**电商**

# 电商模式

**B2C（Business to Consumer）** ：企业与消费者之间的电子商务。B2C是电子商务最普遍的模式，就是通常说的商业零售，借助互联网进行在线销售，直接面向消费者销售产品和服务，消费者线上支付，网上购物。 B2C模式的电商是日常生活中最常见的模式，类似**京东、 天猫**等都是典型的B2C商城。

**B2B（Business to Business）** ：企业与企业之间的电子商务。 供应方（Business） 与采购方（Business） 之间通过运营者（Operator） 达成产品或服务交易的一种电子商务模式。 B2B解决的是供应商的上游到中游的问题，B2B模式的优势在于可降低采购成本，节省周转时间，但是企业采购一般要求较高（资质、 流程、 库存、 货款等） ，需要许多线下沟通，目前B2B电商大部分是促成中小企业的线上交易。 如今B2B平台更多是提供企业之间的信息匹配，要将B2B电商发展成类似B2C一样繁荣，需要在细分行业深度挖掘需求。 类似**阿里巴巴**（1688） 是典型的B2B商城。

**C2C（Consumer to Consumer）** ：消费者与消费者之间的电子商务。C2C模式相对B2C来说更加灵活，对于海量商品和类目进行补充，可以针对用户做个性化定制。 市场的主流趋势仍是B2C，更能保证商品质量和服务。 移动社交的兴起给C2C电商带来了一定机会，给了C端做分销或个性化商品的环境。 **淘宝网**、 微店等都是典型的C2C商城。

C2M（Customer to Manufactory） ：客户与工厂直接对接，即定制化生产消费。 C2M砍掉了架在消费者和制造商之间的品牌商、 零售商。对于消费者而言，这将是重磅福利，因为这样省去了渠道成本和库存成本，消费者将能用非常低的价格买到品质上乘的商品，而且还能定制；对于制造商来说，由于价格低质量好，吸引更多顾客，这也能给他们带去巨大的收益；这种模式还解决了库存顽疾这一大问题，将传统的刚性生产模式变成了柔性生产线。 未来C2M的利润空间更多的将是定制服务和增值服务。

**O2O（Online to Offline）** ：一般是指线上到线下，通过互联网连接本地生活服务，解决日常生活中的衣食住行等需求。 O2O平台在线上提供生活服务信息，从线上获取信息、 购买，到线下享受服务，再到线上评价完成消费的闭环。

# 发展简史

这里主要讲和技术相关的演化过程。先总体有一个大的演化过程概念就行。

## 个人网站

LAMP（Linux+Apache+MySQL+PHP），这个直到现在还是一个很常用的网站架构模型，其优点是：无须编译，发布快速，PHP语言功能强大，能做从页面渲染到数据访问所有的事情，而且用到的技术都是开源、免费的。



mysql

apache

Php\_mod

Pear db

## Java时代

这个时代就是经典的java框架时代，时间长一点的程序员都知道，以前做开发只要知道**SSH**就能有个工作了。

**Spring + Stuts2 + Hibernate**

Oracle(mysql)

Jboss(weblogic)

mvc

Ejb(Spring)

orm

## 分布式(服务化)

说到分布式的产生，最根本的原因还是**业务驱动的**。由于网络的力量，业务的量和复杂度都前所未有的激增，终于有一天到达了一个极限，所有人都受不了了。业务的拆分，可重用性，服务化势在必行。

而这一切带来的技术复杂度也是前所未有的挑战！！

分库-01

分库-02

分表-01

分表-02

分库分表中间件(mycat,sharding-jdbc)

Rpc远程过程调用中间件(dubbo，springcloud,rmi)

消息中间件

Kafuka

Rocketmq

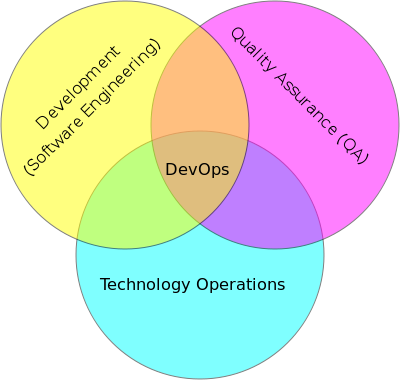
Activemq

rabbitmq

## 微服务

随着业务的拆分，架构由集中式向分布式（服务化）的改变，整个行业服务化的意识和经验也越来越丰富，服务的规模也越来越大，服务的拆分和管理也有了一些标准和原则。例如：**Martin Fowler**的关于微服务的九大特征的描述。

