

# 员工管理、分类管理



黑马程序员  
[www.itheima.com](http://www.itheima.com)

传智教育旗下  
高端IT教育品牌

益喜外卖  
THE SKY TAKE-OUT

营业中

营业状态设置

管理员

工作台

数据统计

订单管理

套餐管理

菜品管理

分类管理

员工管理

分类名称:

分类类型:

查询

新增菜品分类

新增套餐分类

分类名称	分类类型	排序	状态	操作时间	操作
蜀味烤鱼	菜品分类	4	启用	2022-08-31 14:27	修改 删除 禁用
蜀味牛蛙	菜品分类	5	启用	2022-08-31 14:39	修改 删除 禁用
特色蔬菜	菜品分类	6	启用	2022-06-09 22:17	修改 删除 禁用
新鲜时蔬	菜品分类	7	启用	2022-06-09 22:18	修改 删除 禁用
水煮鱼	菜品分类	8	启用	2022-06-09 22:23	修改 删除 禁用
传统主食	菜品分类	9	启用	2022-06-09 22:18	修改 删除 禁用
酒水饮料	菜品分类	10	启用	2022-06-09 22:09	修改 删除 禁用
汤类	菜品分类	11	启用	2022-06-10 10:51	修改 删除 禁用
人气套餐	套餐分类	12	启用	2022-06-10 11:04	修改 删除 禁用
商务套餐	套餐分类	13	启用	2022-06-10 11:04	修改 删除 禁用



# 目录

Contents

- ◆ 新增员工
- ◆ 员工分页查询
- ◆ 启用禁用员工账号
- ◆ 编辑员工
- ◆ 导入分类模块功能代码



## 新增员工

- 需求分析和设计
- 代码开发
- 功能测试
- 代码完善

## 需求分析和设计

产品原型:

添加员工

\* 账号:

请输入账号

账号必须是唯一的

\* 员工姓名:

请输入员工姓名

\* 手机号:

手机号

手机号为合法的11位手机号码

\* 性别:

☐ 男

☒ 女

身份证号:

身份证

身份证号为合法的18位身份证号码

密码默认为123456

保存

保存并继续添加员工

返回

## 需求分析和设计

### 接口设计:

#### 基本信息

**Path:** /admin/employee

**Method:** POST

**接口描述:**

#### 请求参数

##### Headers

参数名称	参数值	是否必须	示例	备注
Content-Type	application/json	是		

##### Body

名称	类型	是否必须	默认值	备注	其他信息
id	integer	非必须		员工id	format: int64
idNumber	string	必须		身份证	
name	string	必须		姓名	
phone	string	必须		手机号	
sex	string	必须		性别	
username	string	必须		用户名	

#### 返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object	非必须			
msg	string	非必须			

### 本项目约定:

- **管理端**发出的请求, 统一使用 **/admin** 作为前缀
- **用户端**发出的请求, 统一使用 **/user** 作为前缀

## 需求分析和设计

数据库设计 (employee表) :

字段名	数据类型	说明	备注
id	bigint	主键	自增
name	varchar(32)	姓名	
username	varchar(32)	用户名	唯一
password	varchar(64)	密码	
phone	varchar(11)	手机号	
sex	varchar(2)	性别	
id_number	varchar(18)	身份证号	
status	Int	账号状态	1正常 0锁定
create_time	Datetime	创建时间	
update_time	datetime	最后修改时间	
create_user	bigint	创建人id	
update_user	bigint	最后修改人id	



## 新增员工

- 需求分析和设计
- 代码开发
- 功能测试
- 代码完善



## 代码开发

根据新增员工接口设计对应的DTO:

Body

名称	类型	是否必须	默认值	备注	其他信息
id	integer	非必须		员工id	format: int64
idNumber	string	必须		身份证	
name	string	必须		姓名	
phone	string	必须		手机号	
sex	string	必须		性别	
username	string	必须		用户名	



```
private Long id;  
  
private String username;  
  
private String name;  
  
private String password;  
  
private String phone;  
  
private String sex;  
  
private String idNumber;  
  
private Integer status;  
  
private LocalDateTime createTime;  
  
private LocalDateTime updateTime;  
  
private Long createUser;  
  
private Long updateUser;
```

implements Serializable

注意：当前端提交的数据和实体类中对应的属性差别比较大时，建议使用DTO来封装数据

## 代码开发

在EmployeeController中创建新增员工方法，接收前端提交的参数：

```
/**
 * 新增员工
 * @param employeeDTO
 * @return
 */
@PostMapping
@ApiOperation("新增员工接口")
public Result save(@RequestBody EmployeeDTO employeeDTO){
    log.info("新增员工: {}",employeeDTO);
    return Result.success();
}
```

EmployeeController

## 代码开发

在EmployeeService接口中声明新增员工方法：

```
/**
 * 新增员工
 * @param employeeDTO
 */
void save(EmployeeDTO employeeDTO);
```

EmployeeService

## 代码开发

在EmployeeServiceImpl中实现新增员工方法：

```
public void save(EmployeeDTO employeeDTO) {  
    Employee employee = new Employee();  
    //属性拷贝  
    BeanUtils.copyProperties(employeeDTO, employee);  
  
    //账号状态默认为1, 正常状态  
    employee.setStatus(StatusConstant.ENABLE);  
    //默认密码为123456  
    employee.setPassword(DigestUtils.md5DigestAsHex(PasswordConstant.DEFAULT_PASSWORD.getBytes()));  
    //创建人、创建时间、修改人、修改时间  
    employee.setCreateTime(LocalDate.now());  
    employee.setUpdateTime(LocalDate.now());  
    employee.setCreateUser(10L);  
    employee.setUpdateUser(10L);  
  
    employeeMapper.insert(employee);  
}
```

EmployeeServiceImpl

## 代码开发

在EmployeeMapper中声明insert方法:

```
/**
 * 插入数据
 * @param employee
 */
@Insert("insert into employee" +
        "(name, username, password, phone, sex, id_number, status, create_time, update_time, create_user, update_user)" +
        "VALUES " +
        "(#{name}, #{username}, #{password}, #{phone}, #{sex}, #{idNumber}, #{status},
#{createTime},#{updateTime},#{createUser}, #{updateUser})")
void insert(Employee employee);
```

