login.jsp</welcome-file>
        </welcome-file-list>
        <taglib>
            <taglib-uri>http://navigationTag.com/common/</taglib-uri>
            <taglib-location>/WEB-INF/tld/commons.tld</taglib-location>
        </taglib>
</web-app>
```

commons.tld

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE taglib
    PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
    "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">
<taglib>
    <!-- 指定标签库的版本号 -->
    <tlib-version>2.0</tlib-version>
    <!-- 指定JSP的版本号 -->
    <jsp-version>1.2</jsp-version>
    <!-- 指定标签库的名称 -->
    <short-name>common</short-name>
    <!-- 指定标签库的URI -->
    <uri>http://navigationTag.com/common/</uri>
    <!-- 指定标签库的显示名称 -->
    <display-name>Common Tag</display-name>
    <!-- 指定标签库的描述 -->
    <description>Common Tag library</description>
```

```xml
    <!-- 注册一个自定义标签 -->
    <tag>
        <!-- 指定注册的自定义标签名称 -->
        <name>page</name>
        <!-- 指定自定义标签的标签处理器类 -->
        <tag-class>com.example.demotest.util.NavigationTag</tag-class>
        <!-- 指定标签体类型 -->
        <body-content>JSP</body-content>
        <!-- 描述 -->
        <description>create navigation for paging</description>
        <!-- 指定标签中的属性 -->
        <attribute>
            <!-- 指定属性名称 -->
            <name>url</name>
            <!-- 该属性为true时表示其指定是属性为必须属性 -->
            <required>true</required>
            <!-- 该属性用于指定能不能使用表达式来动态指定数据，为true时表示可以 -->
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>bean</name>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>number</name>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
    </tag>
</taglib>
```

繁琐的配置文件搞定一半了接下来放入分页标签的工具类，新建工具类包utils放入PageData.java以及 NavigationTag.java，至此分页标签配置完毕

PageData.java

```java
import java.util.List;
public class PageData<T> {
    private int total;    // 总条数
    private int page;      // 当前页
    private int size;      // 每页数
    private List<T> rows; // 结果集
    public int getTotal() {
        return total;
    }
    public void setTotal(int total) {
        this.total = total;
    }
    public int getPage() {
        return page;
    }
    public void setPage(int page) {
        this.page = page;
    }
    public int getSize() {
        return size;
    }
}
```

```java
        public void setSize(int size) {
            this.size = size;
        }
        public List<T> getRows() {
            return rows;
        }
        public void setRows(List<T> rows) {
            this.rows = rows;
        }
    }
```

NavigationTag.java

```java
import java.io.IOException;
import java.util.Map;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.TagSupport;
/**
 * 显示格式：首页 上一页 1 2 3 4 5下一页 尾页
 */
public class NavigationTag extends TagSupport {
    static final long serialVersionUID = 2372405317744358833L;
    /**
     * request 中用于保存Page<E> 对象的变量名,默认为"page"
     */
    private String bean = "page";
    /**
     * 分页跳转的url地址,此属性必须
     */
    private String url = null;
    /**
     * 显示页码数量
     */
    private int number = 5;

    @Override
    public int doStartTag() throws JspException {

        System.out.println("doStartTag");
        JspWriter writer = pageContext.getOut();
        HttpServletRequest request =
                (HttpServletRequest) pageContext.getRequest();
        PageData page = (PageData) request.getAttribute(bean);
        if (page == null)
            return SKIP_BODY;
        url = resolveUrl(url, pageContext);
        try {
            // 计算总页数
            int pageCount = page.getTotal() / page.getSize();
            if (page.getTotal() % page.getSize() > 0) {
                pageCount++;
            }
            writer.print("<nav><ul class=\"pagination\">");
```

```java
            //首页链接路径
            String homeUrl = append(url, "page", 1);
            //末页链接路径
            String backUrl = append(url, "page", pageCount);
            // 显示"上一页"按钮
            if (page.getPage() > 1) {
                String preUrl = append(url, "page", page.getPage() - 1);
                preUrl = append(preUrl, "rows", page.getSize());
                writer.print("<li><a href=\"" + homeUrl + "\">" + "首页</a>
</li>");
                writer.print("<li><a href=\"" + preUrl + "\">" + "上一页</a>
</li>");
            } else {
                writer.print("<li class=\"disabled\"><a href=\"#\">" + "首页 </a>
</li>");
                writer.print("<li class=\"disabled\"><a href=\"#\">" + "上一页
</a></li>");
            }
            // 显示当前页码的前2页码和后两页码
            // 若1 则 1 2 3 4 5, 若2 则 1 2 3 4 5, 若3 则1 2 3 4 5,
            // 若4 则 2 3 4 5 6 ,若10 则 8 9 10 11 12
            int indexPage =1;
            if(page.getPage() - 2 <=0){
                indexPage=1;
            }else if(pageCount-page.getPage() <=2){
                indexPage=pageCount-4;
            }else{
                indexPage= page.getPage() - 2;
            }
            for (int i= 1;i <= number && indexPage <= pageCount;indexPage++,i++)
{
                if (indexPage == page.getPage()) {
                    writer.print("<li class=\"active\"><a href=\"#\">" +
indexPage
                        +"<spanclass=\"sr-only\"></span></a></li>");
                    continue;
                }
                String pageUrl = append(url, "page", indexPage);
                pageUrl = append(pageUrl, "rows", page.getSize());
                writer.print("<li><a href=\"" + pageUrl + "\">" + indexPage + "
</a></li>");
            }
            // 显示"下一页"按钮
            if (page.getPage() < pageCount) {
                String nextUrl = append(url, "page", page.getPage() + 1);
                nextUrl = append(nextUrl, "rows", page.getSize());
                writer.print("<li><a href=\"" + nextUrl + "\">" + "下一页</a>
</li>");
                writer.print("<li><a href=\"" + backUrl + "\">" + "尾页</a>
</li>");
            } else {
                writer.print("<li class=\"disabled\"><a href=\"#\">" + "下一页
</a></li>");
                writer.print("<li class=\"disabled\"><a href=\"#\">" + "尾页</a>
</li>");
            }
            writer.print("</nav>");
        } catch (IOException e) {
```

```java
                System.out.println("IOException");
                e.printStackTrace();
            }
            return SKIP_BODY;
    }

    private String append(String url, String key, int value) {
        return append(url, key, String.valueOf(value));
    }
    /**
     * 为url 参加参数对儿
     */
    private String append(String url, String key, String value) {
        if (url == null || url.trim().length() == 0) {
            return "";
        }
        if (url.indexOf("?") == -1) {
            url = url + "?" + key + "=" + value;
        } else {
            if (url.endsWith("?")) {
                url = url + key + "=" + value;
            } else {
                url = url + "&amp;" + key + "=" + value;
            }
        }
        return url;
    }
    /**
     * 为url 添加翻页请求参数
     */
    private String resolveUrl(String url,
                              javax.servlet.jsp.PageContext pageContext) throws
JspException {
        Map params = pageContext.getRequest().getParameterMap();
        for (Object key : params.keySet()) {
            if ("page".equals(key) || "rows".equals(key)){
                continue;
            }
            Object value = params.get(key);
            if (value == null){
                continue;
            }
            if (value.getClass().isArray()) {
                System.out.println("key.toString()"+key.toString());
                url = append(url, key.toString(), ((String[]) value)[0]);
            } else if (value instanceof String) {
                url = append(url, key.toString(), value.toString());
            }
        }
        return url;
    }
    public String getBean() {
        return bean;
    }
    public void setBean(String bean) {
        this.bean = bean;
    }
    public String getUrl() {
```

```
            return url;
        }
    public void setUrl(String url) {
        this.url = url;
    }
    public void setNumber(int number) {
        this.number = number;
    }
}
```

