登录及身份验证使用**JWT令牌**，自定义拦截器完成认证。通过ThreadLocal配合拦截器来进行token的校验，判断用户是否处于登录状态。此外，使用**Redis**存储热点数据，如菜品，套餐等，缓解多次访问数据库的压力。使用**Nginx**用作HTTP服务器，部署静态资源，反向代理和负载均衡。

JWT令牌技术

> 什么是JWT?

JSON Web Token，通过数字签名的方式，以JSON对象为载体，在不同的服务终端之间安全的传输信息。

jwt就是将原始的json数据格式进行了安全的封装，这样就可以直接基于jwt在通信双方安全的进行信息传输。

> JWT有什么用?

JWT最常见的场景就是授权认证，一旦用户登录，后续的每个请求都将包含JWT，系统在每次处理用户的请求之前，都要先进行JWT安全校验，通过之后再进行处理。

> JWT的组成

JWT的组成：　(JWT令牌由三个部分组成，三个部分之间使用英文的点来分割)

第一部分：Header(头)， 记录令牌类型、签名算法等。 例如：{“alg”:“HS256”, “type”:“JWT”}

第二部分：Payload(有效载荷)，携带一些自定义信息、默认信息等。 例如：{“id”:“1”,“username”:“Tom”}

第三部分：Signature(签名)，防止Token被篡改、确保安全性。将header、payload，并加入指定秘钥，通过指定签名算法计算而来。

Day01

后端搭建-前后端连调

nginx反向代理的好处：
提高访问速度、进行负载均衡、保证后端服务安全

nginx反向代理的配置方式：

```
server {
        listen       80;
        server_name  localhost;
        #charset koi8-r;
        #access_log  logs/host.access.log  main;
        location / {
            root   html/sky;
            index  index.html index.htm;
        }
```

nginx负载均衡策略：

```
| 名称 | 说明|
| -------- | -------- |
| weight   | 权重方式，默认为1，权重越高，被分配的客户端就越多 |
| ip_hash  | 依据ip分配方式，每个访客可以固定访问一个后端服务  |
| least_coon | 最少连接方式，把优先请求分配给连接数少的后端服务  |
```

Swagger

其用来生成接口文档，以及在线接口调试界面

官网:https://swagger.io/

Knife4j是为Java MVC框架集成Swagger生成API文档的增强解决方案

使用方式

1、导入knife4j的maven坐标

2、在配置类中加入knife4j相关配置

3、设置静态资源文档映射，否则接口文档页面无法访问

常用注解

注解	说明
@Api	用在类上，例如Controller，表示对类的说明
@ApiModel	用在类上，例如entity、DTO、VO
@ApiModelProperty	用在属性上，描述属性信息
@ApiOperation	用在方法上，例如Controller的方法，说明方法的用途，

