

# 菜品管理



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

苍穹外卖  
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

菜品名称:








菜品分类:

售卖状态:

查询

批量删除

+ 新建菜品

<input type="checkbox"/>	菜品名称	图片	菜品分类	售价	售卖状态	最后操作时间	操作
<input type="checkbox"/>	测试菜品		蜀味烤鱼	¥ 58	• 停售	2022-09-20 18:30	<a href="#">修改</a> <a href="#">删除</a> <a href="#">启售</a>
<input type="checkbox"/>	平菇豆腐汤		汤类	¥ 6	• 启售	2022-06-10 10:55	<a href="#">修改</a> <a href="#">删除</a> <a href="#">停售</a>
<input type="checkbox"/>	鸡蛋汤		汤类	¥ 4	• 启售	2022-06-10 10:54	<a href="#">修改</a> <a href="#">删除</a> <a href="#">停售</a>
<input type="checkbox"/>	鲈鱼2斤		蜀味烤鱼	¥ 72	• 启售	2022-06-10 10:43	<a href="#">修改</a> <a href="#">删除</a> <a href="#">停售</a>
<input type="checkbox"/>	江团鱼2斤		蜀味烤鱼	¥ 119	• 启售	2022-06-10 10:42	<a href="#">修改</a> <a href="#">删除</a> <a href="#">停售</a>
<input type="checkbox"/>	草鱼2斤		蜀味烤鱼	¥ 68	• 启售	2022-06-10 10:41	<a href="#">修改</a> <a href="#">删除</a> <a href="#">停售</a>
<input type="checkbox"/>	馋嘴牛蛙		蜀味牛蛙	¥ 88	• 启售	2022-06-10 10:37	<a href="#">修改</a> <a href="#">删除</a> <a href="#">停售</a>
<input type="checkbox"/>	香锅牛蛙		蜀味牛蛙	¥ 88	• 启售	2022-06-10 10:35	<a href="#">修改</a> <a href="#">删除</a> <a href="#">停售</a>



# 目录

Contents

- ◆ 公共字段自动填充
- ◆ 新增菜品
- ◆ 菜品分页查询
- ◆ 删除菜品
- ◆ 修改菜品



# 公共字段自动填充

- 问题分析
- 实现思路
- 代码开发
- 功能测试

## 问题分析

业务表中的公共字段：

序号	字段名	含义	数据类型
1	create_time	创建时间	datetime
2	create_user	创建人id	bigint
3	update_time	修改时间	datetime
4	update_user	修改人id	bigint

```
//设置当前记录的创建时间、修改时间、创建人、修改人  
employee.setCreateTime(LocalDateTime.now());  
employee.setUpdateTime(LocalDateTime.now());  
employee.setCreateUser(BaseContext.getCurrentId());  
employee.setUpdateUser(BaseContext.getCurrentId());
```

```
//设置创建时间、修改时间、创建人、修改人  
category.setCreateTime(LocalDateTime.now());  
category.setUpdateTime(LocalDateTime.now());  
category.setCreateUser(BaseContext.getCurrentId());  
category.setUpdateUser(BaseContext.getCurrentId());
```

问题：代码冗余、不便于后期维护



# 公共字段自动填充

- 问题分析
- 实现思路
- 代码开发
- 功能测试

## 实现思路

序号	字段名	含义	数据类型	操作类型
1	create_time	创建时间	datetime	insert
2	create_user	创建人id	bigint	
3	update_time	修改时间	datetime	insert、 update
4	update_user	修改人id	bigint	

- 自定义注解 AutoFill，用于标识需要进行公共字段自动填充的方法
- 自定义切面类 AutoFillAspect，统一拦截加入了 AutoFill 注解的方法，通过反射为公共字段赋值
- 在 Mapper 的方法上加入 AutoFill 注解

技术点：枚举、注解、AOP、反射



# 公共字段自动填充

- 问题分析
- 实现思路
- 代码开发
- 功能测试



## 代码开发

- 自定义注解 AutoFill

```
/**
 * 自动填充
 */
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface AutoFill {

    /**
     * 数据库操作类型
     * @return
     */
    OperationType value();
}
```

## 代码开发

- 自定义切面 AutoFillAspect

```
/**
 * 自定义切面类，统一为公共字段赋值
 */
@Aspect
@Component
@Slf4j
public class AutoFillAspect {
    /**
     * 切入点
     */
    @Pointcut("execution(* com.sky.mapper.*(..)) && @annotation(com.sky.annotation.AutoFill)")
    public void autoFillPointCut() {}

    /**
     * 通知 自动填充公共字段
     * @param joinPoint
     */
    @Before("autoFillPointCut()")
    public void autoFill(JoinPoint joinPoint) {
        log.info("公共字段自动填充...");
    }
}
```

