

瑞吉外卖

Day-----1

搭建数据库

图形界面或则命令行

```
//创建数据库
create database reggie CHARACTER SET utf8mb4
```

导入sql文件命令
`source d:/sql_file/db_reggie.sql`;（路径不要有中文）

搭建maven

maven

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.4.5</version>
  </parent>
  <groupId>org.hbx.project</groupId>
  <artifactId>reggie</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <java.version>1.8</java.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

```

</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <scope>compile</scope>
</dependency>

<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-boot-starter</artifactId>
    <version>3.4.2</version>
</dependency>

<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.20</version>
</dependency>

<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>fastjson</artifactId>
    <version>1.2.76</version>
</dependency>

<dependency>
    <groupId>commons-lang</groupId>
    <artifactId>commons-lang</artifactId>
    <version>2.6</version>
</dependency>

<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
</dependency>

<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid-spring-boot-starter</artifactId>
    <version>1.1.23</version>
</dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <version>2.4.5</version>
        </plugin>
    </plugins>
</build>

</project>

```

yaml配置文件

```
server:
  port: 8888
spring:
  application:
    name: reggie_take_out #设置这个项目的名字（默认是项目文件的名字）
  datasource:
    url: jdbc:mysql://localhost:13306/reggie?
    useSSL=false&useUnicode=true&characterEncoding=utf8
    username: root
    password: abc123
    driver-class-name: com.mysql.jdbc.Driver
  mybatis-plus:
    configuration:
      #在映射实体或者属性时，将数据库中表名和字段名中的下划线去掉，按照驼峰命名法映射
      map-underscore-to-camel-case: true
      log-impl: org.apache.ibatis.logging.stdout.StdoutImpl
    global-config:
      db-config:
        id-type: ASSIGN_ID
```

主程序---springbootapplication

导入静态资源-----backed文件和front文件----放在static目录下

或者自己添加静态资源目录

```
@Configuration
public class staticConfig implements WebMvcConfigurer {
    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {

        registry.addResourceHandler("/front/**").addResourceLocations("classpath:/front/");

    }
}
```

后台登陆功能

思路速记

后台登录功能开发

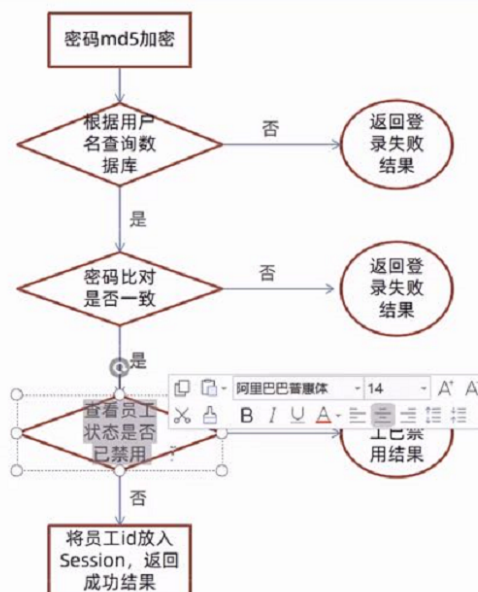


代码开发

4) 在Controller中创建登录方法

处理逻辑如下:

- 1、将页面提交的密码password进行md5加密处理
- 2、根据页面提交的用户名username查询数据库
- 3、如果没有查询到则返回登录失败结果
- 4、密码比对, 如果不一致则返回登录失败结果
- 5、查看员工状态, 如果为已禁用状态, 则返回员工已禁用结果
- 6、登录成功, 将员工id存入Session并返回登录成功结果



controller - service-mapper-查询---返回dao (employee)

前端vue代码的method需要的变量值

规定服务器返回的封装为类型R (code、data、msg、map) ---配合前端

登陆handler---

接受json

保存的session

处理逻辑

(数据库里面是) md5加密 (DigetsUtils.M5D) ----》查用户名-----》查密码

配置过滤器、拦截器---阻止跳过登陆直接访问LoginCheckFilter----- (原生注入的方式)

的好url---筛查是否需要过滤 (/employee/login./dmployee/logout静态资源)

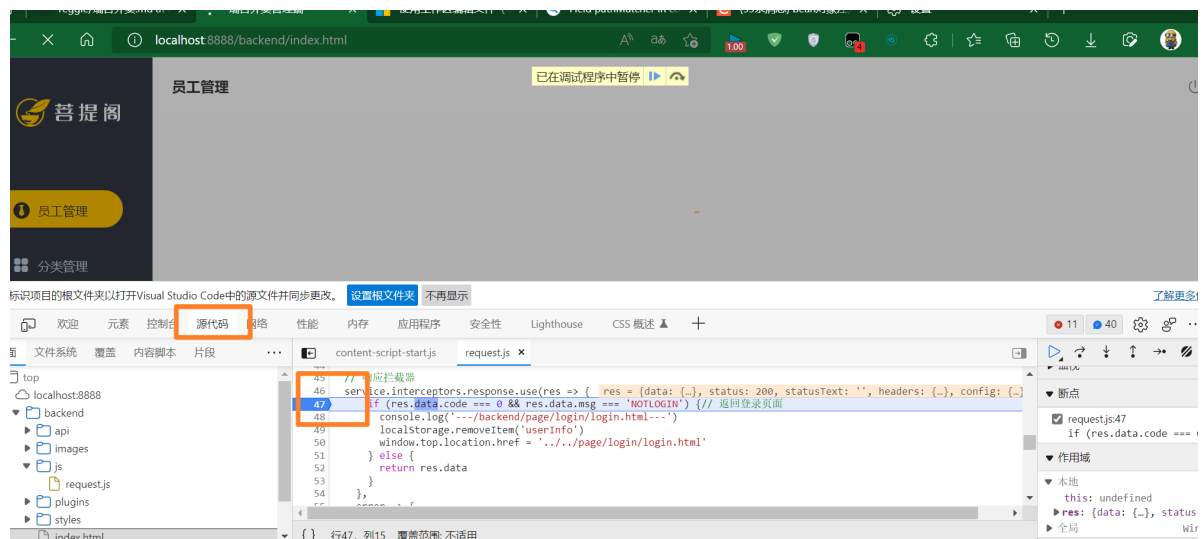
AntPathMatcher-----路径匹配器 解决静态资源 (match方法)