配置完分页该去配置Redis，在工具包下新建redis包放入RedisKeyUtil.java，对于配置文件要加@Component注释，@component（把普通pojo实例化到spring容器中，相当于配置文件中的 `<bean id="" class=""/>`）

```
import ch.qos.logback.classic.LoggerContext;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.core.StringRedisTemplate;
import org.springframework.data.redis.core.ValueOperations;
import org.springframework.stereotype.Component;
import java.text.MessageFormat;
import java.util.Iterator;
import java.util.List;
import java.util.Set;
import java.util.concurrent.TimeUnit;

@Component
public class RedisKeyUtil {
    private final Logger log= LoggerFactory.getLogger(LoggerContext.class);

    /** 字符串缓存模板 */
    @Autowired
    private StringRedisTemplate stringRedisTemplate;
    /** 对象，集合缓存模板  */
    @Autowired
    private RedisTemplate<Object, Object> redisTemplate;

    public void reset(String key, Long seconds){
        stringRedisTemplate.expire(key, seconds, TimeUnit.SECONDS);
    }

    /**
     * 获取匹配的key
     * @param pattern
     * @return Set<String>
     */
    public Set<String> keys(String pattern){
        return stringRedisTemplate.keys(pattern);
    }

    /**
     * 批量删除keys
     * @param pattern
     */
```

```java
public void delKeys(String pattern){
    redisTemplate.delete(stringRedisTemplate.keys(pattern));
}

/**
 * 添加Set集合
 * @param key
 * @param set
 */
public void addSet(String key ,Set<?> set){
    try{
        redisTemplate.opsForSet().add(key, set);
    }catch(Exception ex){
        ex.printStackTrace();
    }
}

/**
 * 获取Set集合
 *
 * @return
 */
public Set<?> getSet(String key){
    try{
        return redisTemplate.opsForSet().members(key);
    }catch(Exception ex){
        ex.printStackTrace();
    }
    return null ;
}

public String getString(String key) {
    String result = "";
    try {
        result = stringRedisTemplate.opsForValue().get(key);
    } catch (Exception e) {
        log.warn(spellString("getString {0}", key), e);
    }
    return result;
}

public  void delString(String key) {
    try {
        stringRedisTemplate.delete(key);
    } catch (Exception e) {
        log.warn(spellString("delString {0}", key),e);
    }
}

public void delAllString(String key) {
    if(key==null || "".equals(key)){
        return;
    }
    try {
        if (!key.endsWith("*")) {
            key += "*";
        }
        Set<String> keys = stringRedisTemplate.keys(key);
```

```java
                Iterator<String> it = keys.iterator();
                while (it.hasNext()) {
                    String singleKey = it.next();
                    delString(singleKey);
                }
        } catch (Exception e) {
            log.warn(spellString("delString {0}", key), e);
        }
    }


    public void addObj(String key ,Object obj, Long seconds){
        try {
            //对象redis存储
            ValueOperations<Object, Object> objOps =
redisTemplate.opsForValue();
            if(seconds!=null){
                objOps.set(key, obj, seconds, TimeUnit.SECONDS);
            }else{
                objOps.set(key, obj);
            }
        } catch (Exception e) {
            log.warn(spellString("addObj {0}={1},{2}", key,obj,seconds),e);
        }
    }



    /**
     * @param  key
     * @return Object
     */
    public Object getObject(String key) {
        Object object = null;
        try {
            object = redisTemplate.opsForValue().get(key);
        } catch (Exception e) {
            log.warn(spellString("getObj {0}", key), e);
        }
        return object;
    }



    /**

     * @return Object
     */
    @SuppressWarnings({ "unchecked"})
    public <T> T getObj(String key, T t) {
        Object o = null;
        try {
            o = redisTemplate.opsForValue().get(key);
        } catch (Exception e) {
            log.warn(spellString("getObj {0}->{1}", key, t), e);
        }
        return o == null ? null : (T) o;
    }
```

```java
public void expire(String key,long second){
    try {
        stringRedisTemplate.expire(key, second, TimeUnit.SECONDS);
    } catch (Exception e) {
        log.warn(spellString("expire {0}={1}", key, second),e);
    }
}

/**
 * @param key
 */
public void delObj(String key) {
    try {
        redisTemplate.delete(key);
    } catch (Exception e) {
        log.warn(spellString("delObj {0}", key),e);
    }
}



/**
 * 压栈
 *
 * @param key
 * @param value
 * @return
 */
public Long push(String key, String value) {
    Long result = 0l;
    try {
        result = stringRedisTemplate.opsForList().leftPush(key, value);
    } catch (Exception e) {
        log.warn(spellString("push {0}={1}", key,value),e);
    }
    return result;
}

/**
 * 出栈
 *
 * @param key
 * @return
 */
public String pop(String key) {
    String popResult = "";
    try {
        popResult = stringRedisTemplate.opsForList().leftPop(key);
    } catch (Exception e) {
        log.warn(spellString("pop {0}", key), e);
    }
    return popResult;
}

/**
 * 入队
 *
```

```java
 * @param key
 * @param value
 * @return
 */
public Long in(String key, String value) {
    Long inResult = 0l;
    try {
        inResult = stringRedisTemplate.opsForList().rightPush(key, value);
    } catch (Exception e) {
        log.warn(spellString("in {0}={1}", key, value), e);
    }
    return inResult;
}

/**
 * 出队
 *
 * @param key
 * @return
 */
public String out(String key) {
    String outResult = "";
    try {
        outResult = stringRedisTemplate.opsForList().leftPop(key);
    } catch (Exception e) {
        log.warn(spellString("out {0}", key),e);
    }
    return outResult;
}

/**
 * 栈/队列长
 *
 * @param key
 * @return
 */
public Long length(String key) {
    Long lengthResult = 0l;
    try {
        lengthResult = stringRedisTemplate.opsForList().size(key);
    } catch (Exception e) {
        log.warn(spellString("length {0}", key), e);
    }
    return lengthResult;
}

/**
 * 范围检索
 *
 * @param key
 * @param start
 * @param end
 * @return
 */
public List<String> range(String key, int start, int end) {
    List<String> rangeResult = null;
    try {
```

```java
            rangeResult = stringRedisTemplate.opsForList().range(key, start,
end);
        } catch (Exception e) {
            log.warn(spellString("range {0},{1}-{2}", key, start, end), e);
        }
        return rangeResult;
    }

    /**
     * 移除
     *
     * @param key
     * @param i
     * @param value
     */
    public void remove(String key, long i, String value) {
        try {
            stringRedisTemplate.opsForList().remove(key, i, value);
        } catch (Exception e) {
            log.warn(spellString("remove {0}={1},{2}", key,value,i),e);
        }
    }

    /**
     * 检索
     *
     * @param key
     * @param index
     * @return
     */
    public String index(String key, long index) {
        String indexResult = "";
        try {
            indexResult = stringRedisTemplate.opsForList().index(key, index);
        } catch (Exception e) {
            log.warn(spellString("index {0}", key), e);
        }
        return indexResult;
    }

    /**
     * 置值
     *
     * @param key
     * @param index
     * @param value
     */
    public void setObject(String key,  Object value,long index) {
        try {
            redisTemplate.opsForValue().set(key,value,index);
        } catch (Exception e) {
            log.warn(spellString("set {0}={1},{2}", key,value,index),e);
        }
    }


    public boolean setString(String key,String value ){
        try {
```

```
                stringRedisTemplate.opsForValue().set(key,value);
                return true;
        } catch (Exception e) {
                e.printStackTrace();
                return false;
        }
    }




    /**
     * 裁剪
     *
     * @param key
     * @param start
     * @param end
     */
    public void trim(String key, long start, int end) {
        try {
                stringRedisTemplate.opsForList().trim(key, start, end);
        } catch (Exception e) {
                log.warn(spellString("trim {0},{1}-{2}", key,start,end),e);
        }
    }

    /**
     * 方法说明：原子性自增
     * @param key 自增的key
     * @param value 每次自增的值
     * @time: 2017年3月9日 下午4:28:21
     * @return: Long
     */
    public Long incr(String key, long value) {
        Long incrResult = 0L;
        try {
                incrResult = stringRedisTemplate.opsForValue().increment(key,
value);
        } catch (Exception e) {
                log.warn(spellString("incr {0}={1}", key, value), e);
        }
        return incrResult;
    }

    /**
     * 拼异常内容
     * @param errStr
     * @param arguments
     * @return
     */
    private String spellString(String errStr,Object ... arguments){
        return MessageFormat.format(errStr,arguments);
    }

}
```

由于在JPA中配置实体类ID自增映射Oracle数据库自增字段比较麻烦，所以采用雪花算法生成Long类型ID，但因为一般雪花算法生成的是18位ID，生成Long类型的长Id返回给前端精度丢失，JS 使用 IEEE 754 的双精度数表示数字，1 位符号，10 位指数，53 位底数。所以 JS 数字精度近似为 15.95 位 10 进制（`10 ** 15.95`）。因此需要对雪花算法进行调整生成15位，放置SystemClock.java和SequenceUtils.java

SystemClock

```java
package com.example.demotest.util.snowflakeId;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicLong;

/**
 * 缓存时间戳解决System.currentTimeMillis()高并发下性能问题<br>
 *      问题根源分析：http://pzemtsov.github.io/2017/07/23/the-slow-
currenttimemillis.html
 **/
public class SystemClock {

    private final long period;
    private final AtomicLong now;

    private SystemClock(long period) {
        this.period = period;
        this.now = new AtomicLong(System.currentTimeMillis());
        scheduleClockUpdating();
    }

    /**
     * 尝试下枚举单例法
     */
    private enum SystemClockEnum {
        SYSTEM_CLOCK;
        private SystemClock systemClock;
        SystemClockEnum() {
            systemClock = new SystemClock(1);
        }
        public SystemClock getInstance() {
            return systemClock;
        }
    }

    /**
     * 获取单例对象
     * @return com.cmallshop.module.core.commons.util.sequence.SystemClock
     */
    private static SystemClock getInstance() {
        return SystemClockEnum.SYSTEM_CLOCK.getInstance();
    }

    /**
     * 获取当前毫秒时间戳
     * @return long
     */
    public static long now() {
```

```java
        return getInstance().now.get();
    }

    /**
     * 起一个线程定时刷新时间戳
     */
    private void scheduleClockUpdating() {
        ScheduledExecutorService scheduler =
Executors.newSingleThreadScheduledExecutor(runnable -> {
            Thread thread = new Thread(runnable, "System Clock");
            thread.setDaemon(true);
            return thread;
        });
        scheduler.scheduleAtFixedRate(() -> now.set(System.currentTimeMillis()),
period, period, TimeUnit.MILLISECONDS);
    }

}
```

SequenceUtils

```java
package com.example.demotest.util.snowflakeId;

import lombok.extern.slf4j.Slf4j;

import java.net.NetworkInterface;
import java.net.SocketException;
import java.util.Enumeration;

import static java.net.NetworkInterface.getNetworkInterfaces;

/**
 * 雪花算法分布式唯一ID生成器<br>
 * 每个机器号最高支持每秒65535个序列，当秒序列不足时启用备份机器号，若备份机器也不足时借用备份
机器下一秒可用序列<br>
 * 53 bits 趋势自增ID结构如下:
 *
 * |00000000|00011111|11111111|11111111|11111111|11111111|11111111|11111111|
 * |-----------|##########32bit 秒级时间戳##########|-----|-----------------|
 * |-------------------------------------5bit机器位|xxxxx|-----------------|
 * |-------------------------------------16bit自增序列|xxxxxxxx|xxxxxxxx|
 **/
@Slf4j
public class SequenceUtils {

    /** 初始偏移时间戳 */
    private static final long OFFSET = 1546300800L;

    /** 机器id (0~15 保留 16~31作为备份机器) */
    private static long WORKER_ID = 0;
    /** 机器id所占位数 (5bit, 支持最大机器数 2^5 = 32)*/
    private static final long WORKER_ID_BITS = 5L;
    /** 自增序列所占位数 (16bit, 支持最大每秒生成 2^16 = 65536) */
    private static final long SEQUENCE_ID_BITS = 16L;
    /** 机器id偏移位数 */
    private static final long WORKER_SHIFT_BITS = SEQUENCE_ID_BITS;
    /** 自增序列偏移位数 */
```

```java
    private static final long OFFSET_SHIFT_BITS = SEQUENCE_ID_BITS +
WORKER_ID_BITS;
    /** 机器标识最大值 (2^5 / 2 - 1 = 15) */
    private static final long WORKER_ID_MAX = ((1 << WORKER_ID_BITS) - 1) >> 1;
    /** 备份机器ID开始位置 (2^5 / 2 = 16) */
    private static final long BACK_WORKER_ID_BEGIN = (1 << WORKER_ID_BITS) >> 1;
    /** 自增序列最大值 (2^16 - 1 = 65535) */
    private static final long SEQUENCE_MAX = (1 << SEQUENCE_ID_BITS) - 1;
    /** 发生时间回拨时容忍的最大回拨时间（秒）*/
    private static final long BACK_TIME_MAX = 1L;

    /** 上次生成ID的时间戳（秒）*/
    private static long lastTimestamp = 0L;
    /** 当前秒内序列 (2^16)*/
    private static long sequence = 0L;
    /** 备份机器上次生成ID的时间戳（秒）*/
    private static long lastTimestampBak = 0L;
    /** 备份机器当前秒内序列 (2^16)*/
    private static long sequenceBak = 0L;

    {
        // 初始化机器ID
        // 伪代码：由你的配置文件获取节点ID
        long workerId = this.getMachineNum();
        if (workerId < 0 || workerId > WORKER_ID_MAX) {
            throw new IllegalArgumentException(String.format("cmallshop.workerId
范围: 0 ~ %d 目前: %d", WORKER_ID_MAX, workerId));
        }
        WORKER_ID = workerId;
    }



    /** 私有构造函数禁止外部访问 */
    private SequenceUtils() {}

    /**
     * 获取自增序列
     * @return long
     */
    public static long nextId() {
        return nextId(SystemClock.now() / 1000);
    }

    /**
     * 主机器自增序列
     * @param timestamp 当前Unix时间戳
     * @return long
     */
    private static synchronized long nextId(long timestamp) {
        // 时钟回拨检查
        if (timestamp < lastTimestamp) {
            // 发生时钟回拨
            log.warn("时钟回拨，启用备份机器ID: now: [{}] last: [{}]", timestamp,
lastTimestamp);
            return nextIdBackup(timestamp);
        }
```

```java
            // 开始下一秒
            if (timestamp != lastTimestamp) {
                lastTimestamp = timestamp;
                sequence = 0L;
            }
            if (0L == (++sequence & SEQUENCE_MAX)) {
                // 秒内序列用尽
//                log.warn("秒内[{}]序列用尽，启用备份机器ID序列", timestamp);
                sequence--;
                return nextIdBackup(timestamp);
            }

            return ((timestamp - OFFSET) << OFFSET_SHIFT_BITS) | (WORKER_ID <<
WORKER_SHIFT_BITS) | sequence;
    }

    /**
     * 备份机器自增序列
     * @param timestamp timestamp 当前Unix时间戳
     * @return long
     */
    private static long nextIdBackup(long timestamp) {
        if (timestamp < lastTimestampBak) {
            if (lastTimestampBak - SystemClock.now() / 1000 <= BACK_TIME_MAX) {
                timestamp = lastTimestampBak;
            } else {
                throw new RuntimeException(String.format("时钟回拨: now: [%d]
last: [%d]", timestamp, lastTimestampBak));
            }
        }

        if (timestamp != lastTimestampBak) {
            lastTimestampBak = timestamp;
            sequenceBak = 0L;
        }

        if (0L == (++sequenceBak & SEQUENCE_MAX)) {
            // 秒内序列用尽
//              logger.warn("秒内[{}]序列用尽，备份机器ID借取下一秒序列", timestamp);
            return nextIdBackup(timestamp + 1);
        }

        return ((timestamp - OFFSET) << OFFSET_SHIFT_BITS) | ((WORKER_ID ^
BACK_WORKER_ID_BEGIN) << WORKER_SHIFT_BITS) | sequenceBak;
    }

    /**

     * 获取机器编码

     * @return

     */

    private  long getMachineNum(){

        long machinePiece;
```

```
        StringBuilder sb = new StringBuilder();

        Enumeration<NetworkInterface> e = null;

        try {

            e = getNetworkInterfaces();

        } catch (SocketException e1) {

            e1.printStackTrace();

        }

        while (e.hasMoreElements()) {

            NetworkInterface ni = e.nextElement();

            sb.append(ni.toString());

        }

        machinePiece = sb.toString().hashCode();

        return machinePiece;

    }
}
```
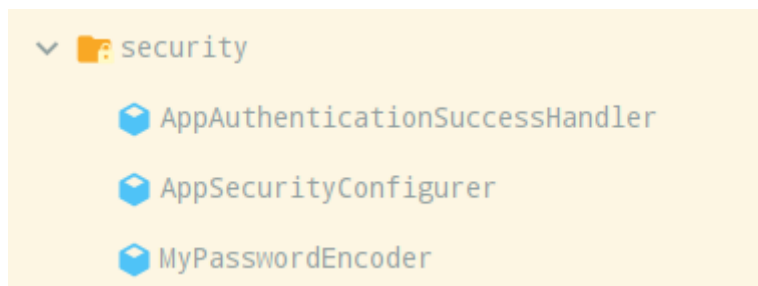
配置Security，引入Security修改一些参数即可