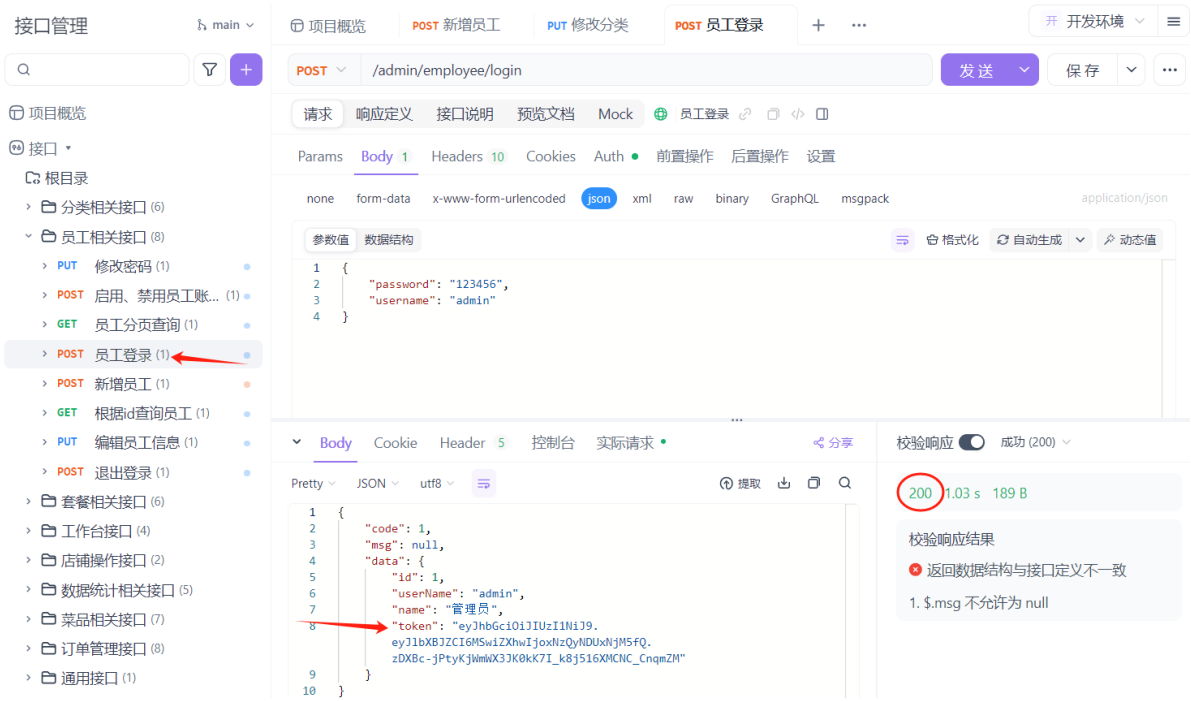
Day02

使用Apifox进行功能测试

添加header的token

先进行登录，获取token。注意的是要先进行**开发环境的服务器配置。**

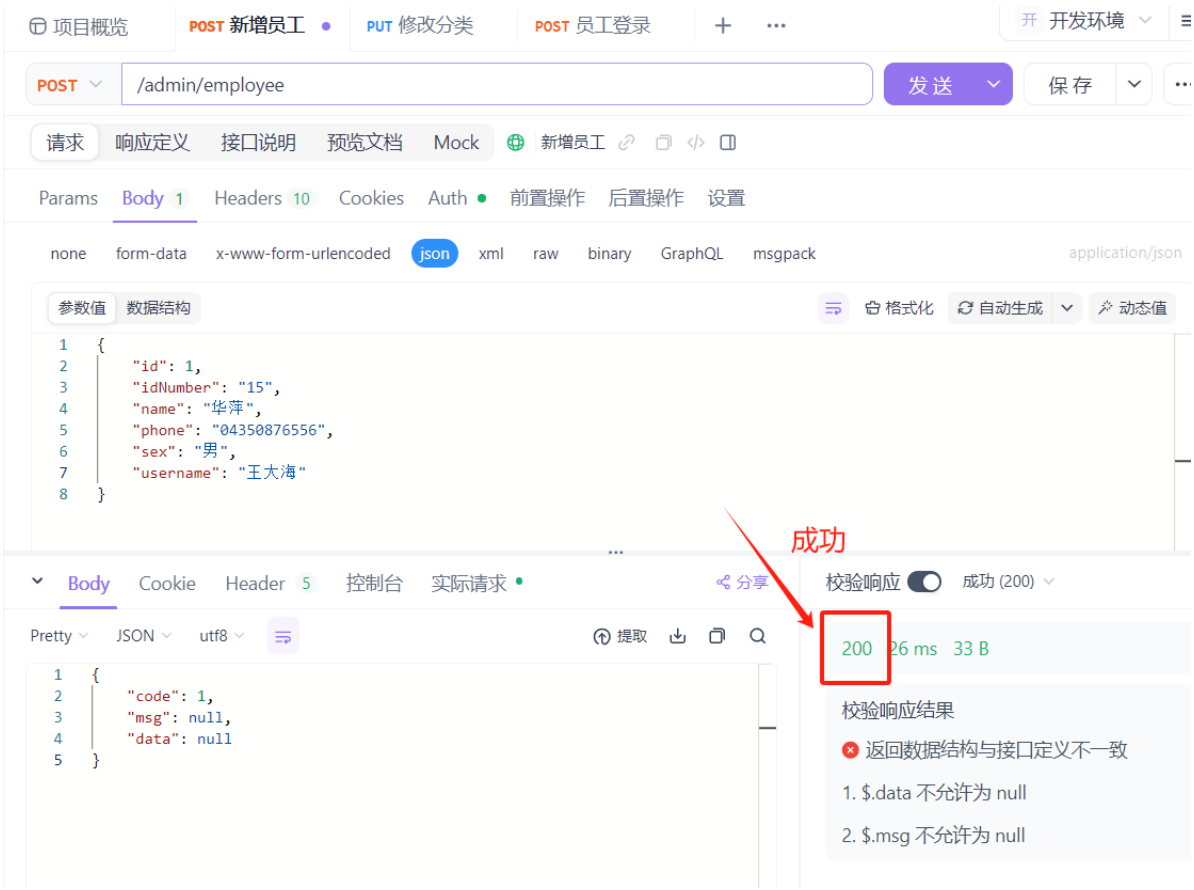
然后登录，获取token。



接下来，在新增员工接口中的**Auth**中选择**API Key**，**保存**之后点击发送即可。

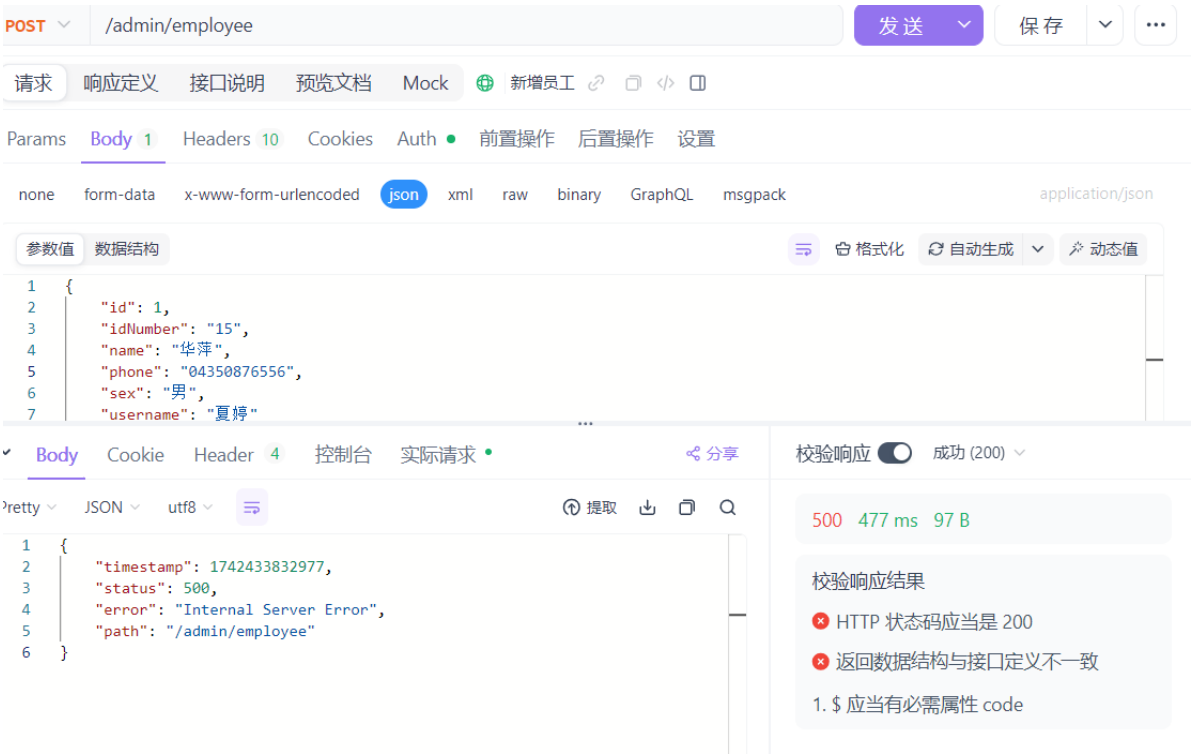
要注意，**token是有时限的**，一般2小时，如果错误，记得**更新token**

成功如下图所示：



500错误

使用Apifox调试登录时报500错误，



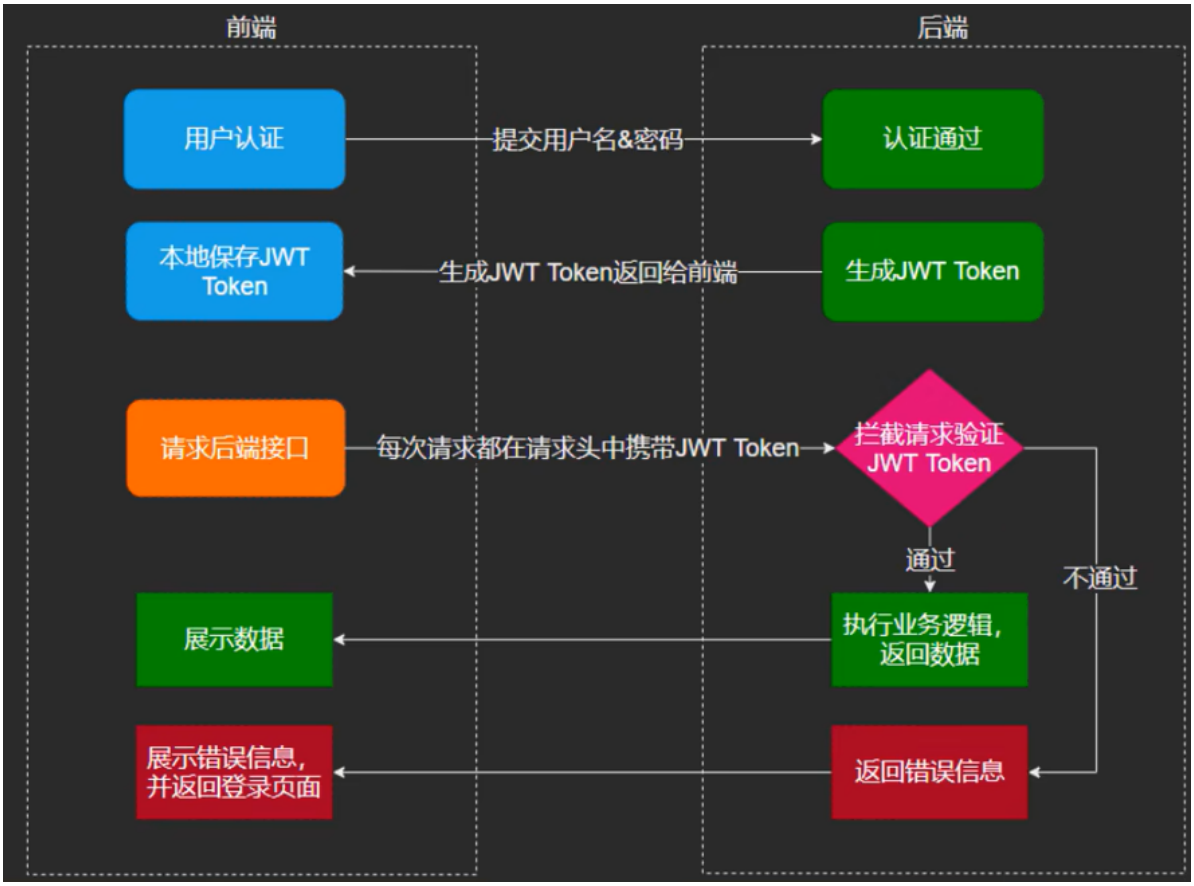
一般是数据库错误。

```
工数据
employee

"insert into employee (name,username,password,phone,sex,id_number,create_time,update_time,create_user,update_use
+
},#{username},#{password},#{phone},#{sex},#{idNumber},#{create_time},#{update_time},#{create_user},#{update_user
ert(Employee employee);
```

下面的值要和employee实体的属性值一一对应，而上面的则是数据库中的字段。

JWT token



JWT 令牌

JWT 令牌（JSON Web Token）是一种用来安全传输信息的字符串，它由三部分组成：**Header（头部）、Payload（负载）和 Signature（签名）**，通过 `.` 进行分隔。例如：

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOjEwMDEsImV4cCI6
MTY5NzMwMjY4MH0.dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk
```

JWT 在 Vue3 + SpringBoot 认证中的流程

1. **用户登录**
   - Vue3 前端提交用户名和密码到 SpringBoot 后端。
   - SpringBoot 校验用户信息，如果正确，生成 JWT 并返回给前端。
2. **前端存储 JWT**
   - Vue3 前端接收 JWT，并存储在 `localStorage` 或 `sessionStorage`（推荐 `sessionStorage` 避免 XSS 攻击）。
   - 之后的 API 请求都会携带 JWT 作为 `Authorization` 头部。
3. **后端验证 JWT**
   - Vue3 前端每次请求 API 时，在 `Authorization` 头部添加 `Bearer JWT_TOKEN`。
   - SpringBoot 拦截请求并解析 JWT，校验有效性。
   - 解析成功则放行，否则返回 401 未授权。
4. **JWT 过期处理**
   - 服务器端通常会设定 JWT 过期时间（如 2 小时）。
   - 过期后，前端需要引导用户重新登录，或者实现 Token 刷新机制。

ThreadLocal

ThreadLocal并不是一个Thread，而是他的局部变量

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

日期格式

解决方法：

1、在属性上加注解，对日期进行格式化

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

2、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