## 代码开发

- 完善自定义切面 AutoFillAspect 的 autoFill 方法

```
log.info("公共字段自动填充...");

//获得方法签名对象
MethodSignature signature = (MethodSignature) joinPoint.getSignature();
//获得方法上的注解
AutoFill autoFill = signature.getMethod().getAnnotation(AutoFill.class);
//获得注解中的操作类型
OperationType operationType = autoFill.value();

//获取当前目标方法的参数
Object[] args = joinPoint.getArgs();

if (args == null || args.length == 0) {
    return;
}

//实体对象
Object entity = args[0];

//准备赋值的数据
LocalDateTime time = LocalDateTime.now();
Long empld = BaseContext.getCurrentId();
```

## 代码开发

- 完善自定义切面 AutoFillAspect 的 autoFill 方法

```
if (operationType == OperationType.INSERT) {  
    //当前执行的是insert操作，为4个字段赋值  
    try {  
        //获得set方法对象----Method  
        Method setCreateTime = entity.getClass().getDeclaredMethod(AutoFillConstant.SET_CREATE_TIME, LocalDateTime.class);  
        Method setUpdateTime = entity.getClass().getDeclaredMethod(AutoFillConstant.SET_UPDATE_TIME, LocalDateTime.class);  
        Method setCreateUser = entity.getClass().getDeclaredMethod(AutoFillConstant.SET_CREATE_USER, Long.class);  
        Method setUpdateUser = entity.getClass().getDeclaredMethod(AutoFillConstant.SET_UPDATE_USER, Long.class);  
        //通过反射调用目标对象的方法  
        setCreateTime.invoke(entity, time);  
        setUpdateTime.invoke(entity, time);  
        setCreateUser.invoke(entity, empId);  
        setUpdateUser.invoke(entity, empId);  
    } catch (Exception ex) {  
        log.error("公共字段自动填充失败: {}", ex.getMessage());  
    }  
}
```

## 代码开发

- 完善自定义切面 AutoFillAspect 的 autoFill 方法

```
else {  
    //当前执行的是update操作，为2个字段赋值  
    try {  
        //获得set方法对象----Method  
        Method setUpdateTime = entity.getClass().getDeclaredMethod(AutoFillConstant.SET_UPDATE_TIME, LocalDateTime.class);  
        Method setUpdateUser = entity.getClass().getDeclaredMethod(AutoFillConstant.SET_UPDATE_USER, Long.class);  
        //通过反射调用目标对象的方法  
        setUpdateTime.invoke(entity, time);  
        setUpdateUser.invoke(entity, empId);  
    } catch (Exception ex) {  
        log.error("公共字段自动填充失败: {}", ex.getMessage());  
    }  
}
```

## 代码开发

- 在Mapper接口的方法上加入 AutoFill 注解

```
/**
 * 插入数据
 * @param category
 */
@AutoFill(OperationType.INSERT)
@Insert("insert into category(type, name, sort, status, create_time, update_time, create_user, update_user)" +
        " VALUES" +
        " (#{type}, #{name}, #{sort}, #{status}, #{createTime}, #{updateTime}, #{createUser}, #{updateUser})")
void insert(Category category);

/**
 * 根据id修改分类
 * @param category
 */
@AutoFill(OperationType.UPDATE)
void update(Category category);
```

## 代码开发

- 将业务层为公共字段赋值的代码注释掉



# 公共字段自动填充

- 问题分析
- 实现思路
- 代码开发
- 功能测试



## 功能测试

通过观察控制台输出的SQL来确定公共字段填充是否完成

开始进行公共字段自动填充...

```
==> Preparing: update employee SET name = ?, username = ?, phone = ?, sex = ?, id_Number = ?, update_Time = ?, update_User = ? where id = ?
```

```
==> Parameters: 王五(String), wangwu(String), 13312345678(String), 1(String), 222333444555666111(String), 2022-10-10T15:27:28.372093900(LocalDateTime), 1(Long), 14(Long)
```

```
<== Updates: 1
```



# 目录

Contents

- ◆ 公共字段自动填充
- ◆ 新增菜品
- ◆ 菜品分页查询
- ◆ 删除菜品
- ◆ 修改菜品



## 新增菜品

- 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：

新建菜品

\*菜品名称:

\*菜品分类:

\*菜品价格:

元

口味做法配置:

口味名 (3个字内)

口味标签 (输入后, 回车添加)

甜度

半糖 X

半糖 X

半糖 X

半糖 X

删除

删除

添加口味

\*菜品图  
片:

图片大小不超过2M  
仅能上传PNG JPG JPEG类型图片  
建议上传200\*200或300\*300尺寸的图片

菜品描述:

菜品描述, 最长200字

保存

保存并继续添加菜品

返回

## 需求分析和设计

业务规则：

- 菜品名称必须是唯一的
- 菜品必须属于某个分类下，不能单独存在
- 新增菜品时可以根据情况选择菜品的口味
- 每个菜品必须对应一张图片

## 需求分析和设计

接口设计:

- 根据类型查询分类 (已完成)
- 文件上传
- 新增菜品

新建菜品

\*菜品名称:

\*菜品分类:

\*菜品价格:

元

口味做法配置:

口味名 (3个字内)

口味标签 (输入后, 回车添加)

甜度

半糖 X

半糖 X

半糖 X

半糖 X

删除

删除

添加口味

\*菜品图  
片:

图片大小不超过2M  
仅能上传PNG JPG JPEG类型图片  
建议上传200\*200或300\*300尺寸的图片

菜品描述:

菜品描述, 最长200字

保存

保存并继续添加菜品

返回

## 需求分析和设计

接口设计：

- 根据类型查询分类

### 基本信息

**Path:** /admin/category/list

**Method:** GET

**接口描述:**

### 请求参数

**Query**

参数名称	是否必须	示例	备注
type	否	2	分类类型：1为菜品分类，2为套餐分类

### 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			item 类型: object
└─ createTime	string	非必须			format: date-time
└─ createUser	integer	非必须			format: int64
└─ id	integer	非必须			format: int64
└─ name	string	非必须			
└─ sort	integer	非必须			format: int32
└─ type	integer	非必须			format: int32
└─ updateTime	string	非必须			format: date-time
└─ updateUser	integer	非必须			format: int64
msg	string	非必须			

## 需求分析和设计

接口设计：

- 文件上传

### 基本信息

**Path:** /admin/common/upload

**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	multipart/form-data	是		

#### Body

参数名称	参数类型	是否必须	示例	备注
file	file	是		文件

### 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	string	必须		文件上传路径	
msg	string	非必须			



## 需求分析和设计

接口设计：

- 新增菜品

### 基本信息

**Path:** /admin/dish

**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
categoryId	integer	必须		分类id	format: int64
description	string	非必须		菜品描述	
flavors	object []	非必须		口味	item 类型: object
└─ dishId	integer	非必须		菜品id	format: int64
└─ id	integer	非必须		口味id	format: int64
└─ name	string	必须		口味名称	
└─ value	string	必须		口味值	
id	integer	非必须		菜品id	format: int64
image	string	必须		菜品图片路径	
name	string	必须		菜品名称	
price	number	必须		菜品价格	
status	integer	非必须		菜品状态: 1为起售, 0为停售	format: int32

### 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			

## 需求分析和设计

数据库设计（dish菜品表和dish\_flavor口味表）：

字段名	数据类型	说明	备注
id	bigint	主键	自增
name	varchar(32)	菜品名称	唯一
category_id	bigint	分类id	逻辑外键
price	decimal(10,2)	菜品价格	
image	varchar(255)	图片路径	
description	varchar(255)	菜品描述	
status	int	售卖状态	1起售 0停售
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	

字段名	数据类型	说明	备注
id	bigint	主键	自增
dish_id	bigint	菜品id	逻辑外键
name	varchar(32)	口味名称	
value	varchar(255)	口味值	



## 新增菜品

- 需求分析和设计
- 代码开发
- 功能测试

## 代码开发

开发文件上传接口:



## 代码开发

开发文件上传接口：

```
sky:  
  alioss:  
    endpoint: oss-cn-beijing.aliyuncs.com  
    access-key-id: LTAI5tPeFLzsPPT8gG3LPW64  
    access-key-secret: U6k1brOZ8gaOIXv3nXbulGTUzy6Pd7  
    bucket-name: sky-itcast
```

application-dev.yml

```
sky:  
  alioss:  
    endpoint: ${sky.alioss.endpoint}  
    access-key-id: ${sky.alioss.access-key-id}  
    access-key-secret: ${sky.alioss.access-key-secret}  
    bucket-name: ${sky.alioss.bucket-name}
```

application.yml

## 代码开发

开发文件上传接口：

```
@Configuration
@Slf4j
public class OssConfiguration {

    /**
     * 通过spring管理对象
     *
     * @param aliOssProperties
     * @return
     */
    @Bean
    @ConditionalOnMissingBean
    public AliOssUtil aliOssUtil(AliOssProperties aliOssProperties) {
        log.info("开始创建阿里云OSS工具类...");
        return new AliOssUtil(aliOssProperties.getEndpoint(),
            aliOssProperties.getAccessKeyId(),
            aliOssProperties.getAccessKeySecret(),
            aliOssProperties.getBucketName());
    }
}
```

