**畅购电商系统开发**

**第1天**

传智播客.黑马程序员.深圳

# 学习目标

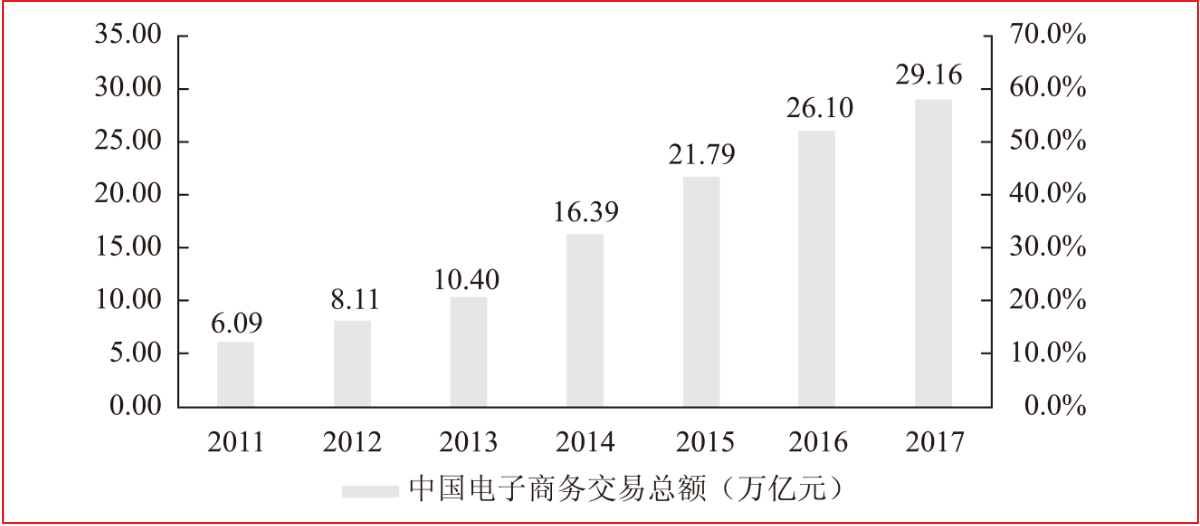
* 了解电商
* 了解畅购架构
* 了解畅购工程结构
* 畅购工程搭建
* 商品微服务搭建
* 品牌增删改查

# 走进电商

## 电商行业分析

近年来，世界经济正向数字化转型，大力发展数字经济成为全球共识。党的十九大报告明确提出要建设“数字中国”“网络强国”，我国数字经济发展进入新阶段，市场规模位居全球第二，数字经济与实体经济深度融合，有力促进了供给侧结构性改革。电子商务是数字经济的重要组成部分，是数字经济最活跃、最集中的表现形式之一。2017年，在政府和市场共同推动下，我国电子商务发展更加注重效率、质量和创新，取得了一系列新的进展，在壮大数字经济、共建“一带一路”、助力乡村振兴、带动创新创业、促进经济转型升级等诸多方面发挥了重要作用，成为我国经济增长的新动力。

2017年，我国电子商务交易规模继续扩大，并保持高速增长态势。国家统计局数据显示，2017年全国电子商务交易额达29.16万亿元，同比增长11.7%；网上零售额7.18万亿元，同比增长32.2%。我国电子商务优势进一步扩大，网络零售规模全球最大、产业创新活力世界领先。数据显示，截止2017年底，全国网络购物用户规模达5.33亿，同比增长14.3%；非银行支付机构发生网络支付金额达143.26万亿元，同比增长44.32%；全国快递服务企业业务量累计完成400.6亿件，同比增长28%；电子商务直接从业人员和间接带动就业达4250万人。



2016天猫全天成交额记录



根据阿里巴巴公布的实时数据，截至2016年11月11日24时，2016天猫双11全球狂欢节总交易额超1207亿，进入一天千亿时代!!!

双11全球购物狂欢节，从2014年5000万元的成交额，到2015年的912亿元，到2016突破一天千亿1207亿，到2017年的1682亿，再到2018年的2135亿元。

## 电商行业技术特点

* 海量数据
* 业务复杂
* 技术新
* 技术范围广
* 分布式
* 高并发，集群，负载均衡，高可用
* 系统安全

**分布式(软件级别)**：一件事情，拆开来做

一个业务分拆多个子业务，分成不同的工程，部署在不同的服务器上

**集群(硬件级别)**：同样的事情，大家一起做

集群是个物理形态，同一个业务，部署在多个服务器上

**先有集群然后才发展出分布式,这是进一步细化,减轻服务器压力,同时也可以对热门资源进行偏斜的资源分配,增加热门资源的服务器分配.**

## 主要电商模式

### IT界的两种行业

（IT）传统行业：OA，CRM，ERP，BOS物流、医药采购管理系统，银行系统，财务软件

（IT）互联网行业：电商（淘宝、京东、畅购），搜索（百度、谷歌），社交（人人网，天涯论坛），新浪微博，门户网站（新浪，搜狐），12306.cn

### B2B--企业对企业

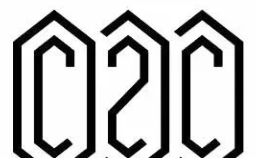
B2B （ Business to Business）是指进行电子商务交易的供需双方都是商家（或企业、公司），她（他）们使用了互联网的技术或各种商务网络平台，完成商务交易的过程。电子商务是现代 B2B marketing的一种具体主要的表现形式。



案例：阿里巴巴、慧聪网

### C2C--个人对个人

C2C即 Customer（Consumer） to Customer（Consumer），意思就是消费者个人间的电子商务行为。比如一个消费者有一台电脑，通过网络进行交易，把它出售给另外一个消费者，此种交易类型就称为C2C电子商务。



案例：淘宝、易趣、瓜子二手车

### B2C--企业对个人

B2C是Business-to-Customer的缩写，而其中文简称为“商对客”。“商对客”是电子商务的一种模式，也就是通常说的直接面向消费者销售产品和服务商业零售模式。这种形式的电子商务一般以网络零售业为主，主要借助于互联网开展在线销售活动。B2C即企业通过互联网为消费者提供一个新型的购物环境——网上商店，消费者通过网络在网上购物、网上支付等消费行为。



案例：唯品会、乐蜂网

### C2B--个人对企业

C2B（Consumer to Business，即消费者到企业），是互联网经济时代新的商业模式。这一模式改变了原有生产者（企业和机构）和消费者的关系，是一种消费者贡献价值（Create Value）， 企业和机构消费价值（Consume Value）。

C2B模式和我们熟知的供需模式（DSM, Demand Supply Model）恰恰相反，真正的C2B 应该先有消费者需求产生而后有企业生产，即先有消费者提出需求，后有生产企业按需求组织生产。通常情况为消费者根据自身需求定制产品和价格，或主动参与产品设计、生产和定价，产品、价格等彰显消费者的个性化需求，生产企业进行定制化生产。



案例：海尔商城、 尚品宅配

### O2O--线上到线下

O2O即Online To Offline（在线离线/线上到线下），是指将线下的商务机会与互联网结合，让互联网成为线下交易的平台，这个概念最早来源于美国。O2O的概念非常广泛，既可涉及到线上，又可涉及到线下,可以通称为O2O。主流商业管理课程均对O2O这种新型的商业模式有所介绍及关注。



案例：美团、饿了么

### F2C--工厂到个人

F2C指的是Factory to customer，即从厂商到消费者的电子商务模式。



### B2B2C --企业-企业-个人

B2B2C是一种电子商务类型的网络购物商业模式，B是BUSINESS的简称，C是CUSTOMER的简称，第一个B指的是商品或服务的供应商，第二个B指的是从事电子商务的企业，C则是表示消费者。

第一个BUSINESS，并不仅仅局限于品牌供应商、影视制作公司和图书出版商，任何的商品供应商或服务供应商都能可以成为第一个BUSINESS；第二B是B2B2C模式的电子商务企业，通过统一的经营管理对商品和服务、消费者终端同时进行整合，是广大供应商和消费者之间的桥梁，为供应商和消费者提供优质的服务，是互联网电子商务服务供应商。C表示消费者，在第二个B构建的统一电子商务平台购物的消费者；

B2B2C的来源于目前的B2B、B2C模式的演变和完善，把B2C和C2C完美地结合起来，通过B2B2C模式的电子商务企业构建自己的物流供应链系统，提供统一的服务。