那么究竟“微服务”是什么，微服务其实不仅仅是架构层面的一个概念了，时至今日，现在的微服务，已经是集开发、测试、集成、发布、部署以及维护为一体的一个生态圈了。由此产生了“devops”理念，以及实现此理念的**CI、CD及容器化**技术。



而我们今天的技术人学微服务，也绝不是“赶时髦”，而是顺势而为，因为如今显然已经是**云的时代**。

云服务

虚拟化技术

pc

手机

微信

ipad

# 项目概述

Goshop类似于JD的一个自营模式的B2C商城。

我们最后一起来完成这个项目，为什么选电商项目，因为在各种项目中，电商的业务复杂，灵活性高，版本更新频繁，关键是各种分布式、并发性的问题，电商的解决方案是最具有代表性的。可以这么说，你能做电商项目，那么其他的传统项目你就都能做。

# 产品架构

前端

商品系统

订单系统

内容管理系统

促销系统

库存系统

调度系统

支付系统

会员系统

财务管理

采购中心

店铺管理

评价系统

客服系统

风控系统

物流系统

仓储系统

权限系统

PC

APP/H5

微信

ipad

支撑

后端

# 技术架构

应用层

商品系统

订单系统

会员系统

......

服务层

商品服务

订单服务

会员服务

......

存储层

mysql

oss

redis

......

git

maven

jenkins

nginx

cloud

mq

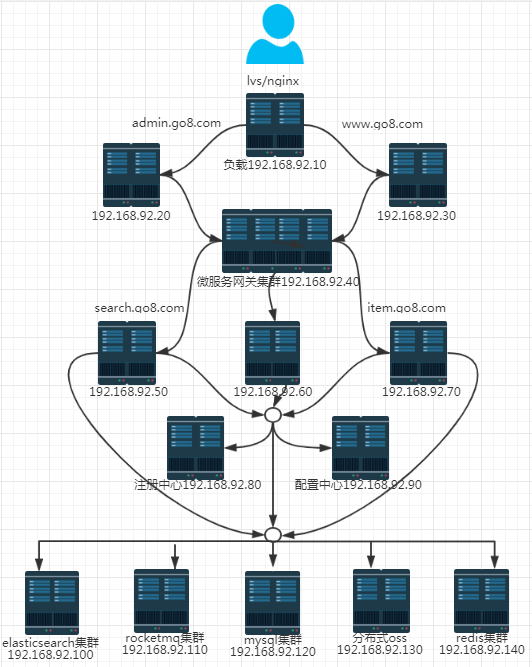
docker

Thymeleaf等静态化解决方案

Zookeeper分布式协同解决方案

高性能文件、缓存解决方案

# 网络架构



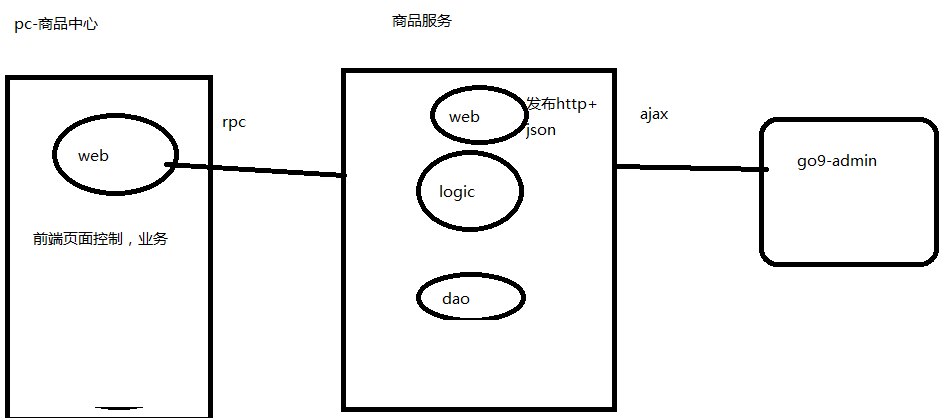
# 项目演示

整体了解即可。

# 软件开发模式

五个阶段，分别是瀑布模型（waterfall），**敏捷开发**（agile），看板（KANBAN），SCRUM 和精益软件开发（lean）。

# 工程分析



在项目初期一般都是集中式的结构，大家前面的项目都是这样的对吧，从今天开始我们来学习创建微服务结构的项目。微服务项目工程搭建总体较难，难在对业务边界的划分，以及服务拆分粒度的合理选择。太细导致系统太复杂，太粗难以复用，耦合度高。