OssConfiguration

## 代码开发

开发文件上传接口:

```
@RestController
@RequestMapping("/admin/common")
@Slf4j
@Api(tags = "通用接口")
public class CommonController {
    @Autowired
    private AliOssUtil aliOssUtil;

    /**
     * 文件上传
     * @param file
     * @return
     */
    @PostMapping("/upload")
    @ApiOperation("文件上传")
    public Result<String> upload(MultipartFile file){
        log.info(file.getName());

        //原始文件名
        String originalFilename = file.getOriginalFilename();
        String extension = originalFilename.substring(originalFilename.lastIndexOf("."));

        //将文件上传的阿里云
        String fileName = UUID.randomUUID().toString() + extension;
        try {
            String filePath = aliOssUtil.upload(file.getBytes(), fileName);
            return Result.success(filePath);
        } catch (IOException e) {
            log.error("文件上传失败:{}", e.getMessage());
        }

        return Result.error(MessageConstant.UPLOAD_FAILED);
    }
}
```

OssConfiguration

## 代码开发

开发新增菜品接口：

Body

名称	类型	是否必须	默认值	备注
categoryId	integer	必须		分类id
description	string	非必须		菜品描述
flavors	object []	非必须		口味
└─ dishId	integer	非必须		菜品id
└─ id	integer	非必须		口味id
└─ name	string	必须		口味名称
└─ value	string	必须		口味值
id	integer	非必须		菜品id
image	string	必须		菜品图片路径
name	string	必须		菜品名称
price	number	必须		菜品价格
status	integer	非必须		菜品状态：1为起售，0为停售



```
@Data
public class DishDTO implements Serializable {

    private Long id;
    //菜品名称
    private String name;
    //菜品分类id
    private Long categoryId;
    //菜品价格
    private BigDecimal price;
    //图片
    private String image;
    //描述信息
    private String description;
    //0 停售 1 起售
    private Integer status;
    //口味
    private List<DishFlavor> flavors = new ArrayList<>();

}
```



## 代码开发

开发新增菜品接口：

```
@RestController
@RequestMapping("/admin/dish")
@Api(tags = "菜品相关接口")
@Slf4j
public class DishController {

    @Autowired
    private DishService dishService;

    /**
     * 新增菜品
     * @param dishDTO
     * @return
     */
    @PostMapping
    @ApiOperation("新增菜品")
    public Result<String> save(@RequestBody DishDTO dishDTO){
        log.info("新增菜品: {}", dishDTO);
        dishService.saveWithFlavor(dishDTO);

        return Result.success();
    }
}
```

DishController

## 代码开发

开发新增菜品接口：

```
public interface DishService {  
    /**  
     * 新增菜品  
     * @param dishDTO  
     */  
    void saveWithFlavor(DishDTO dishDTO);  
}
```

DishService

## 代码开发

开发新增菜品接口：

```
@Service
public class DishServiceImpl implements DishService {

    @Autowired
    private DishMapper dishMapper;
    @Autowired
    private DishFlavorMapper dishFlavorMapper;

    /**
     * 新增菜品
     * @param dishDTO
     */
    @Transactional
    public void saveWithFlavor(DishDTO dishDTO) {
        Dish dish = new Dish();
        BeanUtils.copyProperties(dishDTO, dish);
        //向菜品表dish插入1条数据
        dishMapper.insert(dish);

        //获取菜品的主键值
        Long dishId = dish.getId();

        List<DishFlavor> flavors = dishDTO.getFlavors();
        if(flavors != null && flavors.size() > 0){
            //向口味表dish_flavor插入n条
            flavors.forEach(dishFlavor -> {
                dishFlavor.setDishId(dishId);
            });
            //批量插入
            dishFlavorMapper.insertBatch(flavors);
        }
    }
}
```

DishServiceImpl

## 代码开发

开发新增菜品接口：

```
/**
 * 插入菜品数据
 * @param dish
 */
@AutoFill(OperationType.INSERT)
void insert(Dish dish);
```

DishMapper

```
<!--
    useGeneratedKeys:true 表示获取主键值
    keyProperty="id" 表示将主键值赋给id属性
-->
<insert id="insert" useGeneratedKeys="true" keyProperty="id">
    insert into dish
    (status, name, category_id, price, image, description, create_time, update_time, create_user, update_user)
    values
    (#{status}, #{name}, #{categoryId}, #{price}, #{image}, #{description}, #{createTime}, #{updateTime}, #{createUser}, #{updateUser})
</insert>
```

