第7章 POI报表的入门

- 理解员工管理的的业务逻辑
- 能够说出Eureka和Feign的作用
- 理解报表的两种形式和POI的基本操作
- 熟练使用POI完成Excel的导入导出操作

1 员工管理

1.1 需求分析

企业员工管理是人事资源管理系统中最重要的一个环节，分为对员工入职，转正，离职，调岗，员工报表导入导出等业务逻辑。需求看似复杂，实际上都是对数据库表的基本操作。

1.2 数据库表概述

对于员工操作而言，涉及到的数据库表如下表格说明：

数据库表名称	说明
em_archive	月度员工归档表
em_positive	转正申请表
em_resignation	离职申请表
em_transferposition	员工调岗申请表
em_user_company_jobs	员工岗位信息表
em_user_company_personal	员工详细信息表

1.3 代码实现

由于此部分内容全部围绕的基本CRUD操作，为了节省课程时间，员工管理的代码以资料的形式给各位学员下发，学员们直接导入到工程即可。重点功能突出讲解即可

1.3.1 服务端实现

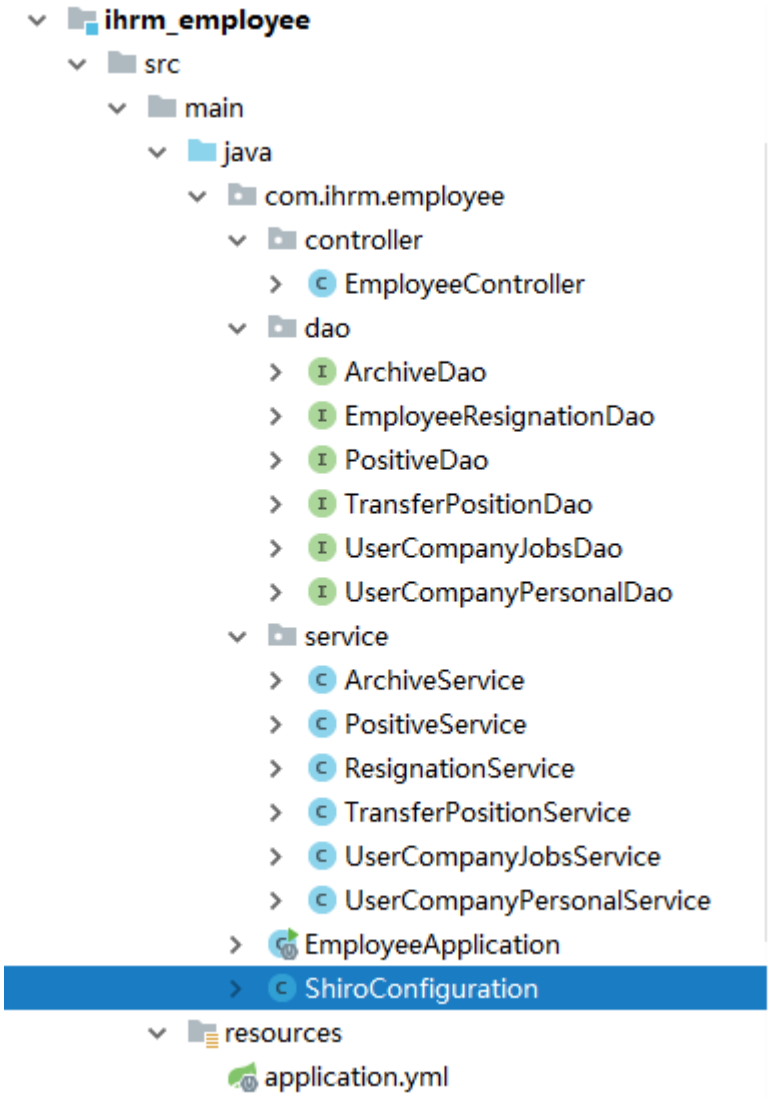
（1）创建员工微服务`ihrm_employee`

（2）配置文件 application.yml

（3）配置Shiro核心配置类ShiroConfiguration

（4）配置启动类EmployeeApplication

（5）导入资源中提供的基本Controller，Service，Dao，Domain代码



1.3.2 前端实现

导入资源中提供的前端代码。

1.4 服务发现组件 Eureka

Eureka是Netflix开发的服务发现框架，SpringCloud将它集成在自己的子项目spring-cloud-netflix中，实现SpringCloud的服务发现功能。Eureka包含两个组件：Eureka Server和Eureka Client。