封装check方法检测是否需要放行

获得session对象

没登陆返回---配合JS文件中的响应拦截器-----根据需要的code和msg(后端返回一个输出流 (json的R队形) 就行)

试试调试一下js代码（网页）



```
service.interceptors.response.use(res => {  
  if (res.data.code === 0 && res.data.msg === 'NOTLOGIN') { // 返回登录页面  
    console.log('---/backend/page/login/login.html---')  
    localStorage.removeItem('userInfo')  
    window.top.location.href = '../page/login/login.html'  
  } else {  
    return res.data  
  }  
},
```

backend目录位置导致的问题，因为放在了static文件下，所以还要多退一级目录

```
window.top.location.href = '../../page/login/login.html'  
} else {
```

知识点

mybatisplus

mybatisplus----只需要接口继承父类接口就行

mapper接口继承BaseMapper

service接口继承IService

service实现类继承ServiceImpl<mapper,dao>并且实现service接口

函数介绍

lambdaQuery--eq里面写查询条件

//查询数据库是否右这个用户名

```
LambdaQueryWrapper<Employee> queryWrapper = new LambdaQueryWrapper<>();
queryWrapper.eq(Employee::getUsername, "admin");
Employee one = empService.getOne(queryWrapper);
```

注意@Service这些注解是标在实现类上而不是接口上的

html知识

这个方法绑定了登陆按钮

```
methods: {
  async handleLogin() {
    this.$refs.loginForm.validate(async (valid) => {
      if (valid) {
        this.loading = true
        let res = await loginApi(this.loginForm)
        if (String(res.code) === '1') {
          localStorage.setItem('userInfo', JSON.stringify(res.data))
          window.location.href= '../index.html'
        } else {
          this.$message.error(res.msg)
          this.loading = false
        }
      }
    })
  }
}
```

对应的js

```
function loginApi(data) {
  return $axios({
    'url': '/employee/login',    control的url
    'method': 'post',
    data
  })
}
```

发给服务器的是

```
data() {
  return {
    loginForm: {
      username: 'admin',
      password: '123456'
    }
  }
}
```

是一个json

debug的时候改一下前端的超时时间

Md5工具类--DigestUtils

```
String md5 = DigestUtils.md5DigestAsHex(password.getBytes());
```

后台退出功能

思路速记

右上角展示当前用户

退出按钮的前端代码分析

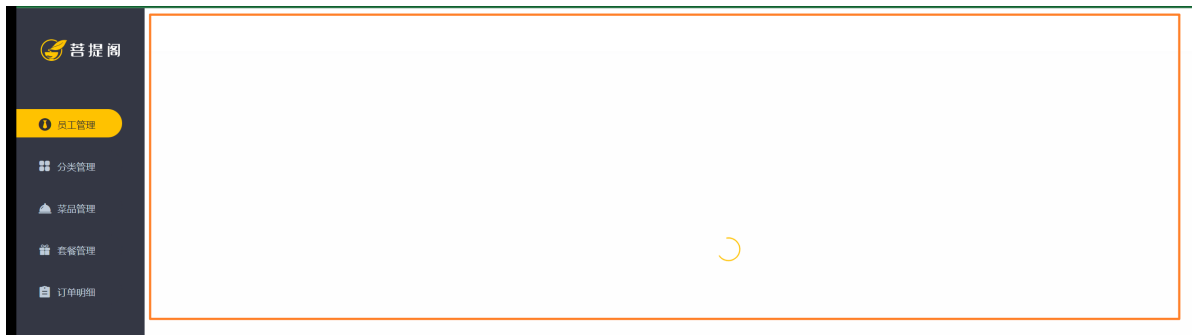
handler

清除缓存

HTML知识点

```
<iframe
  id="cIframe"
  class="c_iframe"
  name="cIframe"
  :src="iframeUrl"
  width="100%"
  height="auto"
  frameborder="0"
  v-show="!loading"
></iframe>
```

相当于留了一片空间用来动态展示src:src="iframeUrl"



利用v-for, v-if, v-else

```
<el-menu-item v-else :index="item.id"
@click="menuHandle(item,false)">
  <i class="iconfont" :class="item.icon"></i>
  <span slot="title">{{item.name}}</span>
</el-menu-item>
```

```
menuList: [
  // {
  //   id: '1',
  //   name: '门店管理',
  //   children: [
  //     {
  //       id: '2',
  //       name: '员工管理',
  //       url: 'page/member/list.html',
  //       icon: 'icon-member'
  //     },
  //   ],
  // },
```



绑定的方法

```
methods: {
  logout() {
    logoutApi().then((res)=>{
      if(res.code === 1){
        localStorage.removeItem('userInfo')
        window.location.href = 'page/login/login.html'
      }
    })
  },
}
```

对应的js
function logoutApi(){
return \$axios({


```

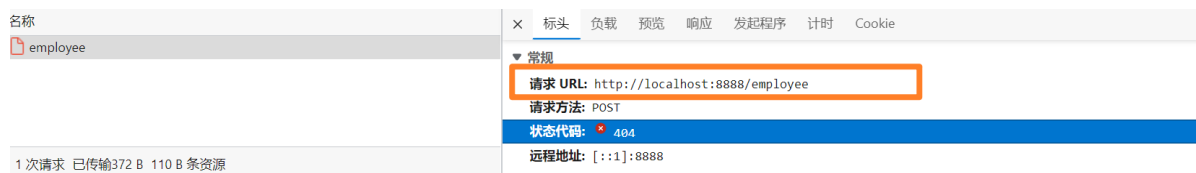
    'url': '/employee/logout', ----handlerurl
    'method': 'post',
  })
}

```

DAY-----2

新增员工功能

页面发送Ajax请求



```

addEmployee(params).then(res => {
    if (res.code === 1) { //-----返回这个就行
        this.$message.success('员工添加成功!')
        if (!st) {
            this.goBack()
        } else {
            this.ruleForm = {