EmployeeMapper



## 新增员工

- 需求分析和设计
- 代码开发
- 功能测试
- 代码完善

## 功能测试

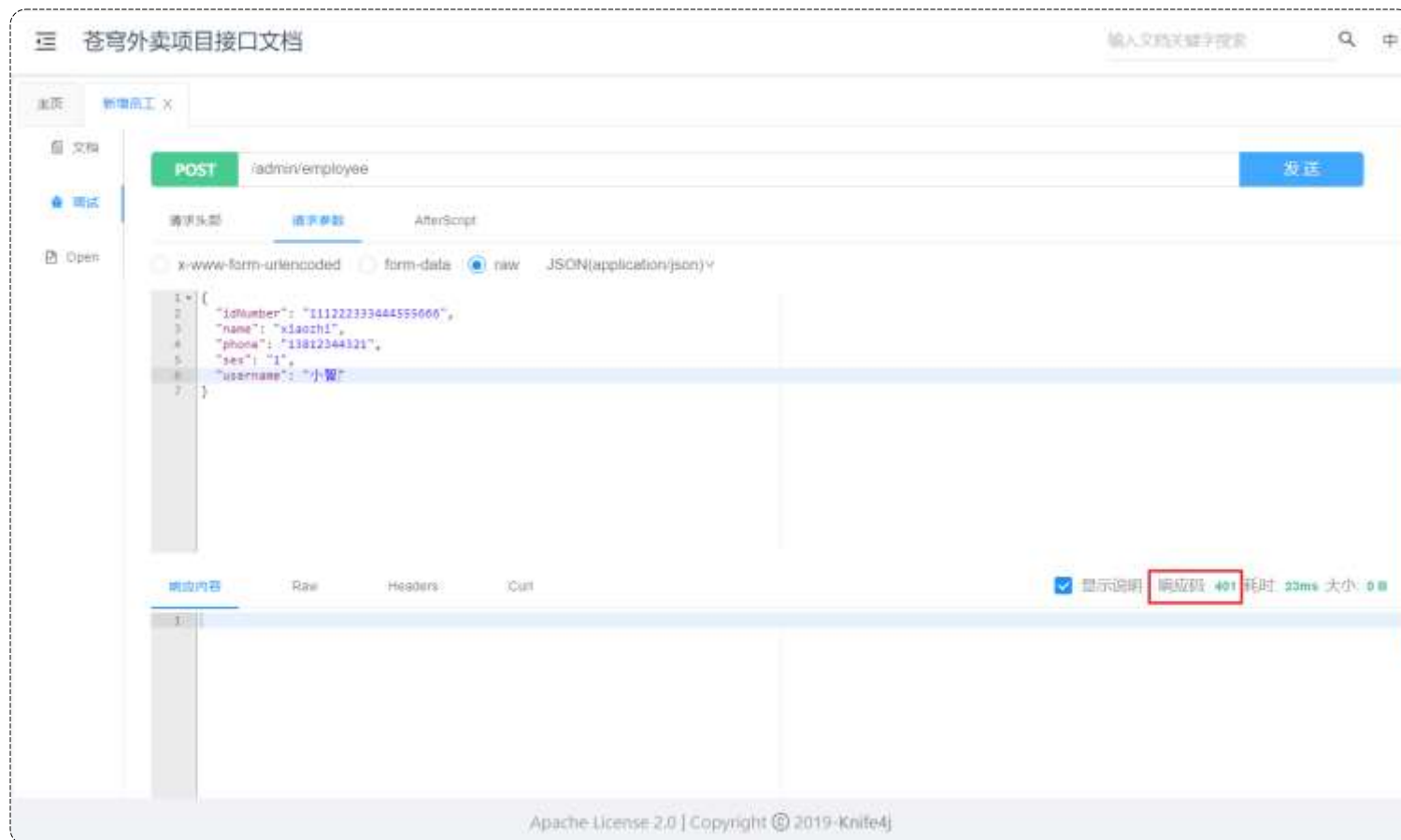
功能测试方式：

- 通过接口文档测试
- 通过前后端联调测试

**注意：**由于开发阶段前端和后端是并行开发的，后端完成某个功能后，此时前端对应的功能可能还没有开发完成，导致无法进行前后端联调测试。所以在开发阶段，后端测试主要以接口文档测试为主。

## 功能测试

通过接口文档进行功能测试：





## 功能测试

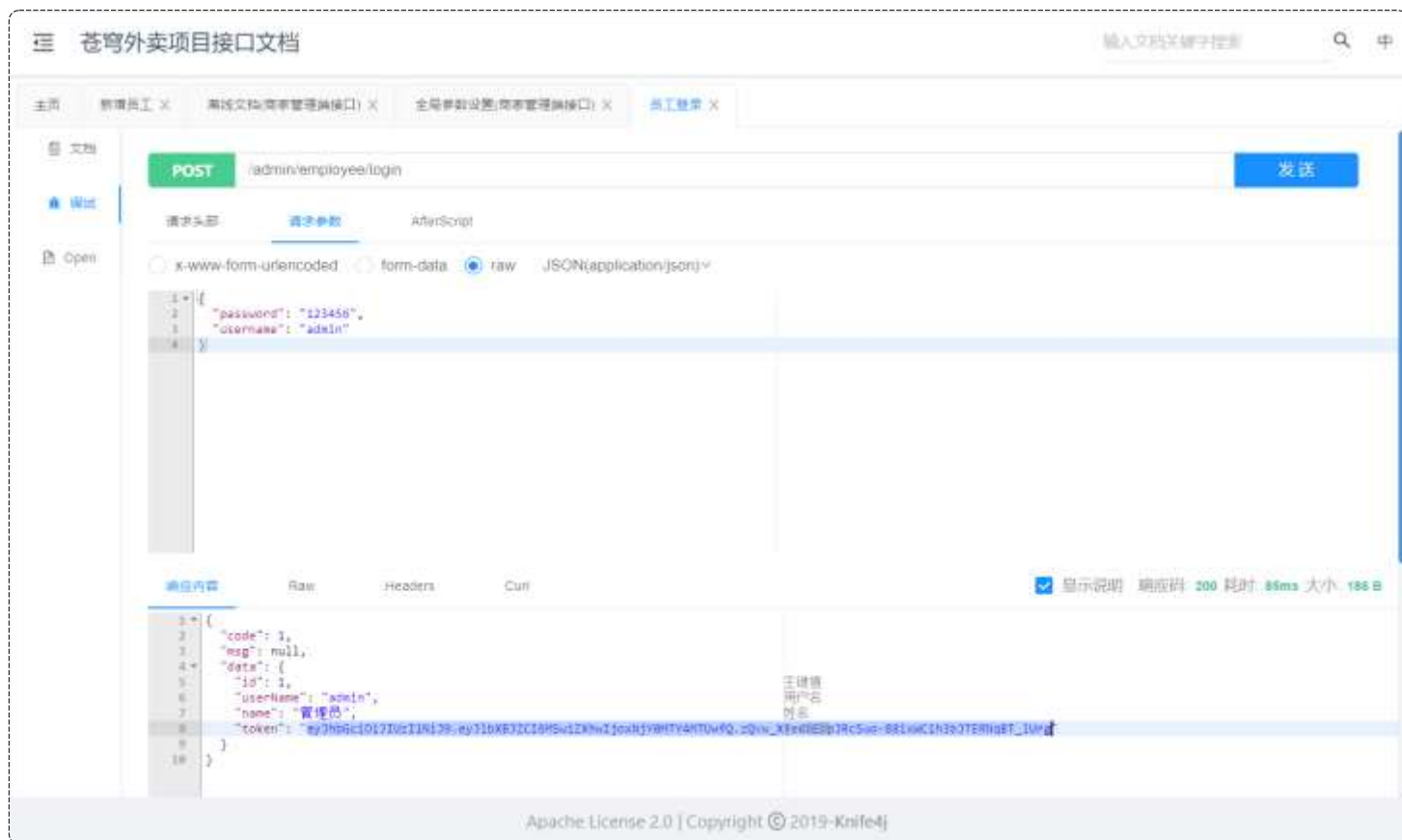
```
public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {  
    Log.info("jwt校验...");  
  
    //判断当前拦截到的是Controller的方法还是其他资源  
    if(!(handler instanceof HandlerMethod)){  
        //当前拦截到的不是动态方法，直接放行  
        return true;  
    }  
  
    //1、从请求头中获取令牌  
    String token = request.getHeader(jwtProperties.getAdminTokenName());  
  
    //2、校验令牌  
    try{  
        Claims claims = JwtUtil.parseJWT(jwtProperties.getAdminSecretKey(), token);  
        Long empId = Long.valueOf(claims.get(JwtClaimsConstant.EMP_ID).toString());  
        //3、通过，放行  
        return true;  
    }catch (Exception ex){  
        //4、不通过，响应401状态码  
        response.setStatus(401);  
        return false;  
    }  
}
```