- Eureka Server提供服务注册服务，各个节点启动后，会在Eureka Server中进行注册，这样EurekaServer中的服务注册表中将会存储所有可用服务节点的信息，服务节点的信息可以在界面中直观的看到。
- Eureka Client是一个java客户端，用于简化与Eureka Server的交互，客户端同时也就别一个内置的、使用轮询(round-robin)负载算法的负载均衡器。在应用启动后，将会向Eureka Server发送心跳,默认周期为30秒，如果Eureka Server在多个心跳周期内没有接收到某个节点的心跳，Eureka Server将会从服务注册表中把这个服务节点移除(默认90秒)。

1.4.1 Eureka服务端开发

（1）创建ihrm_eureka模块

（2）引入依赖 父工程pom.xml定义SpringCloud版本

```
<dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>Finchley.M9</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>
```

ihrm_eureka模块pom.xml引入eureka-server

```
<dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-
server</artifactId>
        </dependency>
    </dependencies>
```

（3）添加application.yml

```
server:
  port: 6868 #服务端口
eureka:
  client:
    registerWithEureka: false #是否将自己注册到Eureka服务中，本身就是所有无需
注册
    fetchRegistry: false #是否从Eureka中获取注册信息
    serviceUrl: #Eureka客户端与Eureka服务端进行交互的地址
      defaultZone: http://127.0.0.1:${server.port}/eureka/
```

（4）配置启动类

```
@SpringBootApplication
@EnableEurekaServer
public class EurekaServer {
    public static void main(String[] args) {
        SpringApplication.run(EurekaServer.class, args);
    }
}
```

1.4.2 微服务注册

我们现在就将所有的微服务都注册到Eureka中，这样所有的微服务之间都可以互相调用了。

（1）将其他微服务模块添加依赖

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

（2）修改每个微服务的application.yml，添加注册eureka服务的配置

```
eureka:
  client:
    service-url:
      defaultZone: http://localhost:6868/eureka
  instance:
    prefer-ip-address: true
```

（3）修改每个服务类的启动类，添加注解

```
@EnableEurekaClient
```

1.5 微服务调用组件Feign

1.5.1 简介

Feign是简化Java HTTP客户端开发的工具（java-to-httpclient-binder），它的灵感来自于Retrofit、JAXRS-2.0和WebSocket。Feign的初衷是降低统一绑定Denominator到HTTP API的复杂度，不区分是否为restful

1.5.2 快速体验

我们现在在系统微服务调用企业微服务的方法（根据ID查询部门）

（1）在ihrm_system模块添加依赖

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
```

（2）修改ihrm_system模块的启动类，添加注解

```
@EnableDiscoveryClient
@EnableFeignClients
```

（3）在Ihrm_system模块创建com.ihrm.system.client包，包下创建接口

```
//@FeignClient注解用于指定从哪个服务中调用功能 ，注意里面的名称与被调用的服务名保持一致
@FeignClient(value = "ihrm-company")
public interface DepartmentFeignClient {
    //@RequestMapping注解用于对被调用的微服务进行地址映射
    @RequestMapping(value = "/company/departments/{id}/", method = RequestMethod.GET)
    public Department findById(@PathVariable("id") String id) throws Exception;
}
```

（4）修改Ihrm_system模块的 UserController

```
    @Autowired
    private DepartmentFeignClient departmentFeignClient;

    //测试通过系统微服务调用企业微服务方法
    @RequestMapping(value = "/test/{id}")
    public void findDeptById(@PathVariable  String id){
        Department dept = departmentFeignClient.findById(id);
        System.out.println(dept);
    }
```

（5）配置Feign拦截器添加请求头