MyPasswordEncoder密码加密

```
import org.springframework.security.crypto.password.PasswordEncoder;
public class MyPasswordEncoder implements PasswordEncoder{

    @Override
    public String encode(CharSequence arg0) {
        return arg0.toString();
    }

    @Override
    public boolean matches(CharSequence arg0, String arg1) {
        return arg1.equals(arg0.toString());
    }

}
```

自定义Spring Security认证处理类

```java
import com.example.demotest.Service.AuthService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationProvider;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.password.PasswordEncoder;

/**
 * 自定义Spring Security认证处理类的时候
 * 我们需要继承自WebSecurityConfigurerAdapter来完成，相关配置重写对应 方法即可。
 * */
@Configuration
public class AppSecurityConfigurer extends WebSecurityConfigurerAdapter{

    // 依赖注入用户服务类
    @Autowired
    private AuthService authService;

    // 依赖注入加密接口
    @Autowired
    private PasswordEncoder passwordEncoder;

    // 依赖注入用户认证接口
    @Autowired
    private AuthenticationProvider authenticationProvider;

    // 依赖注入认证处理成功类，验证用户成功后处理不同用户跳转到不同的页面
    @Autowired
    AppAuthenticationSuccessHandler appAuthenticationSuccessHandler;



    // DaoAuthenticationProvider是Spring Security提供AuthenticationProvider的实现
    @Bean
    public AuthenticationProvider authenticationProvider() {
        System.out.println("AuthenticationProvider authenticationProvider");
        // 创建DaoAuthenticationProvider对象
        DaoAuthenticationProvider provider = new DaoAuthenticationProvider();
        // 不要隐藏"用户未找到"的异常
        provider.setHideUserNotFoundExceptions(false);
        // 通过重写configure方法添加自定义的认证方式。
        provider.setUserDetailsService(authService);
        // 设置密码加密程序认证
        provider.setPasswordEncoder(passwordEncoder);
```

```java
            return provider;
    }


    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception
{
        System.out.println("AppSecurityConfigurer configure auth......");
        // 设置认证方式。
        auth.authenticationProvider(authenticationProvider);

    }

    /**
     * 设置了登录页面，而且登录页面任何人都可以访问，然后设置了登录失败地址，也设置了注销请求，
注销请求也是任何人都可以访问的。
     * permitAll表示该请求任何人都可以访问，.anyRequest().authenticated(),表示其他的请
求都必须要有权限认证。
     * */
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        System.out.println("AppSecurityConfigurer configure http......");
        http.sessionManagement()
                .sessionCreationPolicy(SessionCreationPolicy.IF_REQUIRED)
                .invalidSessionUrl("/invalidSession.html");

        http.headers().frameOptions().disable();
        http.authorizeRequests()
                // spring-security 5.0 之后需要过滤静态资源

.antMatchers("/goRegisterPage","/login","/static/**","/css/**","/js/**","/img/*"
,"/fonts/**").permitAll()// 过滤静态资源，所有人可以访问
                .antMatchers("/").hasAnyRole("USER","ADMIN")//配置ROLE有权访问的路
径
                .antMatchers("/admin/**").hasRole("ADMIN")
                .anyRequest().authenticated()
                .and()

.formLogin().loginPage("/login").successHandler(appAuthenticationSuccessHandler)
//配置登录页面访问的路径
                .usernameParameter("username").passwordParameter("password")  //
配置登录页面 所需用户名和密码
                .and()
                .logout().permitAll()
                .and()
                .exceptionHandling().accessDeniedPage("/accessDenied")
                .and()
                .csrf().disable();
    }

}
```

```java
/**
 * 设置了登录页面，而且登录页面任何人都可以访问，然后设置了登录失败地址，也设置了注销请求，注销请求也是任何人都可以访问的。
 * permitAll 表示该请求任何人都可以访问，.anyRequest().authenticated(),表示其他的请求都必须要有权限认证。
 * */
@Override
protected void configure(HttpSecurity http) throws Exception {
    System.out.println("AppSecurityConfigurer configure http......");
    http.sessionManagement()
            .sessionCreationPolicy(SessionCreationPolicy.IF_REQUIRED)
            .invalidSessionUrl("/invalidSession.html");

    http.headers().frameOptions().disable();
    http.authorizeRequests() ExpressionUrlAuthorizationConfigurer<H>.ExpressionInterceptUrlRegistry
            // spring-security 5.0 之后需要过滤静态资源                                            静态资源过滤
            .antMatchers( ...antPatterns: "/goRegisterPage","/login","/static/**","/css/**","/js/**","/img/*","/fonts/**").permitAll()
            .antMatchers( ...antPatterns: "/", "/user/**").hasAnyRole( ...roles: "USER","ADMIN")
            .antMatchers( ...antPatterns: "/admin/**").hasRole("ADMIN")        路径鉴权
            .anyRequest().authenticated()
            .and() HttpSecurity
            .formLogin().loginPage("/login").successHandler(appAuthenticationSuccessHandler) FormLoginConfigurer<HttpSecurity>
            .usernameParameter("username").passwordParameter("password")
            .and() HttpSecurity
            .logout().permitAll() LogoutConfigurer<HttpSecurity>
            .and() HttpSecurity
            .exceptionHandling().accessDeniedPage("/accessDenied") ExceptionHandlingConfigurer<HttpSecurity>
            .and() HttpSecurity           无权限跳转路径
            .csrf().disable();
}
```

AppAuthenticationSuccessHandler 负责所有重定向事务

```java
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.web.DefaultRedirectStrategy;
import org.springframework.security.web.RedirectStrategy;
import
org.springframework.security.web.authentication.SimpleUrlAuthenticationSuccessHa
ndler;
import org.springframework.stereotype.Component;
@Component
public class AppAuthenticationSuccessHandler extends
SimpleUrlAuthenticationSuccessHandler{
    // Spring Security 通过RedirectStrategy对象负责所有重定向事务
    private RedirectStrategy redirectStrategy = new DefaultRedirectStrategy();

    /*
     * 重写handle方法，方法中通过RedirectStrategy对象重定向到指定的url
     * */
    @Override
    protected void handle(HttpServletRequest request, HttpServletResponse
response,
                          Authentication authentication)
            throws IOException {
        // 通过determineTargetUrl方法返回需要跳转的url
        String targetUrl = determineTargetUrl(authentication);
        // 重定向请求到指定的url
        redirectStrategy.sendRedirect(request, response, targetUrl);
    }
```

```java
    /*
     * 从Authentication对象中提取角色提取当前登录用户的角色，并根据其角色返回适当的URL。
     */
    protected String determineTargetUrl(Authentication authentication) {
        String url = "";

        // 获取当前登录用户的角色权限集合
        Collection<? extends GrantedAuthority> authorities =
authentication.getAuthorities();

        List<String> roles = new ArrayList<String>();

        // 将角色名称添加到List集合
        for (GrantedAuthority a : authorities) {
            roles.add(a.getAuthority());
        }

        // 判断不同角色跳转到不同的url
        if (isUser(roles)) {
            url = "/user/indexR";
        } else if (isAdmin(roles)) {
            url = "/admin/indexA";
        }else {
            url = "/accessDenied";
        }
        System.out.println("url = " + url);
        return url;
    }

    private boolean isUser(List<String> roles) {
        if (roles.contains("ROLE_USER")) {
            return true;
        }
        return false;
    }

    private boolean isAdmin(List<String> roles) {
        if (roles.contains("ROLE_ADMIN")) {
            return true;
        }
        return false;
    }

    public void setRedirectStrategy(RedirectStrategy redirectStrategy) {
        this.redirectStrategy = redirectStrategy;
    }

    protected RedirectStrategy getRedirectStrategy() {
        return redirectStrategy;
    }
}
```

现在权限验证也弄完了，该回到系统主题了，使用Spring data jpa框架时需要先去构建实体类，配合权限验证需要role类，admin类，其次需要role和admin之间的多对多关系，还需要User类也就是员工类，由于使用了lombok,因此只需要显式的注释@Data来表明Setter() Getter()方法，而使用JPA生成表以及映射表时需要@Entity注明实体，@Table(name="tb_admin")注明表名，类需要implements Serializable，主键字段注明@Id

Role.java 角色表

```java
@Data
@Entity
@Table(name="tb_role")
public class Role implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @Column(name="id")
    private Long id;   //主键id
    @Column(name="authority")  //指定生成字段名称
    private String authority;//角色
    private String authorityDesc;//角色描述

    public Role() {
        super();
        // TODO Auto-generated constructor stub
    }

}
```

Admin.java 管理员表，需要配置与Role的多对多关系

```java
@Data
@Entity
@Table(name="tb_admin")
public class Admin implements Serializable{

    private static final long serialVersionUID = 1L;
    @Id
    private Long id;
    private String username;
    private String password;

    @ManyToMany(cascade = {CascadeType.REFRESH},fetch = FetchType.EAGER)  //级联
    @JoinTable(name="tb_admin_role",    //多对多生成的表
            joinColumns={@JoinColumn(name="aid")},
            inverseJoinColumns={@JoinColumn(name="roleid")})
    private List<Role> roles;
}
```