DishMapper.xml

## 代码开发

开发新增菜品接口：

```
@Mapper
public interface DishFlavorMapper {

    /**
     * 批量插入口味数据
     * @param flavors
     */
    void insertBatch(List<DishFlavor> flavors);
}
```

DishFlavorMapper

```
<insert id="insertBatch">
    insert into dish_flavor(dish_id, name, value) values
    <foreach collection="flavors" item="dishFlavor" separator=",">
        (#{dishFlavor.dishId},#{dishFlavor.name},#{dishFlavor.value})
    </foreach>
</insert>
```

DishFlavorMapper.xml



## 新增菜品

- 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，也可以进行前后端联调测试



# 目录

Contents

- ◆ 公共字段自动填充
- ◆ 新增菜品
- ◆ 菜品分页查询
- ◆ 删除菜品
- ◆ 修改菜品





## 菜品分页查询

- 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：

1 菜品名称 请输入菜品名称 菜品分类 请输入菜品分类 售卖状态 请选择 搜索

批量删除 2 新建菜品

<input type="checkbox"/>	菜品名称	图片	菜品分类	售价	售卖状态	最后操作时间	3 操作
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	停售	2019-02-02 11:11	修改 删除 启售 4
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售

总共 85 个项目 < 1 2 3 4 5 >

## 需求分析和设计

业务规则：

- 根据页码展示菜品信息
- 每页展示10条数据
- 分页查询时可以根据需要输入菜品名称、菜品分类、菜品状态进行查询

## 需求分析和设计

接口设计：

### 基本信息

**Path:** /admin/dish/page

**Method:** GET

**接口描述:**

### 请求参数

#### Query

参数名称	是否必须	示例	备注
page	是	1	页码
pageSize	是	10	每页记录数
name	否	鱼香肉丝	菜品名称
categoryId	否	1	分类id
status	否	1	菜品售卖状态

### 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	必须			
└─ records	object []	非必须			item 类型: object
└─ id	number	必须			
└─ name	string	必须			
└─ price	number	必须			
└─ image	string	必须			
└─ description	string	必须			
└─ status	integer	必须			
└─ updateTime	string	必须			
└─ categoryName	string	必须		分类名称	
└─ total	integer	必须			format: int64
msg	string	非必须			



## 菜品分页查询

- 需求分析和设计
- 代码开发
- 功能测试

## 代码开发

根据菜品分页查询接口定义设计对应的DTO:

### 请求参数

#### Query

参数名称	是否必须	示例	备注
page	是	1	页码
pageSize	是	10	每页记录数
name	否	鱼香肉丝	菜品名称
categoryId	否	1	分类id
status	否	1	菜品售卖状态



```
@Data
public class DishPageQueryDTO implements Serializable {

    private int page;

    private int pageSize;

    //菜品名称
    private String name;

    //分类id
    private Integer categoryId;

    //状态 0表示禁用 1表示启用
    private Integer status;

}
```

## 代码开发

根据菜品分页查询接口定义设计对应的VO:

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	必须			
├ records	object []	非必须			item 类型: object
├ id	number	必须			
├ name	string	必须			
├ price	number	必须			
├ image	string	必须			
├ description	string	必须			
├ status	integer	必须			
├ updateTime	string	必须			
├ categoryName	string	必须		分类名称	
├ total	integer	必须			format: int64
msg	string	非必须			



```
public class DishVO implements Serializable {  
  
    private Long id;  
    //菜品名称  
    private String name;  
    //菜品分类id  
    private Long categoryId;  
    //菜品价格  
    private BigDecimal price;  
    //图片  
    private String image;  
    //描述信息  
    private String description;  
    //0 停售 1 起售  
    private Integer status;  
    //更新时间  
    private LocalDateTime updateTime;  
    //分类名称  
    private String categoryName;  
    //菜品关联的口味  
    private List<DishFlavor> flavors = new ArrayList<>();  
  
}
```

## 代码开发

根据接口定义创建DishController的page分页查询方法：

```
/**
 * 菜品分页查询
 * @param dishPageQueryDTO
 * @return
 */
@GetMapping("/page")
@ApiOperation("菜品分页查询")
public Result<PageResult> page(DishPageQueryDTO dishPageQueryDTO){
    log.info("菜品分页查询: {}", dishPageQueryDTO);
    PageResult pageResult = dishService.pageQuery(dishPageQueryDTO);
    return Result.success(pageResult);
}
```