```
/**
 * FeignConfiguration 过滤器，配置请求头信息
 */
@Configuration
public class FeignConfiguration {
    @Bean
    public RequestInterceptor requestInterceptor() {
        return new RequestInterceptor() {
            @Override
            public void apply(RequestTemplate template) {
                ServletRequestAttributes attributes = (ServletRequestAttributes)
RequestContextHolder
                        .getRequestAttributes();
                if (attributes != null) {
                    HttpServletRequest request = attributes.getRequest();
                    Enumeration<String> headerNames = request.getHeaderNames();
                    if (headerNames != null) {
                        while (headerNames.hasMoreElements()) {
                            String name = headerNames.nextElement();
                            String values = request.getHeader(name);
                            template.header(name, values);
                        }
                    }
                }
            }
        };
    }
}
```

2 POI报表的概述

2.1 需求说明

在企业级应用开发中，Excel报表是一种最常见的报表需求。Excel报表开发一般分为两种形式：

- 为了方便操作，基于Excel的报表批量上传数据
- 通过java代码生成Excel报表。

在Saas-HRM系统中，也有大量的报表操作，那么接下来的课程就是一起来学习企业级的报表开发。

2.2 Excel的两种形式

目前世面上的Excel分为两个大的版本Excel2003和Excel2007及以上两个版本，两者之间的区别如下：

	Excel 2003	Excel 2007
后缀	xls	xlsx
结构	二进制格式，其核心结构是复合文档类型的结构	XML类型结构
单sheet数据量	行：65535；列：256	行：1048576；列：16384
特点	存储容量有限	基于xml压缩，占用空间小，操作效率高

Excel2003是一个特有的二进制格式，其核心结构是复合文档类型的结构，存储数据量较小；Excel2007 的核心结构是 XML 类型的结构，采用的是基于 XML 的压缩方式，使其占用的空间更小，操作效率更高

2.3 常见excel操作工具

Java中常见的用来操作Excl的方式一般有2种：JXL和POI。

- JXL只能对Excel进行操作,属于比较老的框架，它只支持到Excel 95-2000的版本。现在已经停止更新和维护。
- POI是apache的项目,可对微软的Word,Excel,Ppt进行操作,包括office2003和2007,Excl2003和2007。poi现在一直有更新。所以现在主流使用POI。

2.4 POI的概述

Apache POI是Apache软件基金会的开源项目，由Java编写的免费开源的跨平台的 Java API，Apache POI提供API给Java语言操作Microsoft Office的功能。

2.5 POI的应用场景

1. 数据报表生成
2. 数据备份
3. 数据批量上传

3 POI的入门操作

3.1 搭建环境

```
<dependencies>
    <dependency>
        <groupId>org.apache.poi</groupId>
        <artifactId>poi</artifactId>
        <version>4.0.1</version>
    </dependency>
    <dependency>
        <groupId>org.apache.poi</groupId>
        <artifactId>poi-ooxml</artifactId>
        <version>4.0.1</version>
    </dependency>
    <dependency>
        <groupId>org.apache.poi</groupId>
        <artifactId>poi-ooxml-schemas</artifactId>
        <version>4.0.1</version>
    </dependency>
</dependencies>
```

3.2 POI结构说明

HSSF提供读写Microsoft Excel XLS格式档案的功能。

XSSF提供读写Microsoft Excel OOXML XLSX格式档案的功能。

HWPF提供读写Microsoft Word DOC格式档案的功能。

HSLF提供读写Microsoft PowerPoint格式档案的功能。

HDGF提供读Microsoft Visio格式档案的功能。

HPBF提供读Microsoft Publisher格式档案的功能。

HSMF提供读Microsoft Outlook格式档案的功能。

3.3 API介绍

API名称	
Workbook	Excel的文档对象,针对不同的Excel类型分为：HSSFWorkbook（2003）和XSSFWorkbool（2007）
Sheet	Excel的表单
Row	Excel的行
Cell	Excel的格子单元
Font	Excel字体
CellStyle	格子单元样式

3.4 基本操作

3.4.1 创建Excel

```
public class PoiTest01 {
    //测试创建excel文件
    public static void main(String[] args) throws Exception {
        //1.创建workbook工作簿
        Workbook wb = new XSSFWorkbook();
        //2.创建表单Sheet
        Sheet sheet = wb.createSheet("test");
        //3.文件流
        FileOutputStream fos = new FileOutputStream("E:\\test.xlsx");
        //4.写入文件
        wb.write(fos);
        fos.close();
    }
}
```

3.4.2 创建单元格