再复习下单体式（**monolith**）

代码 编译、打包 部署

war

jar

jar

jar

project

project

project

# 工程结构图

goshop

go8-common

go8-admin

go8-goods-service

go8-member-service

go8-web

go8-trade-service

go8-search-service

继承

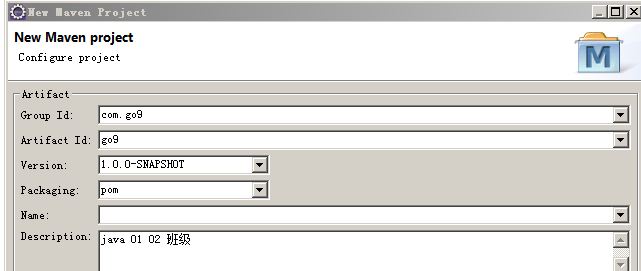
继承

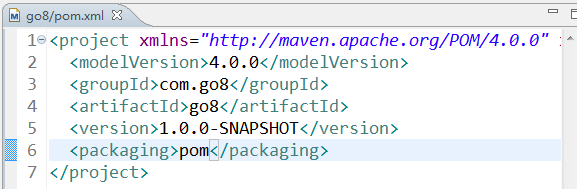
依赖

前后端完全分离的

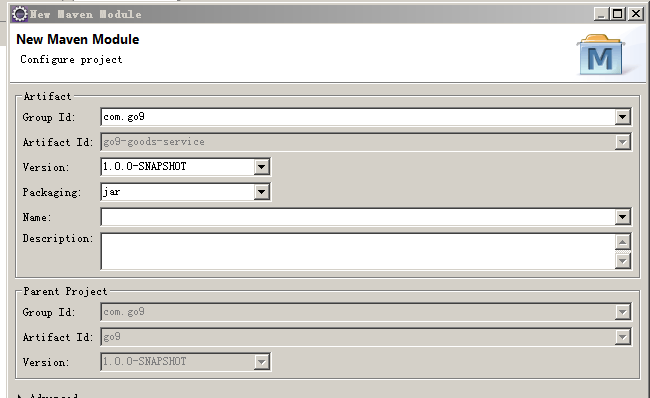
系统后台js项目

# 创建父工程





# 创建go8-goods-service



关键点：

这里需要将子工程公共配置提取到父工程的pom中。例如文件编码，编译器版本，spring-cloud配置及公共仓库等等。

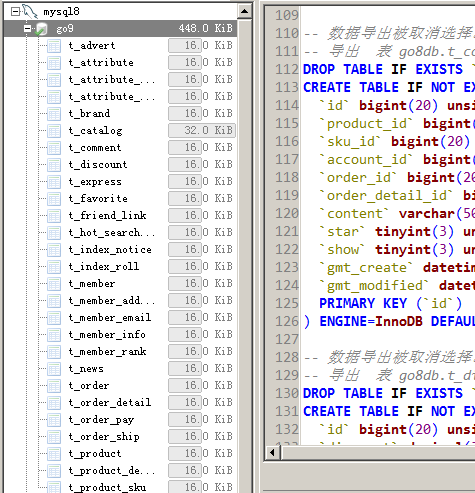
**提示：这里可以不用引入spring-cloud的配置，暂时还用不到。**

|  |
| --- |
| <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>2.1.3.RELEASE</version>  <relativePath /> <!-- lookup parent from repository -->  </parent>  <groupId>com.go9</groupId>  <artifactId>go9</artifactId>  <version>1.0.0-SNAPSHOT</version>  <packaging>pom</packaging>  <description>java 01 02 班级</description>  <!-- 公用的配置 -->  <modules>  <module>go9-goods-service</module>  </modules>    <properties>  <java.version>1.8</java.version>  </properties>    <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

|  |
| --- |
| spring:  datasource:  type: com.alibaba.druid.pool.DruidDataSource  url: jdbc:mysql://localhost:3306/test?characterEncoding=utf-8&autoReconnect=true&failOverReadOnly=false  username: root  password: 111111  driver-class-name: com.mysql.cj.jdbc.Driver  server:  port: 9000 |

# 数据库

导入数据库脚本



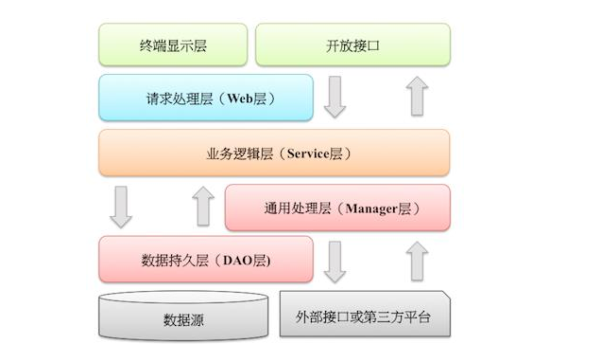
# 商品分类

让我们从商品分类开始，用这个分类业务帮我们建立起一个完整的微服务工程出来。每个微服务工程内部按照分层的原则进行设计。那么这里就自然涉及很多跟设计有关的问题。例如：包命名规则，表命名规则，API如何定义等等？？这些确实在开发之前就需要统一想好的，一般企业会先将文档写好，各个组再严格按照文档定义来做。尤其现在是前后端分离的时代，没有共同的规则约定，是无法分工协作的。