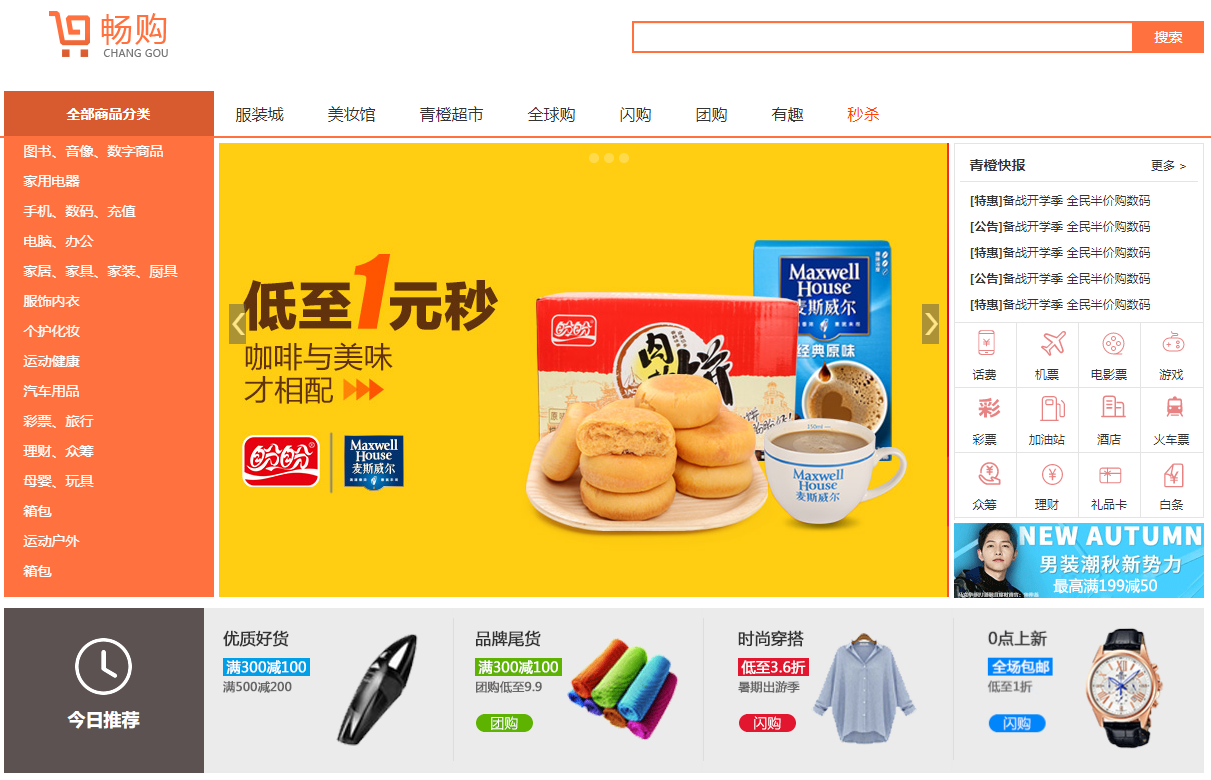
案例：京东商城、天猫商城

注：我们《畅购电商系统开发》课程采用B2C模式，之后的项目实战《畅购电商系统开发实战》采用B2B2C模式。

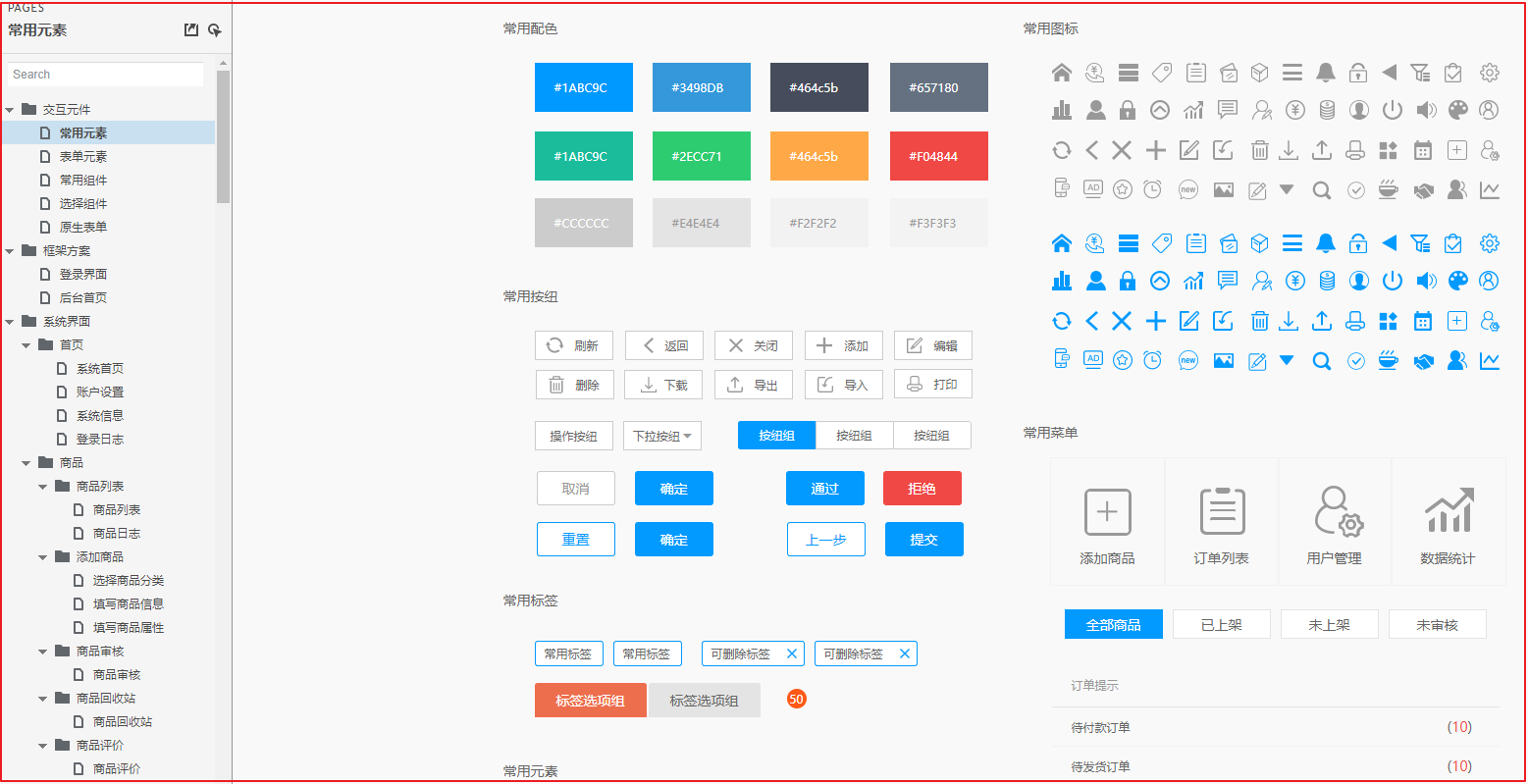
# 畅购-需求分析与系统设计

## 需求分析

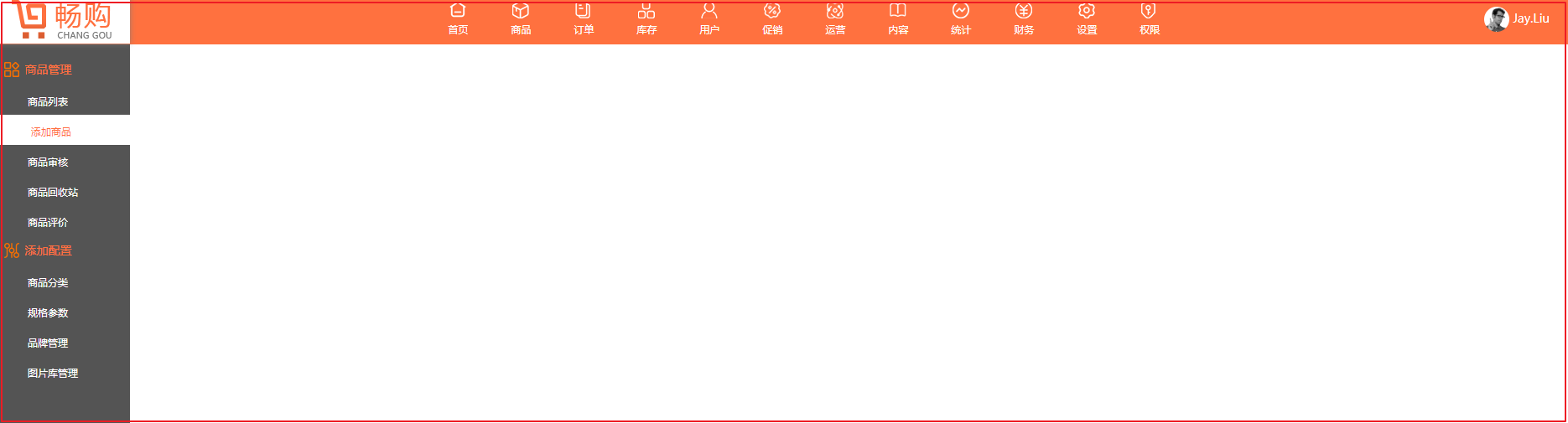
网站前台静态原型演示，打开资料\页面\前台\project-changgou-portal-fis3-master，首页index.html



网站管理后台静态原型演示:[http://czpm.itcast.cn/青橙后台/#g=1&p=后台首页](http://czpm.itcast.cn/青橙后台/%23g=1&p=后台首页)



打开资料\页面\后台\project-changgou-cmm-fis3-master\pages,首页all-medical-main.html



## 系统设计

畅购商城属于B2C电商模式，运营商将自己的产品发布到网站上，会员注册后，在网站上将商品添加到购物车，并且下单，完成线上支付，用户还可以参与秒杀抢购。

### 前后端分离

网站后台的部分采用前后端分离方式。

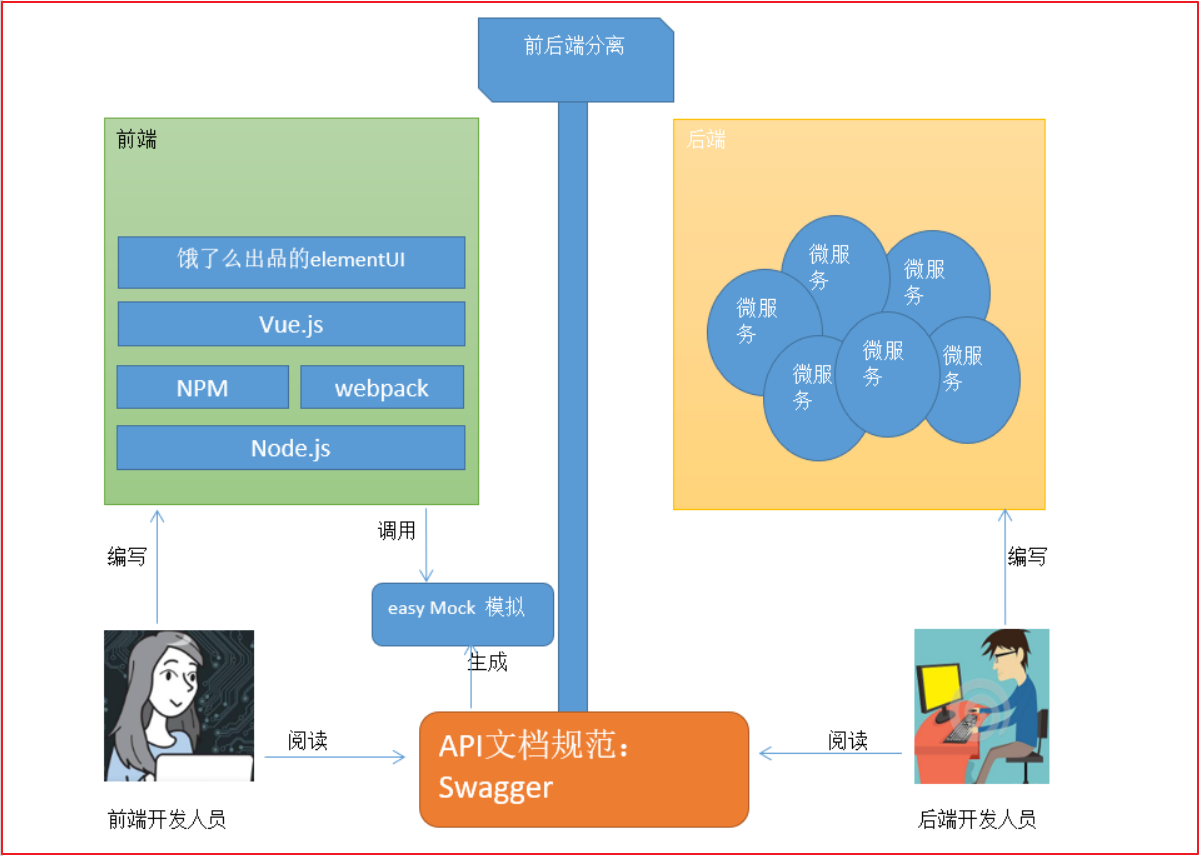
以前的JavaWeb项目大多数都是java程序员又当爹又当妈，又搞前端，又搞后端。随着时代的发展，渐渐的许多大中小公司开始把前后端的界限分的越来越明确，前端工程师只管前端的事情，后端工程师只管后端的事情。正所谓术业有专攻，一个人如果什么都会，那么他毕竟什么都不精。

