0.学习目标

- 了解电商行业
- 了解商城项目结构
- 能独立搭建项目基本框架

1.了解电商行业

学习电商项目，自然要先了解这个行业，所以我们首先来聊聊电商行业

1.1.项目分类

主要从需求方、盈利模式、技术侧重点这三个方面来看它们的不同

1.1.1.传统项目

各种企业里面用的管理系统（ERP、HR、OA、CRM、物流管理系统。。。。。。。）

- 需求方：公司、企业内部
- 盈利模式：项目本身卖钱
- 技术侧重点：业务功能

1.1.2.互联网项目

门户网站、电商网站：baidu.com、qq.com、taobao.com、jd.com ……

- 需求方：广大用户群体
- 盈利模式：虚拟币、增值服务、广告收益……
- 技术侧重点：网站性能、业务功能

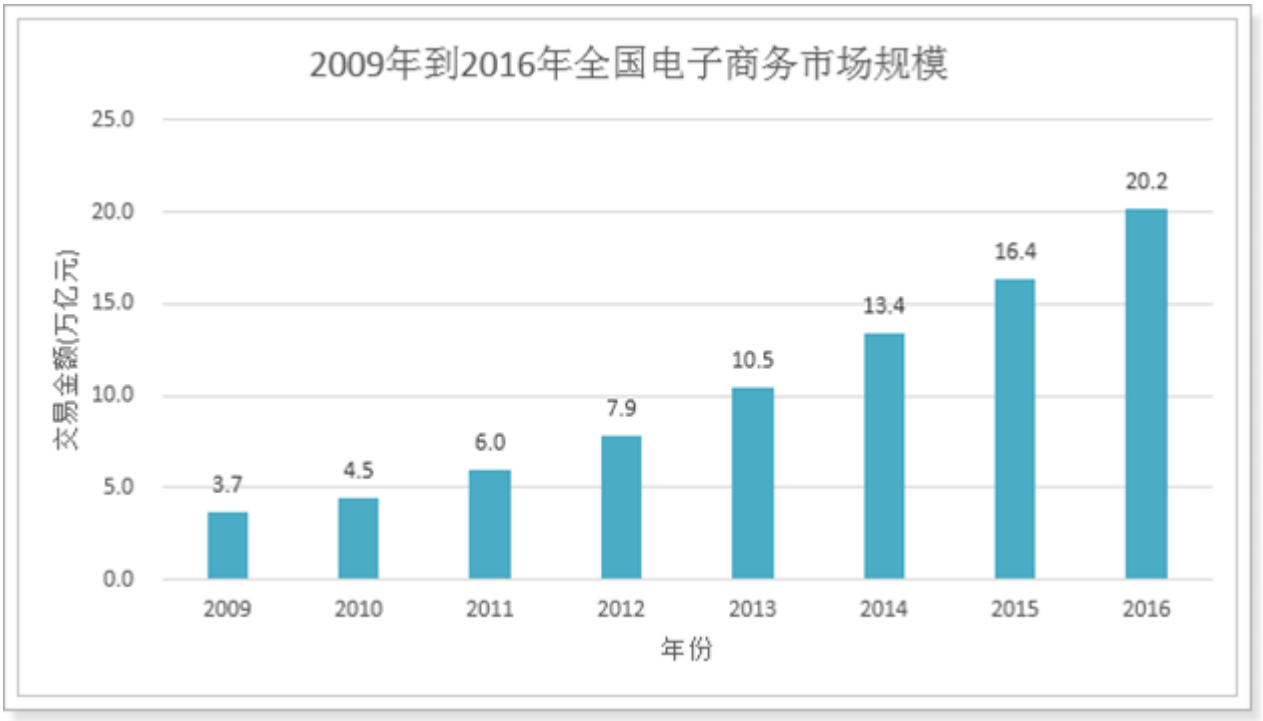
而我们今天要聊的就是互联网项目中的重要角色：电商

1.2.电商行业的发展

1.2.1.钱景

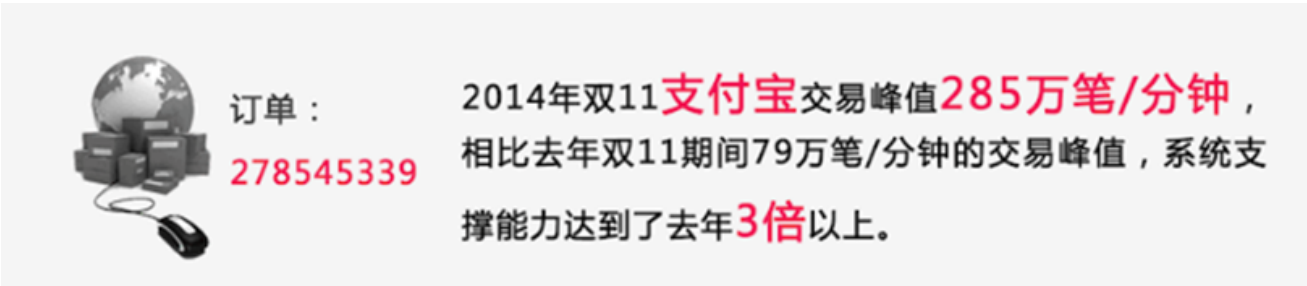
近年来，中国的电子商务快速发展，交易额连创新高，电子商务在各领域的应用不断拓展和深化、相关服务业蓬勃发展、支撑体系不断健全完善、创新的动力和能力不断增强。电子商务正在与实体经济深度融合，进入规模性发展阶段，对经济社会生活的影响不断增大，正成为我国经济发展的新引擎。

中国电子商务研究中心数据显示，截止到 2012 年底，中国电子商务市场交易规模达 7.85万亿人民币，同比增长 30.83%。其中，B2B 电子商务交易额达 6.25 万亿，同比增长 27%。而 2011 年全年，中国电子商务市场交易额达 6 万亿人民币，同比增长 33%，占 GDP 比重上升到 13%；2012 年，电子商务占 GDP 的比重已经高达 15%。



1.2.2.数据

来看看双十一的成交数据：