这里暂不提供文档，完整的文档内容太多，大家也容易忘记，我们用到什么就讲什么。我们下面先来看下行业内有哪些好的设计经验值得分享的。

# 设计经验

## 工程结构

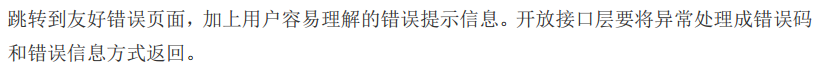


可以参考一下**阿里内部的工程约定**，上图。

## 异常处理

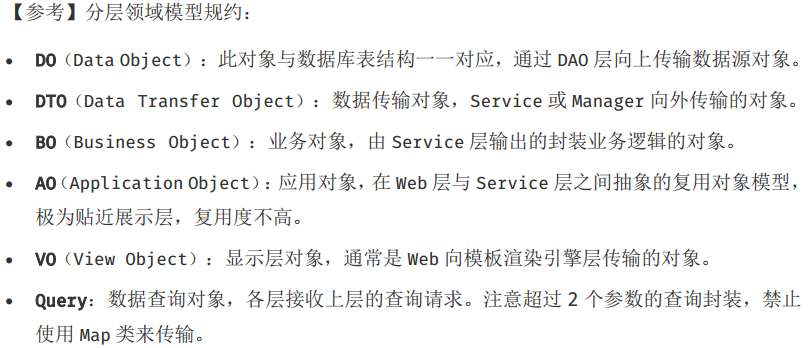
那么异常、还有接口消息如何处理呢？？也来看下阿里的建议！！





## 模型对象

那么有经验的你，肯定还有关于模型对象的问题，对的，我们项目中有大量的数据对象，业务对象及视图对象，那么这又怎么做，才会好点呢？让我们也能看看阿里是如何建议的。



## 接口设计

最后再说一个现在的微服务接口都会暴露成http协议的，那么现在都会使用**restful**风格的，请一定严格遵守，如果你们事先已经定义好了的话。

REST：REpresentational State Transfer，中译为“表属性状态传递”。这个名字源自于国外一个博士的论文，有兴趣的的同学可以去网上翻一翻，当年并没有受到重视，但是现在已经是微服务接口的事实标准了。

**一些最佳实践原则：**

1 使用HTTP动词表示增删改查资源， GET：查询，POST：新增，PUT：更新，DELETE：删除

2 返回结果必须使用JSON

3 HTTP状态码，在REST中都有特定的意义：200，201,202,204,400,401,403,500。比如401表示用户身份认证失败，403表示你验证身份通过了，但这个资源你不能操作。

4 如果出现错误，返回一个错误码。

5 API必须有版本的概念，v1，v2，v3

6 使用Token令牌来做用户身份的校验与权限分级，而不是Cookie。

7 url中大小写不敏感，不要出现大写字母

8 有一份漂亮的文档~（很重要）

**举个例子：**<https://api.z.cn/v1/product/recent?page=3&size=20>

**大家觉得这个url怎么样，是不是一看就懂了。其实开发中有时候最头疼的事不是业务，更不是技术，而是起名字，想想小时候你父母给你起个名字多头疼，那只是一个名字，我们程序员要天天起名字啊。头疼了没？？这真的是很有艺术性的一份工作，在后面的开发中慢慢体会吧。**

## 共同约定

这里重点说下接口返回的消息协议问题，格式是json没错，但是具体的错误码还是得约定一下的，不然乱套了，猜猜淘宝的错误码有多少了？？在约定之前，先看哈HTTP如何使用状态码的，学习从模仿开始嘛。

HTTP状态码分如下大类：

* 1xx：信息
* 2xx：成功
* 3xx：重定向（注意304视为重定向的一种可以理解为需要从缓存读取）
* 4xx：客户端错误
* 5xx：服务器错误

这里有些同学可能会误解，这些不是已经定义好了吗，不是有错误码吗？为什么我们还要定义啊，童鞋，我们这里自定义的错误码，是用来区分不同的**业务**错误的，而以上错误码只是通用的用来表示通信状态的。