**对于后端java工程师**：

把精力放在设计模式，spring+springmvc，linux，mysql事务隔离与锁机制，mongodb，http/tcp，多线程，分布式架构，弹性计算架构，微服务架构，java性能优化，以及相关的项目管理等等。

**对于前端工程师**：

把精力放在html5，css3，vuejs，webpack，nodejs，Google V8引擎，javascript多线程，模块化，面向切面编程，设计模式，浏览器兼容性，性能优化等等。

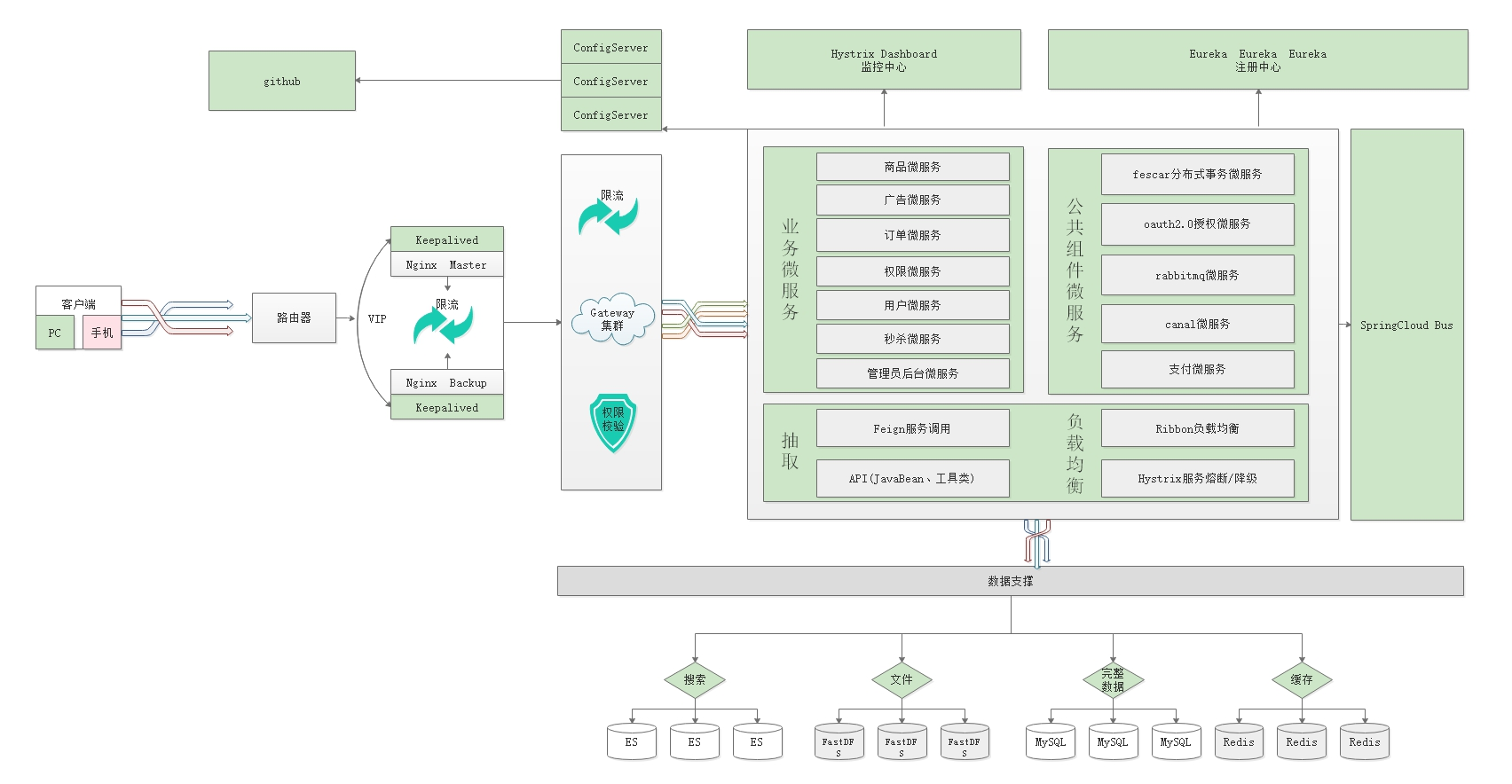


我们在本课程中提供与项目课程配套的管理后台的前端代码，但是不讲解前端的内容。这样我们会将更多的精力放在\*\*后端代码\*\*的开发上！

### 技术架构



### 系统架构图



# 畅购-框架搭建

## 环境准备

（1）VMware Workstation Pro安装centos7 镜像

（2）安装docker

（3）拉取mySQL镜像，并创建容器

（4）客户端连接mysql容器，建库建表（建库建表语句在资源文件夹中提供）

（5）课程使用已经装好的虚拟机：\资料\虚拟机

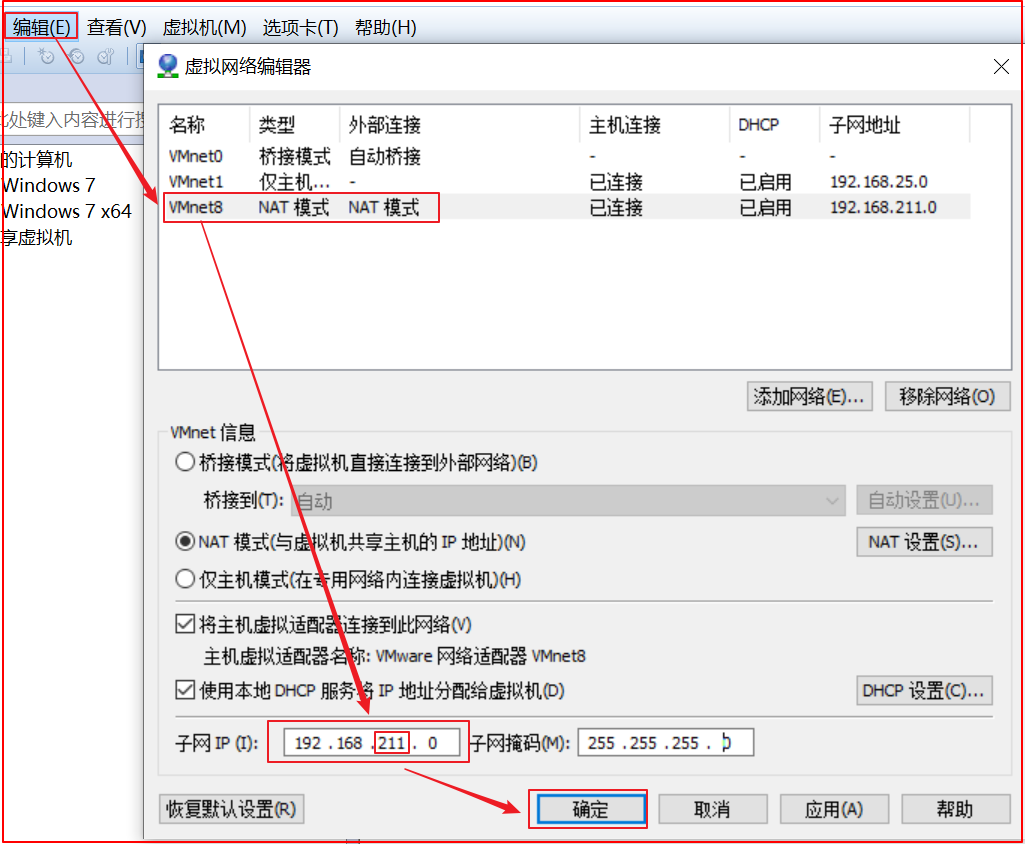
虚拟机数据：

* 虚拟机IP:192.168.211.132
* 虚拟机账号：root 密码：123456
* 数据库端口:3306
* 数据库账号：root 密码：123456

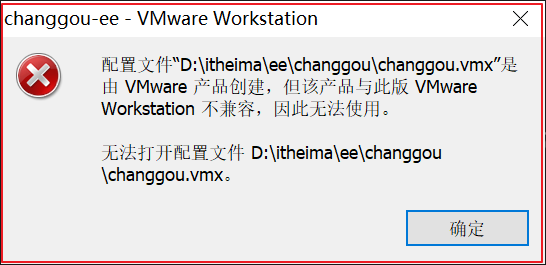
数据库脚本：资料\数据库脚本



**虚拟机设置**



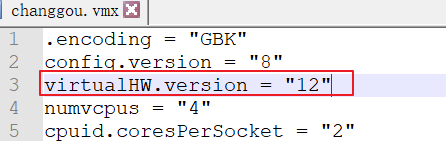
如果发生此异常



找到磁盘上该虚拟机的配置文件.vmx文件



找到：virtualHW.version= "12" ，这里的12可能是其他数字，请检查你的虚拟机版本是多少，将这里的数字修改为和你VMware workstation虚拟机相同的版本，比如你的是：10.01，就修改为10，也就是修改为大版本号，保存即可。



## 项目结构说明



结构说明：

**changgou-gateway**

网关模块，根据网站的规模和需要，可以将综合逻辑相关的服务用网关路由组合到一起。在这里还可以做鉴权和限流相关操作。

**changgou-service**

微服务模块，该模块用于存放所有独立的微服务工程。

**changgou-service-api**

对应工程的JavaBean、Feign、以及Hystrix配置，该工程主要对外提供依赖。

**changgou-transaction-fescar**

分布式事务模块，将分布式事务抽取到该工程中，任何工程如需要使用分布式事务，只需依赖该工程即可。

**changgou-web**

web服务工程，对应功能模块如需要调用多个微服务，可以将他们写入到该模块中，例如网站后台、网站前台等

## 公共工程搭建

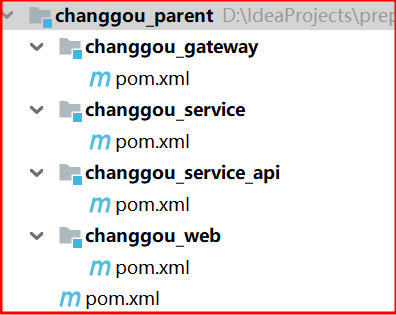
### 父工程搭建

创建父工程 changgou\_parent ，pom.xml文件中加入相关依赖，并删除src文件夹。

*<?***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**>  
  
 <**groupId**>com.changgou</**groupId**>  
 <**artifactId**>changgou\_parent</**artifactId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
  