DishController



## 代码开发

在 DishService 中扩展分页查询方法：

```
/**  
 * 菜品分页查询  
 * @param dishPageQueryDTO  
 * @return  
 */  
PageResult pageQuery(DishPageQueryDTO dishPageQueryDTO);
```

DishService

## 代码开发

在 DishServiceImpl 中实现分页查询方法：

```
/**
 * 菜品分页查询
 * @param dishPageQueryDTO
 * @return
 */
public PageResult pageQuery(DishPageQueryDTO dishPageQueryDTO) {
    PageHelper.startPage(dishPageQueryDTO.getPage(), dishPageQueryDTO.getPageSize());
    Page<DishVO> page = dishMapper.pageQuery(dishPageQueryDTO);
    return new PageResult(page.getTotal(), page.getResult());
}
```

DishServiceImpl

## 代码开发

在 DishMapper 接口中声明 pageQuery 方法：

```
/**  
 * 菜品分页查询  
 * @param dishPageQueryDTO  
 * @return  
 */  
Page<DishVO> pageQuery(DishPageQueryDTO dishPageQueryDTO);
```

DishMapper

## 代码开发

在 DishMapper.xml 中编写SQL：

```
<select id="pageQuery" resultType="com.sky.vo.DishVO">
  select d.*,c.name categoryName from dish d left join category c on d.category_id = c.id
  <where>
    <if test="name != null">
      and d.name like concat('%',{name},%')
    </if>
    <if test="categoryId != null">
      and d.category_id = #{categoryId}
    </if>
    <if test="status != null">
      and d.status = #{status}
    </if>
  </where>
  order by d.create_time desc
</select>
```

DishMapper.xml



## 菜品分页查询

- 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

- ◆ 公共字段自动填充
- ◆ 新增菜品
- ◆ 菜品分页查询
- ◆ 删除菜品
- ◆ 修改菜品



## 删除菜品

- 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：

菜品名称

请输入菜品名称

菜品分类

请输入菜品分类

售卖状态

请选择

搜索

批量删除

2 新建菜品

<input type="checkbox"/>	菜品名称	图片	菜品分类	售价	售卖状态	最后操作时间	3 操作
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	停售	2019-02-02 11:11	修改 删除 启售 4
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售

## 需求分析和设计

业务规则：

- 可以一次删除一个菜品，也可以批量删除菜品
- 起售中的菜品不能删除
- 被套餐关联的菜品不能删除
- 删除菜品后，关联的口味数据也需要删除掉

## 需求分析和设计

接口设计：

### 基本信息

**Path:** /admin/dish

**Method:** DELETE

接口描述：

### 请求参数

Query

参数名称	是否必须	示例	备注
ids	是	1,2,3	菜品id，之间用逗号分隔

### 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			
data	string	非必须			
msg	string	非必须			

## 需求分析和设计

数据库设计:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	
⚡ 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"/>	dish表
🔹 name	VARCHAR(64)	<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"/>	
🔹 category_id	BIGINT	<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"/>	
🔹 price	DECIMAL(10,2)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
🔹 image	VARCHAR(200)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
🔹 description	VARCHAR(400)	<input type="checkbox"/>	<input 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 type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
🔹 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"/>	
🔹 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"/>	
🔹 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"/>	
🔹 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"/>	

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	
⚡ 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"/>	dish_flavor表
🔹 dish_id	BIGINT	<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"/>	
🔹 name	VARCHAR(64)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
🔹 value	VARCHAR(128)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	
⚡ 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"/>	
🔹 setmeal_id	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"/>	
🔹 dish_id	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"/>	setmeal_dish表
🔹 name	VARCHAR(32)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
🔹 price	DECIMAL(10,2)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
🔹 copies	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	



## 删除菜品

- 需求分析和设计
- 代码开发
- 功能测试

## 代码开发

根据删除菜品的接口定义在DishController中创建方法：

```
/**
 * 菜品批量删除
 * @param ids
 * @return
 */
@DeleteMapping
@ApiOperation("菜品批量删除")
public Result delete(@RequestParam List<Long> ids){
    log.info("菜品批量删除: {}", ids);
    dishService.deleteBatch(ids);
    return Result.success();
}
```

DishController

## 代码开发

在DishService接口中声明deleteBatch方法：

```
/**  
 * 菜品批量删除  
 * @param ids  
 */  
void deleteBatch(List<Long> ids);
```

