# 项目介绍

## 项目开发过程

1. 需求分析：需求规格说明书
2. 概要设计说明书
3. 详细设计说明书
4. 开发过程：团队协作
5. 测试报告

## 功能需求

### 后台管理系统

* 省市县管理
* 地址管理
* 用户管理
* 管理员管理
* 支付方式管理
* 轮播图管理
* 商品类别管理
* 商品管理
* 购物车管理
* 订单管理
* 订单状态管理
* 管理员登录

### 前端网页

* 用户注册
* 用户登录
* 商品类别展示
* 商品展示：分页
* 商品查询：主查询框，根据商品名称或简单简介模糊查询
* 分类查询商品
* 加入购物车
* 查看购物车
* 生成订单
* 查看自己的订单
* 在线支付

## 技术架构

|  |  |  |
| --- | --- | --- |
| 框架 | | Spring+SpringMVC+MyBatis |
| 页面展示 | | JSP、JSTL、EL |
| 数据的传递 | | MVC模式 |
| 三层开发模型 | | 表现层+业务逻辑层+数据访问层 |
| 数据库 | | MySQL |
| 持久层 | 数据库连接池 | HikariCP |
| 持久层框架 | MyBatis+tk.mybatis |
| 分页 | pagehelper |
| 事务 | Spring-tx |
| 日志 | | Log4j2 |
| 数据校验 | | * 前端：jquery-validator * 后端：Hibernate-validator |
| Json数据处理 | | Jackson |
| 代码生成 | | Lombok |
| 数据类型转换 | | MapStruct |
| 单元测试 | | Junit5 |
| AOP | | AspectJ |
| 文件（图片）服务器 | | Minio |

# 技术

## IDEA快捷键

* 快速生成代码：Alt+insert
* 自动生成代码：alt+enter

## 技术相关博客

* 命令设计模式：

<https://blog.csdn.net/lianghecai52171314/article/details/102563085>

* 建造者设计模式

<https://hcshow.blog.csdn.net/article/details/102548116>

* intellij常用优化与设置汇总：

<https://hcshow.blog.csdn.net/article/details/116761325>

* intellij常用插件汇总：

<https://hcshow.blog.csdn.net/article/details/105633426>

* intellij创建Maven项目：

<https://blog.csdn.net/lianghecai52171314/article/details/102907427>

* tomcat启动控制台中文乱码

<https://blog.csdn.net/lianghecai52171314/article/details/103164879>

* 常见电商模式

<https://hcshow.blog.csdn.net/article/details/121795624>

* JRebel插件

<https://hcshow.blog.csdn.net/article/details/105637251>

* lombok简介

<https://blog.csdn.net/lianghecai52171314/article/details/105921598>

* lombok常用注解

<https://blog.csdn.net/lianghecai52171314/article/details/105408173>

* Mapstruct

<https://blog.csdn.net/lianghecai52171314/article/details/121332412>

* MyBatisCodeHelperPro

<https://blog.csdn.net/lianghecai52171314/article/details/105511508>

* 动态SQL

<https://blog.csdn.net/lianghecai52171314/article/details/105611742>

* 使用MBG生成MyBatis代码

<https://blog.csdn.net/lianghecai52171314/article/details/105610045>

* MySQL 中的批量操作

<https://hcshow.blog.csdn.net/article/details/130491538>

* JSTL

<https://blog.csdn.net/lianghecai52171314/article/details/103009357>

* Json语法

<https://blog.csdn.net/lianghecai52171314/article/details/102600262>

* jackson

<https://blog.csdn.net/lianghecai52171314/article/details/102587931>

* JSP页面引入技术（了解）

<https://blog.csdn.net/lianghecai52171314/article/details/102960239>

* JSTL 自定义标签（了解）

<https://blog.csdn.net/lianghecai52171314/article/details/125214591>

* 请求转发与重定向（重点）

<https://blog.csdn.net/lianghecai52171314/article/details/102576175>

* redirect重定向Controller中方法之间传递参数

<https://blog.csdn.net/lianghecai52171314/article/details/107707731>

* AOP概念（重点了解）

<https://blog.csdn.net/lianghecai52171314/article/details/124402394>

* generateO2O

<https://blog.csdn.net/lianghecai52171314/article/details/103832278>

* Servlet监听器

<https://blog.csdn.net/lianghecai52171314/article/details/103410934>

* windows下安装Minio

<https://blog.csdn.net/lianghecai52171314/article/details/129920439>

* Minio工具类

<https://blog.csdn.net/lianghecai52171314/article/details/129920363>

* Alibaba Java Coding Guidelines

<https://blog.csdn.net/lianghecai52171314/article/details/103083470>

* Intellij代码自动生成技巧

https://hcshow.blog.csdn.net/article/details/103832177

* mybatis缓存机制详解

<https://blog.csdn.net/lianghecai52171314/article/details/121968433>

* 解决SpringMVC中@ResponseBody返回中文乱码

<https://blog.csdn.net/lianghecai52171314/article/details/131720531>

## 接口

接口是用来定义游戏规则的

### 语法：

|  |
| --- |
| public interface 接口名{  常量  抽象方法  静态方法  默认方法  私有方法  } |

### 示例一：

|  |
| --- |
| public interface Person {  *//常量* public static final Integer *PAGE\_SIZE* =16;  *//抽象方法* public abstract void fun1();  *//静态方法* public static void fun2(){  System.*out*.println("静态方法");  }  *//默认方法* public default void fun3(){  System.*out*.println("默认方法");  }  *//私有方法* private void fun4(){  System.*out*.println("私有方法");  } } |

## 类

类是用来按照游戏规则具体做事情的

### 公式：

|  |
| --- |
| public class类名{  成员变量  静态变量  构造方法  成员方法  静态方法  构造代码块  静态代码块  } |

### 示例一：

|  |
| --- |
| public class Employee {  *//成员变量* private Long empno;  private String ename;  private String job;  *//静态变量* private static int *aa*;   *//构造方法：用来创建对象 重载* public Employee() {  }  public Employee(Long empno, String ename, String job) {  this.empno = empno;  this.ename = ename;  this.job = job;  }   *//普通方法* public void fun1(){   }  *//静态方法* public static void fun2(){   }   *//构造代码块* {   }  *//静态代码块* static {   } } |

### 示例二：

成员变量/成员方法，只能通过对象调用

静态变量/静态方法，可以通过对象和类名调用，但强烈建议通过类名调用

|  |
| --- |
| public class Employee {  private int a1;  private void fun1(){  }   private static int *a2*;  private static void fun2(){  }   public static void main(String[] args) {  *//类名可以调用静态变量和静态方法* System.*out*.println(Employee.*a2*);  Employee.*fun2*();   *// 对象可以调用成员变量和成员方法* Employee obj = new Employee();  System.*out*.println(obj.a1);  obj.fun1();  }  } |

### 示例三：静态代码块和构造代码块

* 程序一执行，就会先执行静态代码块，静态代码块永远只调用一次。
* 创建对象时会先执行构造代码块，每创建一次对象就会执行一次构造代码块

|  |
| --- |
| public class Employee {  {  System.*out*.println("构造代码块");  }  public Employee() {  System.*out*.println("默认构造方法");  }  static {  System.*out*.println("静态代码块");  }  public static void main(String[] args) {  new Employee();  System.*out*.println("main方法");  } } |

示例：

|  |
| --- |
| public class Employee {  public static int *a* =88;   {  System.*out*.println("构造代码块");  }  static {  System.*out*.println("静态代码块");  }   public static void main(String[] args) {  System.*out*.println(Employee.*a*);  System.*out*.println(Employee.*a*);  System.*out*.println(Employee.*a*);   new Employee();  new Employee();  new Employee();  } } |

## 匿名内部类

用来创建只使用一次的对象

### 语法：

创建匿名内部类对象：

|  |
| --- |
| new 父类名/接口名() {  重写的方法  新定义的方法  } |

一般父类是抽象类

### 示例一：

* 父类

|  |
| --- |
| public abstract class A {  public abstract void fun(); } |

* 测试类

|  |
| --- |
| public class DemoTest {  public static void main(String[] args) {  A obj = new A(){  @Override  public void fun() {  System.*out*.println("\*\*");  }  };  obj.fun();  } } |

### 示例二：

接口

|  |
| --- |
| public interface BB {  public abstract void fun(); } |

测试类

|  |
| --- |
| public class DemoTest2 {  public static void main(String[] args) {  new BB() {  @Override  public void fun() {  System.*out*.println(33+44);  }  }.fun();  } } |