```
@Override
    protected void
extendMessageConverters(List<HttpMessageConverter<?>> converters)
{
        log.info("扩展消息转换器....");
        //创建一个消息转换器对象
        MappingJackson2HttpMessageConverter converter = new
MappingJackson2HttpMessageConverter();
        //需要为消息转换器设置一个对象转换器，对象转化器可以将Java对象序列化为
json数据
        converter.setObjectMapper(new JacksonObjectMapper());
        //将自己的消息转换器加入容器中
        converters.add(0,converter);
    }
```

修改密码

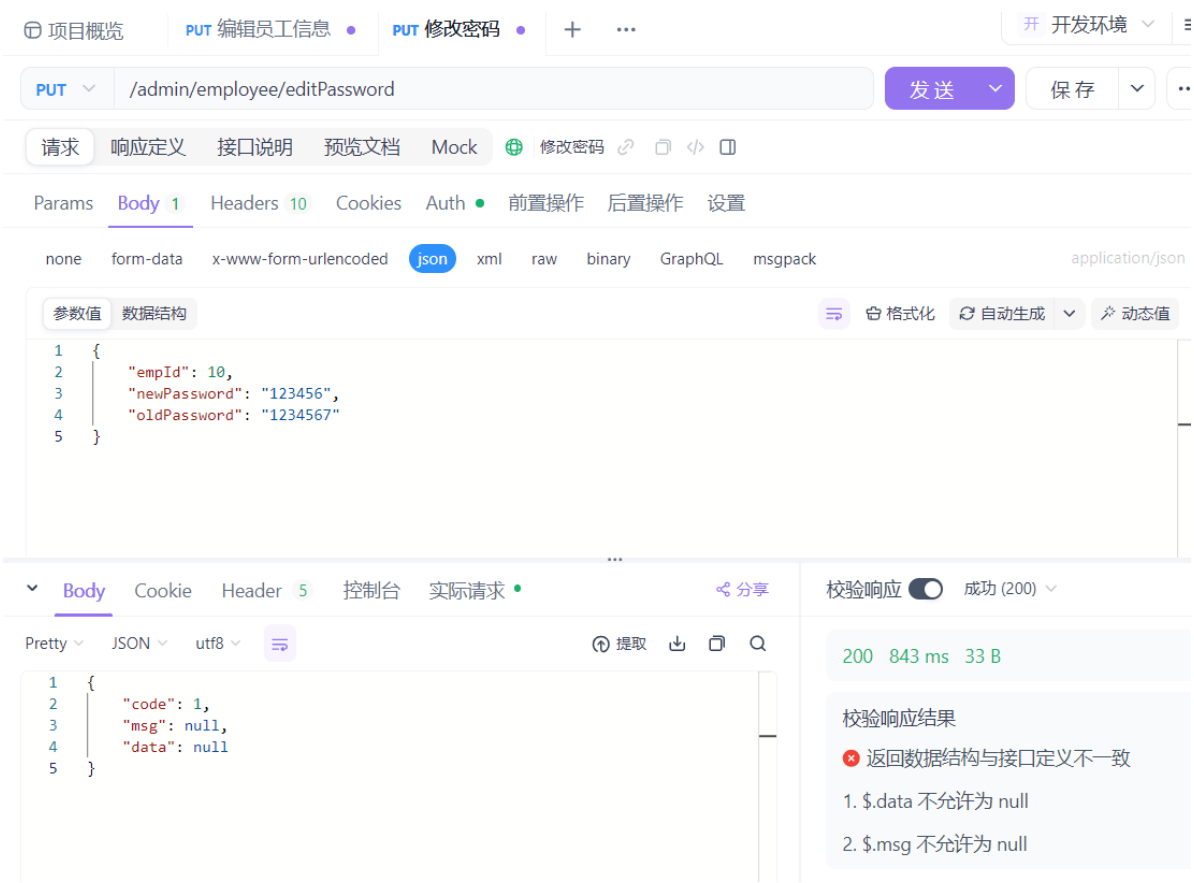
WHERE ⇅ ORDER BY

id	name	username	password	phone	sex	id_r
1	管理员	admin	e10adc3949ba59abbe56e057f20f…	13812312312	1	110101
2	华萍	夏婷	e10adc3949ba59abbe56e057f20f…	04350876556	男	15
4	华萍	王大海	e10adc3949ba59abbe56e057f20f…	04350876556	男	15
5	李四	lisi	e10adc3949ba59abbe56e057f20f…	15975312312	1	111222
10	豌梓诚	wangwu	1234567	15977531212	1	111222

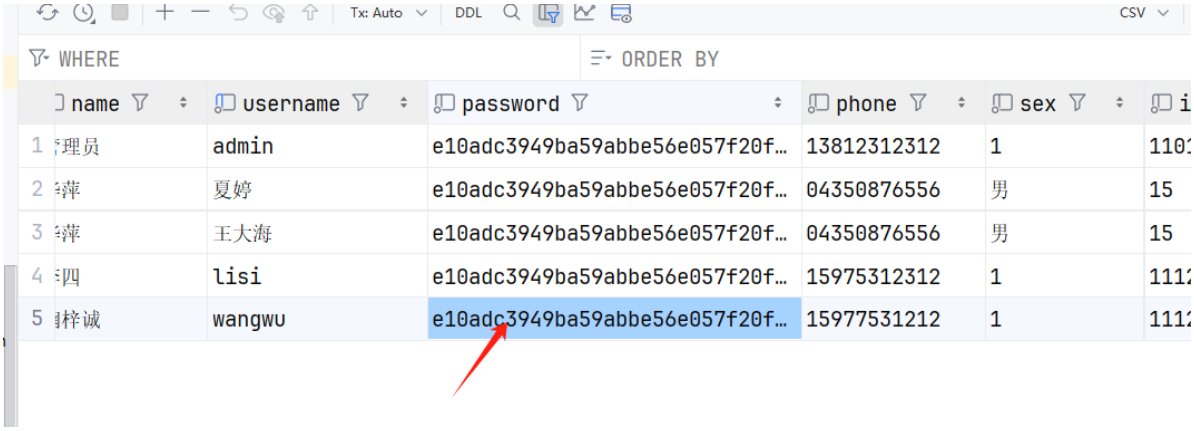
之前修改忘记买md5加密了。

可以看到我的原密码是1234567，那么接下来再修改回之前的123456.

成功如下图所示：



同时查看数据库中，密码确实是**加密**后的123456



步骤如下：

1、创建修改密码的方法

在**EmployeeController**中添加如下方法：

```
@PutMapping("/editPassword")
    @ApiOperation("修改员工密码")
    public Result updatePassword(@RequestBody PasswordEditDTO
passwordEditDTO) {
        log.info("修改员工密码{}", passwordEditDTO);
        employeeService.updatePassword(passwordEditDTO);
        return Result.success();
    }
```

其中**PasswordEditDTO**是资料中自带的，在sky-pojo的dto中。**DTO一般用来接收前端的数据。**

DTO（Data Transfer Object，数据传输对象）

它的主要作用是**封装数据**，简化不同模块或系统之间的数据传输，通常用于**传输对象的集合。**DTO 可以减少服务调用过程中的开销，避免直接暴露数据库实体或复杂的数据结构。

为什么要使用 DTO?

假设你有一个复杂的数据库模型（如 `Employee` 类），它包含许多属性，但你并不希望把所有数据传递给客户端（例如，客户端不需要显示密码字段）。这时你可以通过 DTO 来封装需要传输的数据。

2、在Service层声明接口并实现

```
//修改员工密码,EmployeeService中
    void updatePassword(PasswordEditDTO passwordEditDTO);
```

然后实现，注意，由于我们数据库中存储的是**加密后的密码**，因此在**调用mapper之前**要对DTO中的新密码**加密。**

```
/**
     * 修改员工密码
     * @param passwordEditDTO
     */
    @Override
    public void updatePassword(PasswordEditDTO passwordEditDTO) {
        //设置密码

 passwordEditDTO.setNewPassword(DigestUtils.md5DigestAsHex(passwo
rdEditDTO.getNewPassword().getBytes()));
        employeeMapper.updatePassword(passwordEditDTO);
    }
```

3、Mapper进行数据库操作

由于只是修改操作比较简单，因此通过使用注解来实现。

```
/**
     * 修改员工密码
     * @param passwordEditDTO
     */
    @Update("UPDATE employee SET password = #{newPassword} WHERE
id = #{empId}")
    void updatePassword(PasswordEditDTO passwordEditDTO);
```

一些细节：

MyBatis 并不支持像 `#{passwordEditDTO.newPassword}` 这样**通过对象的嵌套属性**访问。

如果想访问 `PasswordEditDTO` 中的 `newPassword` 和 `empId` 属性，应该使用以下方式进行修改