 <**parent**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-parent</**artifactId**>  
 <**version**>2.1.7.RELEASE</**version**>  
 </**parent**>  
  
 *<!--跳过测试-->* <**properties**>  
 <**skipTests**>true</**skipTests**>  
 </**properties**>  
  
 *<!--依赖包-->* <**dependencies**>  
 *<!--测试包-->* <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-test</**artifactId**>  
 </**dependency**>  
 *<!--fastjson-->* <**dependency**>  
 <**groupId**>com.alibaba</**groupId**>  
 <**artifactId**>fastjson</**artifactId**>  
 <**version**>1.2.51</**version**>  
 </**dependency**>  
  
 *<!--swagger文档-->* <**dependency**>  
 <**groupId**>io.springfox</**groupId**>  
 <**artifactId**>springfox-swagger2</**artifactId**>  
 <**version**>2.6.1</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>io.springfox</**groupId**>  
 <**artifactId**>springfox-swagger-ui</**artifactId**>  
 <**version**>2.6.1</**version**>  
 </**dependency**>  
 *<!--  
 http://localhost:9011/swagger-ui.html  
 -->* </**dependencies**>  
 <**dependencyManagement**>  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-dependencies</**artifactId**>  
 <**version**>Greenwich.SR1</**version**>  
 <**type**>pom</**type**>  
 <**scope**>import</**scope**>  
 </**dependency**>  
 </**dependencies**>  
 </**dependencyManagement**>  
</**project**>

### 其他公共模块搭建

创建**changgou\_gateway、changgou\_service、changgou\_service\_api、changgou\_web**工程，工程全部为pom工程，并将所有工程的src文件删除。目录如下：



## Eureka微服务搭建

### pom.xml依赖

创建模块 changgou\_eureka ，pom.xml引入依赖

*<?***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"**>  
 <**parent**>  
 <**artifactId**>changgou\_parent</**artifactId**>  
 <**groupId**>com.changgou</**groupId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
 </**parent**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
  
 <**artifactId**>changgou\_eureka</**artifactId**>  
  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-netflix-eureka-server</**artifactId**>  
 </**dependency**>  
 </**dependencies**>  
</**project**>

### appliation.yml配置

**server**:  
 **port**: 7001 *#启动端口***spring**:  
 **application**:  
 **name**: eureka-server *#应用程序名***eureka**:  
 **instance**:  
 **hostname**: 127.0.0.1  
 **client**:  
 **register-with-eureka**: **false** *#是否将自己注册到eureka中* **fetch-registry**: **false** *#是否从eureka中获取信息* **service-url**:  
 **defaultZone**: http://${**eureka.instance.hostname**}:${**server.port**}/eureka/

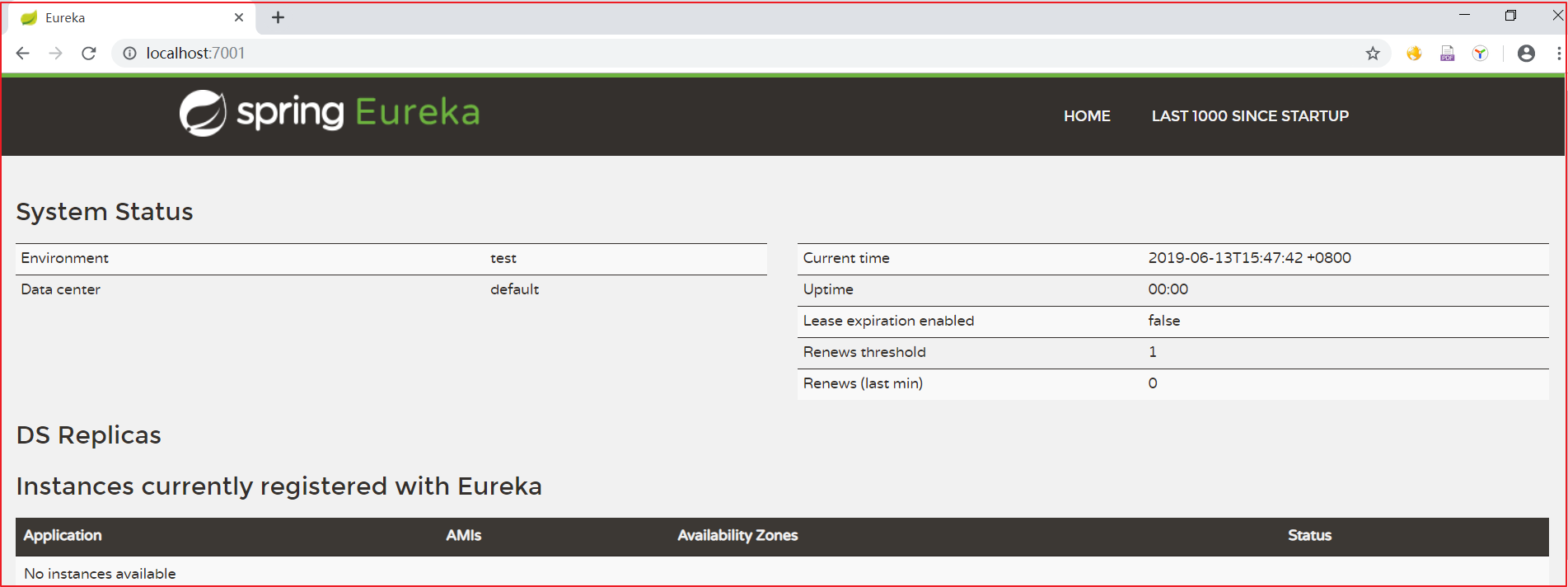
### 启动引导类

创建包com.changgou包下创建启动类EurekaServerApplication，代码如下：

*/\*\*  
 \** ***@author*** *Steven  
 \** ***@description*** *com.changgou  
 \*/*@SpringBootApplication  
@EnableEurekaServer  
**public class** EurekaServerApplication {  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(EurekaServerApplication.**class**, args);  
 }  
}

### 启动与测试

启动引导类，测试访问<http://localhost:7001/> ,效果如下：



## 公共模块搭建

### changgou\_common

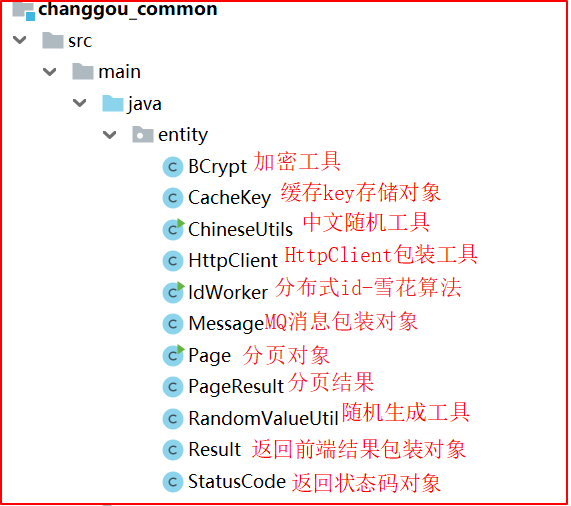
在changgou\_parent中，创建公共子模块，pom.xml引入依赖

*<?***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"**>  
 <**parent**>  
 <**artifactId**>changgou\_parent</**artifactId**>  
 <**groupId**>com.changgou</**groupId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
 </**parent**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
 <**artifactId**>changgou\_common</**artifactId**>  
  
 <**dependencies**>  
 *<!--web起步依赖-->* <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-web</**artifactId**>  
 </**dependency**>  
 *<!-- redis 使用-->* <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-data-redis</**artifactId**>  
 </**dependency**>  
 *<!--eureka-client-->* <**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-netflix-eureka-client</**artifactId**>  
 </**dependency**>  
 *<!--openfeign-->* <**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-openfeign</**artifactId**>  
 </**dependency**>  
 *<!--微信支付-->* <**dependency**>  
 <**groupId**>com.github.wxpay</**groupId**>  
 <**artifactId**>wxpay-sdk</**artifactId**>  
 <**version**>0.0.3</**version**>  
 </**dependency**>  
 *<!--httpclient支持-->* <**dependency**>  
 <**groupId**>org.apache.httpcomponents</**groupId**>  
 <**artifactId**>httpclient</**artifactId**>  
 </**dependency**>  
 </**dependencies**>  
</**project**>

公共子模块引入这些依赖后，其他微服务引入changgou-common后也自动引入了这些依赖

### 常用对象

创建entity包，把资料\工具类目录下的所有类复制到当前包下



## 数据访问工程搭建

创建公共模块changgou\_common\_db ，pom文件引入依赖，删除src目录

*<?***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"**>  
 <**parent**>  
 <**artifactId**>changgou\_parent</**artifactId**>  
 <**groupId**>com.changgou</**groupId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
 </**parent**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
  
 <**artifactId**>changgou\_common\_db</**artifactId**>  
  
 *<!--依赖-->* <**dependencies**>  
 *<!--对changgou-common的依赖-->* <**dependency**>  
 <**groupId**>com.changgou</**groupId**>  
 <**artifactId**>changgou\_common</**artifactId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
 </**dependency**>  
 *<!--通用mapper起步依赖-->* <**dependency**>  
 <**groupId**>tk.mybatis</**groupId**>  
 <**artifactId**>mapper-spring-boot-starter</**artifactId**>  
 <**version**>2.0.4</**version**>  
 </**dependency**>  
 *<!--MySQL数据库驱动-->* <**dependency**>  
 <**groupId**>mysql</**groupId**>  
 <**artifactId**>mysql-connector-java</**artifactId**>  
 </**dependency**>  
 *<!--mybatis分页插件-->* <**dependency**>  
 <**groupId**>com.github.pagehelper</**groupId**>  
 <**artifactId**>pagehelper-spring-boot-starter</**artifactId**>  
 <**version**>1.2.3</**version**>  
 </**dependency**>  
 </**dependencies**>  
</**project**>