## Lambda表达式

Lambda表达式是对只有一个抽象方法的匿名内部类的简写形式。

### 注解@FunctionalInterface

用来标注一个接口中只能有一个抽象方法

### 语法：

|  |
| --- |
| (数据类型1 变量1，数据类型2 变量2……) -> {  //…… 代码  } |

规则：

* 数据类型可以省略

|  |
| --- |
| (变量1，变量2……) -> {  //…… 代码  } |

* 如果大括号中的代码只有一条语句，大括号可以省略

|  |
| --- |
| (变量1，变量2……) -> 代码; |

* 如果大括号中的代码只有一条语句，且这条语句是return，那么大括号和return语句可以省略

|  |
| --- |
| (变量1，变量2……) -> 代码; |

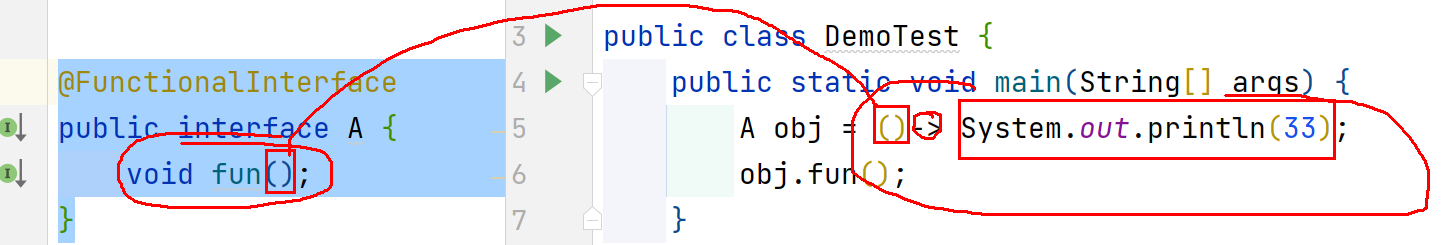
### 示例一：

接口：

|  |
| --- |
| @FunctionalInterface public interface A {  void fun(); } |

测试代码：

|  |
| --- |
| public class DemoTest {  public static void main(String[] args) {  A obj = ()-> System.*out*.println(33);  obj.fun();  } } |



### 示例二：

接口

|  |
| --- |
| public interface A {  int fun(Integer a,Integer b,Integer c); } |

测试代码

|  |
| --- |
| public class DemoTest {  public static void main(String[] args) {  A obj = new A(){  @Override  public int fun(Integer a, Integer b, Integer c) {  int tmp1 = a+b+c;  int tmp2 = a\*b\*c;  return tmp1+tmp2;  }  };  int res1 = obj.fun(33, 44, 55);  System.*out*.println(res1);    A obj2 = (Integer a, Integer b, Integer c) -> {  int tmp1 = a+b+c;  int tmp2 = a\*b\*c;  return tmp1+tmp2;  };  int res2 = obj2.fun(33, 44, 55);  System.*out*.println(res2);    A obj3 = new A(){  @Override  public int fun(Integer a, Integer b, Integer c) {  return a+b+c;  }  };  int res3 = obj3.fun(33, 44, 55);  System.*out*.println(res3);   A obj4 = (Integer a, Integer b, Integer c) -> {  return a+b+c;  };  System.*out*.println(obj4.fun(33, 44, 55));   A obj5 = ( a, b, c) -> {  return a+b+c;  };  System.*out*.println(obj5.fun(33, 44, 55));   A obj6 = ( a, b, c) -> a+b+c;  System.*out*.println(obj6.fun(33, 44, 55));  } } |

## 综合示例

图书管理系统实现对图片的增删改查

### 实体类

|  |
| --- |
| public class Book {  private String isbn;  private String name;  private String auth;  private BigDecimal price;   *//构造方法* public Book() { *//默认构造方法* }  public Book(String isbn, String name, String auth, BigDecimal price) {*//全参构造方法* this.isbn = isbn;  this.name = name;  this.auth = auth;  this.price = price;  }  *//getter/setter* public String getIsbn() {  return isbn;  }   public void setIsbn(String isbn) {  this.isbn = isbn;  }   public String getName() {  return name;  }   public void setName(String name) {  this.name = name;  }   public String getAuth() {  return auth;  }   public void setAuth(String auth) {  this.auth = auth;  }   public BigDecimal getPrice() {  return price;  }   public void setPrice(BigDecimal price) {  this.price = price;  }   *//toString* @Override  public String toString() {  return "Book{" +  "isbn='" + isbn + '\'' +  ", name='" + name + '\'' +  ", auth='" + auth + '\'' +  ", price=" + price +  '}';  } } |

### Dao接口

|  |
| --- |
| public interface BookDao { *//data access object* /\*\*  \* 添加数据  \* @param *book* \* @return  \*/  int insert(Book book);   /\*\*  \* 修改数据  \* @param *book* \* @return  \*/  int update(Book book);   /\*\*  \* 根据主键查询  \* @param *book* \* @return  \*/  Book selectByPrimaryKey(Book book);   /\*\*  \* 查找一共有多少条数据  \* @return  \*/  long selectCount();   /\*\*  \* 查询所有的数据  \* @return  \*/  List<Book> selectList(); } |

### Dao实现类

|  |
| --- |
| public class BookDaoImpl implements BookDao{   @Override  public int insert(Book book) {  return 0;  }   @Override  public int update(Book book) {  return 0;  }   @Override  public Book selectByPrimaryKey(Book book) {  return null;  }   @Override  public long selectCount() {  return 0;  }   @Override  public List<Book> selectList() {  return null;  } } |

## 执行一百万次的for循环需要多少时间

|  |
| --- |
| public class Demo1 {  *//执行一百万次的for循环需要多少时间* public static void main(String[] args) {  *//获取系统当前的时间* long start = System.*currentTimeMillis*();   for (int i = 0; i < 100000; i++) {  *//System.out.println("\*\*\*\*");* }   long end = System.*currentTimeMillis*();   System.*out*.println(end - start);  } } |

## 命令设计模式：求任意一段代码的执行时间

### 模板

|  |
| --- |
| public abstract class CalculateTime {   public abstract void fun();   public long cal(){  long start = System.*nanoTime*();  fun();  long end = System.*nanoTime*();  return end - start;  } } |

### 测试一

|  |
| --- |
| class AA extends CalculateTime {  @Override  public void fun() {  for (int i = 0; i < 10000; i++) {  System.*out*.println("\*\*\*");  }  } } public class Demo1 {  public static void main(String[] args) {  AA a = new AA();  long res = a.cal();  System.*out*.println(res);  } } |

### 测试二

|  |
| --- |
| class BB extends CalculateTime{  @Override  public void fun() {  try {  Thread.*sleep*(3000);  } catch (InterruptedException e) {  e.printStackTrace();  }  } } public class Demo1 {  public static void main(String[] args) {  BB b = new BB();  long res2 = b.cal();  System.*out*.println(res2);  } } |

## 基于内部类实现：命令设计模式

模板

|  |
| --- |
| public abstract class CalculateTime {   public abstract void fun();   public long cal(){  long start = System.*nanoTime*();  fun();  long end = System.*nanoTime*();  return end - start;  } } |

测试

|  |
| --- |
| public class Demo1 {  public static void main(String[] args) {  CalculateTime obj = new CalculateTime(){  @Override  public void fun() {  for (int i = 0; i < 10000; i++) {  System.*out*.println("\*\*");  }  }  };  long res = obj.cal();  System.*out*.println(res);  } } |

## 基于Lambda实现：命令设计模式

接口

|  |
| --- |
| @FunctionalInterface public interface CalculateTime {  void fun();   default long cal() {  long start = System.*nanoTime*();  fun();  long end = System.*nanoTime*();  return end - start;  } } |

测试

|  |
| --- |
| public class Demo1 {  public static void main(String[] args) {  CalculateTime obj = ()->{  for (int i = 0; i < 10000; i++) {  System.*out*.println("##");  }  };  System.*out*.println(obj.cal());  } } |

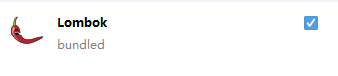
## Lombok

用来生成代码

第一步：

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

第二步：安装插件



第三步： 示例

|  |
| --- |
| @Getter @Setter @ToString @Builder @NoArgsConstructor @AllArgsConstructor public class Province {  private Long id;  private String name;  private String area;  private Integer priority;  private Integer status;  private LocalDateTime createTime;  private LocalDateTime updateTime;    public static void main(String[] args) {  Province province = Province.*builder*()  .id(1234L)  .name("aa")  .area("abc")  .priority(33)  .status(1)  .createTime(LocalDateTime.*now*())  .updateTime(LocalDateTime.*now*())  .build();  System.*out*.println(province);  } } |

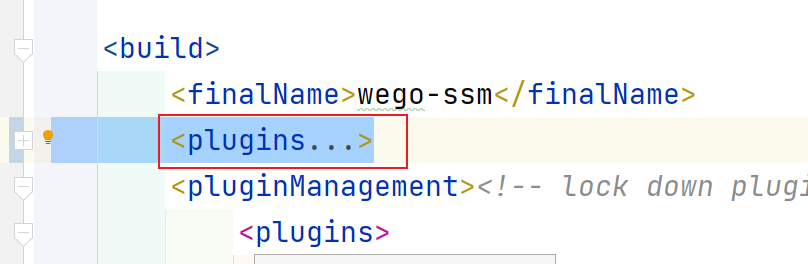
## Mapstruct

数据类型转换的

第一步：添加依赖

|  |
| --- |
| <dependency>  <groupId>org.mapstruct</groupId>  <artifactId>mapstruct</artifactId>  <version>1.5.5.Final</version> </dependency> |

第二步：在pom.xml中添加配置



|  |
| --- |
| <plugins>  *<!--MapStruct-->* <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <version>3.8.0</version>  <configuration>  *<!-- 解决JDK版本自动切换问题-->* <source>17</source>*<!-- 源代码开发版本 -->* <target>17</target>*<!-- java编译版本 -->* <encoding>UTF8</encoding> *<!-- 项目的编码 -->* <annotationProcessorPaths>  <path>  <groupId>org.mapstruct</groupId>  <artifactId>mapstruct-processor</artifactId>  <version>1.5.5.Final</version>  </path>  <path>  <groupId>org.projectlombok</groupId>  <artifactId>lombok</artifactId>  <version>1.18.20</version>  </path>  <path>  <groupId>org.projectlombok</groupId>  <artifactId>lombok-mapstruct-binding</artifactId>  <version>0.2.0</version>  </path>  </annotationProcessorPaths>  </configuration>  </plugin> </plugins> |