User.java 职员表

```java
@Data
@Entity
@JsonIgnoreProperties(value = { "hibernateLazyInitializer", "handler" })
@Table(name="tb_users")
public class Users implements Serializable{

    private static final long serialVersionUID = 1L;
    @Id
    private Long id;          //雪花id  用法 long id = SnowFlake.nextId();
    private String username; //员工姓名
    private String password; //密码
```

```java
    private String sex;        //性别
    private Long eid;        //员工工号
    private String job;        //职位
    private Float basicSalary; //基本工资
    private Float welfarePoints;    //剩余积分
    private String phone;      //手机号
    private String idcard;    //身份证号
    @CreatedDate
    @JSONField(name = "joinTime",format = "yyyy-MM-dd")
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    private Date joinTime;    //入职时间
//    @DateTimeFormat()
    @JSONField(name = "leaveTime",format = "yyyy-MM-dd")
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    private Date leaveTime;    //离职时间
    private String address;   //地址
    private Boolean isWork;   //是否在职

    @ManyToOne( fetch=FetchType.EAGER,
            targetEntity=Role.class)
    @JoinColumn(name="rid",referencedColumnName="id")
    private Role role;

    @ManyToOne( fetch=FetchType.EAGER,
            targetEntity=Salary.class)
    @JoinColumn(name="sid",referencedColumnName="id")
    private Salary salary;
}
```
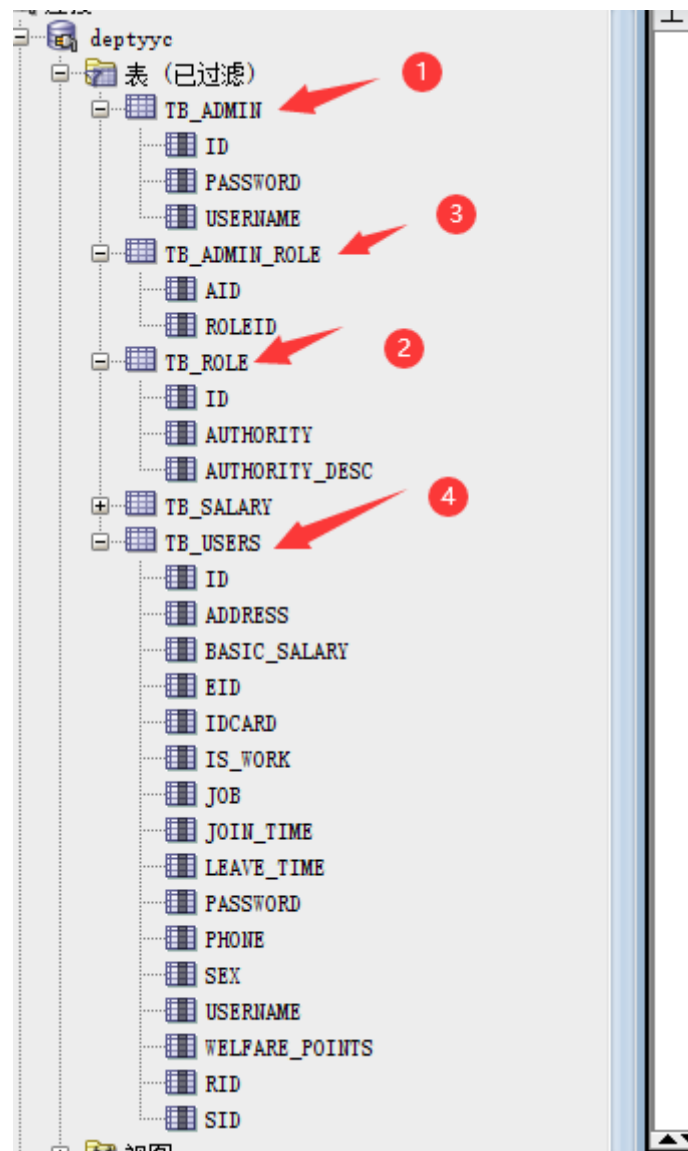
实体类完成，在数据库中建一个数据库，并把配置文件application.properties中JPA配置修改为

```
spring.jpa.hibernate.ddl-auto=create
```

直接启动项目，让JPA生成数据库表后，关闭项目，把配置文件application.properties中JPA配置修改为

```
spring.jpa.hibernate.ddl-auto=update
```

这样子首次启动项目时生成表，后面启动项目时不在重新生成表，ddl-auto=create意味着项目启动即清空表并删除原有表重新生成表

查看数据库中JPA已经生成了四个表，现在为TB_ROLE表，TB_ADMIN表，TB_ADMIN_ROLE表插入数据　TB_ROLE表中AUTHORITY字段一定要对应Security中配置的ROLE

```java
@Override
protected void configure(HttpSecurity http) throws Exception {
    System.out.println("AppSecurityConfigurer configure http......");
    http.sessionManagement()
            .sessionCreationPolicy(SessionCreationPolicy.IF_REQUIRED)
            .invalidSessionUrl("/invalidSession.html");

    http.headers().frameOptions().disable();
    http.authorizeRequests() ExpressionUrlAuthorizationConfigurer<H>.ExpressionInterceptUrlRegistry
            // spring-security 5.0 之后需要过滤静态资源
            .antMatchers( ...antPatterns: "/goRegisterPage","/login","/static/**","/css/**","/js/**","/img/*",
            .antMatchers( ...antPatterns: "/", "/user/**").hasAnyRole( ...roles: "USER","ADMIN")
            .antMatchers( ...antPatterns: "/admin/**").hasRole("ADMIN")
            .anyRequest().authenticated()

    // 判断不同角色跳转到不同的url
    if (isUser(roles)) {
        url = "/user/indexR";
    } else if (isAdmin(roles)) {
        url = "/admin/indexA";
    }else {
        url = "/accessDenied";
    }
    System.out.println("url = " + url);
    return url;
}

private boolean isUser(List<String> roles) {
    if (roles.contains("ROLE_USER")) {
        return true;
    }
    return false;
}

private boolean isAdmin(List<String> roles) {
    if (roles.contains("ROLE_ADMIN")) {
        return true;
    }
    return false;
}
```

```sql
Insert into YYC.TB_ADMIN (ID,PASSWORD,USERNAME) values
(1,'$2a$10$wy95RkpcCOGkLkGkN..Loej480aRBRHOKy7IG.BrXiqpINQ1IOAcW','1');
Insert into YYC.TB_ROLE (ID,AUTHORITY,AUTHORITY_DESC) values (1,'ROLE_ADMIN','管
理员权限');
Insert into YYC.TB_ROLE (ID,AUTHORITY,AUTHORITY_DESC) values (2,'ROLE_USER','普通
用户权限');
Insert into YYC.TB_ADMIN_ROLE (AID,ROLEID) values (1,1);
```

回到Spring boot项目，构建Repository层，Service层，Controller层，这里只对Users职员做详细介绍

UsersDao，注意这里只对3个逻辑复杂的查询方法做了原生查询，其他基本上按照符合JPA接口的命名规范写，方法名称规则既然是基于方法命名规则，那么肯定有一套规则约束方法的命名：

findBy（关键字）＋属性名称（首字母大写）＋查询条件（首字母大写），JPA会生成JPQL语言操作实体类进行SQL操作，添加，修改，删除默认调用提供的接口即可

以下面的表格的形式给出：

关键字	*方法命名*	*sql where字句*
And	findByNameAndPwd	where name= ? and pwd =?
Or	findByNameOrSex	where name= ? or sex=?
Is,Equals	findById,findByIdEquals	where id= ?
Between	findByIdBetween	where id between ? and ?
LessThan	findByIdLessThan	where id < ?
LessThanEquals	findByIdLessThanEquals	where id <= ?
GreaterThan	findByIdGreaterThan	where id > ?
GreaterThanEqual	findByAgeGreaterThanEqual	where age >= ?
After	findByIdAfter	where id > ?
Before	findByIdBefore	where id < ?
IsNull	findByNameIsNull	where name is null
isNotNull,NotNull	findByNameNotNull	where name is not null
Like	findByNameLike	where name like ?
NotLike	findByNameNotLike	where name not like ?
StartingWith	findByNameStartingWith	where name like '?%'
EndingWith	findByNameEndingWith	where name like '%?'
Containing	findByNameContaining	where name like '%?%'
OrderBy	findByIdOrderByXDesc	where id=? order by x desc
Not	findByNameNot	where name <> ?
In	findByIdIn(Collection<?> c)	where id in (?)
NotIn	findByIdNotIn(Collection<?> c)	where id not in (?)
TRUE	findByAaaTue	where aaa = true
FALSE	findByAaaFalse	where aaa = false
IgnoreCase	findByNameIgnoreCase	where UPPER(name)=UPPER(?)
top	findTop100	top 10/where ROWNUM <=10

```
import com.example.demotest.Entity.Users;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.JpaSpecificationExecutor;
import org.springframework.data.jpa.repository.Query;
import java.util.List;
import java.util.Map;
import java.util.Optional;
import java.util.Set;
```

```java
                                                  //Users为实体类名称，Long为主键Id类型，
JpaSpecificationExecutor为分页查询
public interface UsersDao extends JpaRepository<Users,Long> ,
JpaSpecificationExecutor<Users> {
    /**
     * 登录鉴权使用
     * @param username
     * @return
     */
    Optional<Users> findByUsername(String username);

    /**
     * 分职位查询在职的员工数量  饼图
     * @param job
     * @param iswork
     * @return
     */
    Long countByJobAndIsWork(String job,Boolean iswork);

    /**
     * 查询在职员工工资分布情况  柱形图
     * @param iswork
     * @param down
     * @param up
     * @return
     */
    Long countByIsWorkAndBasicSalaryBetween(Boolean iswork,Float down,Float up);

    /**
     * 查询在职员工数量
     * @param isWork
     * @return
     */
    Long countByIsWork(Boolean isWork);

    /**
     * 添加新员工时查询最大的工号
     * @return
     */
    @Query("select  coalesce(max(u.eid),0)  from Users u ")
    Long getMaxEiD( );

    /**
     * 查询最近一个月入职的员工总数
     * @return
     */
    @Query(value = "select count(id) AS total from TB_USERS where
JOIN_TIME>systimestamp-interval'30'day",nativeQuery = true)
    Long countRecentlyJoinedWorker();
    /**
     * 查询最近一个月离职的员工总数
     * @return
     */
    @Query(value = "select count(id) AS total from TB_USERS where
LEAVE_TIME>systimestamp-interval'30'day",nativeQuery = true)
    Long countRecentlyLeaveWorker();

    /**
```

```
     *  查询今年十二个月离职和入职的同事分布情况    折线图
     * @return
     */
    @Query(value = "select t2.datevalue as timeDateMM, nvl(t1.count, 0) as
INScanCount ,nvl(t0.count, 0) as OutScanCount \n" +
            " from (select count(*) as count, month_time\n" +
            "        from (select to_char(TB_USERS.LEAVE_TIME, 'MM') as
month_time\n" +
            "              from TB_USERS)\n" +
            "       group by month_time\n" +
            "       order by month_time asc) t0,\n" +
            "       \n" +
            "       (select count(*) as count, month_time\n" +
            "        from (select to_char(TB_USERS.JOIN_TIME, 'MM') as
month_time\n" +
            "              from TB_USERS)\n" +
            "       group by month_time\n" +
            "       order by month_time asc) t1,\n" +
            "       \n" +
            "      (select '' || lpad(level, 2, 0) datevalue\n" +
            "       from dual\n" +
            "       connect by level < 13) t2\n" +
            "       \n" +
            " where t0.month_time(+) = t2.datevalue AND t1.month_time(+) =
t2.datevalue\n" +
            " order by t2.datevalue\n",nativeQuery = true)
        List<Map<String, Object>> getJoinAndLeaveByMouth();

    /**
     * 查询 like姓名的员工
     * @param
     * @return
     */
    @Query(value = " SELECT USERNAME FROM TB_USERS where USERNAME LIKE '%'||?
1||'%' ",nativeQuery = true)
    Set<String> findUsernameByUsernameLike(String username);
}
```

那么我们该怎么去利用JPA做一个增删查改呢?