约定：

* 1xxx：代表通用错误，例如1001代表输入参数不合法
* 2xxx：操作成功
* 3xxx：商品业务错误
* 4xxx：订单错误
* 5xxx：会员错误

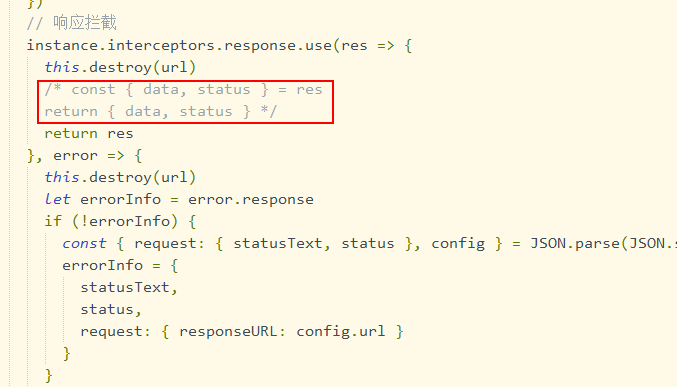
等等。。。。。。。。。。。。

以上讲的设计经验，都比较抽象，大家先听听就行，有这么个意识就好，写一个项目是很容易的，但是要想写出一个好项目，甚至是一个艺术品，真的好难，是需要大量的实践经验积累的。还有千万别教条，具体还是要跟业务走的。

# 前端工程

不要轻易换新版本！！！

修改前端响应消息封装代码



禁用eslint错误提示：

//都是泪水，这行代码我搞了一天，你不写代码，你不知道这有多痛苦

在文件vue.config.js

**lintOnSave: false,**

# 实践

## 项目结构

com.go8.goods.web

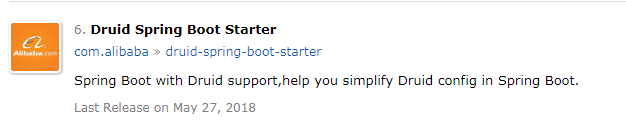
com.go8.goods.service

com.go8.goods.manage

如果有需要的话

com.go8.goods.mapper

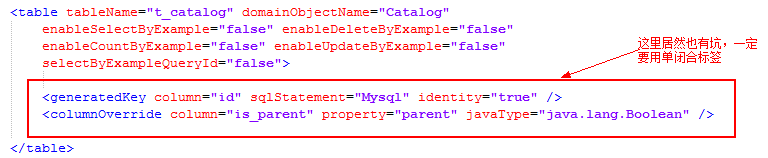
## 集成dao



|  |
| --- |
| <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid-spring-boot-starter</artifactId>  <version>1.1.10</version>  </dependency> |

反向生成dao代码，见提供的资源，解压缩后修改配置文件，再启动运行即可。

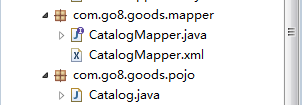
关键配置截图：



执行界面



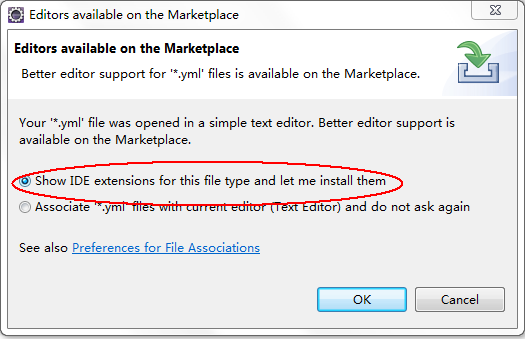
生成结果

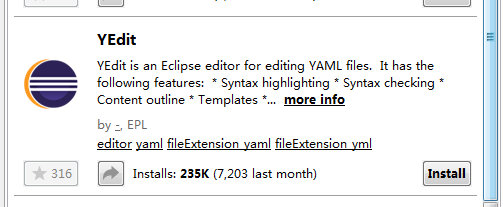


## boot配置

编辑**application.yml**文件

注意，第一次创建yml文件需要安装插件，eclipse如下图





安装下一步即可！！！

目前配置如下

|  |
| --- |
| spring:  datasource:  type: com.alibaba.druid.pool.DruidDataSource  url: jdbc:mysql://localhost:3306/go8db?characterEncoding=utf-8&autoReconnect=true&failOverReadOnly=false  username: root  password: 111111  driver-class-name: com.mysql.cj.jdbc.Driver  #druid 配置，线上千万别这么配置，使用默认值就好，否则不能自动重连接  druid:  connectionErrorRetryAttempts: 0  breakAfterAcquireFailure: **true** |