第三步：安装插件



第四步：示例

* 原类

|  |
| --- |
| @Getter @Setter @Builder @ToString @NoArgsConstructor @AllArgsConstructor public class Goods implements Serializable {  /\*\*  \* 编号  \*/  private Long id;   /\*\*  \* 商品名称  \*/  private String name;   /\*\*  \* 主图  \*/  private String pic;   /\*\*  \* 图集  \*/  private String imgs;   /\*\*  \* 市场价格  \*/  private BigDecimal price1;   /\*\*  \* 会员价格  \*/  private BigDecimal price2;   /\*\*  \* 单位  \*/  private String unit;  /\*\*  \* 库存  \*/  private Integer storage;   /\*\*  \* 卖点  \*/  private String sellingPoint;   /\*\*  \* 显示优先级  \*/  private Integer priority;   /\*\*  \* 类别编号  \*/  private Long categoryId;   /\*\*  \* 商品规格  \*/  private String specs;   /\*\*  \* 简介  \*/  private String info;   /\*\*  \* 状态：1上架 2下架  \*/  private Integer status;   /\*\*  \* 创建时间  \*/  private LocalDateTime createTime;   /\*\*  \* 更新时间  \*/  private LocalDateTime updateTime; } |

* 待转换成的类

|  |
| --- |
| @Getter @Setter @Builder @ToString @NoArgsConstructor @AllArgsConstructor public class GoodsVO {  /\*\*  \* 编号  \*/  private Long id;   /\*\*  \* 商品名称  \*/  private String name;   /\*\*  \* 主图  \*/  private String pic;   /\*\*  \* 市场价格  \*/  private BigDecimal price1;   /\*\*  \* 会员价格  \*/  private BigDecimal price2;   /\*\*  \* 单位  \*/  private String unit;   /\*\*  \* 卖点  \*/  private String sellingPoint;  } |

* 接口

|  |
| --- |
| @Mapper public interface GoodsConverter {  GoodsConverter *INSTANCE* = Mappers.*getMapper*(GoodsConverter.class);   GoodsVO goods2GoodsVO(Goods goods); } |

* 测试代码

|  |
| --- |
| public static void main(String[] args) {  Goods goods = new Goods(88L,"apple","apple.jpg","a1.jpg,a2.jpg,a2.jpg",new BigDecimal(88),new BigDecimal(22),"个",100,"好吃不贵",32,11L,"color:red,size:33","hahaah",2, LocalDateTime.*now*(),LocalDateTime.*now*());   GoodsVO goodsVO = GoodsConverter.*INSTANCE*.goods2GoodsVO(goods);  System.*out*.println(goodsVO); } |

## Spring

### 常用注解

@Component：

@Service：service

@Repository：dao

@Autowired：让Spring自动注入

### IOC

依赖注入，简单说就是让Spring给我们生成对象。

#### 示例一

第一步：创建实体类：JavaBean

|  |
| --- |
| @Getter @Setter @Builder @ToString @NoArgsConstructor @AllArgsConstructor public class Province {  private Long id;  private String name;  private String area;  private Integer priority;  private Integer status;  private LocalDateTime createTime;  private LocalDateTime updateTime; } |

第二步：在resources目录下创建配置文件applicationContext.xml，并在其中配置JavaBean：

|  |
| --- |
| *<?*xml version="1.0" encoding="UTF-8"*?>* <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd">  <bean id="province" class="com.wego.entity.domain.Province">  <property name="id" value="1001"/>  <property name="name" value="aa"/>  </bean> </beans> |

第三步：使用：

|  |
| --- |
| public static void main(String[] args) {  *//之前代码* Province province = new Province();  System.*out*.println(province);   *//IOC* ApplicationContext act = new ClassPathXmlApplicationContext("applicationContext.xml");  Province p2 = (Province) act.getBean("province");  System.*out*.println(p2); } |

#### 示例二

* applicationContext.xml中添加自动扫描

|  |
| --- |
| <context:component-scan base-package="com.wego"/> |

* Dao接口

|  |
| --- |
| public interface ProvinceDao {  int insert(Province province);   int update(Province province);   int deleteById(Province province);   Province selectById(Long id);   List<Province> selectList(); } |

* Dao接口实现类

|  |
| --- |
| @Repository public class ProvinceDaoImpl implements ProvinceDao {  @Override  public int insert(Province province) {  System.*out*.println("insert");  return 0;  }  @Override  public int update(Province province) {  System.*out*.println("update");  return 0;  }   @Override  public int deleteById(Province province) {  System.*out*.println("deleteById");  return 0;  }   @Override  public Province selectById(Long id) {  System.*out*.println("selectById");  return null;  }   @Override  public List<Province> selectList() {  System.*out*.println("selectList");  return null;  } } |

* 测试代码

|  |
| --- |
| public static void main(String[] args) {  ApplicationContext act = new ClassPathXmlApplicationContext("applicationContext.xml");  ProvinceDao provinceDao = (ProvinceDao) act.getBean("provinceDaoImpl");  System.*out*.println(provinceDao);} |

#### 示例三

* Serivce接口

|  |
| --- |
| public interface ProvinceService {  int insert(Province province);   int update(Province province);   int deleteById(Province province);   Province selectById(Long id);   List<Province> selectList(); } |

* Service接口实现类

|  |
| --- |
| @Service public class ProvinceServiceImpl implements ProvinceService {   @Autowired  private ProvinceDao provinceDao;   @Override  public int insert(Province province) {  int res = provinceDao.insert(province);  return res;  }   @Override  public int update(Province province) {  int res = provinceDao.update(province);  return res;  }   @Override  public int deleteById(Province province) {  return 0;  }   @Override  public Province selectById(Long id) {  return null;  }   @Override  public List<Province> selectList() {  return null;  } } |

* 测试代码

|  |
| --- |
| public static void main(String[] args) {  ApplicationContext act = new ClassPathXmlApplicationContext("applicationContext.xml");   ProvinceService provinceService = (ProvinceService) act.getBean("provinceServiceImpl");  provinceService.update(null); } |