DishService

## 代码开发

在DishServiceImpl中实现deleteBatch方法：

```
@Transactional
public void deleteBatch(List<Long> ids) {
    ids.forEach(id->{
        Dish dish = dishMapper.getById(id);
        //判断当前要删除的菜品状态是否为起售中
        if(dish.getStatus() == StatusConstant.ENABLE){
            //如果是起售中，抛出业务异常
            throw new DeletionNotAllowedException(MessageConstant.DISH_ON_SALE);
        }
    });

    //判断当前要删除的菜品是否被套餐关联了
    List<Long> setmealIds = setmealDishMapper.getSetmealIdsByDishIds(ids);
    if(setmealIds != null && setmealIds.size() > 0){
        //如果关联了，抛出业务异常
        throw new DeletionNotAllowedException(MessageConstant.DISH_BE_RELATED_BY_SETMEAL);
    }

    //删除菜品表中的数据
    ids.forEach(id -> {
        dishMapper.deleteById(id);
        //删除口味表中的数据
        dishFlavorMapper.deleteByDishId(id);
    });
}
```

DishServiceImpl



## 代码开发

在DishMapper中声明getById方法，并配置SQL：

```
/**  
 * 根据主键查询菜品数据  
 * @param id  
 * @return  
 */  
@Select("select * from dish where id = #{id}")  
Dish getById(Long id);
```

DishMapper

## 代码开发

创建SetmealDishMapper，声明getSetmealIdsByDishIds方法，并在xml文件中编写SQL：

```
@Mapper
public interface SetmealDishMapper {

    /**
     * 根据菜品id查询关联的套餐id
     * @param ids
     * @return
     */
    List<Long> getSetmealIdsByDishIds(List<Long> ids);
}
```

SetmealDishMapper

```
<select id="getSetmealIdsByDishIds" resultType="java.lang.Long">
    select setmeal_id from setmeal_dish where dish_id in
    <foreach collection="ids" separator="," open="(" close=")" item="dishId">
        #{dishId}
    </foreach>
</select>
```

SetmealDishMapper.xml

## 代码开发

在DishMapper.xml中声明deleteById方法并配置SQL：

```
/**
 * 根据主键删除菜品数据
 * @param id
 */
@Delete("delete from dish where id = #{id}")
void deleteById(Long id);
```

DishMapper

## 代码开发

在DishFlavorMapper中声明deleteByDishId方法并配置SQL：

```
/**
 * 根据菜品id删除口味数据
 * @param dishId
 */
@Delete("delete from dish_flavor where dish_id = #{dishId}")
void deleteByDishId(Long dishId);
```

DishFlavorMapper



## 删除菜品

- 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

- ◆ 公共字段自动填充
- ◆ 新增菜品
- ◆ 菜品分页查询
- ◆ 删除菜品
- ◆ 修改菜品



## 修改菜品

- 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:

\*菜品名称:

\*菜品分类:

\*菜品价格:

元

口味做法配置:

口味名 (3个字内)

口味标签 (输入后, 回车添加)

甜度

半糖 X 半糖 X 半糖 X 半糖 X

删除

删除

添加口味

1 \*菜品图片:

图片大小不超过2M  
仅能上传PNG JPG JPEG类型图片  
建议上传200\*200或300\*300尺寸的图片

菜品描述:

菜品描述, 最长200字

保存

保存并继续添加菜品

返回

## 需求分析和设计

接口设计:

- 根据id查询菜品
- 根据类型查询分类 (已实现)
- 文件上传 (已实现)
- 修改菜品

\*菜品名称:

\*菜品分类:

\*菜品价格:

元

口味做法配置:

口味名 (3个字内)

口味标签 (输入后, 回车添加)

甜度

半糖 X

半糖 X

半糖 X

半糖 X

删除

删除

添加口味

1

\*菜品图  
片:

图片大小不超过2M  
仅能上传PNG JPG JPEG类型图片  
建议上传200\*200或300\*300尺寸的图片

菜品描述:

菜品描述, 最长200字

保存

保存并继续添加菜品

返回

## 需求分析和设计

接口设计：

- 根据id查询菜品

### 基本信息

**Path:** /admin/dish/{id}

**Method:** GET

**接口描述:**

### 请求参数

#### 路径参数

参数名称	示例	备注
id	101	菜品id

### 返回数据

名称	类型	是否必须	默认值	备注	其他信息
data	object	非必须			
└─ categoryId	integer	非必须			format: int64
└─ description	string	非必须			
└─ flavors	object []	非必须			item 类型: object
└─ dishId	integer	非必须			format: int64
└─ id	integer	非必须			format: int64
└─ name	string	非必须			
└─ value	string	非必须			
└─ id	integer	非必须			format: int64
└─ image	string	非必须			
└─ name	string	非必须			
└─ price	number	非必须			
msg	string	非必须			
code	number	必须			