```
    //测试创建单元格
    public static void main(String[] args) throws Exception {
        //1.创建workbook工作簿
        Workbook wb = new XSSFWorkbook();
        //2.创建表单Sheet
        Sheet sheet = wb.createSheet("test");
        //3.创建行对象，从0开始
        Row row = sheet.createRow(3);
        //4.创建单元格，从0开始
        Cell cell = row.createCell(0);
        //5.单元格写入数据
        cell.setCellValue("传智播客");
        //6.文件流
        FileOutputStream fos = new FileOutputStream("E:\\test.xlsx");
        //7.写入文件
```

```
        wb.write(fos);
        fos.close();
    }
```

3.4.3 设置格式

```
        //创建单元格样式对象
        CellStyle cellStyle = wb.createCellStyle();

        //设置边框
        cellStyle.setBorderBottom(BorderStyle.DASH_DOT);//下边框
        cellStyle.setBorderTop(BorderStyle.HAIR);//上边框

        //设置字体
        Font font = wb.createFont();//创建字体对象
        font.setFontName("华文行楷");//设置字体
        font.setFontHeightInPoints((short)28);//设置字号
        cellStyle.setFont(font);

        //设置宽高
        sheet.setColumnWidth(0, 31 * 256);//设置第一列的宽度是31个字符宽度
        row.setHeightInPoints(50);//设置行的高度是50个点

        //设置居中显示
        cellStyle.setAlignment(HorizontalAlignment.CENTER);//水平居中
        cellStyle.setVerticalAlignment(VerticalAlignment.CENTER);//垂直居中

        //设置单元格样式
        cell.setCellStyle(cellStyle);

        //合并单元格
        CellRangeAddress  region =new CellRangeAddress(0, 3, 0, 2);
        sheet.addMergedRegion(region);
```

3.4.4 绘制图形

```
    //绘制图形
    public static void main(String[] args) throws Exception {
        //1.创建workbook工作簿
        Workbook wb = new XSSFWorkbook();
        //2.创建表单Sheet
        Sheet sheet = wb.createSheet("test");
        //读取图片流
        FileInputStream stream=new FileInputStream("e:\\logo.jpg");
        byte[] bytes= IOUtils.toByteArray(stream);
        //读取图片到二进制数组
        stream.read(bytes);
        //向Excel添加一张图片,并返回该图片在Excel中的图片集合中的下标
        int pictureIdx = wb.addPicture(bytes,Workbook.PICTURE_TYPE_JPEG);
        //绘图工具类
        CreationHelper helper = wb.getCreationHelper();
```

```
        //创建一个绘图对象
        Drawing<?> patriarch = sheet.createDrawingPatriarch();
        //创建锚点,设置图片坐标
        ClientAnchor anchor = helper.createClientAnchor();
        anchor.setCol1(0);//从0开始
        anchor.setRow1(0);//从0开始

        //创建图片
        Picture picture = patriarch.createPicture(anchor, pictureIdx);
        picture.resize();
        //6.文件流
        FileOutputStream fos = new FileOutputStream("E:\\test.xlsx");
        //7.写入文件
        wb.write(fos);
        fos.close();
    }
```

3.4.5 加载Excel

```
public class PoiTest06 {
    //单元格样式
    public static void main(String[] args) throws Exception {
        //1.创建workbook工作簿
        Workbook wb = new XSSFWorkbook("E:\\demo.xlsx");
        //2.获取sheet 从0开始
        Sheet sheet = wb.getSheetAt(0);

        int totalRowNum = sheet.getLastRowNum();

        Row row = null;
        Cell cell = null;

        //循环所有行
        for (int rowNum = 3; rowNum <sheet.getLastRowNum(); rowNum++) {
            row = sheet.getRow(rowNum);
            StringBuilder sb = new StringBuilder();
            //循环每行中的所有单元格
            for(int cellNum = 2; cellNum < row.getLastCellNum();cellNum++) {
                cell = row.getCell(cellNum);
                sb.append(getValue(cell)).append("-");
            }
            System.out.println(sb.toString());
        }
    }

    //获取数据
    private static Object getValue(Cell cell) {
        Object value = null;
        switch (cell.getCellType()) {
            case STRING: //字符串类型
                value = cell.getStringCellValue();
                break;
```