### AOP（现阶段会写代码就可以了）

面向切面编程

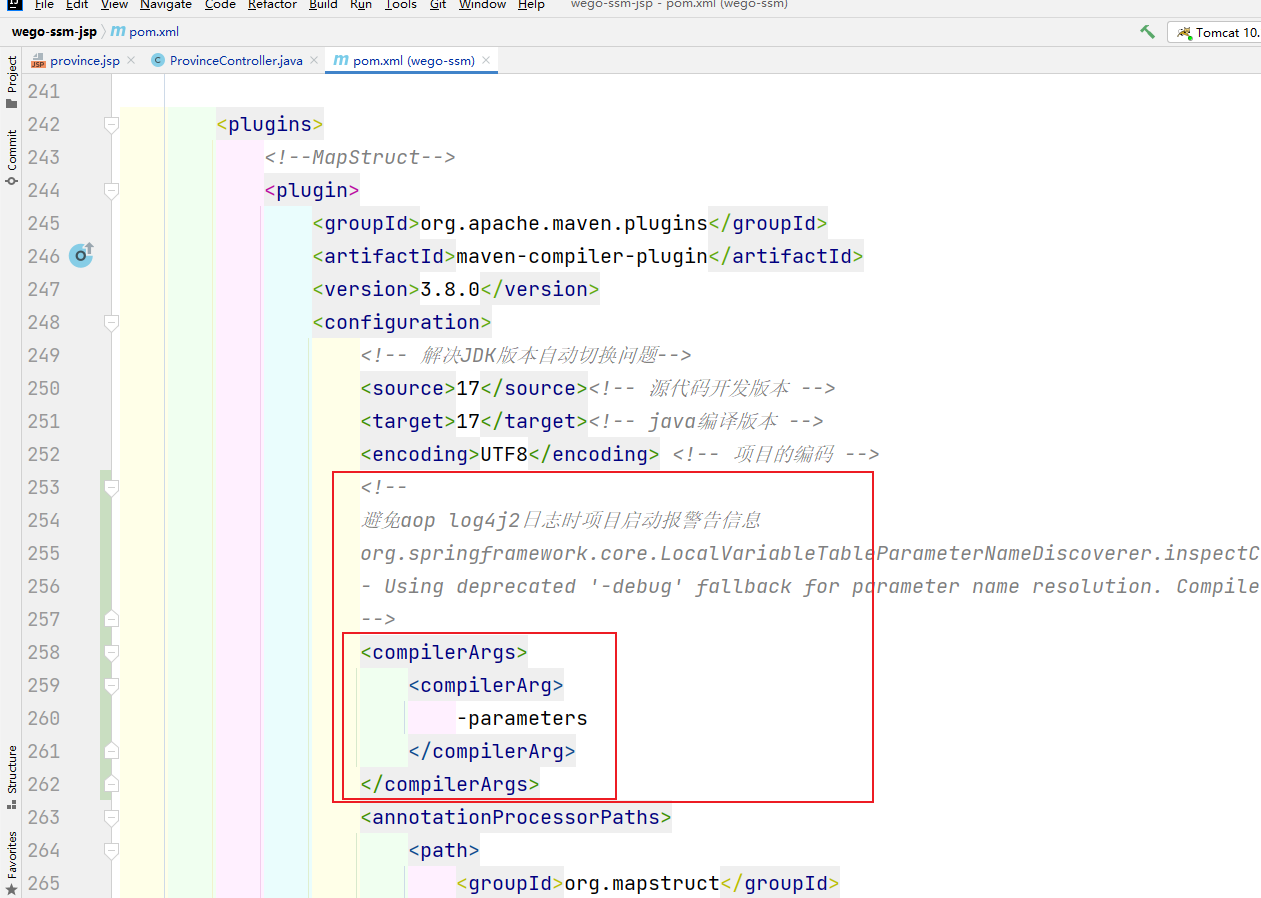
基于注解的日志

#### 示例：aop入门

第一步：添加依赖

|  |
| --- |
| *<!--AOP相关-->* <**dependency**>  <**groupId**>org.aspectj</**groupId**>  <**artifactId**>aspectjrt</**artifactId**>  <**version**>1.9.19</**version**> </**dependency**> <**dependency**>  <**groupId**>org.aspectj</**groupId**>  <**artifactId**>aspectjtools</**artifactId**>  <**version**>1.9.19</**version**> </**dependency**> <**dependency**>  <**groupId**>org.aspectj</**groupId**>  <**artifactId**>aspectjweaver</**artifactId**>  <**version**>1.9.19</**version**> </**dependency**> <**dependency**>  <**groupId**>aopalliance</**groupId**>  <**artifactId**>aopalliance</**artifactId**>  <**version**>1.0</**version**> </**dependency**> *<!--Spring和AspectJ整合中间件-->* <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-aspects</**artifactId**>  <**version**>6.0.9</**version**> </**dependency**> *<!--AOP相关-->* |

消除系统启动时的警告：



第二步：在spring-context中开启aop，添加如下所示配置信息：

|  |
| --- |
| *<!-- 启动对@AspectJ注解的支持 -->* <**aop:aspectj-autoproxy**/> |

第三步：在com.wego.service下创建功能待增强的接口和实现类：

* Demo.java

|  |
| --- |
| **public interface** Demo {  **public int** fun(); } |

* DemoImpl.java

|  |
| --- |
| **public class** DemoImpl **implements** Demo {   @Override  **public int** fun(){  System.***out***.println(**"业务……"**);  **return** 3;  }  } |

第四步：创建AOP类：

|  |
| --- |
| @Aspect @Component **public class** AopDemo {   *//前置增强  //对com.wego.service包及其子包下的，任意类，任意方法名，任意参数，任意返回值，进行功能增强* @Before(**"execution(\* com.wego.service..\*.\*(..))"**)  **public void** before(JoinPoint joinPoint){  String s = joinPoint.getTarget().getClass().getName()  + **"."** + joinPoint.getSignature().getName()  +**"()"**;   System.***out***.println(**"日志：开始执行方法....."** +s);  }   *//后置增强* @After(**"execution(\* com.wego.service..\*.\*(..))"**)  **public void** after(JoinPoint joinPoint){  String s = joinPoint.getTarget().getClass().getName()  + **"."** + joinPoint.getSignature().getName()  +**"()"**;   System.***out***.println(**"日志：方法执行结束....."** +s);  }   *//返回值增强* @AfterReturning(value = **"execution(\* com.wego.service..\*.\*(..))"** ,returning = **"res"**)  **public void** returning(Integer res){  System.***out***.println(**"返回值增强 "**+res);  }   @AfterThrowing(value = **"execution(\* com.wego.service..\*.\*(..))"**,throwing = **"ex"**)  **public void** throwing(Exception ex){  System.***out***.println(**"异常增强 "**+ex);  }   *//环绕增强* @Around(**"execution(\* com.wego.service..\*.\*(..))"**)  **public** Object around(ProceedingJoinPoint joinPoint) {  System.***out***.println(**"前置增强"**);  Object result = **null**;  **try** {  result = joinPoint.proceed();  System.***out***.println(**"返回值增强"** + result);  } **catch** (Throwable e) {  System.***out***.println(**"异常增强 "** + e);  } **finally** {  System.***out***.println(**"后置增强"**);  }  **return** result;  }  } |

第五步：单元测试

|  |
| --- |
| @ExtendWith(SpringExtension.**class**) @ContextConfiguration({**"/spring-context.xml"**, **"/spring-mybatis.xml"**}) **public class** DemoTest {   @Autowired  **private** Demo **demo**;   @Test  **void** fun(){  **demo**.fun();  }  } |

#### 示例：基于AOP的Controller层统一日志

第一步：在spring-mvc.xml中添加：

|  |
| --- |
| *<!-- 启动对@AspectJ注解的支持 -->* <**aop:aspectj-autoproxy**/> |

第二步：AOP类

|  |
| --- |
| @Slf4j @Aspect @Component **public class** LogAspect {  /\*\*  \* 切入点：设置方法会在com.wego.controller包及其子包下的、任意访问控制修饰符修饰的、返回值为任意类型的、包含任何多个参数的、任意方法、执行之前执行  \*/  @Pointcut(**"execution(\* com.wego.controller..\*.\*(..))"**)  **public void** fun() {  }  /\*\*  \* 环绕通知  \* 第一个参数可以定义为org.aspectj.lang.ProceedingJoinPoint类型  \* **@param *joinPoint*** \* **@return** \* **@throws** Throwable  \*/  @Around(**"fun()"**)  **public** Object doAround(ProceedingJoinPoint joinPoint) **throws** Throwable {  Object result = **null**;   *//取得类名和方法名* **final** String fullMethodName = getFullMethodName(joinPoint);   *//相当于前置通知* ***log***.info(fullMethodName+**"方法开始执行"**);   Object[] args= joinPoint.getArgs();  ***log***.info(**"参数信息为：{} "**,args);   **try** {  result = joinPoint.proceed();  ***log***.info(**"结果为：{}"**+result);  } **catch** (Throwable ex) {  *//相当于异常抛出后通知* StackTraceElement stackTraceElement= ex.getStackTrace()[ex.getStackTrace().**length**-1];   ***log***.error (**"{}方法的{}行"**,fullMethodName,stackTraceElement.getLineNumber()+**"抛出异常"**);  ***log***.error(**"异常信息为：{} "**,ex.fillInStackTrace().toString());  **throw** ex;  }**finally**{  *//相当于最终通知* ***log***.info(fullMethodName+**"方法执行结束"**);  }   **return** result;  }   **private** String getFullMethodName(JoinPoint joinPoint) {  **return** joinPoint.getTarget().getClass().getName()  + **"."** + joinPoint.getSignature().getName()  +**"()"**;  } } |

## MyBatis

操作数据库

### 示例：Spring整合MyBatis

第一步：添加依赖，同时在pom.xml的build标签下面添加：

|  |
| --- |
| <resources>  <resource>  <directory>src/main/java</directory>  <includes>  <include>\*\*/\*.xml</include>  </includes>  </resource> </resources> |

第二步：MyBatis配置文件

|  |
| --- |
| *<?*xml version="1.0" encoding="UTF-8" *?>* <!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-config.dtd"*>* <configuration>  *<!--日志-->* <settings>  <setting name="logImpl" value="STDOUT\_LOGGING"/>  </settings>  *<!--分页-->* <plugins>  <plugin interceptor="com.github.pagehelper.PageInterceptor">  <property name="helperDialect" value="mysql"/>  </plugin>  </plugins> </configuration> |

第三步：在applicationContext.xml文件中添加

|  |
| --- |
| *<!-- 配置数据源 -->* <bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">  <property name="driverClassName">  <value>com.mysql.cj.jdbc.Driver</value>  </property>  <property name="url">  <value>jdbc:mysql://localhost:3306/db\_wego\_simple?useSSL=false&amp;serverTimezone= Asia/Shanghai</value>  </property>  <property name="username"><value>root</value></property>  <property name="password"><value>root</value></property> </bean>  *<!-- 创建SqlSessionFactory，同时指定数据源-->* <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">  <property name="dataSource" ref="ds" />  *<!-- 自动扫描mappers.xml文件-->  <!-- 自动扫描mappers.xml文件-->* <property name="mapperLocations" value="classpath\*:mapper/\*.xml"/>  <property name="configLocation" value="classpath:mybatis.xml"/> </bean>  *<!-- 注册Mapper -->* <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">  <property name="basePackage" value="com.wego.mapper" /> </bean> |

第四步：在包com.wego.mapper下创建ProvinceMapper.java

|  |
| --- |
| public interface ProvinceMapper {  int insert(Province province);   int update(Province province);   int deleteById(Province province);   Province selectById(Long id);   List<Province> selectList(); } |

第五步：在resources目录下创建子文件夹mapper，然后在mapper文件夹中创建ProvinceMppaer.xml：

|  |
| --- |
| *<?*xml version="1.0" encoding="UTF-8"*?>* <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd"*>* <mapper namespace="com.wego.mapper.ProvinceMapper">   <select id="selectById" parameterType="long" resultType="com.wego.entity.domain.Province">  select \* from tb\_province where id =${id}  </select>  </mapper> |

第六步：测试

|  |
| --- |
| @ExtendWith(SpringExtension.class) @ContextConfiguration("/applicationContext.xml") class ProvinceMapperTest {  @Autowired  private ProvinceMapper provinceMapper;   @Test  void selectById() {  Province province = provinceMapper.selectById(11L);  System.*out*.println(province);  }  } |

### 示例：Service

第一步：修改ProvinceMapper，在接口上面添加注解@Mapper

|  |
| --- |
| import org.apache.ibatis.annotations.Mapper;  @Mapper public interface ProvinceMapper {  } |