同时创建 boot启动类**GoodsApplication.java**

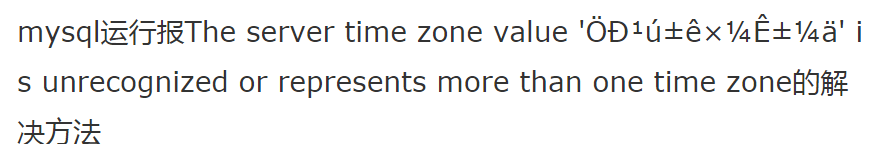
|  |
| --- |
| package com.go8.goods;  import org.mybatis.spring.annotation.MapperScan;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication  @MapperScan("com.go8.goods.mapper")  public class GoodsApplication {  public static void main(String[] args) {  SpringApplication.run(GoodsApplication.class, args);  }  } |

## 单元测试

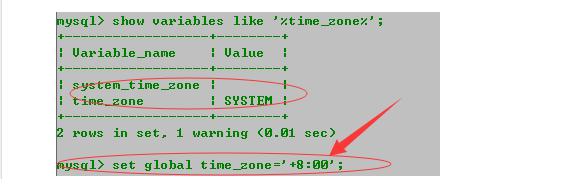
|  |
| --- |
| package com.go8.goods.mapper;  import static org.junit.Assert.\*;  import java.util.Date;  import org.junit.Test;  import org.junit.runner.RunWith;  import org.mybatis.spring.annotation.MapperScan;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.boot.test.context.SpringBootTest;  import org.springframework.test.context.junit4.SpringRunner;  import com.go8.goods.pojo.Catalog;  @RunWith(SpringRunner.class)  @SpringBootTest  public class CatalogMapperTest {  @Autowired  private CatalogMapper catalogMapper;    @Test  public void testInsert() {  Catalog cat = new Catalog();  cat.setCode("yifu");  cat.setName("衣服");  cat.setOrder1(1);  cat.setParent(true);  cat.setPid(0L);  Date now = new Date();  cat.setGmtCreate(now);  cat.setGmtModified(now);  int insert = catalogMapper.insert(cat);  assertEquals(1, insert);  }  } |

## 可能的问题

单元测试插入数据出错，时区的锅（出现在新版本里）



新的版本驱动，包名也改了，会检测数据库的本地时区，默认使用gmt。所以要将数据库时区修改成和系统匹配的真实时区。china的系统都是东八区。

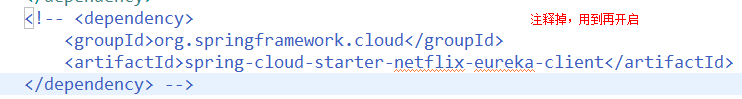


问题思考：为什么有时测试失败时，测试进程没有结束，而是一直在运行，何解？？



网易的一道面试题。

另外，在测试时可以先注释掉微服务的pom配置，避免不必要的干扰。



解决完时区问题，测试应该会正常通过。

## CatalogService

|  |
| --- |
| package com.go8.goods.service;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Service;  import com.go8.goods.mapper.CatalogMapper;  import com.go8.goods.pojo.Catalog;  @Service  public class CatalogServiceImpl implements CatalogService {  @Autowired  private CatalogMapper catalogMapper;    @Override  public void add(Catalog catalog) {  Date now = new Date();  catalog.setGmtCreate(now);  catalog.setGmtModified(now);  catalogMapper.insert(catalog);  }  } |

**依次完成其他查询、修改和删除代码。**

## 发布接口

接口发布的协议，有多种形式，总体分为http和非http的，http形式的多用于对外开放接口，好处不受语言和平台的影响；非http的rpc协议接口，协议种类繁多，如dubbo、thrift等等，多用于内部服务之间的调用，好处速度快点，坏处受平台和语言影响。

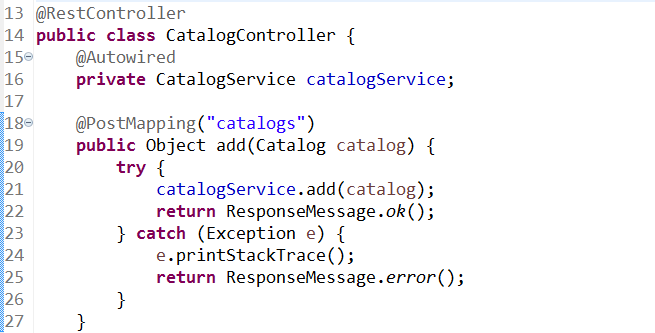
我们目前完成的项目是go8后台管理系统，此系统是一个基于vue的前端项目，语言不同源，因此适合使用http+json的接口协议。

http + json

约定基础接口

|  |  |  |  |
| --- | --- | --- | --- |
| 业务类型 | method | url | return body |
| 添加 | post | catalogs | 返回一个统一的json  对象，  {  “code”: xxx,  “message”:”xxxx”,  “data”:{} or []  } |
| 修改 | put | catalogs/{id} |
| 删除 | delete | catalogs/{id} |
| 查询单体 | get | catalogs/{id} |
| 查询子集 | get | catalogs/sons?pid |
| 查询后代 | get | catalogs/all?pid |
| 唯一性验证 | get | catalogs/code?code |