2016双11开场30分钟，创造**每秒交易峰值17.5万笔**，**每秒**支付峰值**12万笔**的新纪录。菜鸟单日物流订单量超过**4.67亿**，创历史新高。

1.2.3.技术特点

从上面的数据我们不仅要看到钱，更要看到背后的技术实力。正是得益于电商行业的高强度并发压力，促使了BAT等巨头们的技术进步。电商行业有些什么特点呢？

- 技术范围广
- 技术新
- 高并发（分布式、静态化技术、缓存技术、异步并发、池化、队列）
- 高可用（集群、负载均衡、限流、降级、熔断）
- 数据量大
- 业务复杂
- 数据安全

1.3.常见电商模式

电商行业的一些常见模式：

- B2C：商家对个人，如：亚马逊、当当等
- C2C平台：个人对个人，如：咸鱼、拍拍网、ebay
- B2B平台：商家对商家，如：阿里巴巴、八方资源网等
- O2O：线上和线下结合，如：饿了么、电影票、团购等
- P2P：在线金融，贷款，如：网贷之家、人人聚财等。
- B2C平台：天猫、京东、一号店等

1.4.一些专业术语

- SaaS：软件即服务
- SOA：面向服务
- RPC：远程过程调用
- RMI：远程方法调用
- PV：(page view)，即页面浏览量；

  用户每1次对网站中的每个网页访问均被记录1次。用户对同一页面的多次访问，访问量累计
- UV：(unique visitor)，独立访客

  指访问某个站点或点击某条新闻的不同IP地址的人数。在同一天内，uv只记录第一次进入网站的具有独立IP的访问者，在同一天内再次访问该网站则不计数。
- PV与带宽：
  - 计算带宽大小需要关注两个指标：峰值流量和页面的平均大小。
  - 计算公式是：网站带宽=（PV * 平均页面大小（单位MB）* 8)/统计时间（换算到秒）
  - 为什么要乘以8？
    - 网站大小为单位是字节(Byte)，而计算带宽的单位是bit，1Byte=8bit
  - 这个计算的是平均带宽，高峰期还需要扩大一定倍数
- PV、QPS、并发
  - QPS：每秒处理的请求数量。8000/s