```

handler接受data

保存数据库

页面没有设置密码，后端给一个默认的初始密码123456，MD5加密，完善employee对象里面的空字段

username唯一约束---所以重名报异常500（使用异常处理器）---注意这里也要@ResponseBody!!!
因为要响应json给浏览器

@controlleradvice

判断异常信息是否有sql异常信息关键字

截取重复的账号名字

返回错误信息

小知识点

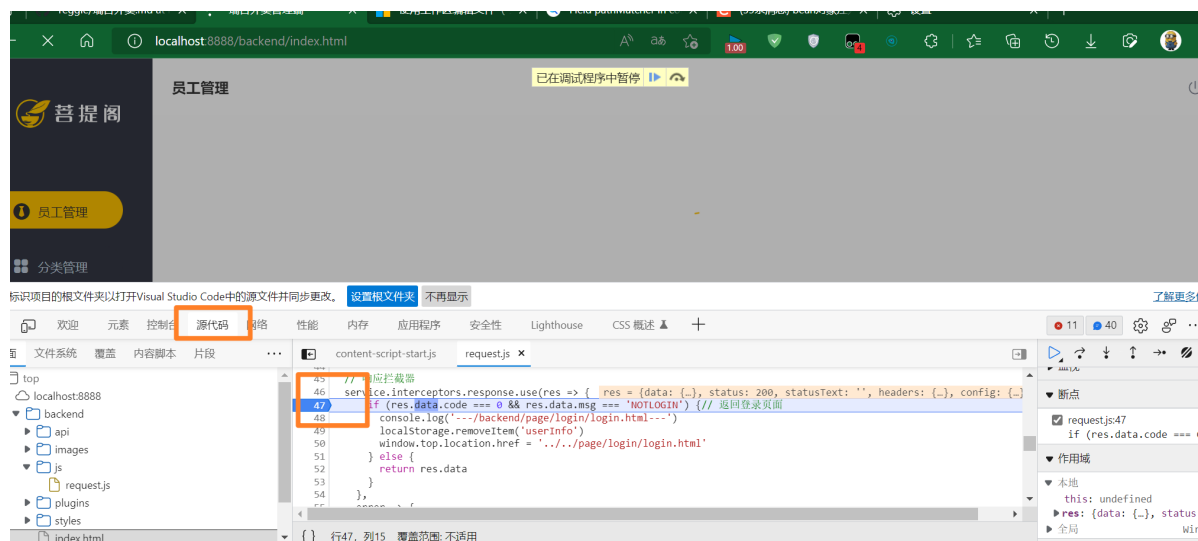
mybatisplus

----save方法-----insert功能

```
empService.save(employee);
```

html

js文件也能进行deug



员工信息分页查询

速记

搜索框

导航栏

ajax请求----接受参数---查询

getMemverList方法要求后端发的数据

mybatisplus分页插件-----配置分页插件的配置类----MybatisplusInceptor

paginationInnerinterceton

handler--uri--返回类型R《page》----接受参数

构造分页构造器

构造条件构造器

排序条件--orderByDesc----updateTime

执行---page ()

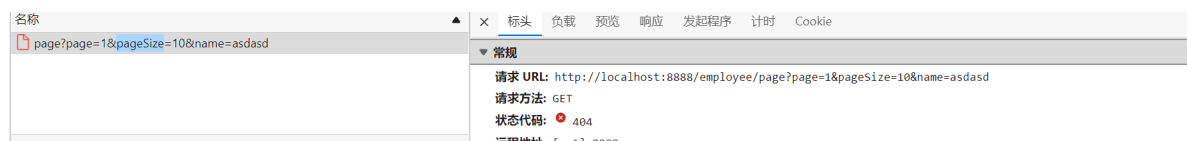
```
//构造分页构造器
Page pageInfo = new Page(page, pageSize); pageInfo: Page@7274 page: 1 pageSize: 10

//构造条件构造器
LambdaQueryWrapper<Employee> queryWrapper = new LambdaQueryWrapper();
//添加过滤条件
queryWrapper.like(StringUtils.isEmpty(name), Employee::getName, name);
```

html--修改导航条显示的数量用于测试

employee/page uri

参数-----int page,int pageSize,string name



```
await getMemberList(params).then(res => {
    if (String(res.code) === '1') {
        this.tableData = res.data.records || []
        this.counts = res.data.total
    }-----所以要返回一个page类型的数据
})
js
function getMemberList (params) {
    return $axios({
        url: '/employee/page',
        method: 'get',
        params
    })
}
```

知识点

mybatisplus

分页插件

配置类

```
@Configuration
//@MapperScan("com.hbx.reggie.mapper")-----如果mapper接口上面标注了@Mapper就不要在这里写了
public class MybatisPlusConfig {
    @Bean
    public MybatisPlusInterceptor mybatisPlusInterceptor(){
        MybatisPlusInterceptor interceptor = new MybatisPlusInterceptor();
        interceptor.addInnerInterceptor(new
        PaginationInnerInterceptor(DbType.MYSQL));
        return interceptor;
    }
}
```

```
Page pageInfo = new Page(page, pageSize);
//条件
LambdaQueryWrapper<Employee> employeeLambdaQueryWrapper = new
LambdaQueryWrapper<>();

employeeLambdaQueryWrapper.like(StringUtils.isNotEmpty(name), Employee::getName,
name);

employeeLambdaQueryWrapper.orderByDesc(Employee::getUpdateTime);
empService.page(pageInfo, employeeLambdaQueryWrapper);
```

html

共 3 条 4条/页 < 1 > 前往 1 页

1条/页

2条/页

3条/页

4条/页

```
<el-pagination
  class="pageList"
  :page-sizes="[1, 2, 3, 4]"
  :page-size="pageSize"
```

```
<script>
  new Vue({
    el: '#member-app',
    data() {
      return {
        input: '',
        counts: 0,
        page: 1,
        pageSize: 10,
        tableData : [],
        id : '',
        status : '',
      }
    }
  })

```

启用/禁用员工账号

速记

admin用户才能操作-----为什么---》前端控制的v-if

uri、参数-----》update 数据库 (status、updatetime、updateuser)

返回参数类型---string

maybatisplus----updateById ()

问题-----前端传过来的id有问题----js处理long形数据丢失精度-----》后端将id转换成字符串再发给前端

配置消息转换器---java--》json

jackson

```


    /**
     * @Override
     * protected void extendMessageConverters(List<HttpMessageConverter<?>> converters) {
     *     //创建消息转换器对象
     *     MappingJackson2HttpMessageConverter messageConverter = new MappingJackson2HttpMessageCon
     *     //设置对象转换器，底层使用Jackson将Java对象转为json
     *     messageConverter.setObjectMapper(new JacksonObjectMapper());
     *     //将上面的消息转换器对象追加到mvc框架的转换器集合中
     *     converters.add(index: 0, messageConverter);
     * }
     */