```
@Update("UPDATE employee SET password = #{newPassword} WHERE id =
#{empId}")
void updatePassword(PasswordEditDTO passwordEditDTO);
```

MyBatis 使用 `#{}` 来引用传递给 SQL 语句的参数。

在的原始 SQL 中，`passwordEditDTO.newPassword` 和 `passwordEditDTO.empId` 这样的写法是错误的，因为 MyBatis 只会根据参数的字段名直接进行映射。

如果你传递的是一个对象，MyBatis 会自动把**对象的属性**与 **SQL 语句中的参数**进行**匹配**。所以应该使用 `#{newPassword}` 和 `#{empId}` 直接引用 `PasswordEditDTO` 的属性。

Day03

1、图片前端无法回显

如果在前端添加菜品的图片无法回显，就将阿里云OSS的读写权限改为公共读即可。

2、useGeneratedKeys

表示在执行完数据库操作之后要获取主键值

```
<insert id="insert" useGeneratedKeys="true" keyProperty="id">
        INSERT INTO dish (name, category_id, price, image,
description, status, create_time, update_time, create_user,
update_user)
        VALUES (#{name}, #{categoryId}, #{price}, #{image}, #
{description}, #{status}, #{createTime}, #{updateTime}, #
{createUser}, #{updateUser});
    </insert>
```

`useGeneratedKeys="true"` 和 `keyProperty="id"`，它的作用是 **自动获取数据库生成的主键 ID 并赋值给 Java 对象的 `id` 属性。**

3、VO和DTO

VO 和 DTO 的定义

名称	全称	作用	主要应用场景
VO（视图对象）	View Object	**封装返回给前端的数据**	只用于 Controller 层，保证返回的数据格式符合前端需求
DTO（数据传输对象）	Data Transfer Object	**用于 Service 层之间的数据传输**	在 Controller 和 Service 之间传递数据，或者用于多个微服务之间传输数据

VO 和 DTO 的区别

对比项	VO（View Object）	DTO（Data Transfer Object）
作用	**封装后端返回给前端的数据**，对数据进行格式化、脱敏	**用于后端不同层之间的数据传输**，通常用于 **Controller 和 Service 层交互**
是否与前端交互	是，VO 主要用于 **返回 JSON** 给前端	否，DTO 主要用于 **内部数据传输**
是否包含业务逻辑	**不包含**，只用于数据展示	**可能包含** 一些基本的业务逻辑，如字段验证
是否与数据库表直接对应	**不一定**，可能是多个表数据的组合	**不一定**，但通常是数据库表字段的映射
字段是否与 Entity 相同	**可能不同**，VO 可能会隐藏一些敏感数据（如 `password`）	**通常相同**，但可能会增加校验规则

4、MyBatis、XML

XML（可扩展标记语言，Extensible Markup Language） 是一种标记语言，用于定义文档的结构，主要用于数据交换和存储。

MyBatis 是一个流行的 Java 持久层框架，它提供了一个支持定制化 SQL、存储过程和高级映射的框架。MyBatis 通过将 Java 对象与数据库表中的数据进行映射，使开发者可以更加灵活地进行数据库操作。与 Hibernate 不同，MyBatis 需要开发者手动编写 SQL 语句，这使得它在一些复杂查询场景下更具优势，尤其是在性能要求较高的项目中。

MyBatis 的核心就是使用 XML 配置文件或注解来定义 SQL 语句，并将结果映射到 Java 对象中。

```xml
<mapper namespace="com.example.UserMapper">
  <select id="getUserById" resultType="com.example.User">
    SELECT * FROM users WHERE id = #{id}
  </select>
</mapper>
```

5、动态SQL

什么是动态 SQL?

动态 SQL 是指 **SQL 语句在运行时动态拼接**，而不是固定不变的 SQL 语句。
在 MyBatis 中，动态 SQL 允许根据 **不同的查询条件** 生成不同的 SQL 语句，以提高查询的灵活性和效率。

为什么需要动态 SQL?

在实际开发中，查询条件往往是 **可选** 的，比如：

- 按 **用户名** 查询用户
- 按 **手机号** 查询用户
- 按 **用户名和手机号** 查询用户

如果每种情况都写一个 `SQL` 方法，代码会很冗余。
而使用 **动态 SQL**，可以 **根据条件动态生成 SQL 语句**，减少重复代码，提高性能。

为什么使用 XML 来写动态 SQL?

在 MyBatis 中，动态 SQL **既可以使用 XML，也可以用注解**，但通常推荐 **使用 XML**，原因如下：

1. XML 代码更清晰

- **如果 SQL 语句较复杂**，比如 `if、foreach、choose` 等，使用 XML 更直观。
- 代码可读性强，方便维护。

2. 方便修改，不需要重新编译代码

- SQL 逻辑变化时，只需要 **修改 XML 文件**，而不用改 Java 代码。
- **不需要重启项目**，比注解方式更灵活。

3. 支持更丰富的动态 SQL 语法

- XML 支持 `<if>、<choose>、<where>、<foreach>`，可以更轻松地实现复杂 SQL 逻辑。
- **注解模式下，SQL 只能拼接字符串**，写多了很难维护。

6、注解相关

@RequestParam

`@RequestParam` 是 Spring MVC 中的一个注解，用于将 HTTP 请求中的参数绑定到方法的参数上。它常用于处理 GET 请求中的查询参数或表单提交的参数。

@PathVariable

`@PathVariable` 是 Spring MVC 中的一个注解，用于从 URL 路径中提取变量并传递给控制器方法的参数。它常用于 RESTful 风格的 API，在路径中直接包含参数，而不是通过查询字符串传递。

7、根据分类id查询菜品

先看接口文档，是GET请求，并且传入的是菜品的id，我们可以发现这个和根据分类id查分类非常相似，于是模仿着写即可。

在**DishController**中添加如下代码：

```
/**
     * 根据分类id查询菜品
     * @param categoryId
     * @return
     */
    @GetMapping("/list")
    @ApiOperation("根据分类id查询菜品")
    public Result<List<Dish>> list(Long categoryId){
        List<Dish> list = dishService.list(categoryId);
        return Result.success(list);
    }
```

在**DishService**中添加如下代码：

```
/**
     * 根据分类id查询菜品
     * @param categoryId
     * @return
     */
    List<Dish> list(Long categoryId);
```

DishServiceImpl中如下：

```
@Override
    public List<Dish> list(Long categoryId) {
        return dishMapper.list(categoryId);
    }
```

DishMapper：

```
/**
     * 根据分类id查询菜品
     * @param categoryId
     * @return
     */
    List<Dish> list(Long categoryId);
```

DishMapper.xml:

```
<select id="list" resultType="Dish">
        select * from dish
        where category_id = #{categoryId}
        order by create_time desc
    </select>
```

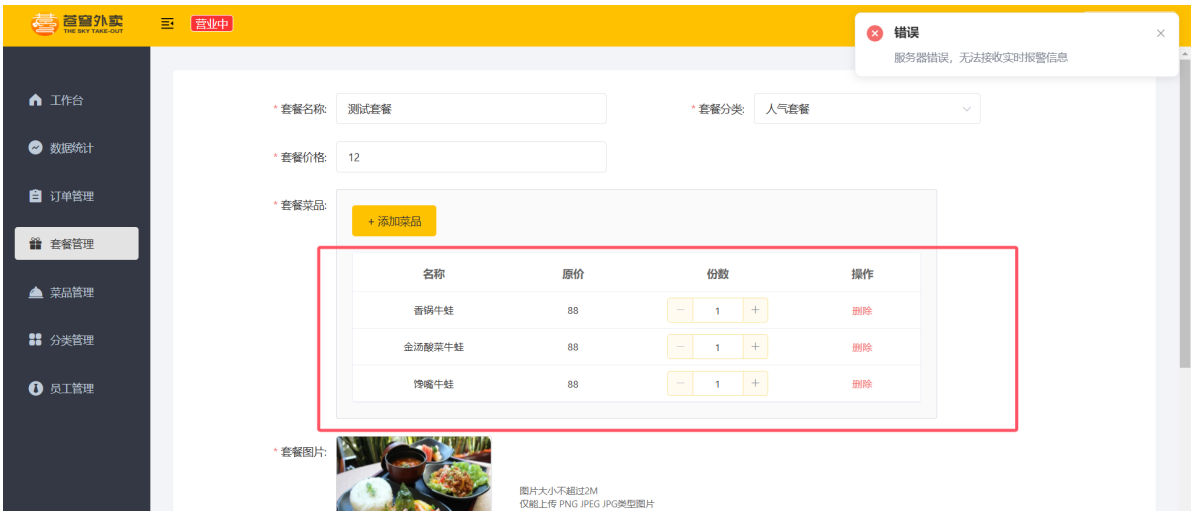
8、菜品的起售停售

与员工的类似不过多赘述

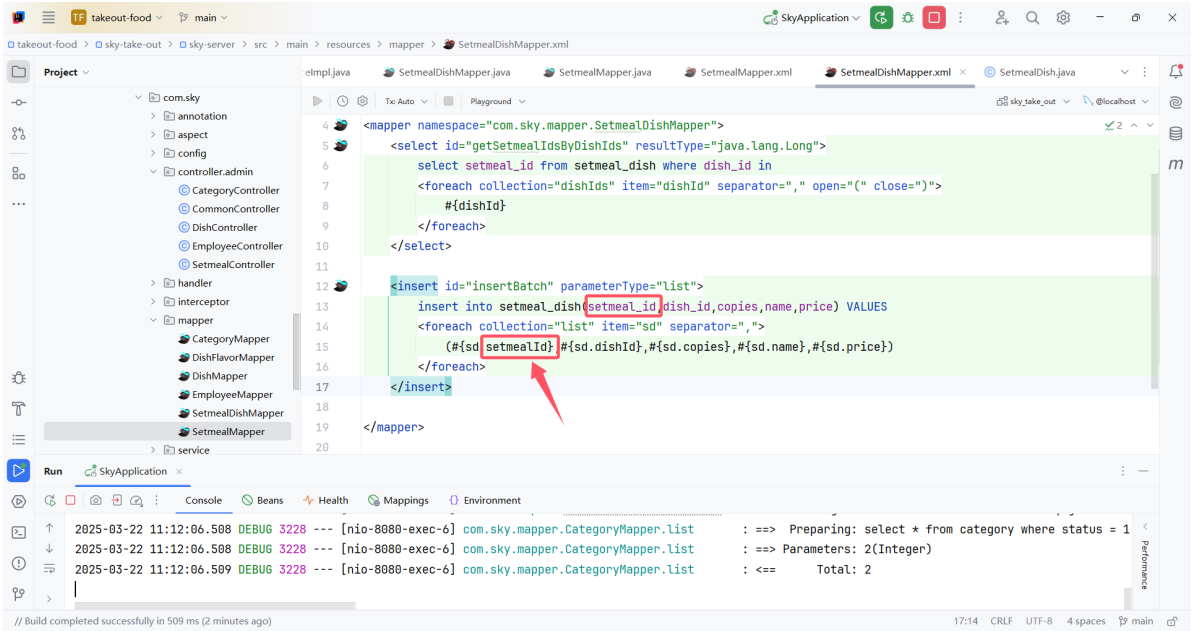
Day04(作业)

与之前菜品类似，模仿即可，这里提一下我自己的错误。

点击修改如果无法回显



点击修改如果无法回显则说明数据库忘记插入了。如下图，记得加上：



数据删除

自己写的时候主要是删除功能没能实现，后面发现是相关的SQL语句不太熟练。

```
<delete id="deleteByDishIds">
        delete from dish_flavor where dish_id in
        <foreach collection="dishIds" open="(" close=")"
separator="," item="dishId">
            #{dishId}
        </foreach>
    </delete>
```

参数名	作用
collection="dishIds"	指定 **Java 传入的参数** `dishIds`，要求它是一个 **列表**（`List<Integer>`）。
item="dishId"	代表**每次遍历时的元素**，即 `dishIds` 列表中的每个 `dishId` 值。
open="("	**在 SQL 语句中添加左括号** `(`。
close=")"	**在 SQL 语句中添加右括号** `)`。
separator=","	**每个 dishId 之间用** `,` **连接**，符合 SQL 语法要求。

Day05

Redis部分

Redis是基于一个**内存**的key-value结构数据库

key	value
id	101
name	小智

- 基于内存存储，读写性能高
- 适合存储热点数据

Redis常用数据类型

在redis中key是字符串类型，value有五种常用的数据类型：
字符串string、哈希hash、列表list、集合set、有序集合sorted set/zset

Redis 主要有 **五种常用数据类型**，分别是：**String、List、Hash、Set、ZSet（Sorted Set）**。每种数据类型都有不同的特点和适用场景，下面我详细讲解一下。

---

1. String（字符串）

特点：

- **最基本的数据类型**，可以存储 **字符串、整数、浮点数、二进制数据（如图片）**。
- **单个 key** 最多能存储 **512MB** 的数据。

应用场景：

- **缓存用户信息**（如 `SET user:123 name "Tom"`）。
- **计数器**（`INCR`、`DECR`）。
- **分布式锁**（`SETNX` 实现原子锁）。

---

2. List（列表）

特点：

- **双向链表结构**，支持 **头部、尾部插入删除**，适用于 **队列、栈** 场景。可以有重复元素，**按照插入顺序排序**
- **可以存储 2^32 - 1（大约 40 亿）个元素。**

应用场景：

- **消息队列**（生产者 `RPUSH`，消费者 `LPOP`）。
- **任务队列**（异步任务处理）。
- **社交动态流**（存储用户发布的动态）。

---

3. Hash（哈希）

特点：

- **类似于 Python 的字典**，用于存储**对象信息**（如用户信息）。
- **适用于存储对象，避免使用多个 String key**（如 `user:123 name Tom age 20`）。

常用命令：

```
HSET key field value  # 设置某个字段的值
HGET key field        # 获取某个字段的值
HGETALL key           # 获取所有字段和值
HDEL key field        # 删除某个字段
HEXISTS key field     # 判断字段是否存在
HINCRBY key field n   # 某个字段的值加 n
```

应用场景：

- **存储用户信息**（如 `HSET user:123 name "Tom"`）。
- **存储商品信息**（如 `HSET product:456 name "iPhone"`）。
- **存储会话数据**（如 `HSET session:789 token "xyz123"`）。

---

4. Set（集合）

特点：

- **无序且不允许重复元素**，适合 **去重** 场景。
- **支持交集、并集、差集** 计算，适合**社交网络**。

常用命令：

```
SADD key value     # 添加元素
SREM key value     # 移除元素
SMEMBERS key       # 获取集合中所有元素
SISMEMBER key value # 判断元素是否存在
SINTER key1 key2   # 求交集
SUNION key1 key2   # 求并集
SDIFF key1 key2    # 求差集
```

应用场景：

- **去重**（如 `SADD ip_blacklist "192.168.1.1"`）。
- **社交关系**（如 `SINTER user:1:friends user:2:friends` 获取共同好友）。
- **标签系统**（如 `SADD user:tags "AI"` 存储用户兴趣标签）。

---

5. ZSet（Sorted Set，有序集合）

特点：

- **带分数的 Set**，元素是 **唯一的**，但可以按照 **分数** 进行排序。
- **适用于排行榜、优先级队列。**

常用命令：

```
ZADD key score value  # 添加元素并指定分数
ZREM key value        # 移除元素
ZRANGE key start stop [WITHSCORES]  # 获取排名范围
ZREVRANGE key start stop [WITHSCORES]  # 逆序获取排名
ZRANK key value       # 获取元素排名（从小到大）
ZREVRANK key value    # 获取元素排名（从大到小）
ZSCORE key value      # 获取元素的分数
```

应用场景：

- **排行榜**（如 `ZADD game_score 100 "Tom"`）。
- **任务优先级队列**（如 `ZADD task_queue 1 "task1"`）。
- **延迟任务调度**（如 `ZADD delay_queue 1681234567 "order:123"` 存储订单超时时间）。

总结

数据类型	结构	主要特点	适用场景
String	普通字符串	存储简单值，支持整数/浮点数操作	**缓存、计数器、分布式锁**
List	双向链表	允许左右插入/删除，适合队列	**消息队列、任务队列、社交动态**
Hash	字典（key-value）	适合存储对象，节省空间	**用户信息、商品信息、会话数据**
Set	无序集合	元素唯一，支持交并差运算，无序且无重复	**去重、好友关系、标签系统**
ZSet	有序集合	元素唯一，支持排序，无重复，根据分数升序排序	**排行榜、优先级队列、延迟任务**

Redis常用命令

1. String（字符串）

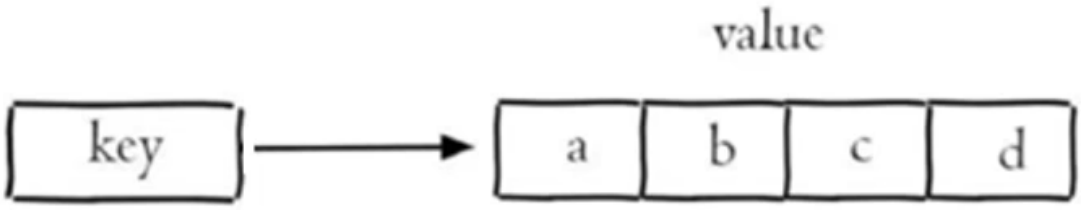
常用命令：