这个公共模块是连接mysql数据库的公共微服务模块，所以需要连接mysql的微服务都依赖自此工程。

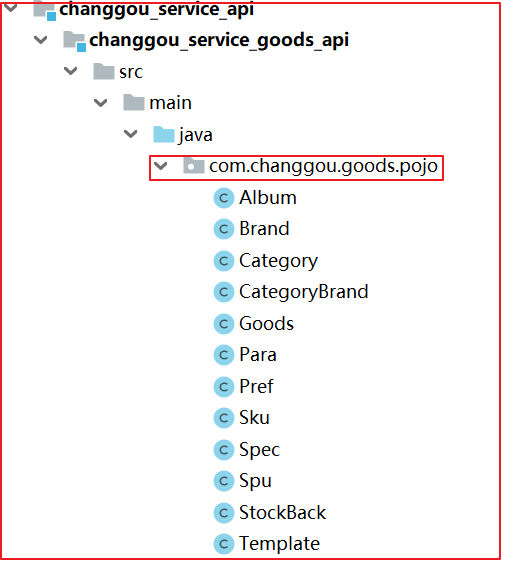
## 商品微服务搭建

商品微服务主要是实现对商品的增删改查相关操作，以及商品相关信息的增删改查。

### 公共组件工程搭建

在changgou\_service\_api创建子模块changgou\_service\_goods\_api

并将资料\javabean\changgou-service-goods-api中的Pojo导入到工程的包com.changgou.goods.pojo中。



修改父工程changgou-service-api的pom.xml，添加persistence-api和changgou-common的依赖，代码如下

<**dependencies**>  
 *<!--通用的common-->* <**dependency**>  
 <**groupId**>com.changgou</**groupId**>  
 <**artifactId**>changgou\_common</**artifactId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
 </**dependency**>  
  
 *<!--每个工程都有Pojo，都需要用到该包对应的注解-->* <**dependency**>  
 <**groupId**>javax.persistence</**groupId**>  
 <**artifactId**>persistence-api</**artifactId**>  
 <**version**>1.0</**version**>  
 <**scope**>compile</**scope**>  
 </**dependency**>  
</**dependencies**>

### 微服务工程搭建

修改changgou\_service的pom.xml引入changgou\_common\_db的依赖，代码如下：

<**dependencies**>  
 <**dependency**>  
 <**groupId**>com.changgou</**groupId**>  
 <**artifactId**>changgou\_common\_db</**artifactId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
 </**dependency**>  
</**dependencies**>

在changgou\_service中创建changgou\_service\_goods ，pom.xml引入依赖

*<?***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"**>  
 <**parent**>  
 <**artifactId**>changgou\_service</**artifactId**>  
 <**groupId**>com.changgou</**groupId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
 </**parent**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
  
 <**artifactId**>changgou\_service\_goods</**artifactId**>  
  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>com.changgou</**groupId**>  
 <**artifactId**>changgou\_service\_goods\_api</**artifactId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
 </**dependency**>  
 </**dependencies**>  
</**project**>

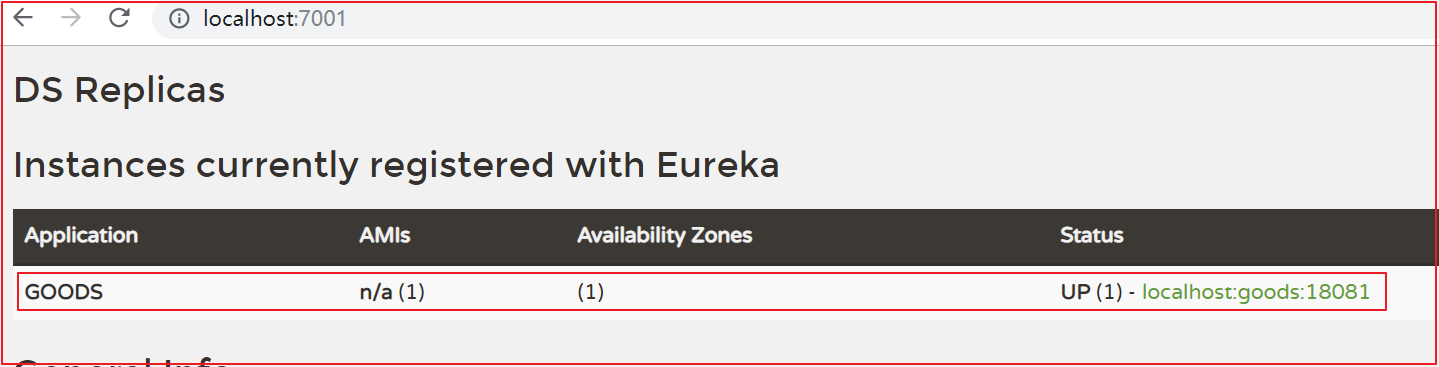
在resources下创建配置文件application.yml

**server**:  
 **port**: 18081  
**spring**:  
 **application**:  
 **name**: goods  
 **datasource**:  
 **driver-class-name**: com.mysql.cj.jdbc.Driver  
 **url**: jdbc:mysql://192.168.211.132:3306/changgou\_goods?useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC  
 **username**: root  
 **password**: 123456  
**eureka**:  
 **client**:  
 **service-url**:  
 **defaultZone**: http://127.0.0.1:7001/eureka  
 **instance**:  
 **prefer-ip-address**: **true  
feign**:  
 **hystrix**:  
 **enabled**: **true***# mybatis配置，使用通用Mapper后可以不配置  
#mybatis:  
# configuration:  
# #开启驼峰命名  
# map-underscore-to-camel-case: true  
# mapper-locations: classpath:mapper/\*Mapper.xml  
# type-aliases-package: com.changgou.goods.pojo*

在包com.changgou.goods 包下创建启动类GoodsApplication，代码如下：

*/\*\*  
 \** ***@author*** *Steven  
 \** ***@description*** *com.changgou.goods  
 \*/*@SpringBootApplication  
@EnableEurekaClient *//开启Eureka客户端服务发现*  
**public class** GoodsApplication {  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(GoodsApplication.**class**, args);  
 }  
}

启动changgou-service-goods 再访问<http://localhost:7001> 效果如下：



# 商品微服务-品牌增删改查

## 需求分析

创建商品微服务，实现对品牌表的增删改查功能。具体包括

（1）查询全部列表数据

（2）根据ID查询实体数据

（3）增加

（4）修改

（5）删除

（6）条件查询

（7）分页查询

（8）分页+条件查询

（9）公共异常处理

## 表结构分析

品牌表：tb\_brand

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名称 | 字段含义 | 字段类型 | 字段长度 | 备注 |
| id | 品牌id | INT |  |  |
| name | 品牌名称 | VARCHAR |  |  |
| image | 品牌图片地址 | VARCHAR |  |  |
| letter | 品牌的首字母 | CHAR |  |  |
| seq | 排序 | INT |  |  |

## Dao集成通用Mapper

### 创建BrandMapper

在changgou\_service\_goods微服务下创建com.changgou.goods.dao.BrandMapper接口，代码如下

*/\*\*  
 \* 品牌信息持久化接口  
 \** ***@author*** *Steven  
 \** ***@description*** *com.changgou.goods.dao  
 \*/***public interface** BrandMapper **extends** Mapper<Brand> {  
}

### 修改启动引导类

修改changgou\_service\_goods微服务下的启动引导类，代码如下

*/\*\*  
 \** ***@author*** *Steven  
 \** ***@description*** *com.changgou.goods  
 \*/*@SpringBootApplication  
@EnableEurekaClient *//开启Eureka客户端服务发现  
//开启通用Mapper的dao包扫描-注意使用tk.mybatis.spring.annotation包下的MapperScan*@MapperScan(basePackages = **"com.changgou.goods.dao"**)  
**public class** GoodsApplication {  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(GoodsApplication.**class**, args);  
 }  
}

## 查询全部品牌列表数据

### 业务层

创建com.changgou.goods.service.BrandService接口，代码如下

**public interface** BrandService {  
  
 */\*\*\*  
 \* 查询所有品牌  
 \** ***@return*** *\*/* List<Brand> findAll();  
}

创建com.changgou.goods.service.impl.BrandServiceImpl实现类，代码如下：

@Service  
**public class** BrandServiceImpl **implements** BrandService {  
  
 @Autowired  
 **private** BrandMapper **brandMapper**;  
 */\*\*  
 \* 全部数据  
 \** ***@return*** *\*/* @Override  
 **public** List<Brand> findAll(){  
 **return brandMapper**.selectAll();  
 }  
}

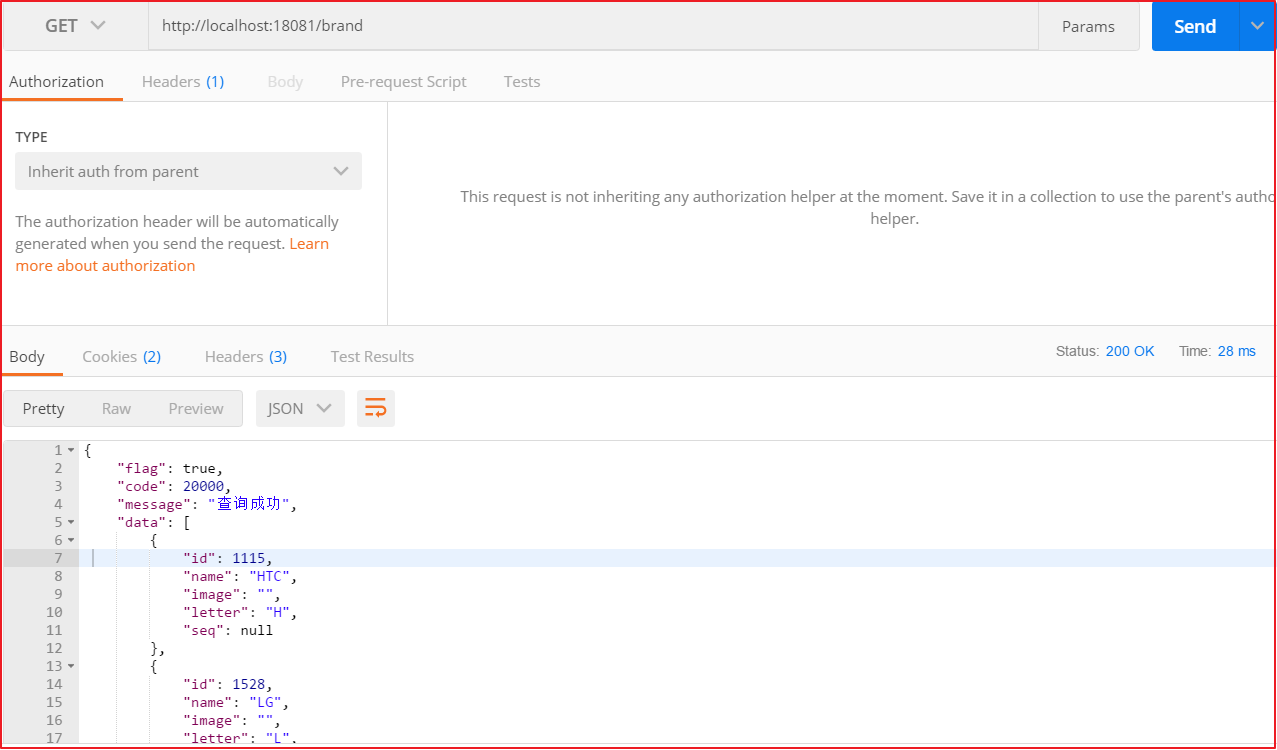
### 控制层

控制层 com.changgou.goods包下创建controller包 ，包下创建类

@RestController  
@RequestMapping(**"brand"**)  
*//用于解决js跨域问题-在微服务中，一般都需要此注解*@CrossOrigin  
**public class** BrandController {  
 @Autowired  
 **private** BrandService **brandService**;  
  