```

▼ 市税

请求 URL: http://localhost:8888/employee

请求方法: PUT

状态代码:  405

响应地址: http://localhost:8888/employee

×

标头

负载

预览

响应

发起程序

计时

Cookie

▼ 请求负载

查看源

▼ {id: 1542793745517342700, status: 0}

id: 1542793745517342700

status: 0

接受参数-----emp对象

```
//状态修改
statusHandle (row) {
  this.id = row.id
  this.status = row.status
  this.$confirm('确认调整该账号的状态?', '提示', {
    'confirmButtonText': '确定',
    'cancelButtonText': '取消',
    'type': 'warning'
  }).then(() => {
    enableOrDisableEmployee({ 'id': this.id, 'status': !this.status ? 1
: 0 }).then(res => {
      console.log('enableOrDisableEmployee',res)
      if (String(res.code) === '1') { ----->返回一个1就行
        this.$message.success('账号状态更改成功! ')
        this.handleQuery()
      }
    })
  })
}
```

知识点

js处理long型数据的会有精度缺失的问题

转换成字符串解决

自定义消息转化器

```
@Override
public void extendMessageConverters(List<HttpMessageConverter<?>>
converters) {
    MappingJackson2HttpMessageConverter mappingJackson2HttpMessageConverter
= new MappingJackson2HttpMessageConverter();
    mappingJackson2HttpMessageConverter.setObjectMapper(new
JacksonObjectMapper());
    converters.add(0,mappingJackson2HttpMessageConverter);
}
```

```
/**
 * 对象映射器:基于jackson将Java对象转为json, 或者将json转为Java对象
 * 将JSON解析为Java对象的过程称为 [从JSON反序列化Java对象]
 * 从Java对象生成JSON的过程称为 [序列化Java对象到JSON]
 */
public class JacksonObjectMapper extends ObjectMapper {

    public static final String DEFAULT_DATE_FORMAT = "yyyy-MM-dd";
    public static final String DEFAULT_DATE_TIME_FORMAT = "yyyy-MM-dd HH:mm:ss";
    public static final String DEFAULT_TIME_FORMAT = "HH:mm:ss";

    public JacksonObjectMapper() {
        super();
        //收到未知属性时不报异常
        this.configure(FAIL_ON_UNKNOWN_PROPERTIES, false);

        //反序列化时，属性不存在的兼容处理

        this.getDeserializationConfig().withoutFeatures(DeserializationFeature.FAIL_ON_
UNKNOWN_PROPERTIES);

        //日期格式化
        SimpleModule simpleModule = new SimpleModule()
            .addDeserializer(LocalDate.class, new
LocalDateDeserialzier(DateTimeFormatter.ofPattern(DEFAULT_DATE_TIME_FORMAT))
        )
            .addDeserializer(LocalDate.class, new
LocalDateDeserialzier(DateTimeFormatter.ofPattern(DEFAULT_DATE_FORMAT)))
            .addDeserializer(LocalTime.class, new
LocalTimeDeserialzier(DateTimeFormatter.ofPattern(DEFAULT_TIME_FORMAT)))
        //long、biginteger等类型转换成json
            .addSerializer(BigInteger.class, ToStringSerializer.instance)
            .addSerializer(Long.class, ToStringSerializer.instance)
            .addSerializer(LocalDate.class, new
LocalDateTimeSerializer(DateTimeFormatter.ofPattern(DEFAULT_DATE_TIME_FORMAT)))
    }
```

```

        .addSerializer(LocalDate.class, new
LocalDateSerializer(DateTimeFormatter.ofPattern(DEFAULT_DATE_FORMAT)))
        .addSerializer(LocalTime.class, new
LocalTimeSerializer(DateTimeFormatter.ofPattern(DEFAULT_TIME_FORMAT)));

    //注册功能模块 例如，可以添加自定义序列化器和反序列化器
    this.registerModule(simpleModule);
}
}

```

mbp

```
empService.updateById(emp);
```

会根据emp对象里面不为空的字符来进行更新

编辑员工信息

速记

编辑按钮绑定的函数-----uri、参数

新增和修改公用add页面

所以有两个handler---一个负责把当前id的信息查询返回给前端显示

▼ 常规

请求 URL: http://localhost:8888/employee/1542793745517342722

请求方法: GET

状态代码: ❌ 404

一个负责update (可以和禁用员工的handler共用)

▼ 常规

请求 URL: http://localhost:8888/employee

请求方法: PUT

状态代码: ✅ 200

远程地址: [::1]:8888

引用站点策略: strict-origin-when-cross-origin


```
▼ 用个实例 显示数据
▼ {name: "黄柏轩", phone: "18845678901", sex: "1", idNumber: "123456789012345678", username: "asdasd"}
  idNumber: "123456789012345678"
  name: "黄柏轩"
  phone: "18845678901"
  sex: "1"
  username: "asdasd"
```

```
created() {
    this.id = requestUrlParam('id')
    this.actionType = this.id ? 'edit' : 'add' -----共用
add页面，通过判断是否有id值来区分（修改的有id值）
    if (this.id) {
        this.init()
    }
},
mounted() {
},
methods: {
    async init () {
        queryEmployeeById(this.id).then(res => {
            console.log(res)
            if (String(res.code) === '1') {
                console.log(res.data)
                this.ruleForm = res.data
                this.ruleForm.sex = res.data.sex === '0' ? '女' : '男'
            }
        })
    }
}

//获取url地址上面的参数
function requestUrlParam(argname){
    var url = location.href
    var arrStr = url.substring(url.indexOf("?")+1).split("&")
    for(var i =0;i<arrStr.length;i++)
    {
        var loc = arrStr[i].indexOf(argname+"=")
        if(loc!=-1){
            return arrStr[i].replace(argname+"=", "").replace("?", "")
        }
    }
}
```

Day3

公共字段自动填充

速记

多表共用的字段，----利用mybatisplus 实现公共字段的自动填充

@TableField（公共属性上）

元数据对象处理器-----实现metaobjecthandler接口

setvalue方法

session的值怎么获得？ ----ThreadLocal

客户端发送的每次http请求，对应的在服务器端都会分配一个新的线程来处理

功能完善

在学习ThreadLocal之前，我们需要先确认一个事情，就是客户端发送的每次http请求，对应的在服务器端都会分配一个新的线程来处理，在处理过程中涉及到下面类中的方法都属于相同的一个线程：