JwtTokenAdminInterceptor

由于JWT令牌校验失败，导致EmployeeController的save方法没有被调用

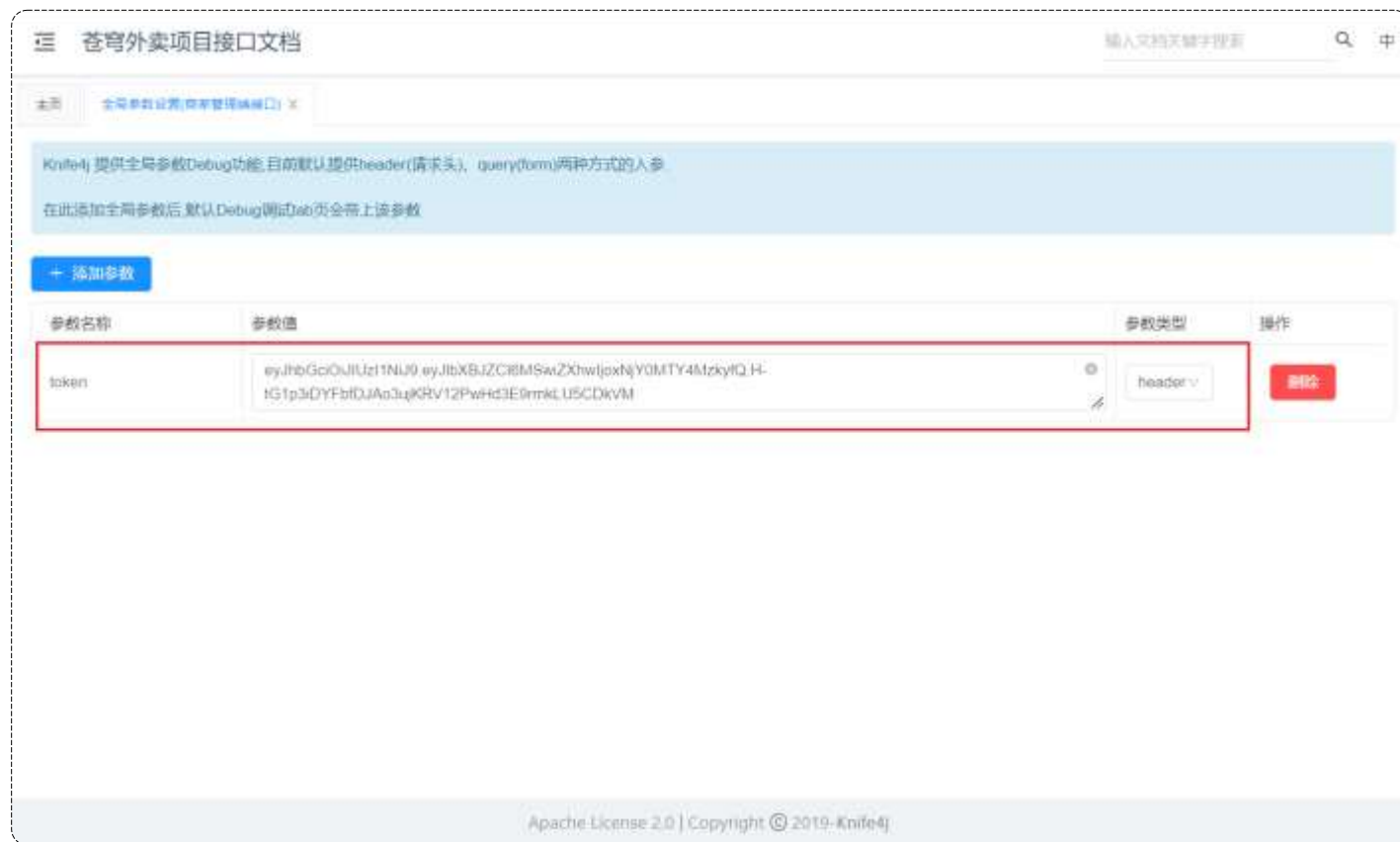
## 功能测试

调用员工登录接口获得一个合法的JWT令牌:



## 功能测试

将合法的JWT令牌添加到全局参数中：





## 功能测试

## 高级软件人才培养专家



## 新增员工

- 需求分析和设计
- 代码开发
- 功能测试
- 代码完善

## 代码完善

程序存在的问题:

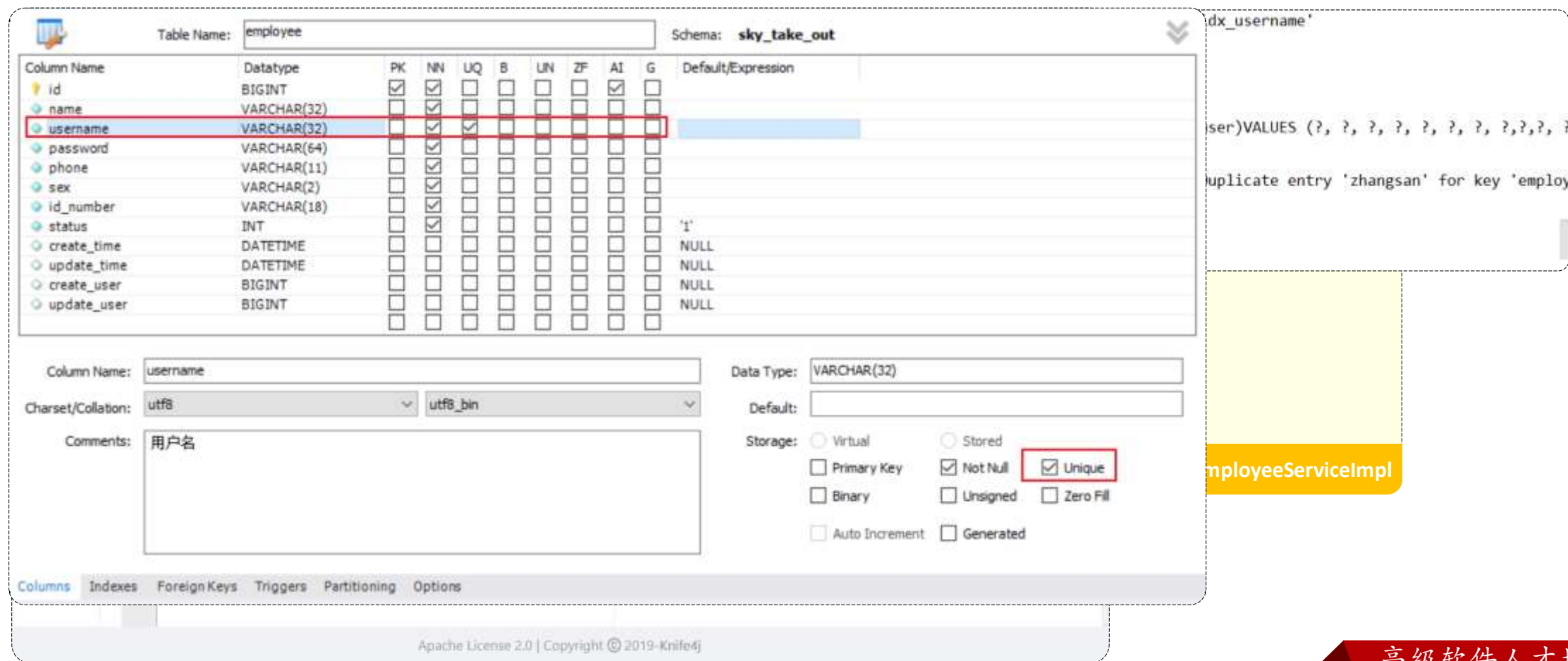


Table Name:  Schema: **sky\_take\_out**

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id	BIGINT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
name	VARCHAR(32)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
username	VARCHAR(32)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
password	VARCHAR(64)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
phone	VARCHAR(11)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
sex	VARCHAR(2)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
id_number	VARCHAR(18)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
status	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	'1'
create_time	DATETIME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
update_time	DATETIME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
create_user	BIGINT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
update_user	BIGINT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL

Column Name:  Data Type:

Charset/Collation:

Comments:

Storage: ☐ Virtual ☐ Stored

☐ Primary Key ☒ Not Null ☒ Unique

☐ Binary ☐ Unsigned ☐ Zero Fill

☐ Auto Increment ☐ Generated

Columns Indexes Foreign Keys Triggers Partitioning Options

Apache License 2.0 | Copyright © 2019-Knife4j

dx\_username'

ser)VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?

uplicate entry 'zhangsan' for key 'employ

mployeeServiceImpl

## 代码完善

针对第一个问题，可以通过全局异常处理器来处理：

```
/**
 * 捕获sql异常
 * @param ex
 * @return
 */
@ExceptionHandler
public Result exceptionHandler(SQLIntegrityConstraintViolationException ex){
    Log.error("异常信息: {}", ex.getMessage());
    String message = ex.getMessage();
    if(message.contains("Duplicate entry")){
        String[] split = message.split(" ");
        String name = split[2];
        return Result.error(name + MessageConstant.ALREADY_EXISTS);
    }
    return Result.error(MessageConstant.UNKNOWN_ERROR);
}
```

GlobalExceptionHandler

## 代码完善

针对第二个问题，需要通过某种方式动态获取当前登录员工的id:





## 代码完善

员工登录成功后会生成JWT令牌并响应给前端：

```
// 登录成功后，生成jwt令牌
Map<String, Object> claims = new HashMap<>();
claims.put(JwtClaimsConstant.EMP_ID, employee.getId());
String token = JwtUtil.createJWT(
    jwtProperties.getAdminSecretKey(),
    jwtProperties.getAdminTtl(),
    claims);
```

EmployeeController

## 代码完善

后续请求中，前端会携带JWT令牌，通过JWT令牌可以解析出当前登录员工id:

```
//1、从请求头中获取令牌
String token = request.getHeader(jwtProperties.getAdminTokenName());

//2、校验令牌
try{
    Claims claims = JwtUtil.parseJWT(jwtProperties.getAdminSecretKey(), token);
    Long empId = Long.valueOf(claims.get(JwtClaimsConstant.EMP_ID).toString());
    //3、通过，放行
    return true;
}catch (Exception ex){
    //4、不通过，响应401状态码
    response.setStatus(401);
    return false;
}
```

JwtTokenAdminInterceptor

解析出登录员工id后，如何传递给Service的save方法?

## 代码完善

**ThreadLocal** 并不是一个Thread，而是Thread的局部变量。

ThreadLocal为每个线程提供单独一份存储空间，具有线程隔离的效果，只有在线程内才能获取到对应的值，线程外则不能访问。

ThreadLocal常用方法：

- `public void set(T value)` 设置当前线程的线程局部变量的值
- `public T get()` 返回当前线程所对应的线程局部变量的值
- `public void remove()` 移除当前线程的线程局部变量

**注意：**客户端发送的每次请求，后端的Tomcat服务器都会分配一个单独的线程来处理请求

## 代码完善

初始工程中已经封装了 ThreadLocal 操作的工具类：

```
public class BaseContext {  
  
    public static ThreadLocal<Long> threadLocal = new ThreadLocal<>();  
  
    public static void setCurrentId(Long id) {  
        threadLocal.set(id);  
    }  
  
    public static Long getCurrentId() {  
        return threadLocal.get();  
    }  
  
    public static void removeCurrentId() {  
        threadLocal.remove();  
    }  
  
}
```

BaseContext

## 代码完善

在拦截器中解析出当前登录员工id，并放入线程局部变量中：

```
//1、从请求头中获取令牌
String token = request.getHeader(jwtProperties.getAdminTokenName());

//2、校验令牌
try{
    Claims claims = JwtUtil.parseJWT(jwtProperties.getAdminSecretKey(), token);
    Long empId = Long.valueOf(claims.get(JwtClaimsConstant.EMP_ID).toString());
    BaseContext.setCurrentId(empId);
    //3、通过, 放行
    return true;
}catch (Exception ex){
    //4、不通过, 响应401状态码
    response.setStatus(401);
    return false;
}
```

JwtTokenAdminInterceptor

## 代码完善

在Service中获取线程局部变量中的值：

```
public void save(EmployeeDTO employeeDTO) {  
    Employee employee = new Employee();  
    //属性拷贝  
    BeanUtils.copyProperties(employeeDTO, employee);  
  
    //账号状态默认为1, 正常状态  
    employee.setStatus(StatusConstant.ENABLE);  
    //默认密码为123456  
    employee.setPassword(DigestUtils.md5DigestAsHex("123456".getBytes()));  
    //创建人、创建时间、修改人、修改时间  
    employee.setCreateTime(LocalDateTime.now());  
    employee.setUpdateTime(LocalDateTime.now());  
    employee.setCreateUser(BaseContext.getCurrentId());  
    employee.setUpdateUser(BaseContext.getCurrentId());  
  
    employeeMapper.insert(employee);  
}
```