发布代码（部分）

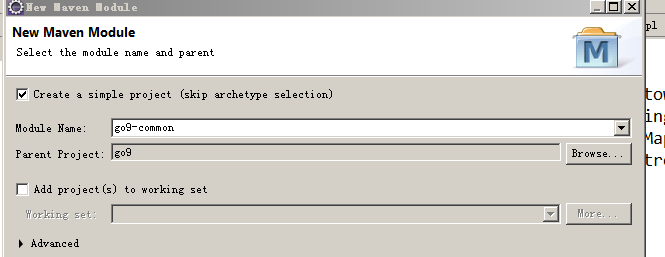


## 接口测试

后台接口开发好以后，我们需要对接口先进行功能性测试，也就是对不对。这里介绍一款专业的api测试工具postman。下载，安装都很简单。注意，这个软件用微软的.net开发的，安装时会下载.net，需要联网，还有耐心。

下载：百度到首页再寻找下载页面，自己下载，也可以去提供的资源包去找。

# 公共模块



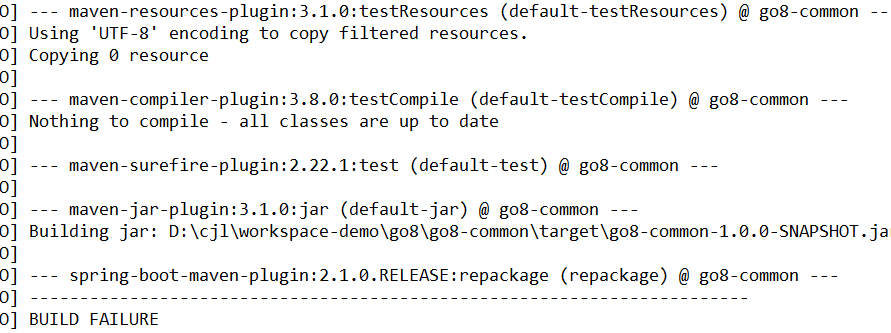


# Cors

|  |
| --- |
| package com.go9.goods.config;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  import org.springframework.web.cors.CorsConfiguration;  import org.springframework.web.filter.CorsFilter;  import org.springframework.web.cors.UrlBasedCorsConfigurationSource;  @Configuration  public class GoodsCorsConfiger {  private CorsConfiguration buildCorsConfiger() {  CorsConfiguration corsc = new CorsConfiguration();  corsc.addAllowedOrigin("\*");  corsc.addAllowedMethod("\*");  corsc.addAllowedHeader("\*");  return corsc;  }    //配置  @Bean  public CorsFilter getCorsFilter() {  UrlBasedCorsConfigurationSource configSource = new UrlBasedCorsConfigurationSource();  configSource.registerCorsConfiguration("/\*\*", buildCorsConfiger());  CorsFilter corsf = new CorsFilter(configSource );  return corsf;  }  } |

# 踩坑记录

发现一个很有意思的事情，在执行**mvn clean install** 这个曾经我们几代人共识的安装命令时，在屏幕上看到了一个爆炸性的输出，眼睛里都是泪水，一时我竟无言以对。



发现什么问题没有，失败，为什么呢，新版本，新版本，千万别轻易尝试**新版本**。

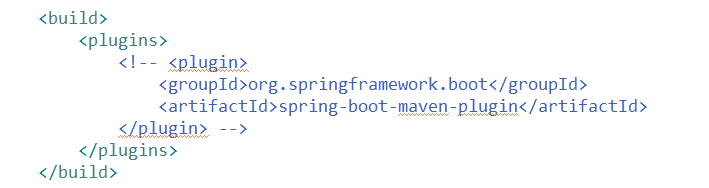
**程序界定律之一：一代版本，一代坑啊！！**

**怎么办啊，我也很绝望啊，同学们！！**

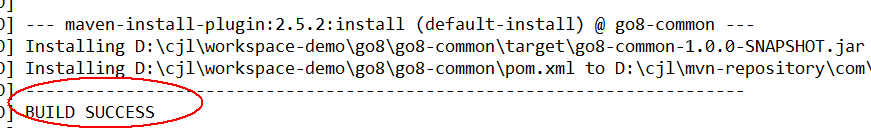
3种办法：

1. 第一种在安装common时工程时，临时注释掉spring-boot-maven插件，哈哈
2. 将父工程里的插件放到各个子工程pom文件中去，按需配置
3. 保持原来的都不变，在子工程中不需要repackage的工程中，修改默认的spring-boot-maven的 执行方式。