第二步：修改ProvinceServiceImpl，在其中采用ProvinceMapper操作数据库

|  |
| --- |
| @Service public class ProvinceServiceImpl implements ProvinceService {  @Autowired  private ProvinceMapper provinceMapper;   @Override  public Province selectById(Long id) {  Province province = provinceMapper.selectById(id);  return province;  }  } |

第三步：测试

|  |
| --- |
| @ExtendWith(SpringExtension.class) @ContextConfiguration("/applicationContext.xml") class ProvinceServiceImplTest {   @Autowired  private ProvinceService provinceService;   @Test  void selectById() {  Province province = provinceService.selectById(12L);  System.*out*.println(province);  }  } |

### 示例：ProvinceMapper.xml完整代码

|  |
| --- |
| *<?*xml version="1.0" encoding="UTF-8"*?>* <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd"*>* <mapper namespace="com.wego.mapper.ProvinceMapper">   <resultMap id="baseResultMap" type="com.wego.entity.domain.Province">  <id column="id" property="id"/>  <result column="name" property="name"/>  <result column="area" property="area"/>  <result column="priority" property="priority"/>  <result column="status" property="status"/>  <result column="create\_time" property="createTime"/>  <result column="update\_time" property="updateTime"/>  </resultMap>   <sql id="baseColumn">  id,name,area,priority,status,create\_time,update\_time  </sql>   <select id="selectById" parameterType="long" resultMap="baseResultMap">  select  <include refid="baseColumn"/>  from tb\_province where id =#{id}  </select>   <insert id="insert" parameterType="com.wego.entity.domain.Province">  insert into tb\_province  (name,area,priority,status,create\_time,update\_time)  values  (#{name},#{area},#{priority},#{status},#{createTime},#{updateTime});  </insert>   <update id="update" parameterType="com.wego.entity.domain.Province">  update tb\_province  set name = #{name},area=#{area},priority=#{priority},status = #{status},create\_time = #{createTime},update\_time=#{updateTime}  where id=#{id}  </update>   <delete id="deleteById">  delete from tb\_province where id=#{id}  </delete>   <select id="selectList" resultMap="baseResultMap">  select  <include refid="baseColumn"/>  from tb\_province;  </select> </mapper> |

### 分页

#### 语法

|  |
| --- |
| select \* from 表名  limit (pageNum-1)\*pageSize, pageSize |

#### 代码

* 用来放置分页信息的Bean

|  |
| --- |
| @Getter @Setter @NoArgsConstructor **public class** PageBean<T> {  /\*\*  \* 每页显示的条数  \*/  **private** Integer **pageSize**;  /\*\*  \* 当前的页码  \*/  **private** Integer **pageNum**;  /\*\*  \* 一共有多少条记录  \*/  **private** Long **total**;  /\*\*  \* 一共有多少页  \*/  **private** Integer **pages**;  /\*\*  \* 每一页所显示的数据  \*/  **private** List<T> **result**;   /\*\*  \* 分页请求路径  \*/  **private** String **url**;   /\*\*  \* 将MyBatis的Page对象转换成我们自定义的PageBean对象  \*  \* **@param *page*** \*/  **public** PageBean(Page<T> page) {  **this**.**pageSize** = page.getPageSize();  **this**.**pageNum** = page.getPageNum();  **this**.**total** = page.getTotal();  *//将Long转换成Integer类型* **this**.**pages** = page.getPages();  **this**.**result** = page.getResult();  } } |

* 在ProvinceService接口中添加方法声明

|  |
| --- |
| PageBean<Province> selectPage(ProvinceQuery provinceQuery); |

* ProvicneServiceImpl中提供分页的代码实现

|  |
| --- |
| **public** PageBean<Province> selectPage(ProvinceQuery provinceQuery) {  *//设置分页信息* Page<Province> page = PageHelper.*startPage*(provinceQuery.getPageNum(), provinceQuery.getPageSize());  *//查询数据* **provinceMapper**.selectList(provinceQuery);  *//将MyBatis提供的Page对象转换成我们自己的PageBean对象* PageBean<Province> pageBean = **new** PageBean<>(page);  **return** pageBean; } |

* 测试

|  |
| --- |
| @Test **void** selectPage(){  ProvinceQuery provinceQuery = **new** ProvinceQuery();  provinceQuery.setPageNum(5);  provinceQuery.setPageSize(3);  *//provinceQuery.setStatus(1);* **provinceService**.selectPage(provinceQuery); } |

### Spring集成tk.mybatis

第一步：添加依赖

|  |
| --- |
| <**dependency**>  <**groupId**>tk.mybatis</**groupId**>  <**artifactId**>mapper</**artifactId**>  <**version**>4.2.3</**version**> </**dependency**> |

第二步：修改spring-mybatis.xml

|  |
| --- |
| *<!-- 注册Mapper -->* <bean class="**tk**.mybatis.spring.mapper.MapperScannerConfigurer">  <property name="basePackage" value="com.wego.mapper"/> </bean> |

第三步：修改mybatis配置文件，在其中指定mapper接口默认包

|  |
| --- |
| <**mappers**>  <**package name="com.wego.mapper"**/> </**mappers**> |

第四步：实体类

|  |
| --- |
| @Getter @Setter @Builder @ToString @AllArgsConstructor @NoArgsConstructor @Table(name = **"tb\_admin"**) **public class** Admin **implements** Serializable {  /\*\*  \* 编号  \*/  @Id  @GeneratedValue(generator = **"JDBC"**)  **private** Long **id**;   /\*\*  \* 昵称  \*/  **private** String **nickname**;   /\*\*  \* 登录名  \*/  **private** String **account**;   /\*\*  \* 登录密码  \*/  **private** String **password**;   /\*\*  \* 用户头像  \*/  **private** String **avatar**;   /\*\*  \* 电话  \*/  **private** String **phone**;   /\*\*  \* 邮箱  \*/  **private** String **email**;   /\*\*  \* QQ  \*/  **private** String **qq**;   /\*\*  \* 微信  \*/  **private** String **wechat**;   /\*\*  \* 简介  \*/  **private** String **intro**;   /\*\*  \* 状态  \*/  **private** Integer **state**;   /\*\*  \* 创建时间  \*/  @Column(name = **"create\_time"**)  **private** LocalDateTime **createTime**;   /\*\*  \* 更新时间  \*/  **private** LocalDateTime **updateTime**; } |

第五步：Mapper接口

|  |
| --- |
| **public interface** AdminMapper **extends** Mapper<Admin> {  } |

第六步：测试

|  |
| --- |
| @ExtendWith(SpringExtension.**class**) @ContextConfiguration(**"/spring-mybatis.xml"**) **class** AdminMapperTest {  @Autowired  **private** AdminMapper **adminMapper**;   @Test  **void** f() {  **final** Admin admin = **adminMapper**.selectByPrimaryKey(1001L);  System.***out***.println(admin);  } } |

## SpringMVC

业务流程 ~ servlet

### 示例：spring整合SpringMVC

第一步：依赖，不能添加以下依赖，加上就报错

|  |
| --- |
| <**dependency**>  <**groupId**>javax</**groupId**>  <**artifactId**>javaee-api</**artifactId**>  <**version**>8.0.1</**version**> </**dependency**> <**dependency**>  <**groupId**>javax.servlet</**groupId**>  <**artifactId**>javax.servlet-api</**artifactId**>  <**version**>4.0.1</**version**> </**dependency**> |

第二步：在com.wego.controller包下创建ProvinceController

|  |
| --- |
| @Controller @RequestMapping(**"/province"**) **public class** ProvinceCtroller {  *//localhost/wego/province/getProvinceById* @RequestMapping(**"/getProvinceById"**)  **public** String getProvinceById() {  **return "aa"**;  } } |

第三步：修改applicationContext.xml,在中添加：

|  |
| --- |
| <**mvc:annotation-driven**/> *<!-- 让SpringMVC不拦截静态资源-->* <**mvc:default-servlet-handler**/>  *<!-- 启动自动扫描：使用注解开发，不用配置controller。该句必须有，否则在web.xml的  DispatcherServlet中不能指定springmvc.xml，而必须指定applicationContext.xml -->* <**context:component-scan base-package="com.wego.controller"**/>  *<!-- 视图解析器 ，可以显式设置，也可以不设置，不设置会依据SpringMVC的默认设置-->* <**bean id="viewResolver"  class="org.springframework.web.servlet.view.InternalResourceViewResolver"**>  <**property name="prefix" value="/WEB-INF/"**/>  <**property name="suffix" value=".jsp"**/> </**bean**> |

注意：在添加这些代码时，idea生成的xml头是错误的，需要修改将tools-mvc：

|  |
| --- |
| <**beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:context="http://www.springframework.org/schema/context"  xmlns:mvc="http://www.springframework.org/schema/mvc"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/context  https://www.springframework.org/schema/context/spring-context.xsd  http://www.springframework.org/schema/mvc  http://www.springframework.org/schema/mvc/spring-mvc.xsd"**> |

第四步：在WEB-INF目录下创建aa.jsp文件

|  |
| --- |
| <%@ **page contentType**="**text/html;charset=UTF-8**" **language**="**java**" %> <**html**> <**head**>  <**title**>Title</**title**> </**head**> <**body**> aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa </**body**> </**html**> |