```
SET key value    # 存储一个值
GET key          # 读取一个值
SETEX key seconds value #设定指定key的值，并将key的过期时间设定为seconds秒
SETNX key value #只有在key不存在时设置key的值
```

2. List（列表）

常用命令：

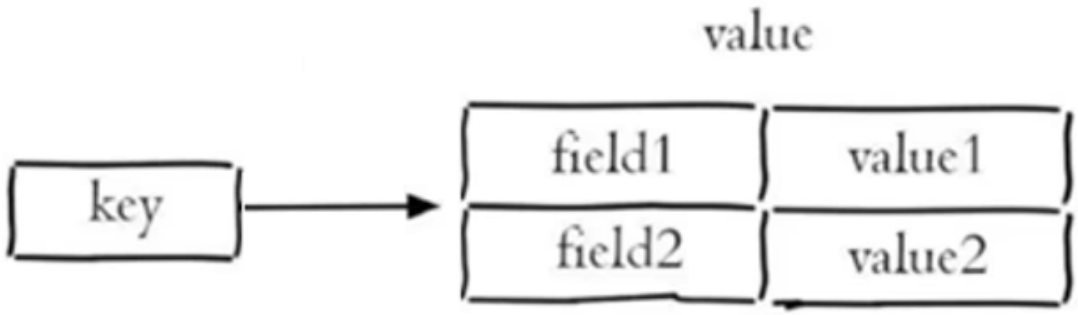
```
LPUSH key value1 [value2]  # 将一个或多个值插入到列表头部
LRANGE key start stop  # 获取指定范围的元素
LLEN key          # 获取列表长度
RPOP key          # 移除并获取列表最后一个元素
```



3. Hash（哈希）

常用命令：

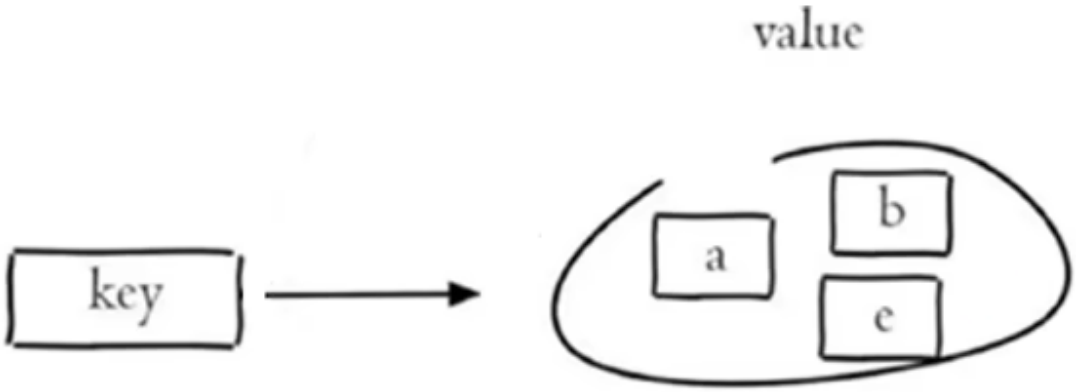
```
HSET key field value  # 设置某个字段的值
HGET key field         # 获取某个字段的值
HDEL key field        #删除存储在哈希表中的指定字段
HKEYS key             #获取哈希表中的所有字段
HVALS key             #获取哈希表中的所有值
```



4. Set（集合）

常用命令：

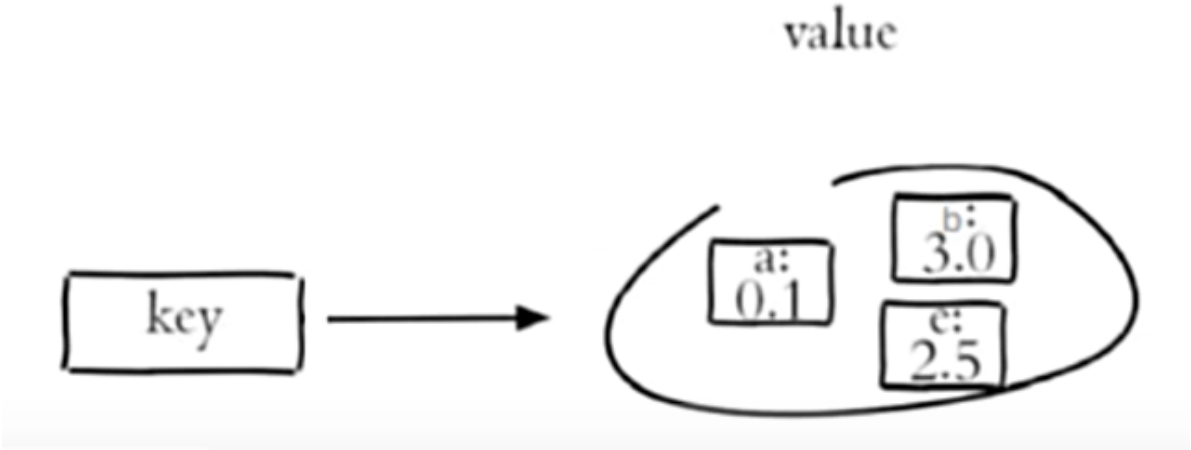
```
SADD key member1 [member2]     # 添加一个或多个元素
SMEMBERS key       # 获取集合中所有元素
SCARD key          #获取集合的成员数
SINTER key1 [key2]  # 返回给定所有集合的交集
SUNION key1 [key2]  # 返回给定所有集合的并集
SREM key [value1] [value2]     # 删除集合中的一个或多个成员
```



5. ZSet（Sorted Set，有序集合）

常用命令：

```
ZADD key score1 value1 [score2 value2]  # 添加元素并指定分数
ZRANGE key start stop [WITHSCORES]  # 获取排名范围
ZINCRBY key increment member #有序集合中对指定成员的分数加上增量
increment
ZREM key member [member2...] #移除有序集合中的一个或多个成员
```



、

6、通用命令