- **分页查询职员****
- 需求: 查询条件（职员姓名（需模糊查询（可进行查询联想），职工号（equal精确查询），性别
  equal精确查询，职位equal精确查询，是否在职equal精确查询），若查询时带有查询条件参数则
  按照带查询条件进行查询，默认查询为分页查询所有职员，使用JpaSpecificationExecutor接口中
  Page findAll(@Nullable Specification var1, Pageable var2);方法即可

  后端实现分页查询

  使用JpaSpecificationExecutor接口方法直接从service层开始写

```
                              //Users为实体类名称，Long为主键Id类型，JpaSpecificationExecutor为分页查询
public interface UsersDao extends JpaRepository<Users,Long> , JpaSpecificationExecutor<Users> {
    /**
```

      注明service层，引入userDao

首先判断查询条件是否为空，在判断 "是否在职条件"为空

```java
//分页查询用户信息
@SuppressWarnings("serial")
public Page<Users> getUsersByPage( final Users user , int pageIndex , int
pageSize ) {
    // 指定排序参数对象：根据id，进行降序查询
    Sort sort = Sort.by(Sort.Direction.DESC, "id");
    // 分页查询用户信息，返回分页实体对象数据,过滤条件为员工编号、姓名、职位，性别，是否在职。
    // pages对象中包含了查询出来的数据信息，以及与分页相关的信息
    Page<Users>  pages= userDao.findAll(new Specification<Users>() {
        @Override
        public Predicate toPredicate(Root<Users> root, CriteriaQuery<?> query,
                                     CriteriaBuilder cb) {
            List<Predicate> predicates = new ArrayList<Predicate>();
            if(user!=null){
                if(user.getIsWork()==null){
                if(!StringUtils.isEmpty(user.getUsername())){
                    predicates.add(cb.like(root.<String> get("username"),
"%"+user.getUsername()+"%"));
                }
                if(user.getEid() != null){
                    predicates.add(cb.equal(root.<String> get("eid"),
user.getEid()));
                }
                if(!StringUtils.isEmpty(user.getSex())){
                    predicates.add(cb.equal(root.<String> get("sex"),
user.getSex()));
                }
                if(!StringUtils.isEmpty(user.getJob())){
                    predicates.add(cb.equal(root.<String> get("job"),
user.getJob()));
                }
                }
                else if (user.getIsWork()){
                    predicates.add(cb.equal(root.<String> get("isWork"),
user.getIsWork()));

                        if (!StringUtils.isEmpty(user.getUsername())) {
                            predicates.add(cb.like(root.<String>get("username"),
"%" + user.getUsername() + "%"));
                        }
                        if (user.getEid() != null) {
                            predicates.add(cb.equal(root.<String>get("eid"),
user.getEid()));
                        }
                        if (!StringUtils.isEmpty(user.getSex())) {
```

```
                                        predicates.add(cb.equal(root.<String>get("sex"),
user.getSex()));
                        }
                        if (!StringUtils.isEmpty(user.getJob())) {
                                predicates.add(cb.equal(root.<String>get("job"),
user.getJob()));
                        }

                }else {
                    predicates.add(cb.equal(root.<String> get("isWork"),
user.getIsWork()));
                        if (user.getEid() != null) {
                            predicates.add(cb.equal(root.<String>get("eid"),
user.getEid()));
                        }
                        if (!StringUtils.isEmpty(user.getSex())) {
                                predicates.add(cb.equal(root.<String>get("sex"),
user.getSex()));
                        }
                        if (!StringUtils.isEmpty(user.getJob())) {
                                predicates.add(cb.equal(root.<String>get("job"),
user.getJob()));
                        }
                    }
                }
                return query.where(predicates.toArray(new
Predicate[predicates.size()])).getRestriction();
        }
    },PageRequest.of(pageIndex-1, pageSize, sort));
    return pages;
};
```

控制层 @Controller注明为控制层，@RequestMapping注明url根路径，引入UsersService职员服务层，RedisKeyUtil 缓存key配置文件以及日志Logger


```
26    @Controller
27    @RequestMapping("/user")
28    public class UserController {
29        @Autowired
30        private UsersService usersService;
31        @Autowired
32        RedisKeyUtil redisKeyUtilss;
33
34        private final Logger log= LoggerFactory.getLogger(LoggerContext.class);
35
```

这里分页查询没有用到异步传值，直接设计method="post"，Model返回信息，在JSP页面放入table中进行c:forEach循环，分页信息默认显示首页，每页十条数据

```
/**
 * 用户管理界面
 * @param page 分页数据
 * @param rows
 * @param user 查询条件
 * @param model 查询条件数据回显
 * @return
 */
```

```java
@RequestMapping(value="/toUser.action")   //到管理员页面
public String toReaderA(@RequestParam(defaultValue="1")Integer page,
                        @RequestParam(defaultValue="10")Integer rows, Users
user,
                        Model model) {
    Page<Users> pageS =usersService.getUsersByPage(user,page, rows );
    List<Users> users = pageS.getContent();
    // 将分页查询出的结果数据进行分析，然后把数据存入到PageData对象中去保存起来响应给浏览器展
示
    PageData pageData = new PageData();
    pageData.setPage(pageS.getNumber()+1);
    pageData.setRows(users);
    pageData.setSize(pageS.getSize());
    pageData.setTotal((int)pageS.getTotalElements());
    model.addAttribute("username",user.getUsername());
    model.addAttribute("eid",user.getEid());
    model.addAttribute("page",pageData);
    log.info("测试分页  查询出Content"+pageS.getContent() +"测试model数据"+model);
    return "admin/user";
}
```

前端JSP页面设计

引入标签库（防止中文乱码，jstl库及分页）

```
<%@ page language="java"  import="java.util.*" contentType="text/html;
charset=UTF-8"
    pageEncoding="UTF-8"%>
 <%@ page trimDirectiveWhitespaces="true"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="navigationTag" uri="http://navigationTag.com/common/"%>
```

配置basePath

```
<%
String path = request.getContextPath();
String basePath =
request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+pa
th+"/";
%>
```

静态资源引入（css，js）

```
<link rel="stylesheet" href="<%=basePath%>css/bootstrap.min.css">

    <link rel="stylesheet" href="<%=basePath%>css/materialize.css">
    <link href="https://fonts.googleapis.com/icon?family=Material+Icons"
rel="stylesheet">

   <script type="text/javascript" src="<%=basePath%>js/js113.js"></script>
   <script type="text/javascript" src="<%=basePath%>js/jquery-1.11.0.min.js">
</script>
    <script type="text/javascript" src="<%=basePath%>js/materialize.js">
</script>
    <script type="text/javascript" src="<%=basePath%>js/materialize.min.js">
</script>
   <script type="text/javascript" src="<%=basePath%>js/bootstrap.min.js">
</script>
    <script src="http://static.runoob.com/assets/jquery-validation-
1.14.0/lib/jquery.js"></script>
    <script src="http://static.runoob.com/assets/jquery-validation-
1.14.0/dist/jquery.validate.min.js"></script>
    <script src="http://static.runoob.com/assets/jquery-validation-
1.14.0/dist/localization/messages_zh.js"></script>
```

CSS设计 这里用到的CSS来源于Materialize-UI

```
<style type="text/css">
    .error{
        color:red;
    }
    .tp{
        font-size: 20px;
    }
    input::-webkit-input-placeholder{
        color:red;
    }
    input::-moz-placeholder{   /* Mozilla Firefox 19+ */
        color:red;
    }
    input:-moz-placeholder{    /* Mozilla Firefox 4 to 18 */
        color:red;
    }
    input:-ms-input-placeholder{  /* Internet Explorer 10-11 */
        color:red;
    }

    /* label color */
    .input-field label {
        color: #000;
    }
    /* label focus color */
    .input-field input[type=text]:focus + label {
        color: #000;
    }
    /* label underline focus color */
    .input-field input[type=text]:focus {
        border-bottom: 1px solid #000;
        box-shadow: 0 1px 0 0 #000;
```

```
        }
        /* valid color */
        .input-field input[type=text].valid {
            border-bottom: 1px solid #000;
            box-shadow: 0 1px 0 0 #000;
        }
        /* invalid color */
        .input-field input[type=text].invalid {
            border-bottom: 1px solid #000;
            box-shadow: 0 1px 0 0 #000;
        }
        /* icon prefix focus color */
        .input-field .prefix.active {
            color: #000;
        }


</style>
    <style type="text/css">
        .autoComplete {margin:8px;position:relative;float:left;}
        .autoComplete input {width:200px;height:25px;margin:0;padding:0;line-
height:25px;}
        .autoComplete ul {z-index:-12;padding:0px;margin:0px;border:1px #333
solid;width:200px;background:white;display:none;position:absolute;left:0;top:28p
x;*margin-left:9px;*margin-top:2px;margin-top:1px\0;}
        .autoComplete li {list-style:none;}
        .autoComplete li a {display:block;color:#000;text-
decoration:none;padding:1px 0 1px 5px;_width:97%;}
        .autoComplete li a:hover {color:#000;background:#ccc;border:none;}
    </style>
```

带有查询条件的分页查询，样式设计来源Materialize-UI

```
<form class="form-inline" method="post"
      action="<%=basePath%>user/toUser.action">
    <div class="form-group">
        <label for="username"><p class="tp" style="color: #0C0C0C"></p></label>
        <div class="autoComplete" style="z-index:19">
            <input type="text" class="form-control" placeholder="姓名:"
id="username" value="${username}" name="username" >
            <ul><li></li></ul>
        </div>
    </div>
    <div class="form-group">
        <label for="eid"><p class="tp" style="color: #0C0C0C">工号:</p></label>
        <input type="text" class="form-control" id="eid" value="${eid}" name="eid"
>
    </div>
                <div class="form-group">
        <p>
          <input name="sex" value="女" type="radio" id="test1"  />
          <label for="test1">女士</label>
        </p>
        <p>
          <input name="sex" value="男" type="radio" id="test2" />
          <label for="test2" >男士</label>
        </p>
```

```
    </div>
        
    <div class="form-group">
        <p>
            <input name="isWork" value="true" type="radio" id="test3"  />
            <label for="test3">在职</label>
        </p>
        <p>
            <input name="isWork" value="false" type="radio" id="test4" />
            <label for="test4" >离职</label>
        </p>
    </div>
        
                <div class="form-group">
    <label  for="job"><p class="tp" style="color: #0C0C0C">Job list</p>
</label>
                        <select class="browser-default" name="job" id="job">
                            <option value="" disabled selected>Choose your
Job</option>

                            <option value ="开发工程师">开发工程师</option>
                            <option value ="实施工程师">实施工程师</option>
                            <option value="DBA">DBA</option>
                            <option value="产品经理">产品经理</option>
                            <option value="项目经理">项目经理</option>
                            <option value="HR">HR</option>
                        </select>
                </div>
    <button class="btn waves-effect waves-light green" type="submit"
name="action">查询
        <i class="material-icons right">send</i>
    </button>
</form>
```

分页显示职员信息table,table样式用的Bootstrap，但重新用Materialize-UI进行修饰，加上了图标以及按钮样式， 标签定义表格的表头，预先定义职员信息描述， 使用c:forEach 循环输出model里的数据

```
    <table class="table  table-hover">
      <thead>
        <tr>
            <th><input type="checkbox" class="i-checks" name="test5" id="all" >
                <label for="all"></label></th>
            <th><i class="small material-icons">explicit</i>工号</th>
            <th><i class="small material-icons">perm_contact_calendar</i>姓名
</th>
            <th><i class="small material-icons">perm_identity</i>性别</th>
            <th><i class="small material-icons">supervisor_account</i>职位</th>
            <th><i class="small material-icons">phone</i>手机号码</th>
            <th><i class="small material-icons">assignment_late</i>是否在职</th>
            <th><a href="#" class="btn waves-effect waves-light red"
onclick="deleteAll()">批量删除</a></th>
        </tr>
      </thead>
      <tbody id="questionlist">

        <c:forEach var="user" items="${page.rows}" varStatus="status" >
            <tr class="aa" draggable="true">
                <td>
```

```
                    <input type="checkbox" class="i-checks" name="idone"
id="idone" value="${user.id}" >
                    <label for="idone"></label>
                </td>
                <td>
                    <c:out value="${user.eid }"/>
                </td>
                <td>
                    <c:out value="${user.username }"/>
                </td>
                <td>
                    <c:out value="${user.sex }"/>
                </td>
                <td>
                    <c:out value="${user.job }"/>
                </td>
                <td>
                    <c:out value="${user.phone }"/>
                </td>
                <td>
                    <c:out value="${user.isWork }"/>
                </td>

                <td>
                    <a href="#" class="btn btn-primary btn-xs" data-toggle="modal"
data-target="#customerEditDialog" onclick= "editUser(${user.id})">详情or修改</a>
                    <a href="#" class="btn waves-effect waves-light red"
onclick="deleteUser(${user.id})">删除</a>

                </td>
            </tr>
        </c:forEach>
    </tbody>
    </table>
    <div class="col-md-12 text-right">
        <navigationTag:page url="${pageContext.request.contextPath
}/user/toUser.action" />
    </div>
</div>
```