```
            case BOOLEAN: //boolean类型
                value = cell.getBooleanCellValue();
                break;
            case NUMERIC: //数字类型（包含日期和普通数字）
                if(DateUtil.isCellDateFormatted(cell)) {
                    value = cell.getDateCellValue();
                }else{
                    value = cell.getNumericCellValue();
                }
                break;
            case FORMULA: //公式类型
                value = cell.getCellFormula();
                break;
            default:
                break;
        }
        return value;
    }
}
```

4 POI报表导入

4.1 需求分析

实现批量导入员工功能，页面端上传excel表格，服务端解析表格获取数据，批量新增用户

4.2 员工导入

4.2.1 搭建环境

父模块pom文件添加依赖

```
        <dependency>
            <groupId>org.apache.poi</groupId>
            <artifactId>poi</artifactId>
```

```
            <version>4.0.1</version>
        </dependency>
        <dependency>
            <groupId>org.apache.poi</groupId>
            <artifactId>poi-ooxml</artifactId>
            <version>4.0.1</version>
        </dependency>
        <dependency>
            <groupId>org.apache.poi</groupId>
            <artifactId>poi-ooxml-schemas</artifactId>
            <version>4.0.1</version>
        </dependency>
```

4.2.2 实现Excel上传

（1）用户实体类配置构造方法

```
    //objs数据位置和excel上传位置一致。
    public User(Object []objs,String companyId,String companyName) {
        //默认手机号excel读取为字符串会存在科学记数法问题，转化处理
        this.mobile = new DecimalFormat("#").format(objs[2]);
        this.username = objs[1].toString();
        this.createTime = new Date();
        this.timeOfEntry = (Date) objs[5];
        this.formOfEmployment = ((Double) objs[4]).intValue() ;
        this.workNumber = new DecimalFormat("#").format(objs[3]).toString();
        this.companyId = companyId;
        this.companyName = companyName;
    }
```

（2）在系统微服务`UserController`中添加上传方法

```
    //批量导入数据
    @RequestMapping(value="/user/import", method = RequestMethod.POST)
    public Result importExcel(@RequestParam(name = "file") MultipartFile attachment)
throws Exception {
        //根据上传流信息创建工作簿
        Workbook workbook = WorkbookFactory.create(attachment.getInputStream());
        //获取第一个sheet
        Sheet sheet = workbook.getSheetAt(0);
        List<User> users = new ArrayList<>();
        //从第二行开始获取数据
        for (int rowNum = 1; rowNum <sheet.getLastRowNum(); rowNum++) {
            Row row = sheet.getRow(rowNum);
            Object objs[] = new Object[row.getLastCellNum()];
            //从第二列获取数据
            for(int cellNum = 1; cellNum < row.getLastCellNum();cellNum++) {
                Cell cell = row.getCell(cellNum);
                objs[cellNum] = getValue(cell);
            }
            //根据每一列构造用户对象
```

```
            User user = new User(objs,companyId,companyName);
            user.setDepartmentId(objs[objs.length-1].toString());
            users.add(user);
        }
        //第一个参数：用户列表，第二个参数：部门编码
        userService.save(users,objs[objs.length-1].toString());
        return Result.SUCCESS();
    }
```

4.2.3 调用企业微服务获取部门数据

（1）在Ihrm_system模块创建com.ihrm.system.client包，包下创建接口

```
//远程调用企业微服务，根据企业编码code和企业名称获取企业信息
@FeignClient(value = "ihrm-company")
public interface DepartmentFeignClient {
    @RequestMapping(value = "/company/departments/search/", method =
RequestMethod.POST)
    public Department findById(@RequestParam(value = "code") String code,
                               @RequestParam(value = "companyId") String companyId)
throws Exception;
}
```

（2）修改 `UserService`，注入 `DepartmentFeignClient`

```
@Autowired
private DepartmentFeignClient departmentFeignClient;
```

4.2.4 保存全部用户

`UserService` 中添加保存全部的方法

```
    @Transactional
    public void save(List<User> users) throws Exception {
        for (User user : users) {
            //配置密码
            user.setPassword(new Md5Hash("123456",user.getMobile(),3).toString());
            //配置id
            user.setId(idWorker.nextId()+"");
            //其他基本属性
            user.setInServiceStatus(1);
            user.setEnableState(1);
            user.setLevel("user");
            //获取部门信息
            Department dept =
departmentFeignClient.findById(user.getDepartmentId(),user.getCompanyId());
            if(dept != null) {
                user.setDepartmentId(dept.getId());
                user.setDepartmentName(dept.getName());
            }
            userDao.save(user);
```