```
KEYS pattern     #查找所有符合给定模式(pattern)的key
EXISTS key       #检查给定key是否存在
TYPE key         #返回key所存储的值的类型
DEL key          #该命令用于在key存在是删除key
```

Sping Data Redis

Sping Data Redis是Spring的一部分，对Redis底层开发包进行了高度封装

Spring Data Redis 使用方式

操作步骤:

1. 导入 Spring Data Redis 的 maven 坐标
2. 配置 Redis 数据源
3. 编写配置类，创建 RedisTemplate 对象
4. 通过 RedisTemplate 对象操作 Redis

Day06

HttpClient

HttpClient是ApacheJakartaCommon下的子项目，可以用来提供高效的、最新的、功能丰富的支持HTTP协议
的客户端编程工具包，并且它支持HTTP协议最新的版本和建议。

微信小程序

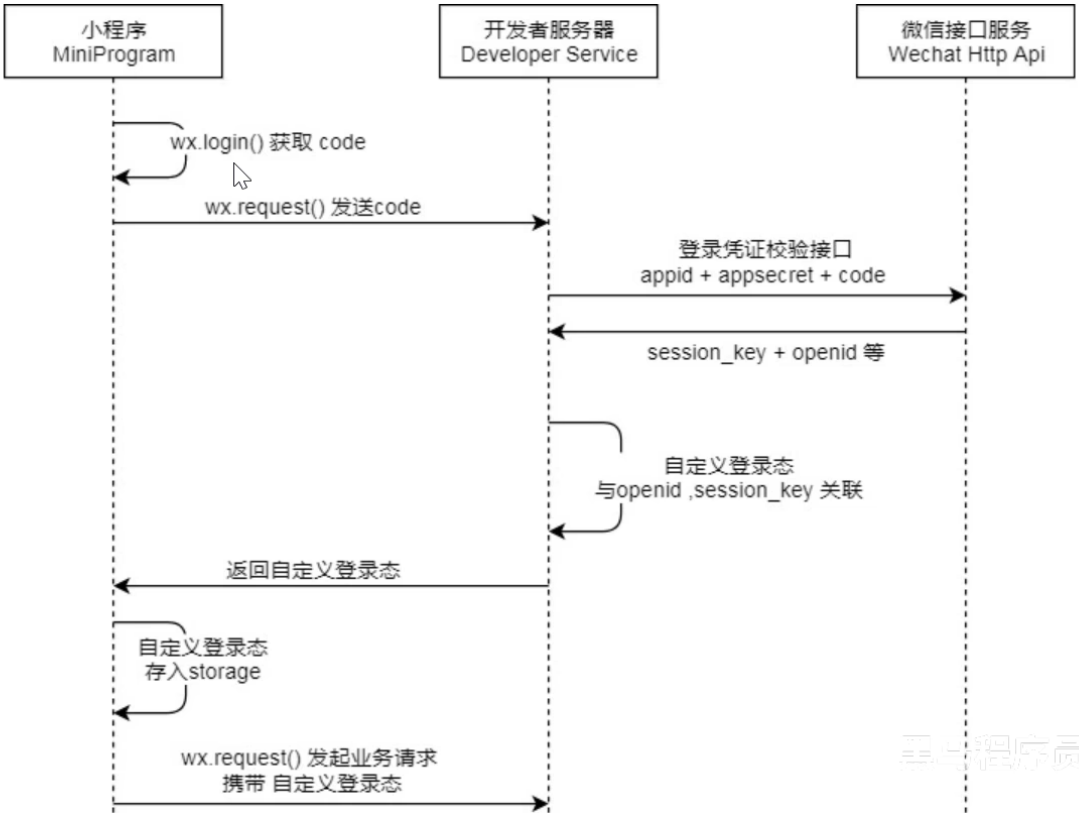
小程序包含一个描述整体程序的 app 和多个描述各自页面的 page。一个小程序主体部分由三个文件组成，必须放在项目的根目录，如下:

文件	必需	作用
app.js	是	小程序逻辑
app.json	是	小程序公共配置
app.wxss	否	小程序公共样式表

一个小程序页面由四个文件组成：

文件类型	必需	作用
js	是	页面逻辑
wxml	是	页面结构
json	否	页面配置
wxss	否	页面样式表

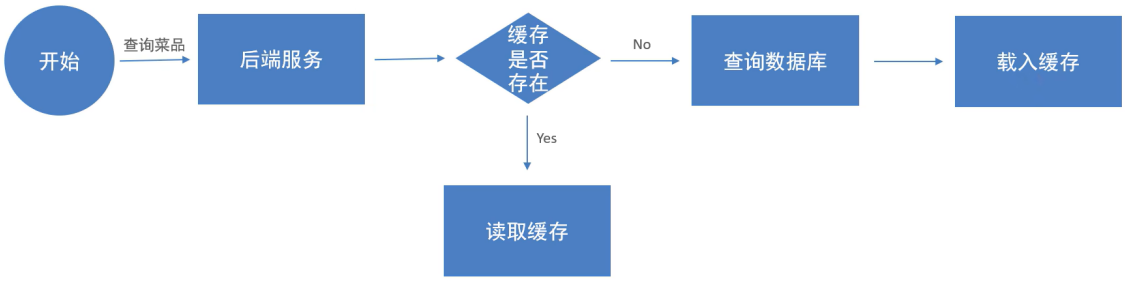
登录流程时序



Day07

缓存菜品

通过Redis来缓存菜品数据，减少数据库查询操作。



缓存逻辑：

缓存逻辑分析：

- 每个分类下的菜品保存一份缓存数据
- 数据库中菜品数据有变更时清理缓存数据

key	value
dish_1	string(...)
dish_2	string(...)
dish_3	string(...)

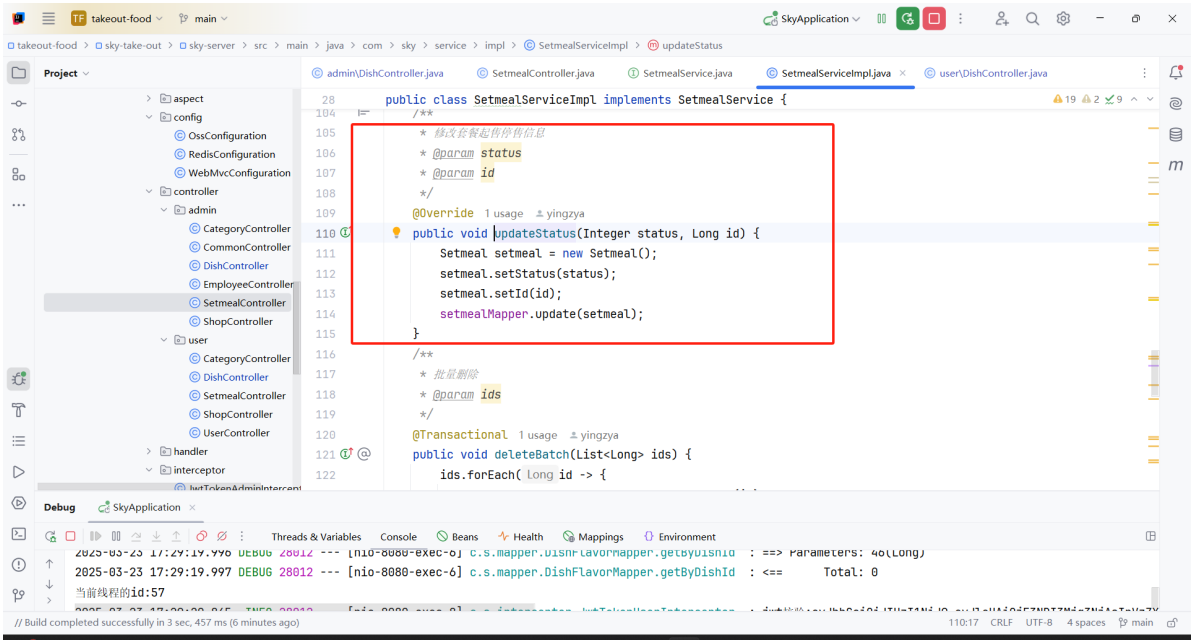
修改管理端接口 DishController 的相关方法，加入清理缓存的逻辑，需要改造的方法：

- 新增菜品
- 修改菜品
- 批量删除菜品
- 起售、停售菜品

一些问题(停售还能查到菜品)：

停售之后，用户端还能**看到菜品。**

经过代码查看，发现原来写的时候只是修改状态，没做相应的判断。



修改如下：(来自**黑马资料**)

5.2.3 SetmealServiceImpl

```
/**
     * 套餐起售、停售
     * @param status
     * @param id
*/
public void startOrStop(Integer status, Long id) {
    //起售套餐时，判断套餐内是否有停售菜品，有停售菜品提示"套餐内包含未启售菜品，无法启售"
    if(status == StatusConstant.ENABLE){
        //select a.* from dish a left join setmeal_dish b on a.id = b.dish_id where b.setmeal_id = ?
        List<Dish> dishList = dishMapper.getBySetmealId(id);
        if(dishList != null && dishList.size() > 0){
            dishList.forEach(dish -> {
                if(StatusConstant.DISABLE == dish.getStatus()){
                    throw new SetmealEnableFailedException(MessageConstant.SETMEAL_ENABLE_FAILED);
                }
            });
        }
    }