- 比如你的程序处理一个请求平均需要0.1S，那么1秒就可以处理10个请求。QPS自然就是10，多线程情况下，这个数字可能就会有所增加。
    - 由PV和QPS如何需要部署的服务器数量？
        - 根据二八原则，80%的请求集中在20%的时间来计算峰值压力：
        - （每日PV * 80%）/（3600s * 24 * 20%）* 每个页面的请求数 = 每个页面每秒的请求数量
    - 然后除以服务器的QPS值，即可计算得出需要部署的服务器数量

1.5.项目开发流程

项目经理：管人

产品经理：设计需求原型

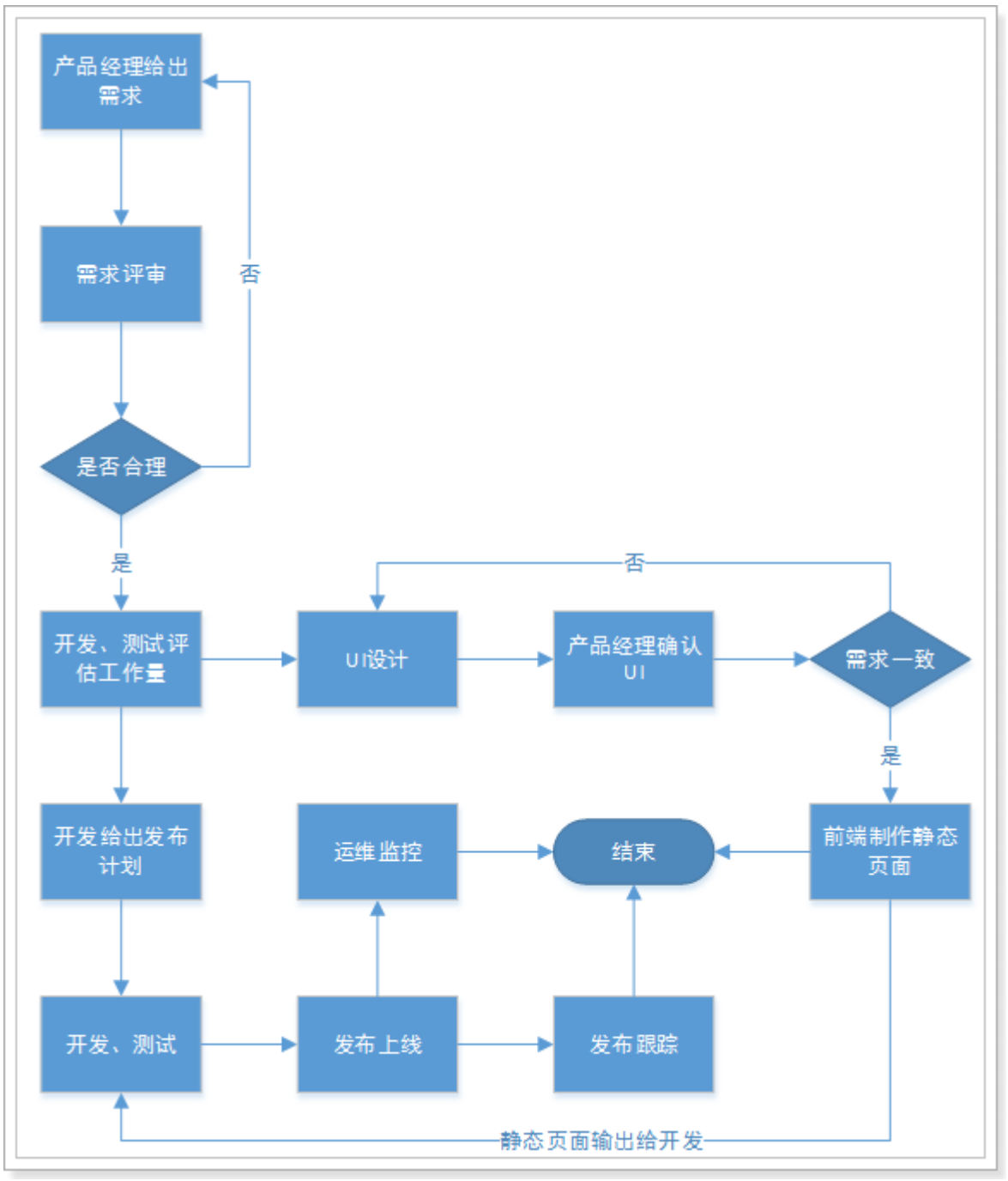
测试：

前端：大前端。node

后端：

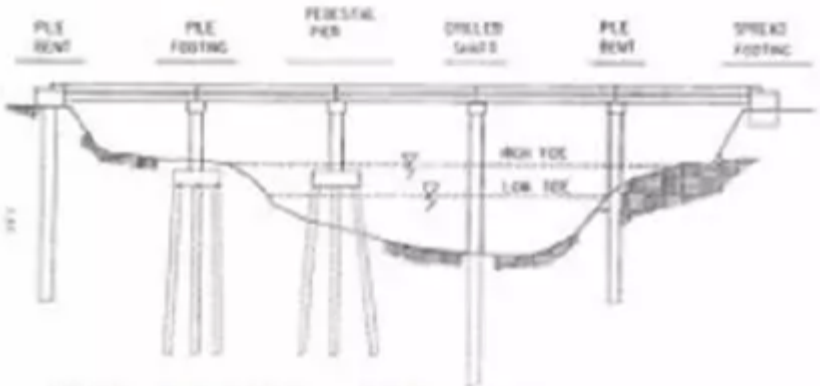