## 需求分析和设计

接口设计：

- 修改菜品

### 基本信息

Path: /admin/dish

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
categoryId	integer	必须			format: int64
description	string	非必须			
flavors	object []	非必须			item 类型: object
└─ dishId	integer	必须			format: int64
└─ id	integer	必须			format: int64
└─ name	string	必须			
└─ value	string	必须			
id	integer	必须			format: int64
image	string	必须			
name	string	必须			
price	number	必须			
status	integer	必须			format: int32

### 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	string	非必须			
msg	string	非必须			



## 修改菜品

- 需求分析和设计
- 代码开发
- 功能测试

## 代码开发

根据id查询菜品 接口开发：

```
/**
 * 根据id查询菜品和关联的口味数据
 * @param id
 * @return
 */
@GetMapping("/{id}")
@ApiOperation("根据id查询菜品和关联的口味数据")
public Result<DishVO> getByld(@PathVariable Long id){
    return Result.success(dishService.getByldWithFlavor(id));
}
```

DishController

## 代码开发

根据id查询菜品 接口开发：

```
/**  
 * 根据id查询菜品和关联的口味数据  
 * @param id  
 * @return  
 */  
DishVO getByIdWithFlavor(Long id);
```

DishService

## 代码开发

根据id查询菜品 接口开发:

```
/**
 * 根据id查询菜品和关联的口味数据
 * @param id
 * @return
 */
public DishVO getByIdWithFlavor(Long id) {
    //查询菜品表
    Dish dish = dishMapper.getById(id);

    //查询关联的口味
    List<DishFlavor> dishFlavorList = dishFlavorMapper.getByDishId(id);

    //封装成VO
    DishVO dishVO = new DishVO();
    BeanUtils.copyProperties(dish, dishVO);
    dishVO.setFlavors(dishFlavorList);

    return dishVO;
}
```

DishServiceImpl



## 代码开发

根据id查询菜品 接口开发：

```
/**
 * 根据菜品id查询对应的口味
 * @param dishId
 * @return
 */
@Select("select * from dish_flavor where dish_id = #{dishId}")
List<DishFlavor> getByDishId(Long dishId);
```

DishFlavorMapper

## 代码开发

修改菜品 接口开发：

```
/**
 * 修改菜品
 * @param dishDTO
 * @return
 */
@PutMapping
@ApiOperation("修改菜品")
public Result update(@RequestBody DishDTO dishDTO){
    log.info("修改菜品: {}", dishDTO);
    dishService.updateWithFlavor(dishDTO);
    return Result.success();
}
```

DishController

## 代码开发

修改菜品 接口开发:

```
/**  
 * 根据id修改菜品和关联的口味  
 * @param dishDTO  
 */  
void updateWithFlavor(DishDTO dishDTO);
```

DishService

## 代码开发

修改菜品 接口开发：

```
/**
 * 根据id修改菜品和关联的口味
 * @param dishDTO
 */
@Transactional
public void updateWithFlavor(DishDTO dishDTO) {
    Dish dish = new Dish();
    BeanUtils.copyProperties(dishDTO, dish);

    //修改菜品表dish, 执行update操作
    dishMapper.update(dish);

    //删除当前菜品关联的口味数据, 操作dish_flavor, 执行delete操作
    dishFlavorMapper.deleteByDishId(dishDTO.getId());

    //插入最新的口味数据, 操作dish_flavor, 执行insert操作
    List<DishFlavor> flavors = dishDTO.getFlavors();
    if(flavors != null && flavors.size() > 0){
        flavors.forEach(dishFlavor -> {
            dishFlavor.setDishId(dishDTO.getId());
        });
        dishFlavorMapper.insertBatch(flavors);
    }
}
```

DishServiceImpl

## 代码开发

修改菜品 接口开发：

```
/**  
 * 根据主键修改菜品信息  
 * @param dish  
 */  
@AutoFill(OperationType.UPDATE)  
void update(Dish dish);
```

DishMapper

## 代码开发

修改菜品 接口开发:

```
<update id="update">
  update dish
  <set>
    <if test="name != null">
      name = #{name},
    </if>
    <if test="categoryId != null">
      category_id = #{categoryId},
    </if>
    <if test="price != null">
      price = #{price},
    </if>
    <if test="image != null">
      image = #{image},
    </if>
    <if test="description != null">
      description = #{description},
    </if>
    <if test="status != null">
      status = #{status},
    </if>
    <if test="updateTime != null">
      update_time = #{updateTime},
    </if>
    <if test="updateUser != null">
      update_user = #{updateUser},
    </if>
  </set>
  where id = #{id}
</update>
```

DishMapper.xml



## 修改菜品

- 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





传智教育旗下高端IT教育品牌