 */\*\*  
 \* 查询所有品牌  
 \** ***@return*** *\*/* @GetMapping  
 **public** Result<List<Brand>> findAll(){  
 List<Brand> brandList = **brandService**.findAll();  
 **return new** Result<List<Brand>>(**true**, StatusCode.***OK***,**"查询品牌信息成功"**,brandList);  
 }  
}

### 测试

测试：<http://localhost:18081/brand>



## 根据ID查询品牌

### 业务层

修改com.changgou.goods.service.BrandService接口，添加根据ID查询品牌数据方法，代码如下：

*/\*\*  
 \* 根据ID查询  
 \** ***@param id*** *\** ***@return*** *\*/*Brand findById(Integer id);

修改com.changgou.goods.service.impl.BrandServiceImpl新增方法，代码如下：

*/\*\*  
 \* 根据ID查询  
 \** ***@return*** *\*/*@Override  
**public** Brand findById(Integer id){  
 **return brandMapper**.selectByPrimaryKey(id);  
}

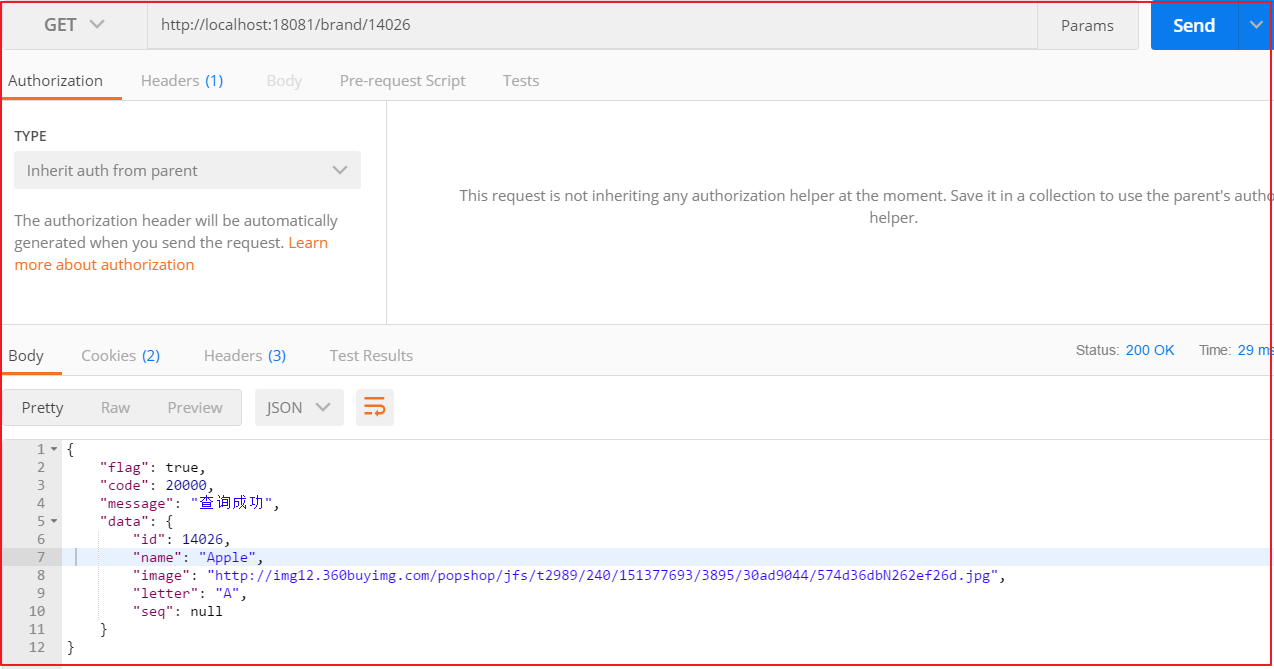
### 控制层

BrandController添加方法

*/\*\*\*  
 \* 根据ID查询品牌数据  
 \** ***@param id*** *\** ***@return*** *\*/*@GetMapping(**"{id}"**)  
**public** Result<Brand> findById(@PathVariable Integer id){  
 *//根据ID查询* Brand brand = **brandService**.findById(id);  
 **return new** Result<Brand>(**true**,StatusCode.***OK***,**"查询品牌成功"**,brand);  
}

### 测试

<http://localhost:18081/brand/14026>



## 新增品牌

### 业务层

修改com.changgou.goods.service.BrandService，新增方法

*/\*\*\*  
 \* 新增品牌  
 \** ***@param brand*** *\*/***void** add(Brand brand);

修改com.changgou.goods.service.impl.BrandServiceImpl，新增增加品牌方法代码如下：

*/\*\*  
 \* 增加  
 \** ***@param brand*** *\*/*@Override  
**public void** add(Brand brand){  
 */\*\*  
 \* insert与insertSelective区别  
 \* insert不管入参对象的属性有没有传入，直接插入所有字段  
 \* insertSelective只会插入，入参里有值的字段  
 \*/* **brandMapper**.insertSelective(brand);  
}

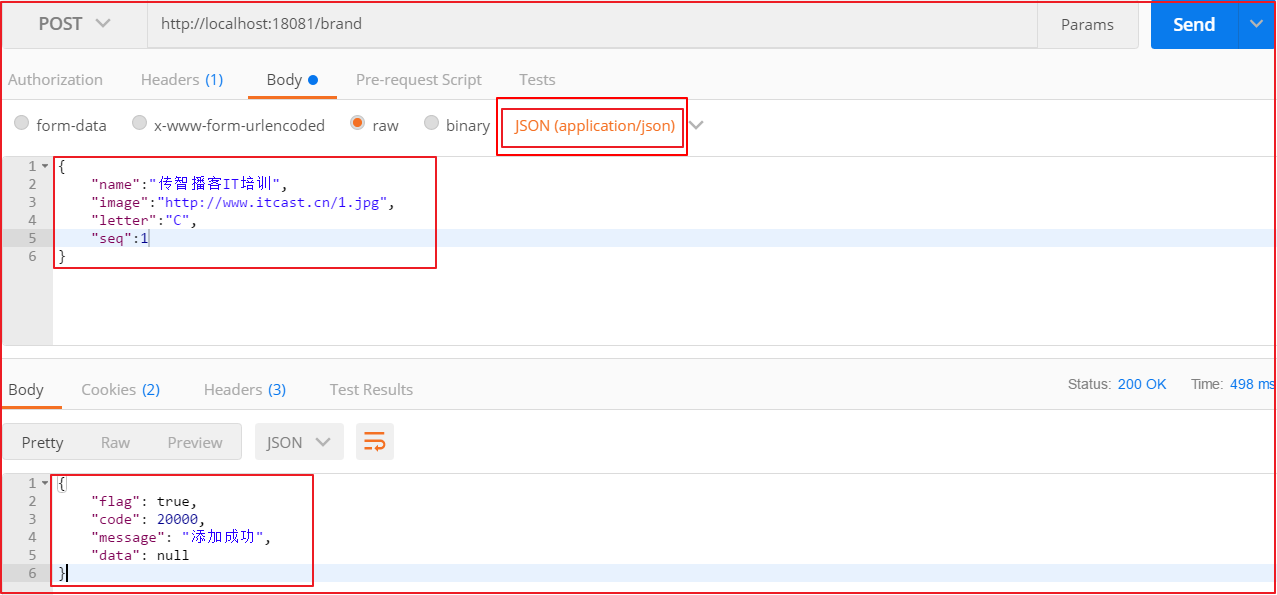
### 控制层

BrandController新增方法

*/\*\*  
 \* 添加品牌  
 \** ***@param brand*** *\** ***@return*** *\*/*@PostMapping  
**public** Result add(@RequestBody Brand brand){  
 **brandService**.add(brand);  
 **return new** Result(**true**,StatusCode.***OK***,**"保存品牌成功"**);  
}

### 测试

<http://localhost:18081/brand>



## 修改品牌

### 业务层

需改com.changgou.goods.service.BrandService,添加修改品牌方法，代码如下：

*/\*\*\*  
 \* 修改品牌数据  
 \** ***@param brand*** *\*/***void** update(Brand brand);

修改com.changgou.goods.service.impl.BrandServiceImpl，添加修改品牌方法，代码如下：

*/\*\*  
 \* 修改  
 \** ***@param brand*** *\*/*@Override  
**public void** update(Brand brand){  
 **brandMapper**.updateByPrimaryKeySelective(brand);  
}