EmployeeServiceImpl



# 目录

Contents

- ◆ 新增员工
- ◆ 员工分页查询
- ◆ 启用禁用员工账号
- ◆ 编辑员工
- ◆ 导入分类模块功能代码



## 员工分页查询

- 需求分析和设计
- 代码开发
- 功能测试
- 代码完善



## 需求分析和设计

产品原型:

员工管理

员工姓名

请输入员工姓名

搜索

添加员工

员工姓名	账号	手机号	账号状态	最后操作时间	操作
管理员	13333333333	13333333333	禁用	2022-4-29 09:39	修改 启用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除

总共 05 个项目 < 1 2 3 4 5 >

业务规则:

- 根据页码展示员工信息
- 每页展示10条数据
- 分页查询时可以根据需要，输入员工姓名进行查询

## 需求分析和设计

接口设计:

### 基本信息

**Path:** /admin/employee/page

**Method:** GET

**接口描述:**

### 请求参数

**Query**

参数名称	是否必须	示例	备注
name	否	张三	员工姓名
page	是	1	页码
pageSize	是	10	每页记录数

### 返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	number	必须			
msg	null	非必须			
data	object	必须			
└─ total	number	必须			
└─ records	object []	必须			item 类型: object
└─ id	number	必须			
└─ username	string	必须			
└─ name	string	必须			
└─ password	string	必须			
└─ phone	string	必须			
└─ sex	string	必须			
└─ idNumber	string	必须			
└─ status	number	必须			
└─ createTime	string,null	必须			
└─ updateTime	string	必须			
└─ createUser	number,null	必须			
└─ updateUser	number	必须			



## 员工分页查询

- 需求分析和设计
- 代码开发
- 功能测试
- 代码完善

## 代码开发

根据分页查询接口设计对应的DTO:

### 请求参数

#### Query

参数名称	是否必须	示例	备注
name	否	张三	员工姓名
page	是	1	页码
pageSize	是	10	每页记录数



```
@Data
public class EmployeePageQueryDTO implements Serializable {

    //员工姓名
    private String name;

    //页码
    private int page;

    //每页显示记录数
    private int pageSize;

}
```

## 代码开发

后面所有的分页查询，统一都封装成PageResult对象：

```
/**
 * 封装分页查询结果
 */
@Data
@AllArgsConstructor
@NoArgsConstructor
public class PageResult implements Serializable {

    private long total; //总记录数

    private List records; //当前页数据集合

}
```

## 代码开发

员工信息分页查询后端返回的对象类型为：Result<PageResult>

名称	类型	是否必须	默认值	备注	其他信息
code	number	非必须			
msg	null	非必须			
data	object	必须			
├ total	number	必须			
├ records	object []	必须			item 类型: object
├ id	number	必须			
├ username	string	必须			
├ name	string	必须			
├ password	string	必须			
├ phone	string	必须			
├ sex	string	必须			

## 代码开发

根据接口定义创建分页查询方法：

```
/**
 * 员工分页查询
 * @param employeePageQueryDTO
 * @return
 */
@GetMapping("/page")
@ApiOperation("员工分页查询")
public Result<PageResult> page(EmployeePageQueryDTO employeePageQueryDTO){
    Log.info("分页查询: {}", employeePageQueryDTO);
    PageResult pageResult = employeeService.pageQuery(employeePageQueryDTO);
    return Result.success(pageResult);
}
```

EmployeeController

## 代码开发

在EmployeeService接口中声明pageQuery方法：

```
/**  
 * 分页查询  
 * @param employeePageQueryDTO  
 * @return  
 */  
PageResult pageQuery(EmployeePageQueryDTO employeePageQueryDTO);
```

EmployeeService



## 代码开发

在 EmployeeServiceImpl 中实现 pageQuery 方法:

```
public PageResult pageQuery(EmployeePageQueryDTO employeePageQueryDTO) {  
    //select * from employee limit 10,20  
    //基于PageHelper插件实现动态分页查询  
    PageHelper.startPage(employeePageQueryDTO.getPage(), employeePageQueryDTO.getPageSize());  
    Page<Employee> page = employeeMapper.pageQuery(employeePageQueryDTO);  
    return new PageResult(page.getTotal(), page.getResult());  
}
```

EmployeeServiceImpl

**注意:** 此处使用 mybatis 的分页插件 PageHelper 来简化分页代码的开发。底层基于 mybatis 的拦截器实现。

```
<dependency>  
    <groupId>com.github.pagehelper</groupId>  
    <artifactId>pagehelper-spring-boot-starter</artifactId>  
    <version>${pagehelper}</version>  
</dependency>
```

pom.xml

## 代码开发

在 EmployeeMapper 中声明 pageQuery 方法:

```
/**
 * 分页查询
 * @param employeePageQueryDTO
 * @return
 */
Page<Employee> pageQuery(EmployeePageQueryDTO employeePageQueryDTO);
```

## 代码开发

在 EmployeeMapper.xml 中编写SQL:

```
<select id="pageQuery" resultType="com.sky.entity.Employee">
  select * from employee
  <where>
    <if test="name != null and name != ''">
      and name like concat('%',{name},%')
    </if>
  </where>
  order by create_time desc
</select>
```



## 员工分页查询

- 需求分析和设计
- 代码开发
- 功能测试
- 代码完善

## 功能测试

可以通过接口文档进行测试，也可以进行前后端联调测试，最后操作时间字段展示有问题，如下：

响应内容 Raw Headers Curl

```
1 {
2   "code": 1,
3   "msg": null,
4   "data": {
5     "total": 3,
6     "records": [
7       {
8         "id": 20,
9         "username": "admin123",
10        "name": "itcast",
11        "password": "e10adc3949ba59abbe56e057f20f883e",
12        "phone": "131",
13        "sex": "1",
14        "idNumber": "123",
15        "status": 1,
16        "createTime": [
17          2022,
18          5,
19          24,
20          11,
21          20,
22          24
23        ],
24        "updateTime": [
25          2022,
26          5,
27          24,
28          11,
29          20,
30          24
31        ],
32        "createUser": 1,
33        "updateUser": 1
34      },
```



账号状态	最后操作时间
• 启用	20215102249
• 启用	2022524112024
• 启用	2022524112934
• 启用	2022524114543
• 启用	202252412945



## 员工分页查询

- 需求分析和设计
- 代码开发
- 功能测试
- 代码完善

## 代码完善

解决方式:

- 方式一：在属性上加入注解，对日期进行格式化

```
@JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss")  
private LocalDateTime updateTime;
```

- 方式二：在 WebMvcConfiguration 中扩展Spring MVC的消息转换器，统一对日期类型进行格式化处理

```
/**  
 * 扩展mvc框架的消息转换器  
 * @param converters  
 */  
protected void extendMessageConverters(List<HttpMessageConverter<?>> converters) {  
    Log.info("开始扩展消息转换器...");  
  
    //创建一个消息转化器对象  
    MappingJackson2HttpMessageConverter converter = new MappingJackson2HttpMessageConverter();  
    //设置对象转换器，可以将Java对象转为json字符串  
    converter.setObjectMapper(new JacksonObjectMapper());  
  
    //将我们自己的转换器放入spring MVC框架的容器中  
    converters.add(0,converter);  
}
```