PageData分页标签使用：

```
<div class="col-md-12 text-right">
    <navigationTag:page url="${pageContext.request.contextPath
}/user/toUser.action" />
</div>
```

最终页面效果为：

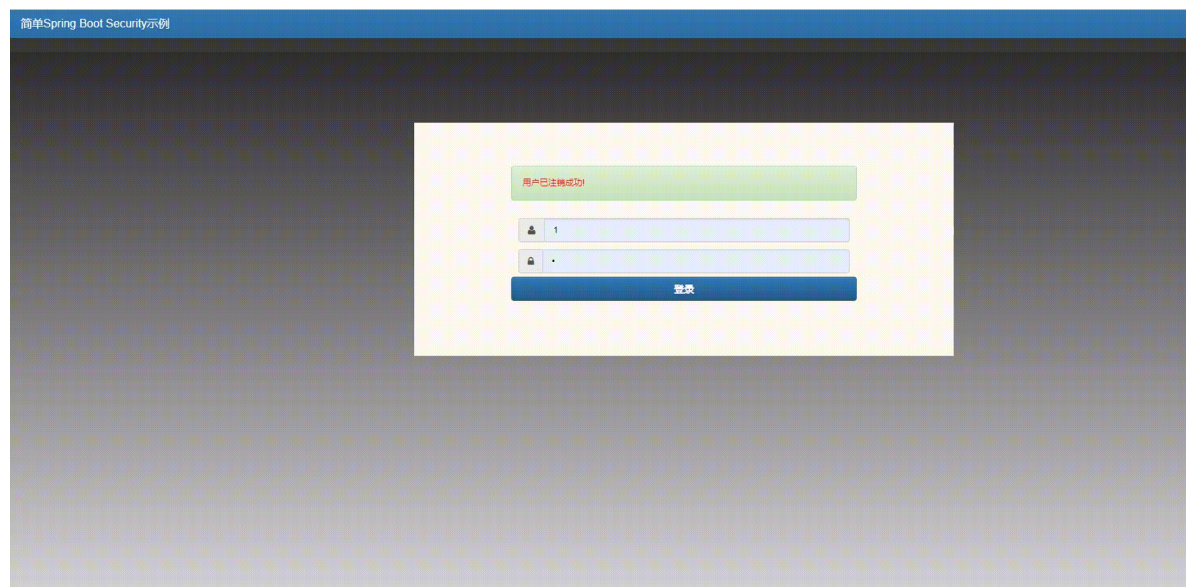- **添加职员**
- 需求：添加职员，ID为雪花生成，入职时间为当前时间，在职情况为TRUE
- 解决方法：职员信息在前端输入完成后通过AJAX异步提交到后端，controller层先判断user参数是否为空，不为空时直接调用Service层添加方法,ID，入职时间，在职情况直接在Service层处理

前端使用Bootstrap模态框生成添加职员模态框 target="#myModal" 为模态框ID

```
<form class="form-inline"  onsubmit="return false">
                    <a class="btn-floating btn-large waves-effect waves-light blue"
type="button"  data-toggle="modal" data-target="#myModal">ADD<i class="material-
icons">add</i></a>
</form>
```

id="myModal" 对应 target="#myModal" class="modal fade bs-example-modal-lg" lg 样式注明
为large巨大模态框

输入框带上id为jq获取输出框值，placeholder为输入框空值时提示 class="validate"为Materialize-UI
样式 required注明该字段需要经过JQ validate校验

姓名 username

```
!-- 模态框开始（Modal） -->

<div class="modal fade bs-example-modal-lg" id="myModal"  style="height:fit-
content" tabindex="-1" role="dialog" aria-labelledby="myLargeModalLabel">
        <div class="modal-dialog modal-lg">
            <div class='yellow'>
            <button type="button" class="close" data-dismiss="modal" aria-
hidden="true">
            </button>
            <h4 class="modal-title" id="myModalLabel"  align="center">NEW WORKER
            </h4>
        </div>

            <div class="row">
                <form id="validateAdd" class="col s12" >
                    <div class="row">
                        <div class=".input-field col s6">
                     <label for="add_username">Name：</label>
                            <i class="material-icons prefix"></i>
                            <input id="add_username" type="text" placeholder="姓
名 username" class="validate" required>
                            </div>
                            <div class=".input-field col s6">
                        <label for="add_phone">tel：</label>
                            <i class="material-icons prefix"></i>
                            <input id="add_phone" type="tel" placeholder="电话
tel"  class="validate" required>

                            </div>
                    </div>
                    <div class="row">
                        <div class=".input-field col s2">
                        <label for="add_sex">性别选择 </label>
                        <select class="browser-default" id="add_sex" required>
                            <option value="" disabled selected>Choose sex</option>
                            <option value="男">小哥哥</option>
                            <option value="女">小姐姐</option>
                        </select>
                            </div>
                        <div class=".input-field col s4">
                            <label  for="add_job">职位 Choose</label>
```

```html
                        <select class="browser-default" id="add_job" required>
                            <option value="" disabled selected>Choose Job</option>
                            <option value ="开发工程师">开发工程师</option>
                            <option value ="实施工程师">实施工程师</option>
                            <option value="DBA">DBA</option>
                            <option value="产品经理">产品经理</option>
                                    <option value="项目经理">项目经理</option>
                            <option value="HR">HR</option>
                        </select>
                    </div>
                        <div class="input-field col s2">
                            <label  for="add_basicSalary">基本工资:</label>
                            <input id="add_basicSalary" type="text"
placeholder="工资 " class="validate" >
                            </div>
                </div>
                    <div class="row">
                        <div class="input-field col s6">
                        <label  for="add_idcard">身份证号码:</label>
                            <input id="add_idcard" type="text" placeholder="身份
证号码 idcard" class="validate" >

                            </div>
                            <div class="input-field col s6">
                        <label  for="add_address">地址:</label>
                            <input id="add_address" type="text" placeholder="地址
address" class="validate" >

                            </div>
                        </div>
                        <button  type="reset" class="btn btn-danger btn-xs">重置
</button>
                </form>
            </div>
        <div class="card-panel">
            <h5>Primary Buttons</h5>
            <button id="buttonAdd" class="btn waves-effect waves-light red"
onclick="addUser()">确认<i class="material-icons right">send</i></button>
                <button type="button" class="btn waves-effect waves-light yellow
" data-dismiss="modal">关闭<i class="material-icons right">cancel</i></button>
<%--        <button class="btn waves-effect waves-light red disabled" data-
dismiss="modal" >Cancel<i class="material-icons right">cancel</i></button>--%>
        </div>
    </div>
    </div>
</div>
<%--添加模态框结束--%>
```

添加职员 JS代码，首先校验字段空值和是否符合对应规则，校验通过后使用AJAX异步 post方法推送数据JSON格式数据

```javascript
function addUser(){
    //form表单校验
   let flag = $('#validateAdd').valid();
   if(!flag) return;
   var username = $('#add_username').val();
   var sex = $('#add_sex').val();
```

```javascript
    var job = $('#add_job').val();
    var basicSalary = $('#add_basicSalary').val();
    var phone = $('#add_phone').val();
    var idcard = $('#add_idcard').val();
    var address = $('#add_address').val();
        /*
        正则检验
         */
    var regPhone = /(^1(3|4|5|6|7|8|9)\d{9}$)/;
    if (!regPhone.test(phone)) {
        layer.msg('手机号码,请输入正确的数值，13/14/15/16/17/18/19开头的号码', {icon: 2,
time: 3000});
        return ;
    };
    var regIdcard = /(^\d{15}$)|(^\d{18}$)|(^\d{17}(\d|X|x)$)/;
    if (!regIdcard.test(idcard)) {
        layer.msg('身份证输入错误,请输入正确的数值', {icon: 2, time: 3000});
        return ;
    };
        var str = address.replace(/(^\s*)|(\s*$)/g, '');//去除空格;
        if (str == '' || str == undefined || str == null) {
            layer.msg('请输入地址，前中后不要包含空格', {icon: 2, time: 3000});
            return ;
        }
    var regBasicSalary = /(^[1-9]([0-9]+)?(\.[0-9]{1,2})?$)|(^(0){1}$)|(^[0-9]\.
[0-9]([0-9])?$)/;
    if (!regBasicSalary.test(basicSalary)) {
        layer.msg('工资输入错误,请输入正确的数值，例如正确：588.00 or 588 错误：558a or
adawd', {icon: 2, time: 3000});
        return ;
    };


$.ajax({
url : "<%=basePath%>user/add.action",
type : "post",
data :JSON.stringify({username: username,
    sex: sex,
    job: job,
    basicSalary: basicSalary,
    phone: phone,
    idcard: idcard,
    address: address}),
// 定义发送请求的数据格式为JSON字符串
contentType : "application/json;charset=UTF-8",
//定义回调响应的数据格式为JSON字符串,该属性可以省略
dataType : "json",
//成功响应的结果
success : function(data){
    if (data.code === 200) {
        layer.msg(data.msg, {icon: 1, time: 1000},function (index){
            window.location.reload();
            layer.close();
        });
    } else {
        layer.msg(data.msg, {icon: 2, time: 3000});
    }
}
```

```
  })
};
```

职员控制层，校验空指针

```java
/**
 *    添加用户  AJAX
 * @param users 用户表单数据
 * @return
 */
@ResponseBody
@PostMapping(value = "/add.action")
    public ResponseEntity<?> addUser(@RequestBody Users users){
    Result result = new Result();
    if(Objects.isNull(users)){
        result.setCode(500);
        result.setMsg("FALSE");
        log.error("添加失败,返回值为"+result);
        return ResponseEntity.ok(result);
    }
        result.setCode(200);
        result.setMsg("OK");
    usersService.saveUser(users);
    log.info("添加成功"+users);
    return ResponseEntity.ok(result);
}
```