- 1、LoginCheckFilter的doFilter方法
- 2、EmployeeController的update方法
- 3、MyMetaObjectHandler的updateFill方法

可以在上面的三个方法中分别加入下面代码（获取当前线程id）：

```
long id = Thread.currentThread().getId();  
log.info("线程id: {}", id);
```

执行编辑员工功能进行验证，通过观察控制台输出可以发现，一次请求对应的线程id是相同的：

```
2021-06-28 15:36:00.708 INFO 11972 --- [nio-8080-exec-7] c.i.reggie.filter.LoginCheckFilter : 线程id:31  
2021-06-28 15:36:00.711 INFO 11972 --- [nio-8080-exec-7] c.i.r.controller.EmployeeController : 线程id:31  
2021-06-28 15:36:00.711 INFO 11972 --- [nio-8080-exec-7] c.i.r.controller.EmployeeController : Employee(id=14078983:  
Creating a new SqlSession  
SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@17f9a154] was not registered for synchronization because  
2021-06-28 15:36:00.715 INFO 11972 --- [nio-8080-exec-7] c.i.reggie.common.MyMetaObjectHandler : 线程id:31
```

什么是ThreadLocal?

ThreadLocal并不是一个Thread，而是Thread的局部变量。当使用ThreadLocal维护变量时，ThreadLocal为每个使用该变量的线程提供独立的变量副本，所以每一个线程都可以独立地改变自己的副本，而不会影响到其它线程所对应的副本。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

我们可以在LoginCheckFilter的doFilter方法中获取当前登录用户id，并调用ThreadLocal的set方法来设置当前线程的线程局部变量的值（用户id），然后在MyMetaObjectHandler的updateFill方法中调用ThreadLocal的get方法来获得当前线程所对应的线程局部变量的值（用户id）。

封装threadlocal工具类basecontext-----封装set和get方法

在filter里面获得保存

知识点

mybatisplus

对于多张表都有的公共字段，每个表都要赋值，代码冗余-----mybatisplus提供了公共字段赋值的api

1、在公共属性是注解@TableField

```
//创建时间
@TableField(fill = FieldFill.INSERT)
private LocalDateTime createTime;

//更新时间
@TableField(fill = FieldFill.INSERT_UPDATE)
private LocalDateTime updateTime;
```

2、实现MetaObjectHandler接口。重写insertFill和updateFill方法

```
@Component
@Slf4j
public class commonfiled implements MetaObjectHandler {
    @Override
    public void insertFill(MetaObject metaObject) {
        log.info("insert start");
        metaObject.setValue("createTime", LocalDateTime.now());
        metaObject.setValue("updateTime", LocalDateTime.now());
        metaObject.setValue("createUser", BaseContext.getId());
        metaObject.setValue("updateUser", BaseContext.getId());
    }

    @Override
    public void updateFill(MetaObject metaObject) {
        log.info("update start");
        metaObject.setValue("updateTime", LocalDateTime.now());
        metaObject.setValue("updateUser", BaseContext.getId());
    }
}
```

ThreadLocal-----获取域对象

需求，公共字段赋值的时候需要从session中拿数据，但是没有request属性

客户端发送的每次http请求，对应的在服务器端都会分配一个新的线程来处理

功能完善

在学习ThreadLocal之前，我们需要先确认一个事情，就是客户端发送的每次http请求，对应的在服务器端都会分配一个新的线程来处理，在处理过程中涉及到下面类中的方法都属于相同的一个线程：

- 1、LoginCheckFilter的doFilter方法
- 2、EmployeeController的update方法
- 3、MyMetaObjectHandler的updateFill方法

可以在上面的三个方法中分别加入下面代码（获取当前线程id）：

```
long id = Thread.currentThread().getId();  
log.info("线程id: {}", id);
```

执行编辑员工功能进行验证，通过观察控制台输出可以发现，一次请求对应的线程id是相同的：

```
2021-06-28 15:36:00.708 INFO 11972 --- [nio-8080-ex] c.i.reggie.filter.LoginCheckFilter : 线程id:31  
2021-06-28 15:36:00.711 INFO 11972 --- [nio-8080-ex] c.i.r.controller.EmployeeController : 线程id:31  
2021-06-28 15:36:00.711 INFO 11972 --- [nio-8080-ex] c.i.r.controller.EmployeeController : Employee(id=14078983,  
Creating a new SqlSession  
SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@17f9a154] was not registered for synchronization because  
2021-06-28 15:36:00.715 INFO 11972 --- [nio-8080-ex] c.i.reggie.common.MyMetaObjectHandler : 线程id:31
```

什么是ThreadLocal?

ThreadLocal并不是一个Thread，而是Thread的局部变量。当使用ThreadLocal维护变量时，ThreadLocal为每个使用该变量的线程提供独立的变量副本，所以每一个线程都可以独立地改变自己的副本，而不会影响其它线程所对应的副本。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

我们可以在LoginCheckFilter的doFilter方法中获取当前登录用户id，并调用ThreadLocal的set方法来设置当前线程的线程局部变量的值（用户id），然后在MyMetaObjectHandler的updateFill方法中调用ThreadLocal的get方法来获得当前线程所对应的线程局部变量的值（用户id）。

封装的工具类

```
public class BaseContext {  
    private static final ThreadLocal<Long> threadLocal = new ThreadLocal<>();  
    public static void setId(Long id){  
        threadLocal.set(id);  
    }  
  
    public static Long getId(){  
        return threadLocal.get();  
    }  
}
```