WebMvcConfiguration

## 代码完善

查看效果:

响应内容 Raw Headers Curl

```
1 {
2   "code": 1,
3   "msg": null,
4   "data": {
5     "total": 3,
6     "records": [
7       {
8         "id": 20,
9         "username": "admin123",
10        "name": "itcast",
11        "password": "e10adc3949ba59abbe56e057f20f883e",
12        "phone": "131",
13        "sex": "1",
14        "idNumber": "123",
15        "status": 1,
16        "createTime": "2022-05-24 11:20",
17        "updateTime": "2022-05-24 11:20",
18        "createUser": 1,
19        "updateUser": 1
20      },
21      {
22        "id": 22,
23        "username": "admin1234",
24        "name": "itcast",
25        "password": "e10adc3949ba59abbe56e057f20f883e",
26        "phone": "131",
27        "sex": "1",
28        "idNumber": "123",
29        "status": 1,
30        "createTime": "2022-05-24 11:45",
31        "updateTime": "2022-05-24 11:45",
32        "createUser": 1,
33        "updateUser": 1
34      },
35    ]
36  }
```



账号状态	最后操作时间
• 启用	2021-05-10 02:24
• 启用	2022-05-24 11:20
• 启用	2022-05-24 11:29
• 启用	2022-05-24 11:45
• 启用	2022-05-24 12:09





# 目录

Contents

- ◆ 新增员工
- ◆ 员工分页查询
- ◆ 启用禁用员工账号
- ◆ 编辑员工
- ◆ 导入分类模块功能代码



## 启用禁用员工账号

- 需求分析和设计
- 代码开发
- 功能测试

## 需求分析和设计

产品原型：

员工姓名

搜索

添加员工

员工姓名	账号	手机号	账号状态	最后操作时间	操作
管理员	13333333333	13333333333	禁用	2022-4-29 09:39	修改 启用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除

## 需求分析和设计

业务规则：

- 可以对状态为“启用”的员工账号进行“禁用”操作
- 可以对状态为“禁用”的员工账号进行“启用”操作
- 状态为“禁用”的员工账号不能登录系统

## 需求分析和设计

### 接口设计:

#### 基本信息

**Path:** /admin/employee/status/{status}

**Method:** POST

接口描述:

#### 请求参数

##### Headers

参数名称	参数值	是否必须	示例	备注
Content-Type	application/json	是		

##### 路径参数

参数名称	示例	备注
status	1	状态, 1为启用 0为禁用

##### Query

参数名称	是否必须	示例	备注
id	是		员工id

#### 返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	number	必须			
msg	string	非必须			
data	string	非必须			



## 启用禁用员工账号

- 需求分析和设计
- 代码开发
- 功能测试

## 代码开发

根据接口设计中的请求参数形式对应的在 EmployeeController 中创建启用禁用员工账号的方法：

```
@PostMapping("/status/{status}")
@ApiOperation("启用禁用员工账号")
public Result<String> startOrStop(@PathVariable Integer status, Long id){
    Log.info("启用禁用员工账号: {},{}",status,id);
    employeeService.startOrStop(status,id);
    return Result.success();
}
```

EmployeeController

## 代码开发

在 EmployeeService 接口中声明启用禁用员工账号的业务方法：

```
/**
 * 启用禁用员工账号
 * @param status
 * @param id
 */
void startOrStop(Integer status, Long id);
```

EmployeeService



## 代码开发

在 EmployeeServiceImpl 中实现启用禁用员工账号的业务方法：

```
/**
 * 启用禁用员工账号
 *
 * @param status
 * @param id
 */
public void startOrStop(Integer status, Long id) {
    Employee employee = Employee.builder()
        .id(id)
        .status(status)
        .build();
    employeeMapper.update(employee);
}
```

EmployeeServiceImpl

## 代码开发

在 EmployeeMapper 接口中声明 update 方法：

```
/**
 * 根据id修改员工信息
 * @param employee
 */
void update(Employee employee);
```

EmployeeMapper

## 代码开发

在 EmployeeMapper.xml 中编写SQL:

```
<update id="update">
  update employee
  <set>
    <if test="username != null">username = #{username},</if>
    <if test="name != null">name = #{name},</if>
    <if test="password != null">password = #{password},</if>
    <if test="phone != null">phone = #{phone},</if>
    <if test="sex != null">sex = #{sex},</if>
    <if test="idNumber != null">id_Number = #{idNumber},</if>
    <if test="updateTime != null">update_Time = #{updateTime},</if>
    <if test="updateUser != null">update_User = #{updateUser},</if>
    <if test="status != null">status = #{status},</if>
  </set>
  where id = #{id}
</update>
```