职员服务层

```java
/**
 * 添加员工  需要事务支持
 * @param user
 */
@Transactional
public void    saveUser(Users user) {
    user.setEid(userDao.getMaxEiD()+1);
    user.setId(SequenceUtils.nextId()); //雪花算法ID生成

user.setPassword("$2a$10$36A/etwTPl2PscyArcEpzeAyvhC3cd1IJZYMXv27jIp67j0BOOpJa")
;
    user.setJoinTime(new Date());
    user.setIsWork(Boolean.TRUE);
    userDao.save(user);
};
```
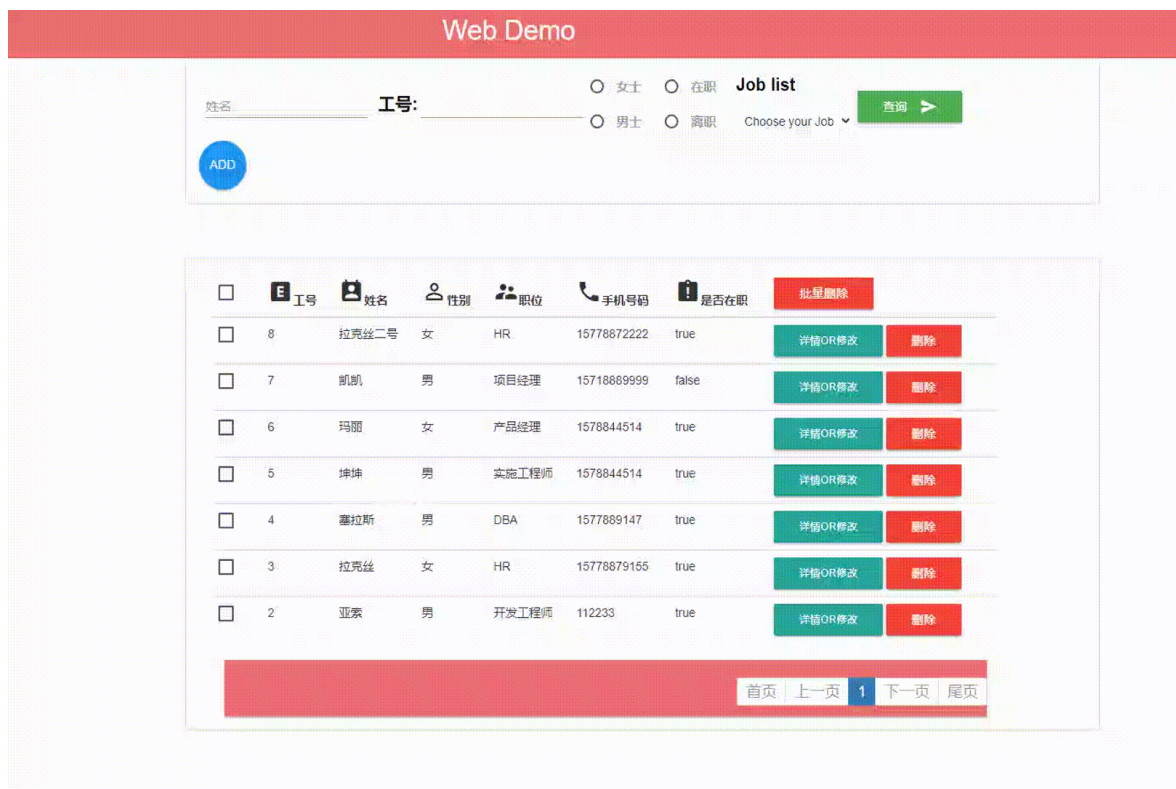
- **详细查看职员***

- 需求：通过ID查询用户

- 解决方法：AJAX异步提交ID到后端，controller层先判断ID参数是否为空，不为空时直接调用Service层查询方法

  前端 生成详情模态框

  - **修改职员***

  - 需求：通过ID修改用户

  - 解决方法：AJAX异步提交ID到后端，controller层先判断ID参数是否为空，不为空时直接调用Service层查询方法，查询得到职员信息AJAX后通过JQ设置前端职员修改模态框信息为查询到的职员信息,修改完成后通过AJAX将表单序列化提交到后端进行处理，controller层先判断表单是否为空，不为空时直接调用Service层修改方法

```
<a href="#" class="btn btn-primary btn-xs" data-toggle="modal" data-
target="#customerEditDialog" onclick= "editUser(${user.id})">详情or修改</a>
```

```
<div class="modal fade bs-example-modal-lg" id="customerEditDialog"
tabindex="-1" role="dialog" aria-labelledby="myLargeModalLabel">
    <div class="modal-dialog modal-lg">
      <div class='red'>
        <button type="button" class="close" data-dismiss="modal" aria-
hidden="true">
        </button>
        <h4 class="modal-title" id="myModalLabel1"  align="center">ALTER
WORKER MESSAGE
        </h4>
    </div>
```

```html
        <div class="row">
            <form class="col s12" id="edit_user_form">
                <input type="hidden" id="edit_id" name="id"/>
                <input type="hidden" id="edit_password" name="password"/>
                <input type="hidden" id="edit_eid" name="eid"/>
                <input type="hidden" id="edit_welfarePoints"
name="welfarePoints"/>
                <input type="hidden" id="edit_joinTime" name="joinTime"/>
                <input type="hidden" id="edit_leaveTime" name="leaveTime"/>
                <div class="row">
                    <div class=".input-field col s6">
                        <label for="edit_username">Name：</label>
                        <i class="material-icons prefix"></i>
                        <input id="edit_username" type="text" placeholder="姓名
username"  name="username" class="validate" required>
                    </div>
                    <div class=".input-field col s6">
                        <label for="edit_phone">tel：</label>
                        <i class="material-icons prefix"></i>
                        <input id="edit_phone" type="tel" placeholder="电话 tel"
name="phone" class="validate" required>

                    </div>
                </div>
                <div class="row">
                    <div class=".input-field col s6"><p>Sex：</p>
                        <p>
                            <input name="sex" value="女" type="radio" id="sexW"
required/>

                            <label for="sexW">女士</label>
                        </p>
                        <p>
                            <input name="sex" value="男" type="radio" id="sexM" />
                            <label for="sexM">男士</label>
                        </p>
                    </div>
                    <div class=".input-field col s6"><p>是否在职：</p>
                        <p>
                            <input name="isWork" value="true" type="radio"
id="workT" required/>
                            <label for="workT">在职</label>
                        </p>
                        <p>
                            <input name="isWork" value="false" type="radio"
id="workF" />
                            <label for="workF">离职</label>
                        </p>
                    </div>
                </div>

                <div class="row">
                    <div class=".input-field col s2">
                        <label  for="edit_job">Job list</label>
                        <select class="browser-default" id="edit_job" name="job"
required>
                            <option value="" disabled selected>Choose your
Job</option>
```

```html
                    <option value ="开发工程师">开发工程师</option>
                    <option value ="实施工程师">实施工程师</option>
                    <option value="DBA">DBA</option>
                    <option value="产品经理">产品经理</option>
                    <option value="项目经理">项目经理</option>
                    <option value="HR">HR</option>
                </select>
            </div>
            <div class=".input-field col s2">
                <label  for="edit_basicSalary">基本工资:</label>
                <input id="edit_basicSalary" name="basicSalary"
type="text" placeholder="工资" class="validate">
            </div>
        </div>
        <div class="row">
            <div class=".input-field col s3">
                <label  for="edit_idCard">身份证号码:</label>
                <input id="edit_idCard" name="idcard" type="text"
placeholder="身份证号码 idcard" class="validate" >

            </div>
            <div class=".input-field col s6">
                <label  for="edit_address">地址:</label>
                <input id="edit_address" name="address" type="text"
placeholder="地址 address" class="validate">

            </div>
        </div>


            <button  type="reset" class="btn btn-danger btn-xs">重置
</button>
        </form>
    </div>
    <div class="card-panel">
        <h5>Primary Buttons</h5>
        <button class="btn waves-effect waves-light red"
onclick="updateUser()">修改<i class="material-icons right">send</i></button>
        <button type="button" class="btn waves-effect waves-light yellow "
data-dismiss="modal">关闭<i class="material-icons right">cancel</i></button>


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

```javascript
function editUser(id) {
    $.ajax({
        type:"get",
        url:"<%=basePath%>user/getUserById.action",
        data:{"id":id},
        success:function(data) {
            $("#edit_id").val(data.id);
            $("#edit_username").val(data.username);
            $("#edit_password").val(data.password);
            // $("#edit_sex").val(data.sex);
            $("#edit_eid").val(data.eid);
            $("#edit_job").val(data.job);
```

```javascript
        $("#edit_basicSalary").val(data.basicSalary);
        $("#edit_welfarePoints").val(data.welfarePoints);
        $("#edit_phone").val(data.phone);
        $("#edit_idCard").val(data.idcard);
        $("#edit_joinTime").val(data.joinTime);
        $("#edit_leaveTime").val(data.leaveTime);
        $("#edit_address").val(data.address);


        if (data.sex===("女")) {
            $("input[id=sexW][value='女']").attr("checked",true);//value=34
的radio被选中
        } else {
            $("input[id=sexM][value='男']").attr("checked",true);//value=34
的radio被选中
        }
        if (data.isWork) {
            $("input[id=workT]
[value='true']").attr("checked",true);//value=34的radio被选中
        } else {
            $("input[id=workF]
[value='false']").attr("checked",true);//value=34的radio被选中
        }

    }
  });
};

function updateUser() {
      let flag = $('#edit_user_form').valid();
      if(!flag) return;
      var basicSalary = $('#edit_basicSalary').val();
      var phone = $('#edit_phone').val();
      var idcard = $('#edit_idCard').val();
      var address = $('#edit_address').val();
      /*
      正则检验
       */
      var regPhone = /(^1(3|4|5|6|7|8|9)\d{9}$)/;
      if (!regPhone.test(phone)) {
          layer.msg('手机号码,请输入正确的数值，13/14/15/16/17/18/19开头的号码',
{icon: 2, time: 3000});
          return ;
      };
      var regIdcard = /(^\d{15}$)|(^\d{18}$)|(^\d{17}(\d|X|x)$)/;
      if (!regIdcard.test(idcard)) {
          layer.msg('身份证输入错误,请输入正确的数值', {icon: 2, time: 3000});
          return ;
      };
      var str = address.replace(/(^\s*)|(\s*$)/g, '');//去除空格;
      if (str == '' || str == undefined || str == null) {
          layer.msg('请输入地址，前中后不要包含空格', {icon: 2, time: 3000});
          return ;
      }
      var regBasicSalary = /(^[1-9]([0-9]+)?(\.[0-9]{1,2})?$)|(^(0){1}$)|
(^[0-9]\.[0-9]([0-9])?$)/;
      if (!regBasicSalary.test(basicSalary)) {
```

```
                    layer.msg('工资输入错误,请输入正确的数值,例如正确: 588.00 or 588 错误:
558a or adawd', {icon: 2, time: 3000});
            return ;
        };


    $.ajax({
        type: "POST",
        dataType: "json",
        contentType: "application/x-www-form-urlencoded;charset=UTF-8",
        url: "<%=basePath%>user/updateUser.action",
        data: $("#edit_user_form").serialize(),
        beforeSend: function () {
        },
        success: function (data) {
            if (data.code === 200) {
                layer.msg(data.msg, {icon: 1, time: 1000},function (index){
                    window.location.reload();
                    layer.close();
                });
            } else {
                layer.msg(data.msg, {icon: 2, time: 3000});
            }
        }
    });
};
```

后端控制层

```
/**
 * 通过id获取职工信息  详细查看或者修改使用
 * @param id
 * @return
 */
@RequestMapping("/getUserById.action")
@ResponseBody
public Users getUserById(Long id) {
    if (id!=null){
            Users users = usersService.findById(id);
            log.info("get Users by id"+id);
             return users;
    }else {
        log.error("数据异常");
        return null;
    }
}
/**
 * 更新职员   form数据序列化
 */
@RequestMapping("/updateUser.action")
@ResponseBody
public  ResponseEntity<?>  userUpdate(Users users) {
    Result result = new Result();
    if(users.getId() != null){
        usersService.updateUser(users);
        result.setCode(200);
```

```
            result.setMsg("OK");
            log.info("操作成功,更新的职员id为"+users.getId());
            return ResponseEntity.ok(result);
    } else {
            result.setCode(500);
            result.setMsg("FALSE");
            log.error("操作失败"+result);
            return   ResponseEntity.ok(result);
        }
 }
```

服务层 修改需要事务支持

```
/**
  *   详情查看 user by id
  * @param id
  * @return
  */
 public Users findById(Long id) {
     return userDao.getOne(id);
 }
   /**
  * 修改员工  需要事务支持
  * @param user
  */
 @Modifying
 @Transactional
 public void   updateUser(Users user) {
     if(!user.getIsWork()&&user.getLeaveTime()==null){
         user.setLeaveTime(new Date());
         userDao.save(user);
     }else {
         user.setLeaveTime(null);
         userDao.save(user);
     }
```


- **删除职员****

- 需求：通过ID删除用户
- AJAX异步提交ID到后端，controller层先判断ID参数是否为空，不为空时直接调用Service层删除方法，如果是批量删除的话后端需要处理前端提供的批量ID组成的字符串，前端将ID放进String字符串按照","进行拼接，利用Ajax传值，后端收到并对字符串进行校验是否为空，是否含有","对其进行String转List处理，循环List将List转为List 调用Service层删除方法即可。

前端

单删除