新增分类

category表---name---unique

pojo--mapper-----service----controller

第一个页面



General

Request URL: http://localhost:8888/category

Request Method: POST

Status Code: 404

套餐分类、菜品分类---共用

返回参数、uri、接受参数

```
▼ {name: "1231231", type: "2", sort: "1123"}
  name: "1231231"
  sort: "1123"
```

```
addCategory({'name': classData.name,'type':this.type, sort:
classData.sort}).then(res => {
  console.log(res)
  if (res.code === 1) {
    this.$message.success('分类添加成功! ')
    if (!st) {
```

unique字段---重复添加---异常---全局的异常处理器捕获

分类信息查询

Request URL: http://localhost:8888/category/page?page=1&pageSize=10

Request Method: GET

设置一个排序的条件构造器

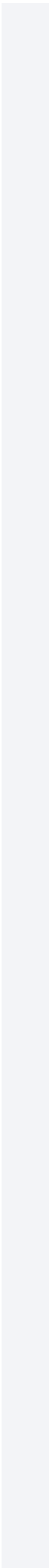
问题

```
==> Preparing: SELECT id,type,name,sort,create_time,update_time,create_user,update_user,is_deleted FROM category LIMIT ?
==> Parameters: 10(Long)
Closing non transactional SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@718c079d]
2022-07-03 12:44:12.662 ERROR 25292 --- [nio-8888-exec-2] o.a.c.c.C.[[/]/[dispatcherServlet]]: Servlet.service() for s
### Error querying database. Cause: java.sql.SQLException: Unknown column 'is_deleted' in 'field list'
```

数据库少了个字段

删除分类

操作
修改 删除
修改 删除
修改 删除
修改 删除
修改 删除
修改 删除
修改 删除
修改 删除
修改 删除



注意---不是简单的删除

如果当前分类有关联的菜品，不允许删除

General

Request URL: http://localhost:8888/category?id=1397844263642378242

Request Method: DELETE

Status Code: 405

dish和setmeal的pojo

关联的字段----categoryId

重写mybatisplus的remove方法

条件构造器

count>0?

不能删除-----自定义一个异常 (Runtimeexception)

-----全局异常处理捕获这个异常

-----将错误信息返回给页面

自定义remove

service接口定义

实现类实现方法

知识点

删除---不再是普通的删除，要考虑是否影响其他表的数据

mbp

自定义sql功能

和mybatis一样，在自己的service接口定义方法，在实现类实现这个方法，再把mbp提供的方法应用到这个方法里面

如：自定义删除功能

```
public interface CateService extends IService<Category> {  
    void remove(Long id);  
}
```



```

@Service
public class CateServiceImp extends ServiceImp<CateMapper, Category> implements
CateService{
    @Autowired
    private DishService dishService;
    @Autowired
    private SetMealService setMealService;
    @Override
    public void remove(Long id) {

        //判断是否有菜品关联
        LambdaQueryWrapper<Dish> dishLambdaQueryWrapper = new
LambdaQueryWrapper<>();
        dishLambdaQueryWrapper.eq(Dish::getCategoryId,id);
        int count = dishService.count(dishLambdaQueryWrapper);
        if(count>0){
            throw new AssociateException("有菜品关联，不允许删除");
        }
        //判断是否有套餐关联
        LambdaQueryWrapper<Setmeal> setmealLambdaQueryWrapper = new
LambdaQueryWrapper<>();
        SetmealLambdaQueryWrapper.eq(Setmeal::getCategoryId,id);
        int count1 = setMealService.count(setmealLambdaQueryWrapper);
        if(count1>0){
            throw new AssociateException("有套餐关联，不允许删除");
        }
        //没有关联，允许删除
        removeById(id);
    }
}

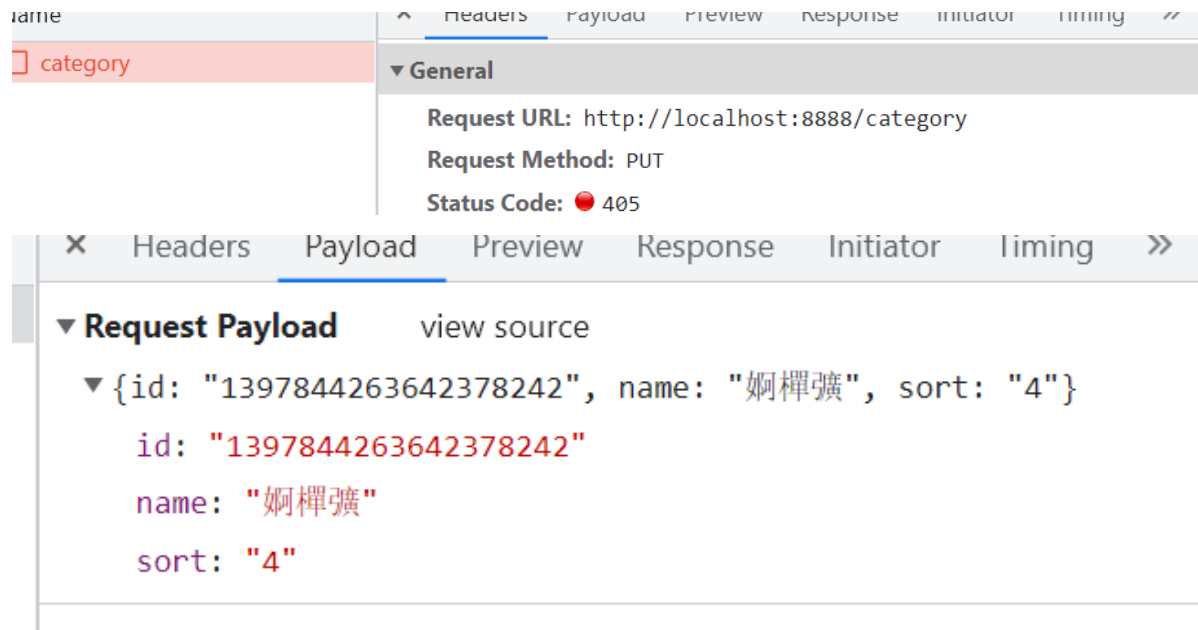
```

修改分类

前端完成了回显功能----怎么实现的？

```
@click="editHandle(scope.row)"
```

```
editHandle(dat) {  
  this.classData.title = '修改分类'  
  this.action = 'edit'  
  this.classData.name = dat.name  
  this.classData.sort = dat.sort  
  this.classData.id = dat.id  
  this.classData.dialogVisible = true  
},
```



```
if (reg.test(this.classData.sort)) {  
  editCategory({'id':this.classData.id,'name': this.classData.name,  
sort: this.classData.sort}).then(res => {  
    if (res.code === 1) {  
      this.$message.success('分类修改成功! ')  
    }  
  })  
}
```