    Setmeal setmeal = Setmeal.builder()
        .id(id)
        .status(status)
        .build();
```

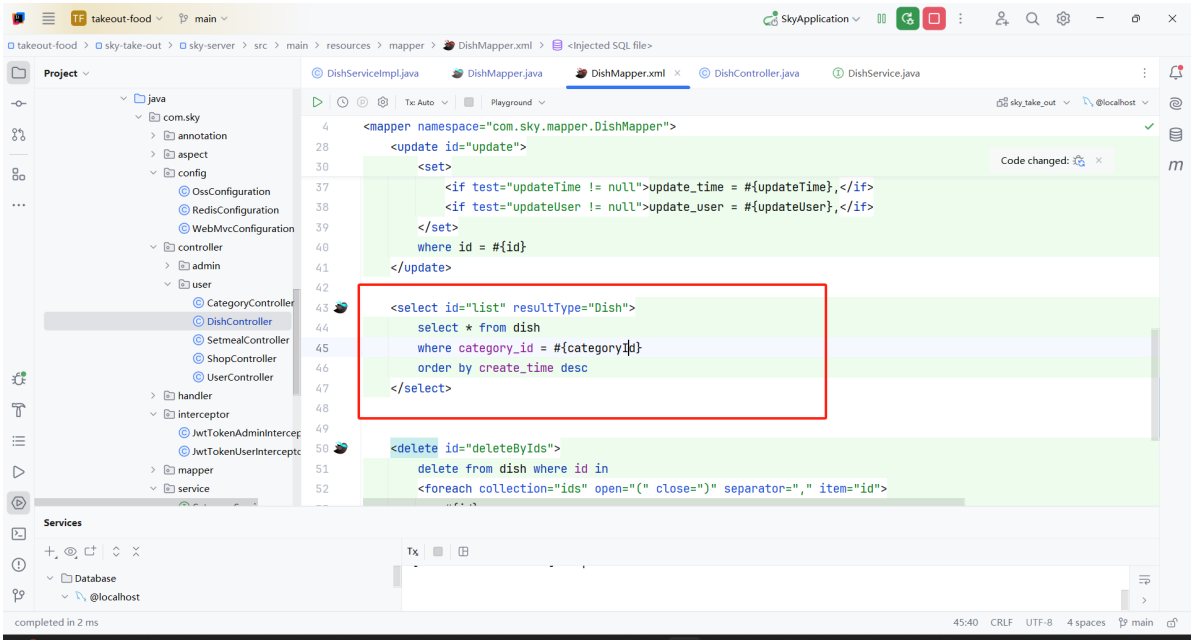
```
        setmealMapper.update(setmeal);
    }
```

5.2.4 DishMapper

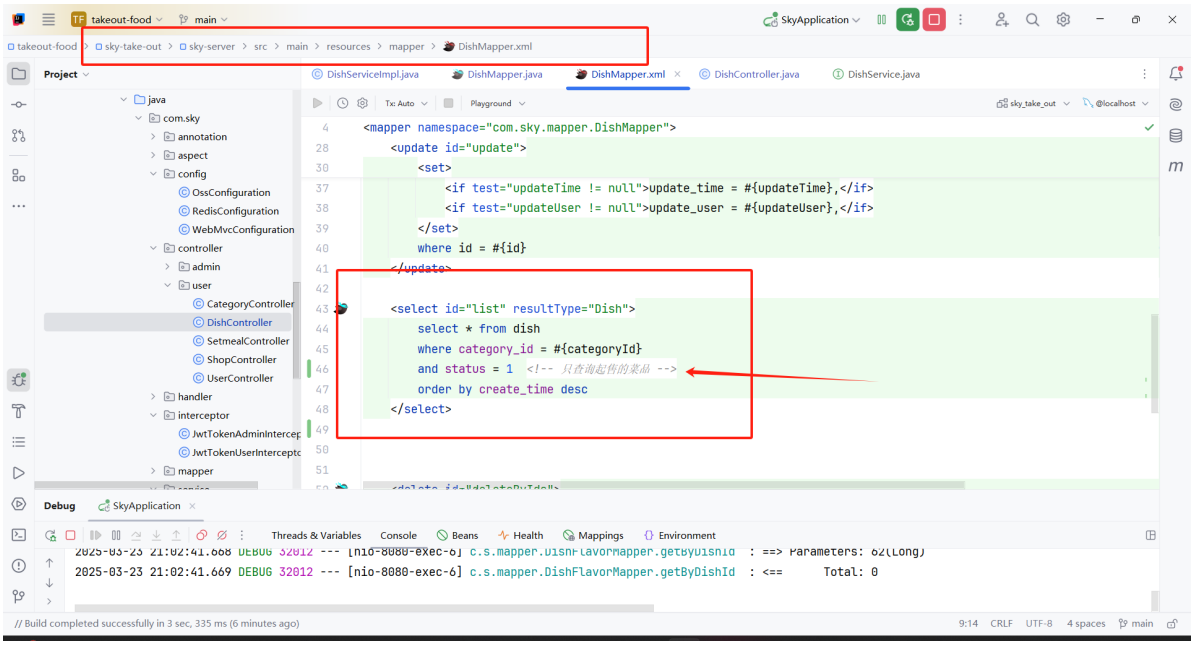
```
/**
     * 根据套餐id查询菜品
     * @param setmealId
     * @return
*/
@Select("select a.* from dish a left join setmeal_dish b on a.id
= b.dish_id where b.setmeal_id = #{setmealId}")
List<Dish> getBySetmealId(Long setmealId);
```

正确修改方法！！：

我以为是没考虑上面的停售起售逻辑，但是通过控制台请求发现，status为0的也被查到了，因此我判断是**数据库的查询**出了问题，如下图所示，我查询的时候只**根据了种类id，没有加售卖状态**，因此导致全查到了。



修改如下即可：



这时即可正常进行查询：

缓存套餐

Spring Cache

Spring Cache 是一个框架，实现了基于**注解**的缓存功能，只需要简单地加一个注解，就能实现缓存功能。

Spring Cache 提供了一层抽象，底层可以切换不同的缓存实现，例如：

- EHCache
- Caffeine
- Redis

Spring Cache 常用注解如下：

注解	说明
@EnableCaching	开启缓存注解功能，通常添加在启动类上。
@Cacheable	方法执行前先查询缓存，若有数据则直接返回缓存数据； 若无缓存数据，调用方法并将返回值存入缓存。
@CachePut	将方法的返回值存入缓存中。
@CacheEvict	从缓存中删除一条或多条数据。

```
@CachePut(cacheNames = "userCache",key = "#user.id") //如果使用springcache缓存数据，key的生成：usercache：xxxx
    public User save(@RequestBody User user){
        userMapper.insert(user);
        return user;
    }
```

`cacheNames = "userCache"`

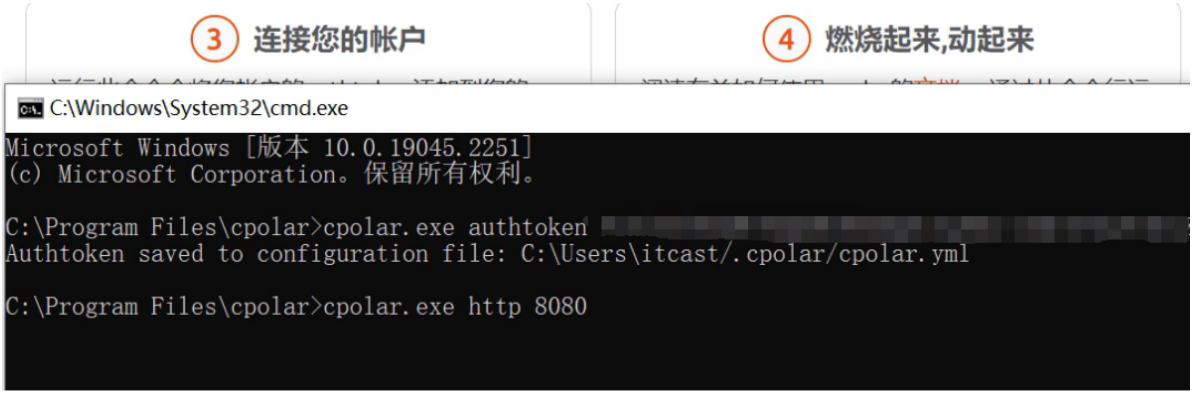
- 这个 `cacheNames`（或 `value`）定义了缓存的名称，类似于一个“缓存的空间”。
- 在不同的地方，如果使用相同的 `cacheNames`，就可以访问相同的缓存数据。

```
@DeleteMapping("/delAll")  //删除所有数据
    @CacheEvict(cacheNames = "userCache",allEntries = true)
    public void deleteAll(){
        userMapper.deleteAll();
    }

    @GetMapping
    @Cacheable(cacheNames = "userCache",key = "#id") //清理一条数据
    public User getById(Long id){
        User user = userMapper.getById(id);
        return user;
    }
```

Day08

订单支付



通过该命令获得临时的公网IP

import org.springframework.beans.factory.annotation.Value;

跳过支付

主要参考这个大佬的笔记：

https://www.bilibili.com/h5/note-app/view?cvid=25087721&pagefrom=comment&richtext=true

Day10

Spring Task

Spring Task是Spring框架提供的任务调度工具，可以按照约定的时间自动执行某个代码逻辑

应用场景：

- 信用卡每月还款提醒
- 银行贷款每月还款提醒
- 火车票售票系统处理未支付订单

定时处理的场景都可以使用Spring Task

cron表达式

cron表达式其实就是一个字符串，通过cron表达式可以定义任务触发的时间

构成规则：分为6或7个域，由空格分隔开，每个域代表一个含义

每个域的含义分别为：秒、分钟、小时、日、月、周、年(可选)

WebSocket

WebSocket 是基于 TCP 的一种新的**网络协议。**它实现了浏览器与服务器全双工通信--浏览器和服务器只需要完成并进行双向数据传输。一次握手，两者之间就可以创建**持久性**的连接，并进行**双向**数据传输。

应用场景：
• 视频弹幕
• 网页聊天
• 体育实况更新
• 股票基金报价实时更新

实现步骤：

1. **直接使用WebSocket.html页面作为WebSocket客户端**
2. **导入WebSocket的Maven坐标**：
3. **导入WebSocket服务端组件WebSocketServer，用于和客户端通信**：
4. **导入配置类WebSocketConfiguration，注册WebSocket的服务端组件**
5. **导入定时任务类WebSocketTask，定时向客户端推送数据**

- 通过WebSocket实现管理端页面和服务端保持长连接状态
- 当客户支付后，调用WebSocket的相关API实现服务端向客户端推送消息
- 客户端浏览器解析服务端推送的消息，判断是来单提醒还是客户催单，进行相应的消息提示和语音播报
- 约定服务端发送给客户端浏览器的数据格式为JSON，字段包括：type、orderId、content

Day11

Apache ECharts

Apache POI

Apache PO! 是一个处理Miscrosoft Office各种文件格式的开源项目。简单来说就是，我们可以使用 POI 在Java 程序中对Miscrosoft Office各种文件进行读写操作。

一般情况下，POI都是用于操作 Excel 文件。