第五步：修改web.xml文件：

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns="http://xmlns.jcp.org/xml/ns/javaee"  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_4\_0.xsd"  version="4.0"**>   *<!-- 配置SpringMVC核心控制器：DispatcherServlet主要负责流程的控制。-->* <**servlet**>  <**servlet-name**>SpringMVC</**servlet-name**>  <**servlet-class**>org.springframework.web.servlet.DispatcherServlet</**servlet-class**>  <**init-param**>  <**param-name**>contextConfigLocation</**param-name**>  <**param-value**>classpath:applicationContext.xml</**param-value**>  </**init-param**>  <**load-on-startup**>1</**load-on-startup**>  </**servlet**>  <**servlet-mapping**>  <**servlet-name**>SpringMVC</**servlet-name**>  <**url-pattern**>/</**url-pattern**>  </**servlet-mapping**>  </**web-app**> |

第六步：测试（必须使用tomcat10）

### 统一异常处理

第一步：自定义异常

|  |
| --- |
| @Getter @Setter @NoArgsConstructor @AllArgsConstructor **public class** GlobalException **extends** RuntimeException {  /\*\*  \* 异常错误编码  \*/  **private** Integer **code**;   /\*\*  \* 异常信息  \*/  **private** String **msg**;   /\*\*  \* 生成异常时页面的url  \*/  **private** String **url**;  } |

第二步：异常处理器

|  |
| --- |
| /\*\*  \* 全局异常处理器  \* **@author** hc  \*/ @ControllerAdvice **public class** GlobalExceptionHandler {   @ExceptionHandler(GlobalException.**class**)  **public** String viewExceptionHandler(RedirectAttributes redirectAttributes, GlobalException e) {  *//将异常信息设置如modelAndView* Notification notification = **new** Notification(**true**, e.getMsg());  redirectAttributes.addFlashAttribute(**"notification"**, notification);   **return "redirect:"**+e.getUrl();  }  } |

第三步：Controller中抛出异常

|  |
| --- |
| @GetMapping(**"/deleteById"**) **public** String deleteById(Long id, RedirectAttributes redirectAttributes) {  **try** {  **boolean** res = **provinceService**.deleteByPrimaryKey(id);  String msg = res ? **"删除成功！"** : **"删除失败"**;  Notification notification = **new** Notification(res, msg);  redirectAttributes.addFlashAttribute(**"notification"**, notification);  } **catch** (Exception e) {  e.printStackTrace();  **throw new** GlobalException(400,**"执行删除操作时发生错误"**,**"list"**);  }  **return "redirect:list"**; } |

## SSM配置文件拆分

### spring-context.xml

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:context="http://www.springframework.org/schema/context"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/context  https://www.springframework.org/schema/context/spring-context.xsd"**>   *<!--自动扫描-->* <**context:component-scan base-package="com.wego"**/>  </**beans**> |

### spring-mvc.xml

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:context="http://www.springframework.org/schema/context"  xmlns:mvc="http://www.springframework.org/schema/mvc"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/context  https://www.springframework.org/schema/context/spring-context.xsd  http://www.springframework.org/schema/mvc  http://www.springframework.org/schema/mvc/spring-mvc.xsd"**>   <**mvc:annotation-driven**/>  *<!-- 让SpringMVC不拦截静态资源-->* <**mvc:default-servlet-handler**/>   *<!-- 启动自动扫描：使用注解开发，不用配置controller。该句必须有，否则在web.xml的  DispatcherServlet中不能指定springmvc.xml，而必须指定applicationContext.xml -->* <**context:component-scan base-package="com.wego.controller"**/>   *<!-- 视图解析器 ，可以显式设置，也可以不设置，不设置会依据SpringMVC的默认设置-->* <**bean id="viewResolver"  class="org.springframework.web.servlet.view.InternalResourceViewResolver"**>  <**property name="prefix" value="/WEB-INF/"**/>  <**property name="suffix" value=".jsp"**/>  </**bean**>  </**beans**> |

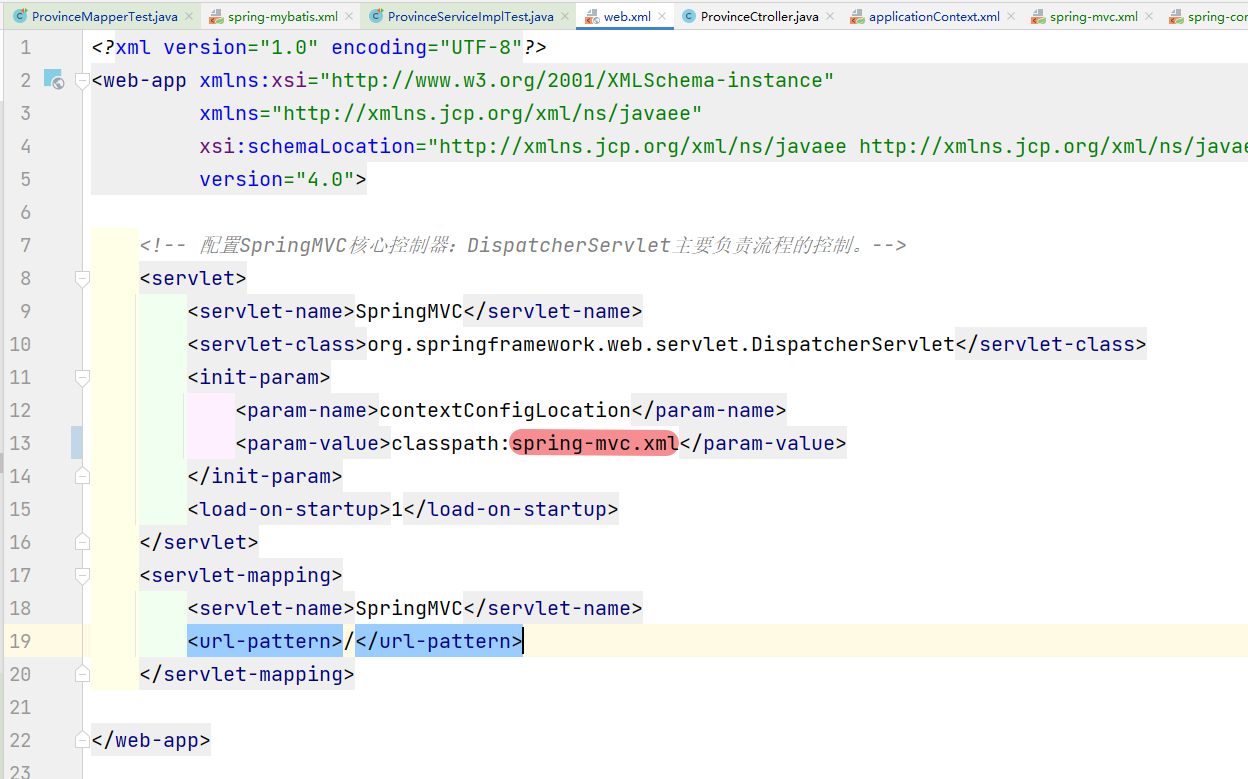
### spring-mybatis.xml

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd"**>   *<!-- 配置数据源 -->* <**bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource"**>  <**property name="driverClassName"**>  <**value**>com.mysql.cj.jdbc.Driver</**value**>  </**property**>  <**property name="url"**>  <**value**>jdbc:mysql://localhost:3306/db\_wego\_simple?useSSL=false**&amp;**serverTimezone=UTC**&amp;**user=root**&amp;**password=**&amp;**useUnicode=true**&amp;**characterEncoding=UTF8**&amp;**autoReconnect=true**&amp;**failOverReadOnly=false**&amp;**allowPublicKeyRetrieval=true</**value**>  </**property**>  <**property name="username"**><**value**>root</**value**></**property**>  <**property name="password"**><**value**>root</**value**></**property**>  </**bean**>   *<!-- 创建SqlSessionFactory，同时指定数据源-->* <**bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean"**>  <**property name="dataSource" ref="ds"** />  *<!-- 自动扫描mappers.xml文件-->* <**property name="mapperLocations" value="classpath\*:mapper/\*.xml"**/>  <**property name="configLocation" value="classpath:mybatis.xml"**/>  </**bean**>   *<!-- 注册Mapper -->* <**bean class="org.mybatis.spring.mapper.MapperScannerConfigurer"**>  <**property name="basePackage" value="com.wego.mapper"** />  </**bean**>   </**beans**> |

### applicationContext.xml

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd"**>   <**import resource="spring-context.xml"**/>  <**import resource="spring-mvc.xml"**/>  <**import resource="spring-mybatis.xml"**/>  </**beans**> |

### web.xml



## ssm集成minio

第一步：添加依赖

|  |
| --- |
| <**dependency**>  <**groupId**>io.minio</**groupId**>  <**artifactId**>minio</**artifactId**>  <**version**>8.5.4</**version**> </**dependency**> |

第二步：在resources目录下创建minio的属性文件minio.properties

|  |
| --- |
| **minio.url**=**http://127.0.0.1:9005 minio.username**=**minioadmin minio.password**=**minioadmin minio.bucketName**=**wego-simple** |

修改web.xml，在DispatcherServlet标签中添加如下内容

|  |
| --- |
| <**multipart-config**>  <**max-file-size**>20848820</**max-file-size**>  <**max-request-size**>418018841</**max-request-size**>  <**file-size-threshold**>1048576</**file-size-threshold**> </**multipart-config**> |