返回参数---string

Day-4---菜品管理

菜品管理							管理员
<input type="text" value="请输入菜品名称"/>							批量删除 批量启售 批量停售 新增菜品
<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"/> 霸王别姬		湘菜	¥ 128	启售	2021-05-27 09:43:08	修改	停售 删除
<input type="checkbox"/> 东安子鸡		湘菜	¥ 88	启售	2021-05-27 09:42:16	修改	停售 删除
<input type="checkbox"/> 绝味鱼翅		湘菜	¥ 48	启售	2021-05-27 09:41:19	修改	停售 删除
<input type="checkbox"/> 毛氏红烧肉		湘菜	¥ 68	启售	2021-05-27 09:40:19	修改	停售 删除
<input type="checkbox"/> 永州血鸭		湘菜	¥ 88	启售	2021-05-27 09:39:30	修改	停售 删除
<input type="checkbox"/> 辣子鸡		湘菜	¥ 78	启售	2021-05-27 09:38:43	修改	停售 删除
<input type="checkbox"/> 剁椒鱼头		湘菜	¥ 68	启售	2021-05-27 09:37:27	修改	停售 删除
共 57 条 10条/页 < 1 2 3 4 5 6 > 前往 6 页							

一下功能自己实现

批量删除

批量启售

批量停售

文件上传和下载

上传

文件上传对于form表单的要求

引入upload文件用于测试功能

▼ General

Request URL: http://localhost:8888/common/upload

Request Method: POST

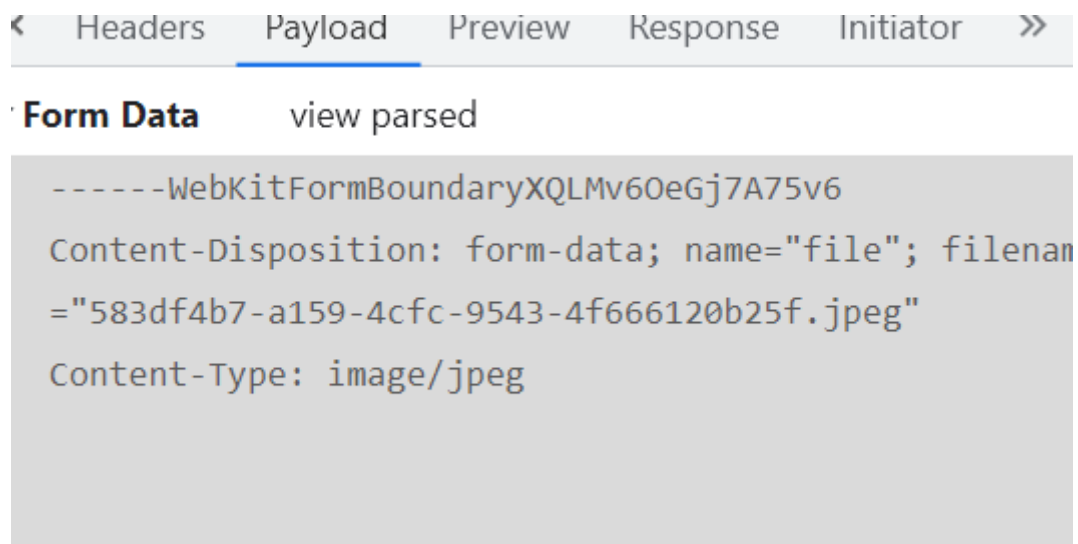
Status Code: ● 404

Remote Address: [::1]:8888

×
Headers
Payload
Preview
Response
Initiator
>>

▼ **Form Data**
view source
view decoded

file: (binary)



handler参数的名字（测试一把两个注解）

转存的位置

--配置文件指定目录（path属性） ---判断目录是否存在

--el表达式

文件名

--uuid

--获取后缀名---substring

返回值？----返回文件的名称（因为要保存到数据库里面）

下载

形式

附件形式

直接在浏览器打开（因为要展示，所以用这种形式）

文件下载，页面端可以使用标签展示下载的图片



General
Request URL: http://localhost:8888/common/download?name=E:\reggie_upload_photos\1ae04c7c-6e5f-4d7e-9f3f-81e389520582.jpg
Request Method: GET
Status Code: 400
Remote Address: [::1]:8888
Referrer Policy: strict-origin-when-cross-origin

返回值--void Get请求

直接写给浏览器

知识点

自定义变量

```
reggie:
  path: E:\reggie_upload_photos\
```

el表达式获取

```
@Value("${reggie.path}")
private String BasePath;
```

上传文件后直接显示的html模板

```
<div class="addBrand-container" id="food-add-app">
  <div class="container">
    <el-upload class="avatar-uploader"
      action="/common/upload"
      :show-file-list="false"
      :on-success="handleAvatarSuccess"
      <!--这里文件上传成功就调用handleAvatarSuccess方法向服务器发请求，服务直接把数据写给浏览器就行-->
      :before-upload="beforeUpload"
      ref="upload">
      </img>
      <i v-else class="el-icon-plus avatar-uploader-icon"></i>
    </el-upload>
  </div>
</div>
```

```
methods: {  
    handleAvatarSuccess (response, file, fileList) {  
        this.imageUrl = `/common/download?name=${response.data}` -----这  
        里拼接的是上传时候服务器返回的文件名字  
    },  
}
```

上传返回给浏览器文件名的时候不能返回全路径

d?name=E:\reggie_upload_photos\d4e3fb8f-7c6f-4e45-91bf-21104f03

只能返回文件名

oad?name=cd6f4660-84e8-4459-a0f5-390339e54a19.jpeg

不然会报错

```
java.lang.IllegalArgumentException: Invalid character found in the request target  
[/common/download?name=E:\reggie_upload_photos\d4e3fb8f-7c6f-4e45-91bf-  
21104f03eeb4.jpg]. The valid characters are defined in RFC 7230 and RFC 3986
```

习惯---存图片到数据库的时候一般保存图片的名字就行

新增菜品

dish

dish_flavor(菜品口味表)

观察表结构

对应的pojo、mapper、ser、contr (同一由于dish操控)