我这里临时用了第一种先解燃眉之急哈，如下图



然后再安装go8-common就可以了

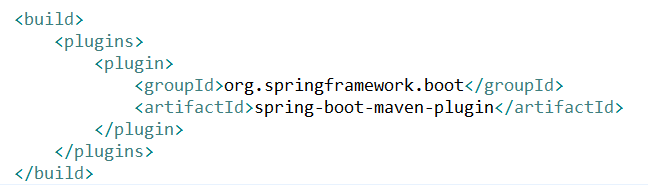


别忘了把父工程也安装一下

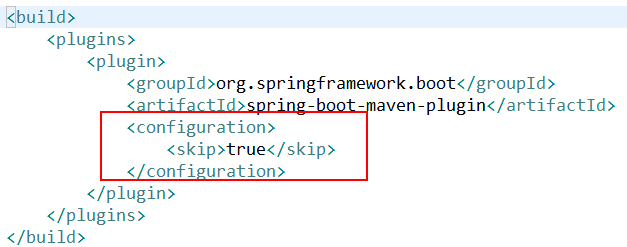
**注意：同学们，还是用第三种方法吧，临时方法只是用来测试而已，有bug的。**

**第三种安全的解决方案如下：**

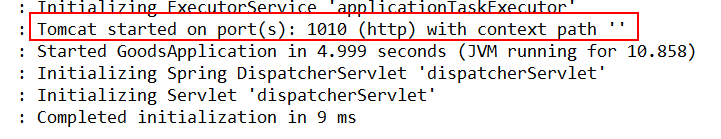
**父工程pom不变**



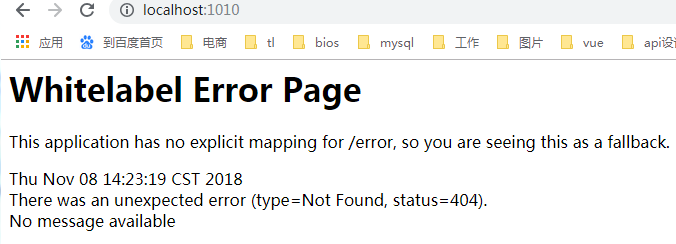
**子工程go8-common的pom如下**



再次启动服务成功，如下图



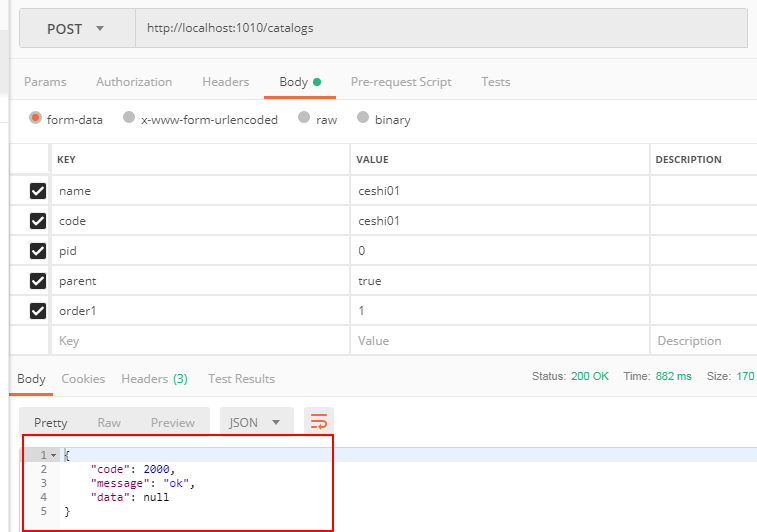
浏览器访问



这里有的同学可能会遇到一个更有意思的事情，就是刚才好不容易测试通过的CatalogMapperTest测试添加一条类目的操作，你再次去测试，会发现一个找不到**CatalogMapper.xml**配置文件的错误，就是提示xxxx is not bound，这种错误啦。这是maven的一个bug啦，而且它不总是会出现。一般会出现在install阶段，现在出现，就是因为你前面执行安装了父工程，安装父工程时会安装子工程，会重新打包，这时基本都会丢失mybatis的配置文件。如果你遇到了，请这样来解决，没遇到的同学，也请添加一下。请在父工程的pom文件里添加资源配置，如下图



## 再测接口



课后有时间的同学最好再测试下异常情况。