第三步：创建minio的配置文件，在其中读取minio.properties中的配置信息，并提供创建MinioClient的方法

|  |
| --- |
| **public class** MinioConfig {  **private** String **url**;  **private** String **username**;  **private** String **password**;  **private** String **bucketName**;   {  **final** Properties properties = **new** Properties();  **final** InputStream is = MinioConfig.**class**.getResourceAsStream(**"/minio.properties"**);  **try** {  properties.load(is);  } **catch** (IOException e) {  e.printStackTrace();  }  **url** = properties.getProperty(**"minio.url"**);  **username** = properties.getProperty(**"minio.username"**);  **password** = properties.getProperty(**"minio.password"**);  **bucketName** = properties.getProperty(**"minio.bucketName"**);  }   **public** String getBucketName() {  **return bucketName**;  }   **public** MinioClient getMinioClient() {  MinioClient minioClient = MinioClient.*builder*().endpoint(**url**)  .credentials(**username**, **password**).build();  **return** minioClient;  }   **public static void** main(String[] args) {  **final** MinioConfig config = **new** MinioConfig();  **final** MinioClient minioClient = config.getMinioClient();  System.***out***.println(minioClient);  System.***out***.println(config.getBucketName());  }  } |

第四步：创建Minio工具类

|  |
| --- |
| @Slf4j @Component **public class** MinioUtil {  **private** MinioClient **minioClient**;  **private** String **bucketName**;   {  **final** MinioConfig minioConfig = **new** MinioConfig();  **minioClient** = minioConfig.getMinioClient();  **bucketName** = minioConfig.getBucketName();  }   *//桶操作* /\*\*  \* 查看bucket是否存在  \* **@return** boolean  \*/  **public boolean** bucketExists() {  **try** {  **return minioClient**.bucketExists(BucketExistsArgs.*builder*()  .bucket(**bucketName**)  .build());  } **catch** (Exception e) {  ***log***.error(**"查看bucket是否存在"**, e);  **return false**;  }  }   /\*\*  \* 创建存储bucket  \* **@return** Boolean  \*/  **public boolean** createBucket() {  **if** (bucketExists()) {  **return true**;  }  **try** {  **minioClient**.makeBucket(MakeBucketArgs.*builder*()  .bucket(**bucketName**)  .build());  **return true**;  } **catch** (Exception e) {  ***log***.error(**"创建桶失败"**, e);  **return false**;  }  }   /\*\*  \* 删除存储bucket  \*  \* **@return** boolean  \*/  **public boolean** removeBucket() {  **if** (!bucketExists()) {  **return true**;  }  *//获取桶中所有的对象* List<Item> items = getBucketObjects();  **if** (items.size() > 0) {  *//有对象文件，则删除失败* **return false**;  }  **try** {  **minioClient**.removeBucket(RemoveBucketArgs.*builder*()  .bucket(**bucketName**)  .build());  **return true**;  } **catch** (Exception e) {  ***log***.error(**"根据名称删除桶失败"**, e);  **return false**;  }  }   /\*\*  \* 获取存储桶策略  \*  \* **@return** json  \*/  **public** String getBucketPolicy() {  String bucketPolicy = **null**;  **try** {  bucketPolicy = **minioClient**.getBucketPolicy(GetBucketPolicyArgs.*builder*()  .bucket(**bucketName**)  .build());  } **catch** (Exception e) {  ***log***.error(**"获取存储桶策略失败"**, e);  }  **return** bucketPolicy;  }   /\*\*  \* 根据bucketName获取信息  \*  \* return 如果不存在返回null  \*/  **public** Bucket getBucket() {  **try** {  **return minioClient**.listBuckets()  .stream()  .filter(b -> b.name().equals(**bucketName**))  .findFirst()  .orElse(**null**);  } **catch** (Exception e) {  ***log***.error(**"根据bucketName获取桶信息"**, e);  }  **return null**;  }   /\*\*  \* 获取全部bucket  \*/  **public** List<Bucket> getAllBuckets() {  **try** {  **return minioClient**.listBuckets();  } **catch** (Exception e) {  ***log***.error(**"获取所有的桶信息"**, e);  }  **return null**;  }   /\*\*  \* 创建文件夹或目录  \*  \* **@param *directoryName*** 目录路径  \*/  **public boolean** createDirectory(String directoryName) {  **if** (!bucketExists()) {  createBucket();  }   **try** {  **minioClient**.putObject(PutObjectArgs.*builder*()  .bucket(**bucketName**)  .object(directoryName)  .stream(**new** ByteArrayInputStream(**new byte**[]{}), 0, -1)  .build());  **return true**;  } **catch** (Exception e) {  ***log***.error(**"创建文件夹或目录失败"**, e);  **return false**;  }  }   *// 文件操作* /\*\*  \* 判断文件是否存在  \*  \* **@param *objectName*** 对象  \* **@return** 存在返回true，不存在发生异常返回false  \*/  **public boolean** objectExist(String objectName) {  **if** (!bucketExists()) {  **return false**;  }  **try** {  **minioClient**.statObject(StatObjectArgs.*builder*()  .bucket(**bucketName**)  .object(objectName)  .build());  **return true**;  } **catch** (Exception e) {  ***log***.error(**"判断文件是否存在失败"**, e);  **return false**;  }  }   /\*\*  \* 判断文件夹是否存在【注意是文件夹而不是目录】  \* **@param *folderName*** 文件夹名称（去掉前后的/）  \* **@return** \*/  **public boolean** folderExist(String folderName) {  **if** (!bucketExists()) {  **return false**;  }  **try** {  Iterable<Result<Item>> results = **minioClient**.listObjects(ListObjectsArgs.*builder*()  .bucket(**bucketName**)  .prefix(folderName)  .recursive(**false**)  .build());  **if** (results != **null**) {  **for** (Result<Item> result : results) {  Item item = result.get();  folderName += **"/"**;  **if** (item.isDir() && folderName.equals(item.objectName())) {  **return true**;  }  }  }  } **catch** (Exception e) {  ***log***.error(**"判断文件夹是否存在失败"**, e);  **return false**;  }  **return false**;  }   /\*\*  \* 文件上传  \* **@param *multipartFile*** 待上传文件  \* **@param *folderName*** 目录  \* **@param *aimFileName*** 最终保存到minio中的文件名，不需要后缀  \* **@return** \*/  **public** String putObject(MultipartFile multipartFile, String folderName, String aimFileName) {  **if** (!bucketExists()) {  createBucket();  }   **if** (!StringUtils.*hasText*(aimFileName)) {  aimFileName = UUID.*randomUUID*().toString();  }  *//获取文件后缀* String originalFilename = multipartFile.getOriginalFilename();  String suffix = originalFilename.substring(originalFilename.lastIndexOf(**"."**));  aimFileName += suffix;   *//带路径的文件名* String lastFileName = **""**;  **if** (StringUtils.*hasText*(folderName)) {  lastFileName = **"/"** + folderName + **"/"** + aimFileName;  } **else** {  lastFileName = aimFileName;  }   **try** (InputStream inputStream = multipartFile.getInputStream();) {  *//上传文件到指定目录,文件名称相同会覆盖* **minioClient**.putObject(PutObjectArgs.*builder*()  .bucket(**bucketName**)  .object(lastFileName)  .stream(inputStream, multipartFile.getSize(), -1)  .contentType(multipartFile.getContentType())  .build());  **return** getObjectUrl(lastFileName);  } **catch** (Exception e) {  ***log***.error(**"文件上传失败"**, e);  **return null**;  }   }   /\*\*  \* 上传文件【不指定文件夹】  \* **@param *multipartFile*** \* **@param *fileName*** \* **@return** \*/  **public** String putObject(MultipartFile multipartFile, String fileName) {  **return** putObject(multipartFile, **null**, fileName);  }   /\*\*  \* 上传文件【不指定文件夹,不指定目标文件名】  \* **@param *multipartFile*** \* **@return** \*/  **public** String putObject(MultipartFile multipartFile) {  **return** putObject(multipartFile, **null**, **null**);  }   /\*\*  \* 自动创建桶并存储文件  \*  \* **@param *inputStream*** \* **@param *aimFileName*** 必须，minio桶中文件的名字，需要带后缀  \* **@return** \*/  **public** String putObject(InputStream inputStream, String aimFileName) {  **if** (!bucketExists()) {  createBucket();  }  **try** {  PutObjectArgs putObjectArgs = PutObjectArgs.*builder*()  .bucket(**bucketName**)  .object(aimFileName)  .stream(inputStream, inputStream.available(), -1)  .build();  **minioClient**.putObject(putObjectArgs);  inputStream.close();  **return** getObjectUrl(aimFileName);  } **catch** (Exception e) {  ***log***.error(**"文件上传失败"**, e);  **return null**;  }  }   /\*\*  \* 拷贝文件  \*  \* **@param *objectName*** 文件名称  \* **@param *srcBucketName*** 目标bucket名称  \* **@param *srcObjectName*** 目标文件名称  \*/  **public boolean** copyObject(String srcBucketName, String srcObjectName, String objectName) {  **try** {  **minioClient**.copyObject(  CopyObjectArgs.*builder*()  .source(CopySource.*builder*()  .bucket(srcBucketName)  .object(srcObjectName)  .build())  .bucket(**bucketName**)  .object(objectName)  .build());  **return true**;  } **catch** (Exception e) {  ***log***.error(**"拷贝文件失败"**, e);  **return false**;  }  }   /\*\*  \* 文件下载  \* **@param *fileName*** 文件名称  \* **@param *response*** response  \* **@return** Boolean  \*/  **public void** getObject(String fileName, HttpServletResponse response) {  **if** (!bucketExists()) {  **return**;  }  GetObjectArgs getObjectArgs = GetObjectArgs.*builder*()  .bucket(**bucketName**)  .object(fileName)  .build();   **try** (ServletOutputStream outputStream = response.getOutputStream();  GetObjectResponse objectResponse = **minioClient**.getObject(getObjectArgs)) {   response.setCharacterEncoding(**"utf-8"**);  *//设置强行下载不打开  //response.setContentType("application/force-download");  //response.setContentType("APPLICATION/OCTET-STREAM");* response.addHeader(**"Content-Disposition"**, **"attachment;filename="** + fileName);  ByteStreams.*copy*(objectResponse, outputStream);  outputStream.flush();  } **catch** (Exception e) {  ***log***.error(**"文件下载失败"**, e);  }  }   /\*\*  \* 以流的形式获取一个文件对象  \*  \* **@param *objectName*** 对象名称  \* **@return** {**@link** InputStream}  \*/  **public** InputStream getObject(String objectName) {  **if** (!bucketExists()) {  **return null**;  }  **try** {  StatObjectResponse statObjectResponse = **minioClient**.statObject(StatObjectArgs.*builder*()  .bucket(**bucketName**)  .object(objectName)  .build());  **if** (statObjectResponse.size() > 0) {  *// 获取objectName的输入流。* **return minioClient**.getObject(GetObjectArgs.*builder*()  .bucket(**bucketName**)  .object(objectName)  .build());  }  } **catch** (Exception e) {  ***log***.error(**"文件下载失败"**, e);  }  **return null**;  }   /\*\*  \* 获取文件信息, 如果抛出异常则说明文件不存在  \*  \* **@param *objectName*** 文件名称  \*/  **public** StatObjectResponse getObjectInfo(String objectName) {  **if** (!bucketExists()) {  **return null**;  }   StatObjectResponse statObjectResponse = **null**;  **try** {  statObjectResponse = **minioClient**.statObject(StatObjectArgs.*builder*()  .bucket(**bucketName**)  .object(objectName)  .build());  } **catch** (Exception e) {  ***log***.error(**"获取文件信息失败"**, e);  }  **return** statObjectResponse;  }   /\*\*  \* 获取图片的路径  \*  \* **@param *fileName*** \* **@return** \*/  **public** String getObjectUrl(String fileName) {  **try** {  **if** (fileName.startsWith(**"/"**)) {  fileName = fileName.substring(1);  }  GetPresignedObjectUrlArgs build = GetPresignedObjectUrlArgs.*builder*()  .bucket(**bucketName**)  .object(fileName)  .method(Method.***GET***)  *//过期时间(分钟数)* .expiry(60 \* 60)  .build();  **return minioClient**.getPresignedObjectUrl(build);  } **catch** (Exception e) {  ***log***.error(**"获取文件路径失败"**, e);  }  **return null**;  }   /\*\*  \* 断点下载  \*  \* **@param *objectName*** 文件名称  \* **@param *offset*** 起始字节的位置  \* **@param *length*** 要读取的长度  \* **@return** 流  \*/  **public** InputStream getObject(String objectName, **long** offset, **long** length) {  **if** (!bucketExists()) {  **return null**;  }  GetObjectResponse objectResponse = **null**;  **try** {  objectResponse = **minioClient**.getObject(GetObjectArgs.*builder*()  .bucket(**bucketName**)  .object(objectName)  .offset(offset)  .length(length)  .build());  } **catch** (Exception e) {  e.printStackTrace();  }  **return** objectResponse;  }   /\*\*  \* 获取指定桶中的所有文件对象  \*  \* **@return** 存储bucket内文件对象信息  \*/  **public** List<Item> getBucketObjects() {  **if** (!bucketExists()) {  **return null**;  }   List<Item> items = **new** ArrayList<>();   Iterable<Result<Item>> results = **minioClient**.listObjects(ListObjectsArgs.*builder*()  .bucket(**bucketName**)  .build());  **if** (results != **null**) {  **try** {  **for** (Result<Item> result : results) {  items.add(result.get());  }  } **catch** (Exception e) {  ***log***.error(**"获取指定桶中的所有文件对象"**, e);  }  }  **return** items;  }   /\*\*  \* 获取路径下文件列表  \*  \* **@param *prefix*** 路径名称  \* **@param *recursive*** 是否递归查找，如果是false,就模拟文件夹结构查找  \* **@return** 二进制流  \*/  **public** Iterable<Result<Item>> getObjects(String prefix, **boolean** recursive) {  **if** (!bucketExists()) {  **return null**;  }  Iterable<Result<Item>> results = **minioClient**.listObjects(ListObjectsArgs.*builder*()  .bucket(**bucketName**)  .prefix(prefix)  .recursive(recursive)  .build());  **return** results;  }   /\*\*  \* 根据文件前置查询文件  \*  \* **@param *prefix*** 前缀  \* **@param *recursive*** 是否递归查询  \* **@return** MinioItem 列表  \*/  **public** List<Item> getAllObjectsByPrefix(String prefix, **boolean** recursive) {  **if** (!bucketExists()) {  **return null**;  }  List<Item> items = **new** ArrayList<>();  Iterable<Result<Item>> objectsIterator = **minioClient**.listObjects(  ListObjectsArgs.*builder*()  .bucket(**bucketName**)  .prefix(prefix)  .recursive(recursive)  .build());  **if** (objectsIterator != **null**) {  **try** {  **for** (Result<Item> o : objectsIterator) {  Item item = o.get();  items.add(item);  }  } **catch** (Exception e) {  e.printStackTrace();  }  }  **return** items;  }   /\*\*  \* 删除文件  \* **@param *fileName*** 文件名  \* **@return** \* **@throws** Exception  \*/  **public boolean** removeObject(String fileName) {  **if** (!bucketExists()) {  **return false**;  }  **try** {  **minioClient**.removeObject(RemoveObjectArgs.*builder*()  .bucket(**bucketName**)  .object(fileName)  .build());  **return true**;  } **catch** (Exception e) {  ***log***.error(**"根据文件删除文件失败"**, e);  **return false**;  }  }   /\*\*  \* 批量删除文件对象【没有测试成功】  \* **@param *objects*** 需要删除的文件列表  \*/  **public boolean** removeObjects(String... objects) {  **if** (!bucketExists()) {  **return false**;  }  List<DeleteObject> deleteObjects = **new** LinkedList<>();  Arrays.*stream*(objects).forEach(s -> {  deleteObjects.add(**new** DeleteObject(s));  });   Iterable<Result<DeleteError>> results = **minioClient**.removeObjects(RemoveObjectsArgs.*builder*()  .bucket(**bucketName**)  .objects(deleteObjects)  .build());   *//Minio处理批量删除的时候, 采用的延迟执行, 需要通过迭代返回的Iterable<Result<DeleteError>>以执行删除* **if** (results != **null**) {  **try** {  **for** (Result<DeleteError> result : results) {  DeleteError error = result.get();  ***log***.error(**"Error in deleting object "** + error.objectName() + **"; "** + error.message());  }  } **catch** (Exception e) {  ***log***.error(**"批量删除文件失败"**, e);  }  }   **return true**;  }  } |