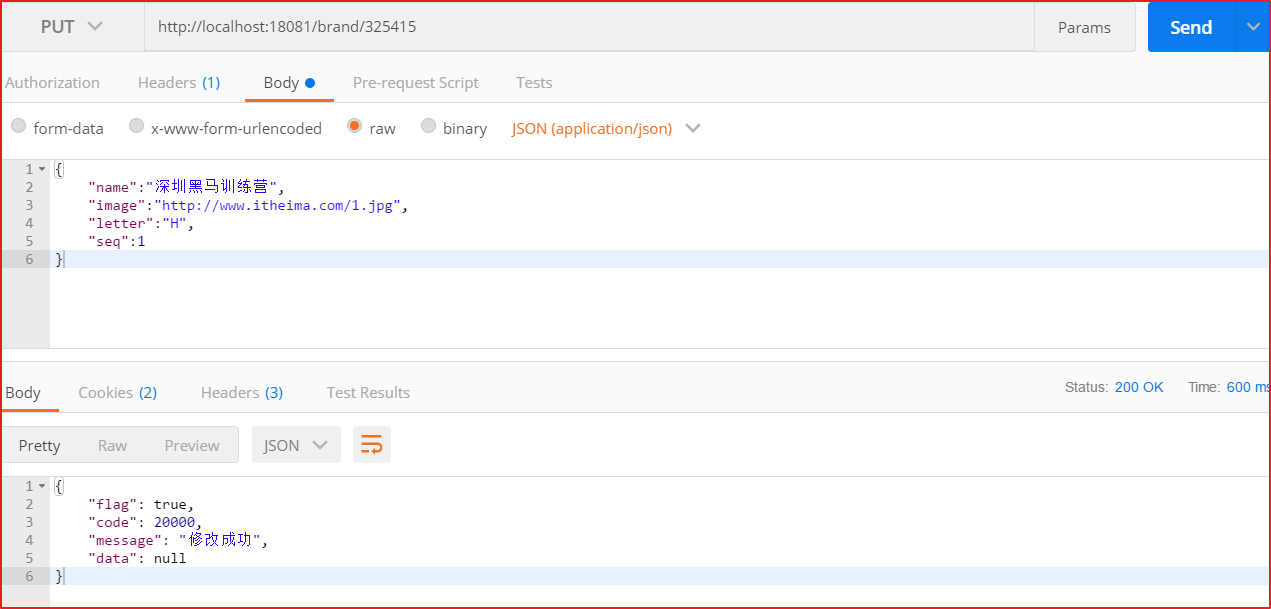
### 控制层

BrandController添加方法

*/\*\*  
 \* 修改品牌数据  
 \** ***@param brand*** *\** ***@param id*** *\** ***@return*** *\*/*@PutMapping(**"{id}"**)  
**public** Result update(@RequestBody Brand brand,@PathVariable Integer id){  
 *//设置ID* brand.setId(id);  
 *//修改数据* **brandService**.update(brand);  
 **return new** Result(**true**,StatusCode.***OK***,**"修改品牌成功"**);  
}

### 测试

<http://localhost:18081/brand/325415>



## 删除品牌

### 业务层

修改com.changgou.goods.service.BrandService，添加删除品牌方法，代码如下：

*/\*\*\*  
 \* 删除品牌  
 \** ***@param id*** *\*/***void** delete(Integer id);

修改com.changgou.goods.service.impl.BrandServiceImpl，新增删除品牌方法，代码如下：

*/\*\*  
 \* 删除  
 \** ***@param id*** *\*/*@Override  
**public void** delete(Integer id){  
 **brandMapper**.deleteByPrimaryKey(id);  
}

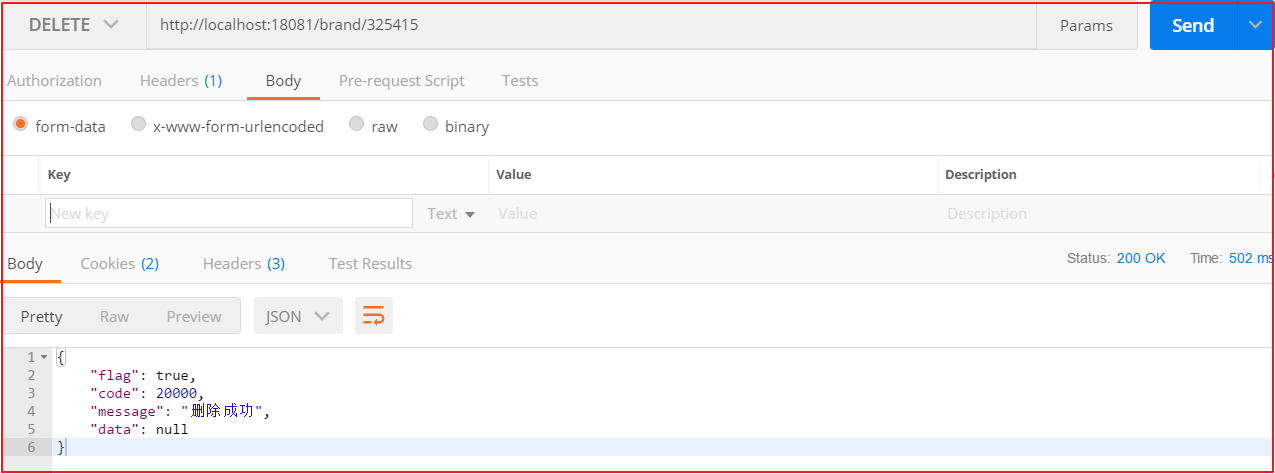
### 控制层

BrandController新增方法

*/\*\*\*  
 \* 根据ID删除品牌数据  
 \** ***@param id*** *\** ***@return*** *\*/*@DeleteMapping(**"{id}"**)  
**public** Result delete(@PathVariable Integer id){  
 **brandService**.delete(id);  
 **return new** Result(**true**,StatusCode.***OK***,**"删除成功"**);  
}

### 测试

<http://localhost:18081/brand/325415>



## 品牌列表条件查询

### 业务层

修改com.changgou.goods.service.BrandService，增加根据条件搜索品牌方法，代码如下：

*/\*\*\*  
 \* 多条件搜索品牌方法  
 \** ***@param brand*** *\** ***@return*** *\*/*List<Brand> findList(Brand brand);

修改com.changgou.goods.service.impl.BrandServiceImpl，添加根据多条件搜索品牌方法的实现，代码如下：

*/\*\*  
 \* 条件查询  
 \** ***@param brand*** *\** ***@return*** *\*/*@Override  
**public** List<Brand> findList(Brand brand){  
 *//构建查询条件* Example example = createExample(brand);  
 *//根据构建的条件查询数据* **return brandMapper**.selectByExample(example);  
}  
  
*/\*\*  
 \* 品牌查询条件构建方法  
 \** ***@param brand*** *\** ***@return*** *\*/***private** Example createExample(Brand brand) {  
 *//构建查询条件* Example example = **new** Example(Brand.**class**);  
 *//创建条件构建器* Example.Criteria criteria = example.createCriteria();  
 **if**(brand!=**null**){  
 *// 品牌名称* **if**(StringUtils.*isNotEmpty*(brand.getName())){  
 criteria.andLike(**"name"**,**"%"**+brand.getName()+**"%"**);  
 }  
 *// 品牌的首字母* **if**(StringUtils.*isNotEmpty*(brand.getLetter())){  
 criteria.andEqualTo(**"letter"**,brand.getLetter());  
 }  
 }  
 **return** example;  
}

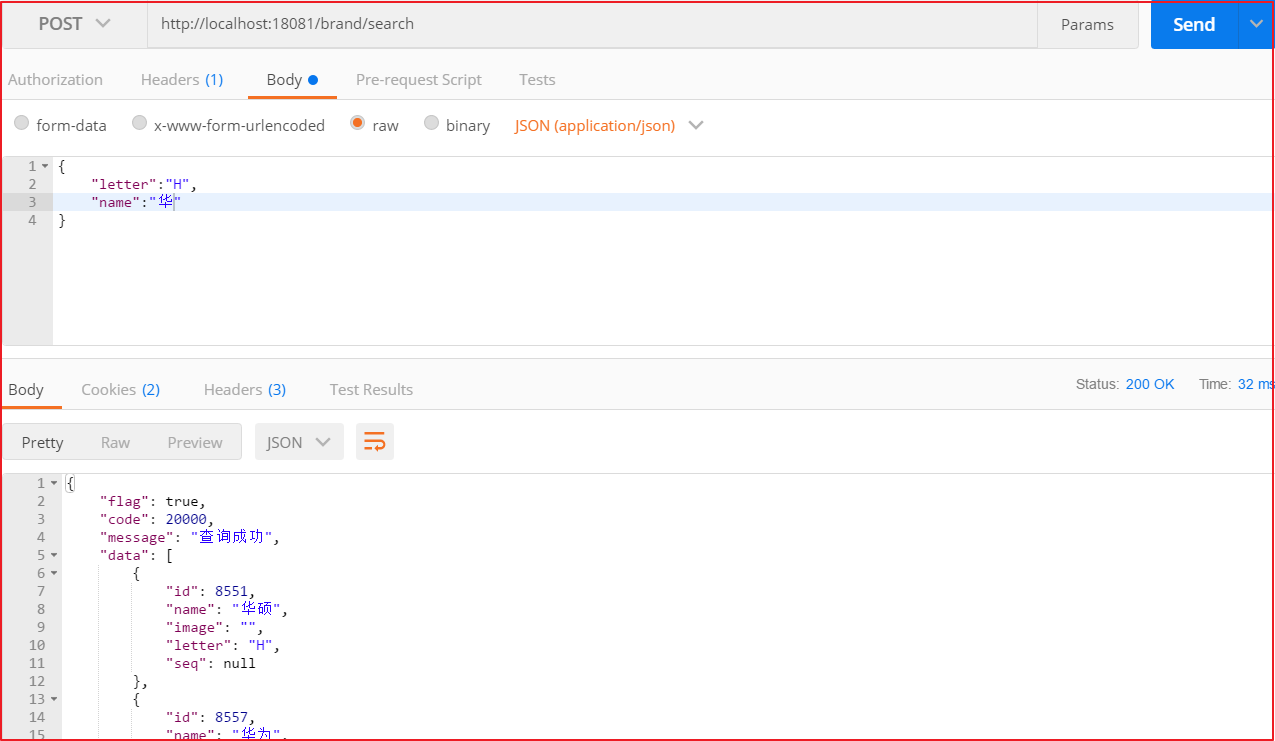
### 控制层

BrandController新增方法

*/\*\*\*  
 \* 多条件搜索品牌数据  
 \** ***@param brand*** *\** ***@return*** *\*/*@PostMapping(value = **"search"** )  
**public** Result<List<Brand>> findList(@RequestBody(required = **false**) Brand brand){  
 List<Brand> list = **brandService**.findList(brand);  
 **return new** Result<List<Brand>>(**true**,StatusCode.***OK***,**"查询成功"**,list);  
}

### 测试

<http://localhost:18081/brand/search>



## 品牌列表分页查询

### 业务层

修改com.changgou.goods.service.BrandService添加分页方法，代码如下：

*/\*\*\*  
 \* 分页查询  
 \** ***@param page*** *\** ***@param size*** *\** ***@return*** *\*/*PageInfo<Brand> findPage(**int** page, **int** size);