+ 新建菜品

先发了一个ajax请求

Request URL: http://localhost:8888/category/list?type=1

Request Method: GET

Status Code: 404

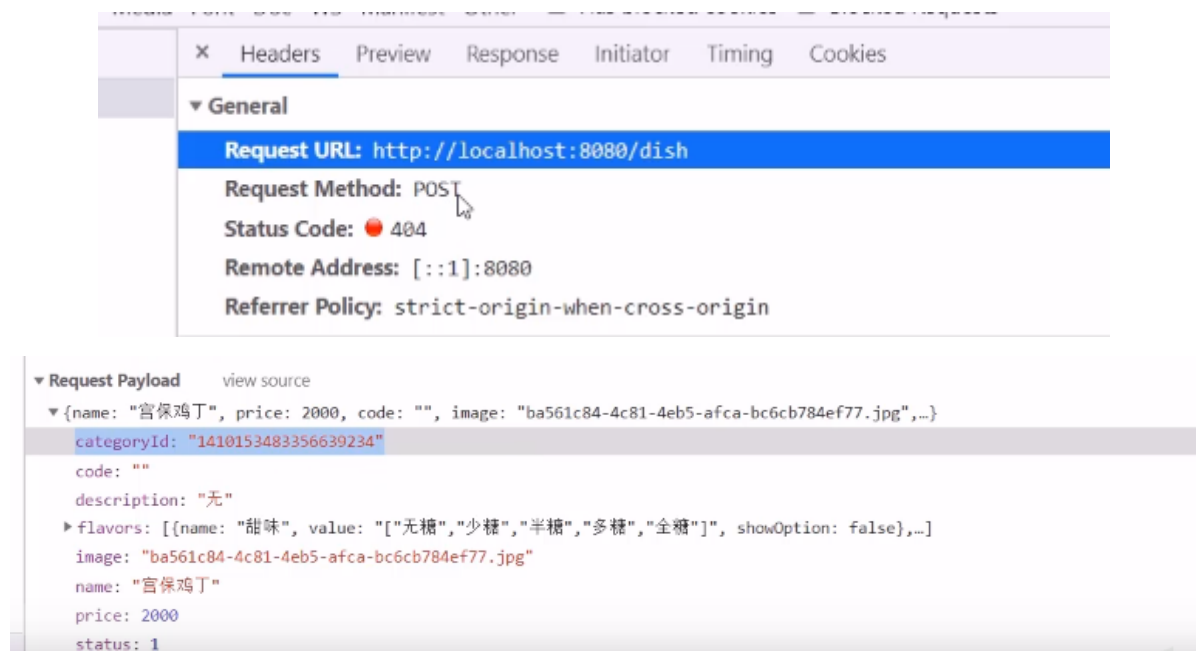
获得菜品类型

参数? --category

list方法

上传图片

保存



参数---? flavors怎么接? ----导入新的类DishDTO---继承dish

DTO:

注意事项

DTO, 全称为Data Transfer Object, 即数据传输对象, 一般用于展示层与服务层之间的数据传输。

为什么要使用DTO---- (实体类和接受参数不对应的时候用)

自定义mapper方法

同时操作两张表

dishid赋给dishflavor

stream api和lambda

//菜品口味

```
List<DishFlavor> flavors = dishDto.getFlavors();
flavors = flavors.stream().map((item) -> {
    item.setDishId(dishId);
    return item;
}).collect(Collectors.toList());
```

事务控制---因为两张表---保持一致性

开始事务

```
INSERT INTO dish ( id, name, category_id, price, code, image, description, status, create_time, update_time, create_user, upd
```

```
this.save(dishDto);
```

save方法绑定pojo

父类是--即可

知识点

mybatisplus-

-查询多条数据

list方法

```
List<Category> list = cateService.list(categoryLambdaQuerywrapper);
```

方法的作用范围

```
@Autowired
private DishFlavorService dishFlavorService;
@Override
public void saveDishAndFlavor(DishDto dishDto) {
    this.save(dishDto);
    Long dishid = dishDto.getId();
    List<DishFlavor> flavors = dishDto.getFlavors();
```

因为this是DishService的实现类，他已经绑定了Dish实体类，所以在执行insert语句的时候只会插入Dish的字段，只要参数是Dish的子类都能通用----这为使用DTO接受参数带来基础

DTO--DATA TRANSFER OBJECT

往往用在多表的操作中

应用场景，客户端发过来的数据和实体类并不能对应

这个时候就可以构造DTO类负责接受参数

套路：

```
public class DishDto extends Dish { //继承对应的实体类

    //构造一个List，泛型为另一张表的实体类
    private List<DishFlavor> flavors = new ArrayList<>();

    //后面可以自己定义额外的属性
    private String categoryName;
```

注意，涉及到多表别忘记加上事务管理---保证一致性

步骤1、注解

2、开启事务

菜品信息分页查询

Request URL: http://localhost:8888/dish/page?page=1&pageSize=10&name=dasd

Request Method: GET

Status Code: 404

Remote Address: [::1]:8888

难点---图片和分类怎么展示

图片---文件下载

分类----新增泛型使用DTO

对象拷贝--BeanUtils.copyproperties ignores属性

```
//对象拷贝
BeanUtils.copyProperties(pageInfo, dishDtoPage, ... ignoreProperties: "records");

List<Dish> records = pageInfo.getRecords();

List<DishDto> list = records.stream().map((item) -> {
    DishDto dishDto = new DishDto();

    BeanUtils.copyProperties(item, dishDto);

    Long categoryId = item.getCategoryId(); //分类id
    //根据id查询分类对象
    Category category = categoryService.getById(categoryId);
    String categoryName = category.getName();
    dishDto.setCategoryName(categoryName);

    return dishDto;
}).collect(Collectors.toList());

dishDtoPage.setRecords(list);
```

知识点 多表---dto

复制--beanutils

知识点

利用工具类拷贝属性

```
BeanUtils.copyProperties(dish, dishDto);
//注意第一个参数是被copy的对象
```

常用

```
public static void copyProperties(Object source, Object target, String...
ignoreProperties){} //第三个参数指定那些熟悉不拷贝
public static void copyProperties(Object source, Object target) throws
BeansException {
    copyProperties(source, target, (Class)null, (String[])null);
}
```

修改菜品

信息回显

Request URL: http://localhost:8888/dish/1397849739276890114

Request Method: GET

Status Code: 🚫 404

返回泛型----dto

接受参数---路径参数

自定义方法---查两张表 (dish、dishflavor)

利用dto

修改

前端新增和修改公用一个页面

两张表同时更新

自定义方法

口味数据

先清除再添加---这样可以用之前的代码逻辑（插入），不需要写update逻辑了

remove方法