第五步：在Controller中使用：

* 前面页面：

|  |
| --- |
| <**html**> <**body**> <**form method="post" enctype="multipart/form-data"  action="${**pageContext.request.contextPath**}/fileUpload"**>  <**input type="file" name="imgFile"**>  <**input type="submit"**> </**form**> </**body**> </**html**> |

控制器

|  |
| --- |
| @Controller **public class** DispatcherController {  @Autowired  **private** MinioUtil **minioUtil**;   @PostMapping(**"/fileUpload"**)  **public void** fileUpload(MultipartFile imgFile) {  **final** String s = **minioUtil**.putObject(imgFile);  System.***out***.println(s);  }   /\*\*  \* 获取图片的路径【带问号后面的内容】  \* **@param *filename*** \* **@return** \*/  @ResponseBody  @GetMapping(**"/fileDownload/{filename}"**)  **public void** download(@PathVariable(**"filename"**) String filename, HttpServletResponse response) {  **minioUtil**.getObject(filename,response);  } } |

## 常用模块：代码自动补全

### 网页

|  |
| --- |
| <%@ **page contentType**="**text/html;charset=UTF-8**" **language**="**java**" %>  <**html**> <**body**> <**input type="text" id="info" onblur="***fun*()**"**> <**div id="show"**>  </**div**> <**button**>搜索</**button**>  <**script src="${**pageContext.request.contextPath**}/static/manager/js/jquery.js"**></**script**> <**script type="text/javascript"**>  **function** *fun*(){  **let** info = **$**(**"#info"**).val();  **$**.ajax({  **type**:**"get"**,  **url**:**"${**pageContext.request.contextPath**}/ajax?info="**+info,  success:**function** (data){  **$**(**"#show"**).empty();  **$**.each(data,**function** (index,item){  **$**(**"#show"**).append(**"<div>"**+item.**info**+**"</div>"**)  })  }  })  } </**script**> </**body**> </**html**> |

### 控制器

|  |
| --- |
| @Controller **public class** CategoryController {   @Autowired  **private** CategoryMapper **categoryMapper**;   @ResponseBody  @GetMapping(**"/ajax"**)  List<Category> ajax(String info){  **final** Example example = **new** Example(Category.**class**);  **final** Example.Criteria criteria = example.createCriteria();  criteria.andLike(**"info"**,info+**"%"**);  **final** List<Category> categoryList = **categoryMapper**.selectByExample(example);  **return** categoryList;  } |
| } |