```
<a href="#" class="btn waves-effect waves-light red"
onclick="deleteUser(${user.id})">删除</a>
```

```
// 删除
function deleteUser(id) {
    layer.confirm('删除后不可恢复，谨慎操作！', {icon: 7, title: '警告'},
function (index) {
        $.post("<%=basePath%>user/deleteUser.action",{"id":id},
            function(data){
                if (data.code === 200) {
                    layer.msg(data.msg, {icon: 1, time: 1000},function (index)
{

                        window.location.reload();
                        layer.close();
                    });
                } else {
                    layer.msg(data.msg, {icon: 2, time: 3000});
                }
            });
    });
}
```

批量删除时需要判断至少选中一个数据

checkbok表头

```
<th><input type="checkbox" class="i-checks" name="test5" id="all" >
    <label for="all"></label></th>
```

checkbok数据

```
<td>
    <input type="checkbox" class="i-checks" name="idone" id="idone"
value="${user.id}" >
    <label for="idone"></label>
</td>
```

批量删除按钮

```
<th><a href="#" class="btn waves-effect waves-light red"
onclick="deleteAll()">批量删除</a></th>
```

全选按钮

```javascript
//全选
var oall=document.getElementById("all");
var oid=document.getElementsByName("idone");
oall.onclick=function(){//勾选全选时
    for(var i=0;i<oid.length;i++){
        //所有的选择框和全选一致
        oid[i].checked=oall.checked;
    }
};
```

批量删除js

```javascript
function deleteAll() {
    var idss = '';
    $('input:checkbox[name="idone"]').each(function () {
        if (this.checked === true) {
            idss += this.value + ',';
        }
    });
    if (!idss){
        layer.msg('请至少选中一个', {icon: 2, time: 1500});
        return false;
    } else {
    layer.confirm('批量删除后不可恢复，谨慎操作！', {icon: 7, title: '警告'}, function
(index) {
        $.ajax({
            type: 'POST',
            url: "<%=basePath%>user/deleteByIds.action",
            data: {ids: idss},
            dataType: 'json',
            success: function (data) {
                if (data.code === 200) {
                    layer.msg(data.msg, {icon: 1, time: 1000},function (index){
                        window.location.reload();
                        layer.close();
                    });
                } else {
                    layer.msg(data.msg, {icon: 2, time: 3000});
                }
            }
        });
    });
    }
}
```

后端控制层

```java
/**
 * 批量删除
 * @param ids
 * @return
 */
@RequestMapping(value = "/deleteByIds.action",method = RequestMethod.POST)
@ResponseBody
public ResponseEntity deleteByIds(String ids) {
    Result result = new Result();
```

```java
            if (ids.contains(",")) {
                List lists = ConvertUtil.string2List(ids, ",");
                try {
                    //批量删除
                    if (lists != null && lists.size() >= 1) {
                        log.info("批量删除开始 测试第一个数据ConvertUtil--" +
ConvertUtil.stringToLong(lists.get(1).toString()));
                        for (int i = 0; i < lists.size(); i++) {

usersService.delete(ConvertUtil.stringToLong(lists.get(i).toString()));
                        }
                    }
                } catch (Exception e) {
                    result.setCode(500);
                    result.setMsg("FALSE");
                    log.error("批量删除操作失败+"+e);
                    return ResponseEntity.ok(result);
                }
                result.setCode(200);
                result.setMsg("OK");
                log.info("批量删除成功,ids为"+lists);
                return ResponseEntity.ok(result);
            }else {
                List lists = ConvertUtil.array2List(ids);
                usersService.delete(ConvertUtil.stringToLong(lists.get(0).toString()));
                result.setCode(200);
                result.setMsg("OK");
                log.info("批量删除失败，进入单个删除,id为"+lists);
                return  ResponseEntity.ok(result);
            }
        }
        /**
         * 删除职员
         */
        @RequestMapping("/deleteUser.action")
        @ResponseBody
        public ResponseEntity<?> userDelete(Long id) {
            Result result = new Result();
            if(id != null ){ //success return
                usersService.delete(id);
                result.setCode(200);
                result.setMsg("OK");
                log.info("删除成功,id为"+id);
                return  ResponseEntity.ok(result);
            } else{  //false return
                result.setCode(500);
                result.setMsg("FALSE");
                log.error("删除失败,id为"+id);
                return   ResponseEntity.ok(result);
            }
        }
```

服务层

```
//删除 需要事务支持
@Transactional
public void delete(Long id) {
    userDao.deleteById(id);
};
```


到这里完成了职员的增删查改，接下来需要依据职员数据进行分析

人事仪盘表

仪表盘主要分析

- 在职员工职位分布
- 在职员工工资分布
- 员工入职/离职情况

首先分析 员工职位分布情况

功能：饼图显示各个职位占比人数

需求：查询各个职位的人数

实现：查询count通过职位以及是否在职,因此需要一个枚举类放置职位名称

```
public enum Job {
    DEVELOPMENTENGINEER("开发工程师"),
    IMPLEMENTATIONENGINEER("实施工程师"),
    DBAENGINEER("DBA"),
    PRODUCTMANAGER("产品经理"),
    PROJECTMANAGER("项目经理"),
    HR("HR");
    private String code;
     Job(String code ) {
        this.code= code;
    }
    public  String getCode(){
        return this.code;
    }
```

```
    }
```

SQL:

```
SELECT COUNT(*)  FROM TB_USERS WHERE JOB='?'
```

JPA: DAO层

```
    /**
     * 分职位查询在职的员工数量  饼图
     * @param job
     * @param iswork
     * @return
     */
    Long countByJobAndIsWork(String job,Boolean iswork);
```

Service层 通过FOR循环Job枚举类，查询各个职位的count存入List，前端取值时通过List下标取值。

```
/*
通过职位查询数量   饼图 p1
 */
public  List getUserCountByJob( ){
    List list = new ArrayList();
    for(Job job: Job.values()){
        Long count = userDao.countByJobAndIsWork(job.getCode(),Boolean.TRUE);
        System.out.println(job.getCode()+count);
        list.add(count);
    }
    return list;
}
```

Controller层

Echarts图表要动态显示时需要异步加载数据，这里选择了AJAX 通过GET请求数据放置图表数据字段

首次加载仪盘表界面时将数据存入Redis中，TTL为一个小时，key类型为String 所以取值时需要经过字符串截取及转Long类型

```
/*
 各职业人数分布  饼图  开发工程师  实施工程师  DBA 产品经理  项目经理  HR  顺序分布（size=6）
 */
@ResponseBody
@RequestMapping("/getPie1")
public ResponseEntity<?> getPie(){
    if (StringUtils.isNotEmpty(redisKeyUtilss.getString("getPie1"))){
        List lists =
ConvertUtil.string2List(redisKeyUtilss.getString("getPie1").substring(1,17), ",
");
        List list2 = new ArrayList();
        for (int i = 0; i < lists.size(); i++){
            list2.add(ConvertUtil.stringToLong(lists.get(i).toString()));
        }
        log.info("从redis中取出各职业人数分布  饼图"+list2);
        return ResponseEntity.ok(list2);
```

```
    } else {
        List list = usersService.getUserCountByJob();

redisKeyUtilss.setString("getPie1",list.toString());redisKeyUtilss.expire("getPi
e1",60*60);
        log.info("存入list 饼图");
        log.info("各职业人数分布  饼图  开发工程师  实施工程师  DBA  产品经理  项目经理  HR  顺序
分布（size=6）"+list);
        return ResponseEntity.ok(list);
    }

}
```

前端

引入echarts.js

```
<script src="https://cdn.staticfile.org/echarts/4.3.0/echarts.min.js"></script>
```

在页面上为饼图分配一个DOM

```
<div class="col-sm-6">
    <div class="ibox float-e-margins">
        <div class="ibox-title">
            <h5>各职业分布</h5>
        </div>
        <div class="ibox-content">
            <div class="echarts" style="height: 430px" id="echarts-pie-chart">
</div>
        </div>
    </div>
</div>
```
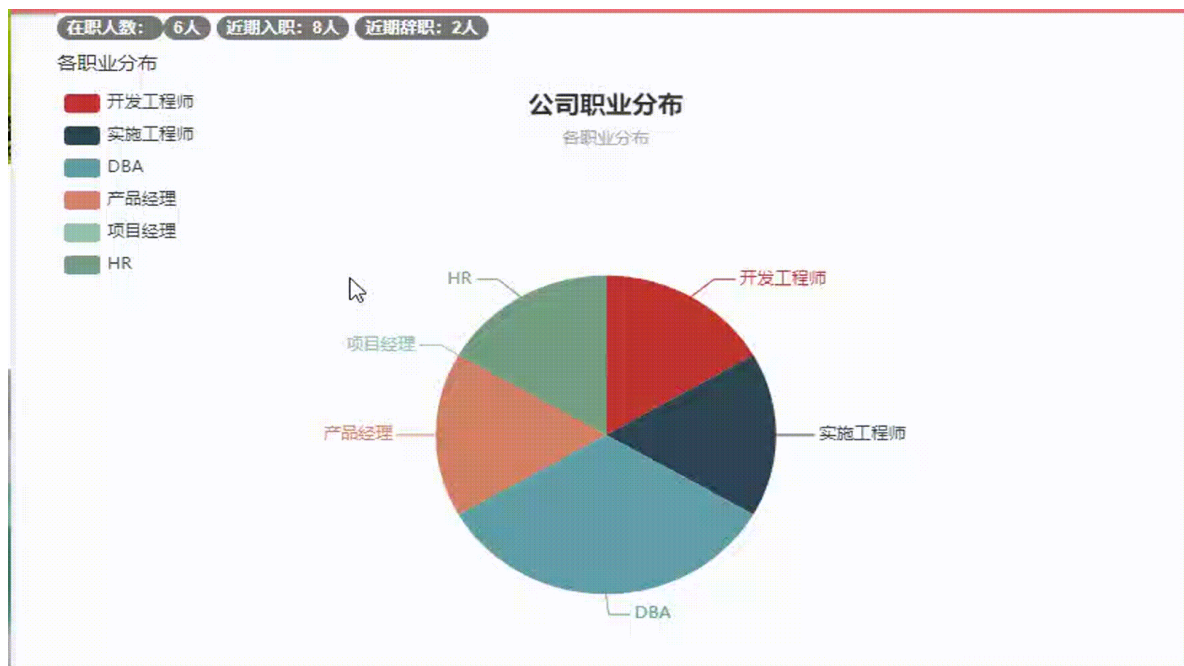
js初始化饼图

```
var pieChart = echarts.init(document.getElementById("echarts-pie-chart"));
var pieoption = null
$.get("<%=basePath%>admin/getPie1", function(data) {
    WorkerType = data;
    console.log(WorkerType);
    var pieoption = {
        title : {
            text: '公司职业分布',
            subtext: '各职业分布',
            x:'center'
        },
        tooltip : {
            trigger: 'item',
            formatter: "{a} <br/>{b} : {c} ({d}%)"
        },
        legend: {
            orient : 'vertical',
            x : 'left',
            data:['开发工程师' ,'实施工程师','DBA','产品经理','项目经理','HR']
        },
        calculable : true,
```

```
        series : [
            {
                name:'人数',
                type:'pie',
                radius : '55%',
                center: ['50%', '60%'],
                data:[
                    {value:WorkerType[0], name:'开发工程师'},
                    {value:WorkerType[1], name:'实施工程师'},
                    {value:WorkerType[2], name:'DBA'},
                    {value:WorkerType[3], name:'产品经理'},
                    {value:WorkerType[4], name:'项目经理'},
                    {value:WorkerType[5], name:'HR'}
                ]
            }
        ]
    };
    pieChart.setOption(pieoption);
    $(window).resize(pieChart.resize);
}) ;
```


总仪盘表页面