EmployeeMapper.xml

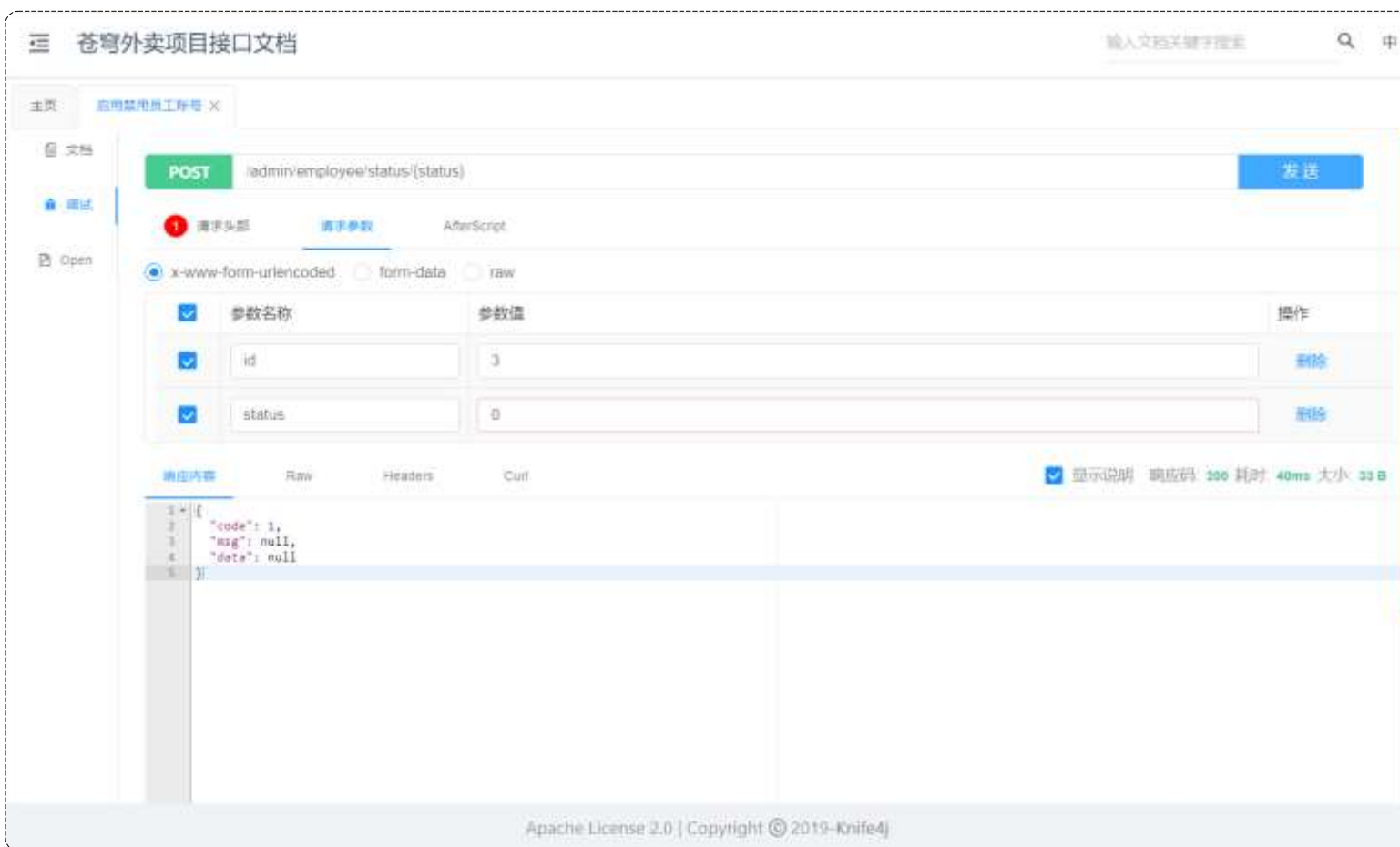


## 启用禁用员工账号

- 需求分析和设计
- 代码开发
- 功能测试

## 功能测试

可以通过接口文档进行测试，最后完成前后端联调测试即可





# 目录

Contents

- ◆ 新增员工
- ◆ 员工分页查询
- ◆ 启用禁用员工账号
- ◆ 编辑员工
- ◆ 导入分类模块功能代码



## 编辑员工

- 需求分析和设计
- 代码开发
- 功能测试

## 需求分析和设计

产品原型:

员工管理

1 员工姓名

请输入员工姓名

搜索

2 添加员工

员工姓名	账号	手机号	账号状态	最后操作时间	3 操作
管理员	13333333333	13333333333	禁用	2022-4-29 09:39	修改 启用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除



## 需求分析和设计

编辑员工功能涉及到两个接口：

- 根据id查询员工信息
- 编辑员工信息

### 基本信息

**Path:** /admin/employee

**Method:** PUT

接口描述：

### 请求参数

#### Headers

参数名称	参数值	是否必须	示例	备注
Content-Type	application/json	是		

#### Body

名称	类型	是否必须	默认值	备注	其他信息
id	integer	必须			format: int64
idNumber	string	必须			
name	string	必须			
phone	string	必须			
sex	string	必须			
username	string	必须			

### 基本信息

**Path:** /admin

**Method:** GET

接口描述：

### 请求参数

#### 路径参数

参数名称	示例
id	

是否必须	默认值	备注	其他信息
必须			format: int32
必须			
必须			format: date-time
必须			format: int64
必须			format: int64
必须			
必须			
必须			
必须			

### 返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object	非必须			
msg	string	非必须			



## 编辑员工

- 需求分析和设计
- 代码开发
- 功能测试

## 代码开发

在 EmployeeController 中创建 getById 方法:

```
/**
 * 根据id查询员工
 * @param id
 * @return
 */
@GetMapping("/{id}")
@ApiOperation("根据id查询员工")
public Result<Employee> getById(@PathVariable Long id){
    return Result.success(employeeService.getById(id));
}
```

EmployeeController

## 代码开发

在 EmployeeService 接口中声明 getById 方法：

```
/**  
 * 根据id查询员工  
 * @param id  
 * @return  
 */  
Employee getById(Long id);
```

EmployeeService

## 代码开发

在 EmployeeServiceImpl 中实现 getById 方法:

```
/**
 * 根据id查询员工
 * @param id
 * @return
 */
public Employee getById(Long id) {
    Employee employee = employeeMapper.getById(id);
    employee.setPassword("****");
    return employee;
}
```

EmployeeServiceImpl

## 代码开发

在 EmployeeMapper 接口中声明 getById 方法:

```
/**
 * 根据id查询员工
 * @param id
 * @return
 */
@Select("select * from employee where id = #{id}")
Employee getById(Long id);
```

EmployeeMapper

可以先通过接口测试确认数据回显是否有问题，如果没有问题再继续开发

## 代码开发

在 EmployeeController 中创建 update 方法:

```
/**
 * 编辑员工信息
 * @param employeeDTO
 * @return
 */
@PutMapping
@ApiOperation("编辑员工信息")
public Result<String> update(@RequestBody EmployeeDTO employeeDTO){
    Log.info("编辑员工: {}", employeeDTO);
    employeeService.update(employeeDTO);
    return Result.success();
}
```

EmployeeController

## 代码开发

在 EmployeeService 接口中声明 update 方法:

```
/**
 * 根据id修改员工信息
 * @param employeeDTO
 */
void update(EmployeeDTO employeeDTO);
```

EmployeeService



## 代码开发

在 EmployeeServiceImpl 中实现 update 方法:

```
/**
 * 修改员工
 * @param employeeDTO
 */
public void update(EmployeeDTO employeeDTO) {
    // update employee set ... where id = ?
    Employee employee = new Employee();
    BeanUtils.copyProperties(employeeDTO, employee);

    //设置修改人和修改时间
    employee.setUpdateUser(BaseContext.getCurrentId());
    employee.setUpdateTime(LocalDateTime.now());

    employeeMapper.update(employee);
}
```

EmployeeServiceImpl



## 编辑员工

- 需求分析和设计
- 代码开发
- 功能测试

## 功能测试

通过Swagger接口文档进行测试，通过后再前后端联调测试即可



# 目录

Contents

- ◆ 新增员工
- ◆ 员工分页查询
- ◆ 启用禁用员工账号
- ◆ 编辑员工
- ◆ 导入分类模块功能代码



## 导入分类模块功能代码

- 需求分析和设计
- 代码导入
- 功能测试

## 需求分析和设计

产品原型:

分类管理

1 分类名称

请输入分类名称

分类类型

请选择分类类型

搜索

新增菜品分类

新增套餐分类

1

分类名称	分类类型	排序	状态	操作时间	3 操作
荤菜	菜品分类	1	启用	2019-01-01 11:11	修改 删除 禁用
素菜	菜品分类	2	禁用	2019-01-01 11:11	修改 删除 启用
凉菜	菜品分类	3	启用	2019-01-01 11:11	修改 删除 禁用
周一套餐	套餐分类	4	禁用	2019-01-02 11:11	修改 删除 启用

## 需求分析和设计

业务规则:

- 分类名称必须是**唯一**的
- 分类按照类型可以分为**菜品分类**和**套餐分类**
- 新添加的分类状态默认为“**禁用**”

## 需求分析和设计

接口设计:

- 新增分类
- 分类分页查询
- 根据id删除分类
- 修改分类
- 启用禁用分类
- 根据类型查询分类



## 需求分析和设计

数据库设计(category表):

字段名	数据类型	说明	备注
id	bigint	主键	自增
name	varchar(32)	分类名称	唯一
type	int	分类类型	1菜品分类 2套餐分类
sort	int	排序字段	用于分类数据的排序
status	int	状态	1启用 0禁用
create_time	datetime	创建时间	
update_time	datetime	最后修改时间	
create_user	bigint	创建人id	
update_user	bigint	最后修改人id	










## 导入分类模块功能代码

- 需求分析和设计
- 代码导入
- 功能测试

## 代码导入

导入资料中的分类管理模块功能代码即可

-  CategoryController.java
-  CategoryMapper.java
-  CategoryMapper.xml
-  CategoryService.java
-  CategoryServiceImpl.java
-  DishMapper.java
-  SetmealMapper.java

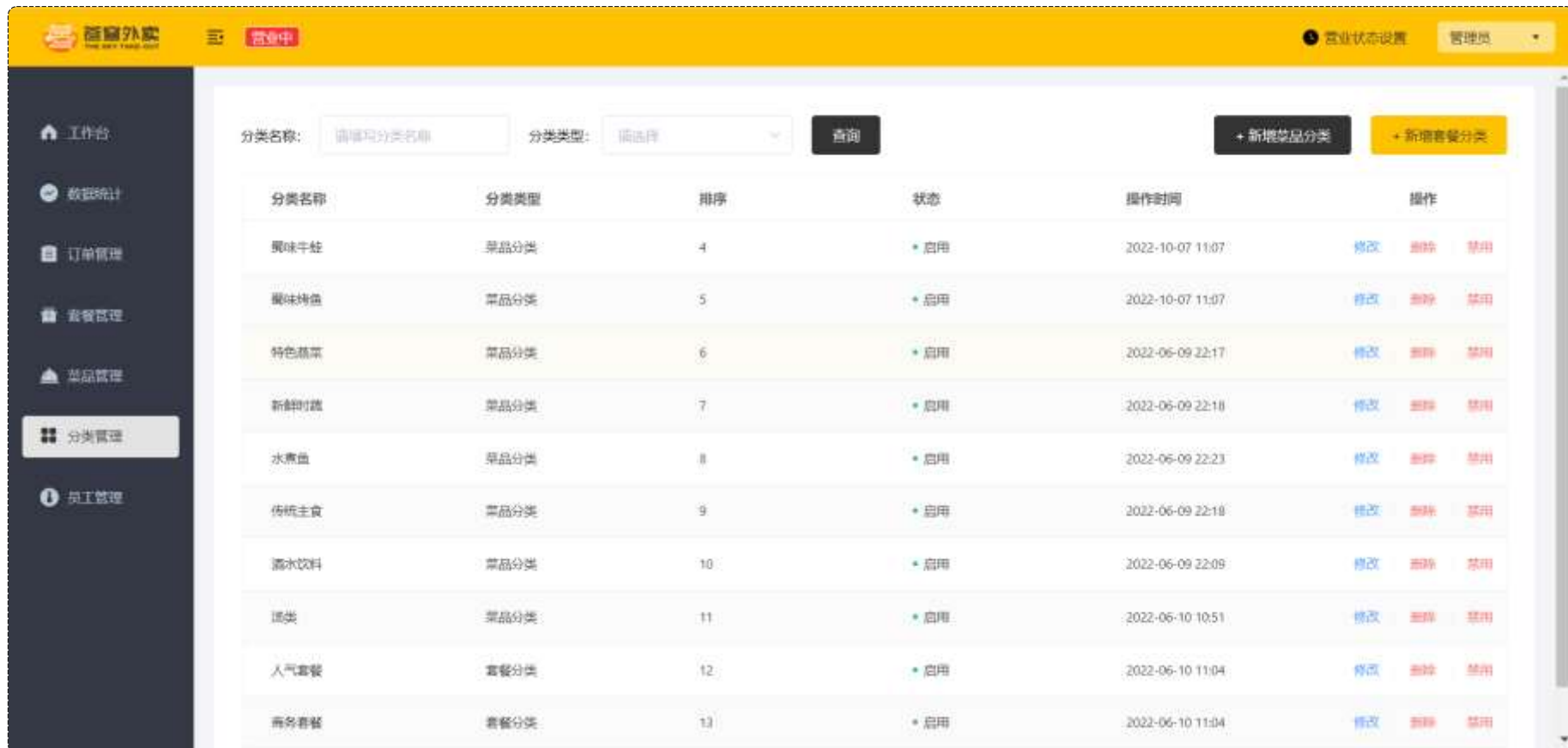


## 导入分类模块功能代码

- 需求分析和设计
- 代码导入
- 功能测试

## 功能测试

直接进行前后端联调测试即可



The screenshot displays a web application for restaurant management. The top navigation bar is yellow and contains the logo, a status indicator '营业中' (Open), and a user profile '管理员'. The left sidebar is dark blue with various management options. The main content area shows a table of food categories with search filters and action buttons.

分类名称	分类类型	排序	状态	操作时间	操作
蟹味牛蛙	菜品分类	4	• 启用	2022-10-07 11:07	<a href="#">修改</a> <a href="#">删除</a> <a href="#">禁用</a>
蟹味烤鱼	菜品分类	5	• 启用	2022-10-07 11:07	<a href="#">修改</a> <a href="#">删除</a> <a href="#">禁用</a>
特色蔬菜	菜品分类	6	• 启用	2022-06-09 22:17	<a href="#">修改</a> <a href="#">删除</a> <a href="#">禁用</a>
新鲜时蔬	菜品分类	7	• 启用	2022-06-09 22:18	<a href="#">修改</a> <a href="#">删除</a> <a href="#">禁用</a>
水煮鱼	菜品分类	8	• 启用	2022-06-09 22:23	<a href="#">修改</a> <a href="#">删除</a> <a href="#">禁用</a>
传统主食	菜品分类	9	• 启用	2022-06-09 22:18	<a href="#">修改</a> <a href="#">删除</a> <a href="#">禁用</a>
汤水饮料	菜品分类	10	• 启用	2022-06-09 22:09	<a href="#">修改</a> <a href="#">删除</a> <a href="#">禁用</a>
酒类	菜品分类	11	• 启用	2022-06-10 10:51	<a href="#">修改</a> <a href="#">删除</a> <a href="#">禁用</a>
人气套餐	套餐分类	12	• 启用	2022-06-10 11:04	<a href="#">修改</a> <a href="#">删除</a> <a href="#">禁用</a>
商务套餐	套餐分类	13	• 启用	2022-06-10 11:04	<a href="#">修改</a> <a href="#">删除</a> <a href="#">禁用</a>