修改com.changgou.goods.service.impl.BrandServiceImpl添加分页方法实现，代码如下：

@Override  
**public** PageInfo<Brand> findPage(**int** page, **int** size) {  
 *//设置分页条件* PageHelper.*startPage*(page, size);  
 *//查询数据* List<Brand> brandList = **brandMapper**.selectAll();  
 *//封装并返回分页对象* **return new** PageInfo<Brand>(brandList);  
}

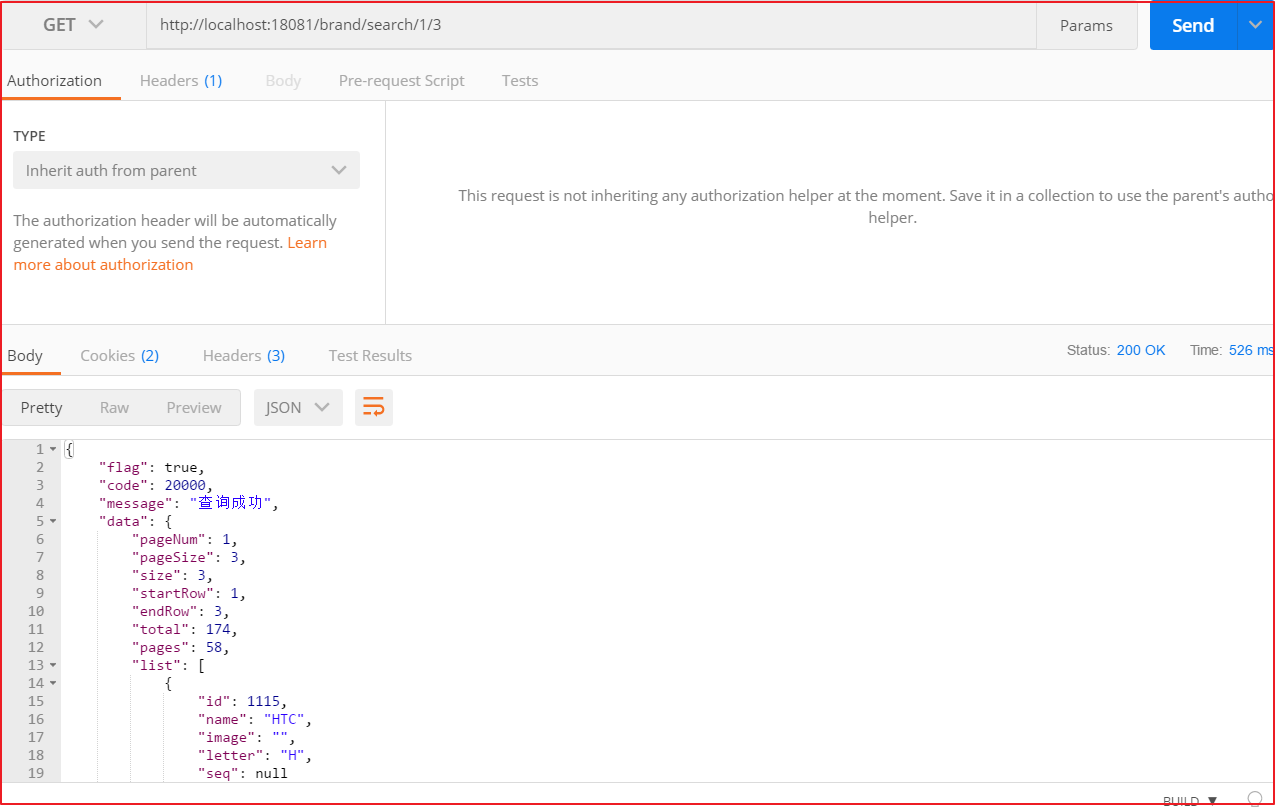
### 控制层

BrandController新增方法

*/\*\*\*  
 \* 分页搜索实现  
 \** ***@param page****:当前页  
 \** ***@param size****:每页显示多少条  
 \** ***@return*** *\*/*@GetMapping(value = **"search/{page}/{size}"** )  
**public** Result<PageInfo> findPage(@PathVariable **int** page, @PathVariable **int** size){  
 *//分页查询* PageInfo<Brand> pageInfo = **brandService**.findPage(page, size);  
 **return new** Result<PageInfo>(**true**,StatusCode.***OK***,**"查询成功"**,pageInfo);  
}

### 测试

<http://localhost:18081/brand/search/1/3>



## 品牌列表条件+分页查询

### 业务层

修改com.changgou.goods.service.BrandService，增加多条件分页查询方法，代码如下：

*/\*\*  
 \* 条件+分页查询  
 \** ***@param brand*** *查询条件  
 \** ***@param page*** *页码  
 \** ***@param size*** *页大小  
 \** ***@return*** *分页结果  
 \*/*PageInfo<Brand> findPage(Brand brand, **int** page, **int** size);

修改com.changgou.goods.service.impl.BrandServiceImpl，添加多条件分页查询方法代码如下：

@Override  
**public** PageInfo<Brand> findPage(Brand brand, **int** page, **int** size) {  
 *//设置分页条件* PageHelper.*startPage*(page, size);  
 *//分页查询数据列表-方式一  
 //List<Brand> brandList = brandMapper.select(brand);  
  
 //方式二：  
 //构建查询条件* Example example = **new** Example(Brand.**class**);  
 Example.Criteria criteria = example.createCriteria();  
 **if**(brand != **null**){  
 *//" " isNotBlank == false isNotEmpty == true* **if** (StringUtils.*isNotBlank*(brand.getName())) {  
 criteria.andLike(**"name"**, **"%"** + brand.getName() + **"%"**);  
 }  
 }  
  
 List<Brand> brandList = **brandMapper**.selectByExample(example);  
  
 *//总记录数* PageInfo<Brand> info = **new** PageInfo<>(brandList);  
 **return** info;  
}

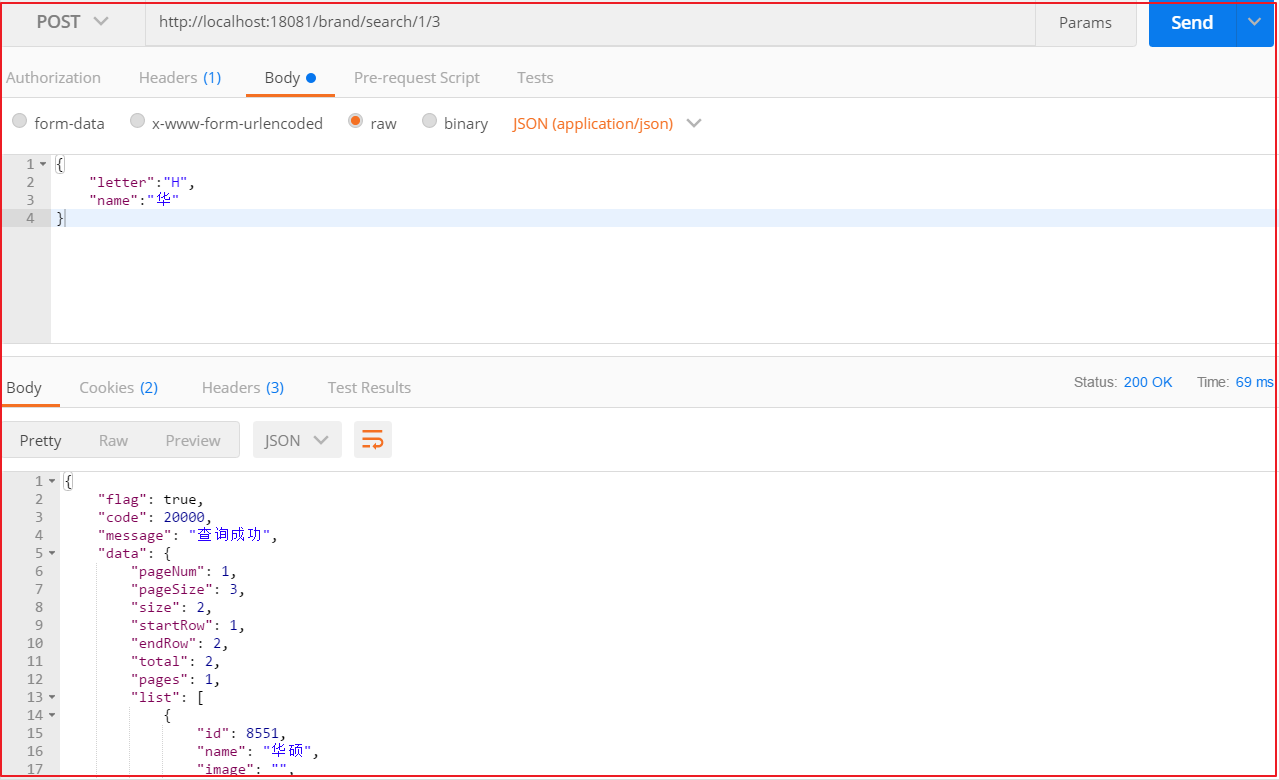
### 控制层

BrandController新增方法

*/\*\*\*  
 \* 条件+分页搜索实现**\** ***@return*** *\*/*@PostMapping(value = **"/search/{page}/{size}"** )  
**public** Result<PageInfo> findPage(@RequestBody(required = **false**) Brand brand,  
 @PathVariable **int** page, @PathVariable **int** size){  
 *//执行搜索* PageInfo<Brand> pageInfo = **brandService**.findPage(brand, page, size);  
 **return new** Result(**true**,StatusCode.***OK***,**"查询成功"**,pageInfo);  
}

### 测试

<http://localhost:18081/brand/search/1/3>



## 公共异常处理

为了使我们的代码更容易维护，我们创建一个类集中处理异常,该异常类可以创建在changgou\_common工程中，创建com.changgou.framework.exception.BaseExceptionHandler，代码如下：

*/\*\*  
 \* 公共异常处理类  
 \** ***@author*** *Steven  
 \** ***@description*** *com.changgou.framework.exception  
 \*/*@ControllerAdvice  
**public class** BaseExceptionHandler {  
 */\*\*\*  
 \* 异常处理**\*/* @ExceptionHandler(value = Exception.**class**)  
 @ResponseBody  
 **public** Result error(Exception e) {  
 e.printStackTrace();  
 **return new** Result(**false**, StatusCode.***ERROR***, e.getMessage());  
 }  
}

介绍：@ControllerAdvice注解，全局捕获异常类，只要方法上有@RequestMapping，所有的异常都会被捕获。

备注：如果要想此类生效，记得要把当前工程的启动引导类，移动到com.changgou包下才行。