```
        }
    }
```

5 POI报表导出

5.1 需求分析

完成当月人事报表的导出：包含当月入职员工信息，离职员工信息

5.2 人事报表导出

5.2.1 步骤分析

- 构造Excel表格数据
- 创建工作簿
- 创建sheet
- 创建行对象
- 创建单元格对象
- 填充数据，设置样式
- 下载

5.2.2 代码实现

（1）配置controller

```
@RequestMapping(value = "/export/{month}", method = RequestMethod.GET)
    public void export(@PathVariable(name = "month") String month) throws Exception {
        //1.构造数据
        List<EmployeeReportResult> list =
userCompanyPersonalService.findByReport(companyId,month+"%");
        //2.创建工作簿
        XSSFWorkbook workbook = new XSSFWorkbook();
        //3.构造sheet
        String[] titles = {"编号", "姓名", "手机","最高学历", "国家地区", "护照号", "籍贯",
"生日", "属相","入职时间","离职类型","离职原因","离职时间"};

        Sheet sheet = workbook.createSheet();
```

```
        Row row = sheet.createRow(0);

        AtomicInteger headersAi = new AtomicInteger();

        for (String title : titles) {
            Cell cell = row.createCell(headersAi.getAndIncrement());
            cell.setCellValue(title);
        }

        AtomicInteger datasAi = new AtomicInteger(1);

        Cell cell = null;
        for (EmployeeReportResult report : list) {
            Row dataRow = sheet.createRow(datasAi.getAndIncrement());
            //编号
            cell = dataRow.createCell(0);
            cell.setCellValue(report.getUserId());
            //姓名
            cell = dataRow.createCell(1);
            cell.setCellValue(report.getUsername());
            //手机
            cell = dataRow.createCell(2);
            cell.setCellValue(report.getMobile());
            //最高学历
            cell = dataRow.createCell(3);
            cell.setCellValue(report.getTheHighestDegreeOfEducation());
            //国家地区
            cell = dataRow.createCell(4);
            cell.setCellValue(report.getNationalArea());
            //护照号
            cell = dataRow.createCell(5);
            cell.setCellValue(report.getPassportNo());
            //籍贯
            cell = dataRow.createCell(6);
            cell.setCellValue(report.getNativePlace());
            //生日
            cell = dataRow.createCell(7);
            cell.setCellValue(report.getBirthday());
            //属相
            cell = dataRow.createCell(8);
            cell.setCellValue(report.getZodiac());
            //入职时间
            cell = dataRow.createCell(9);
            cell.setCellValue(report.getTimeOfEntry());
            //离职类型
            cell = dataRow.createCell(10);
            cell.setCellValue(report.getTypeOfTurnover());
            //离职原因
            cell = dataRow.createCell(11);
            cell.setCellValue(report.getReasonsForLeaving());
            //离职时间
            cell = dataRow.createCell(12);
```

```
            cell.setCellValue(report.getResignationTime());
        }

        String fileName = URLEncoder.encode(month+"人员信息.xlsx", "UTF-8");
        response.setContentType("application/octet-stream");
        response.setHeader("content-disposition", "attachment;filename=" + new
String(fileName.getBytes("ISO8859-1")));
        response.setHeader("filename", fileName);
        workbook.write(response.getOutputStream());
    }
```

（2）添加service

```
    //根据企业id和年月查询
    public List<EmployeeReportResult> findByReport(String companyId, String month) {
        return userCompanyPersonalDao.findByReport(companyId,month);
    }
```

（3）dao层实现

```
    @Query(value = "select new
com.ihrm.domain.employee.response.EmployeeReportResult(a,b) " +
            "FROM UserCompanyPersonal a LEFT JOIN EmployeeResignation b ON
a.userId=b.userId WHERE a.companyId = ?1 AND a.timeOfEntry LIKE ?2 OR
(b.resignationTime LIKE ?2)")
    List<EmployeeReportResult> findByReport(String companyId, String month);
```