移动端：

项目开发流程图：

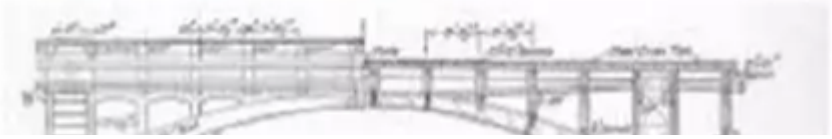

产品经理给出需求
需求评审
否
是否合理
是
否
开发、测试评估工作量
UI设计
产品经理确认UI
需求一致
是
开发给出发布计划
运维监控
结束
前端制作静态页面
开发、测试
发布上线
发布跟踪
静态页面输出给开发

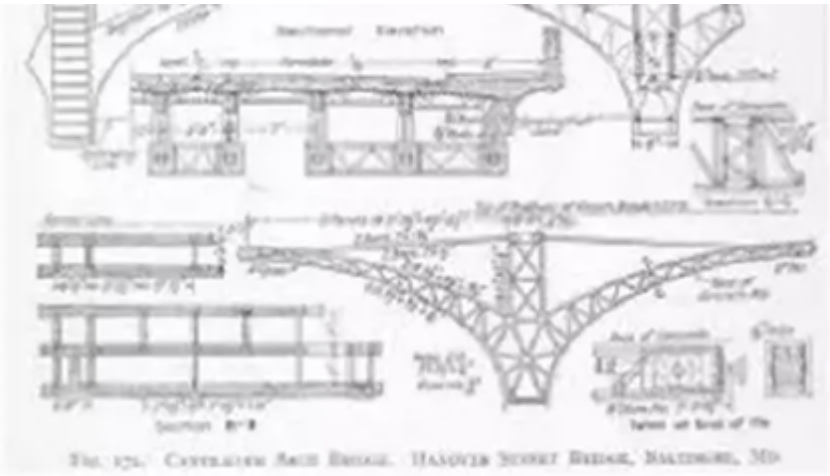
公司现状：

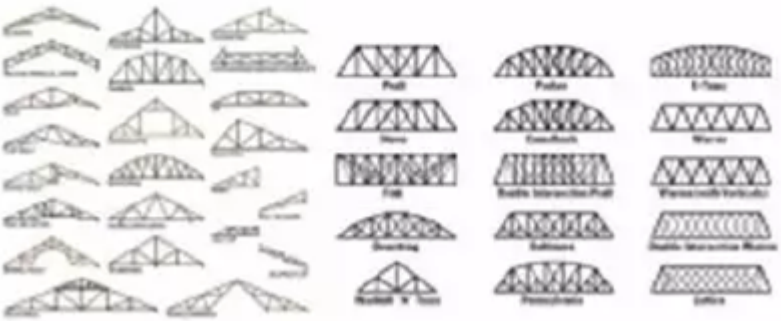
架构师设计



高级工程师分解模块







技术团队时间评估需要1.2年

领导发话：

“互联网精神—先上线，再迭代
给技术团队一个月的时间造桥”


浪来了


处理故障的技术人员
（人肉运维）


对外宣讲

2.商城介绍

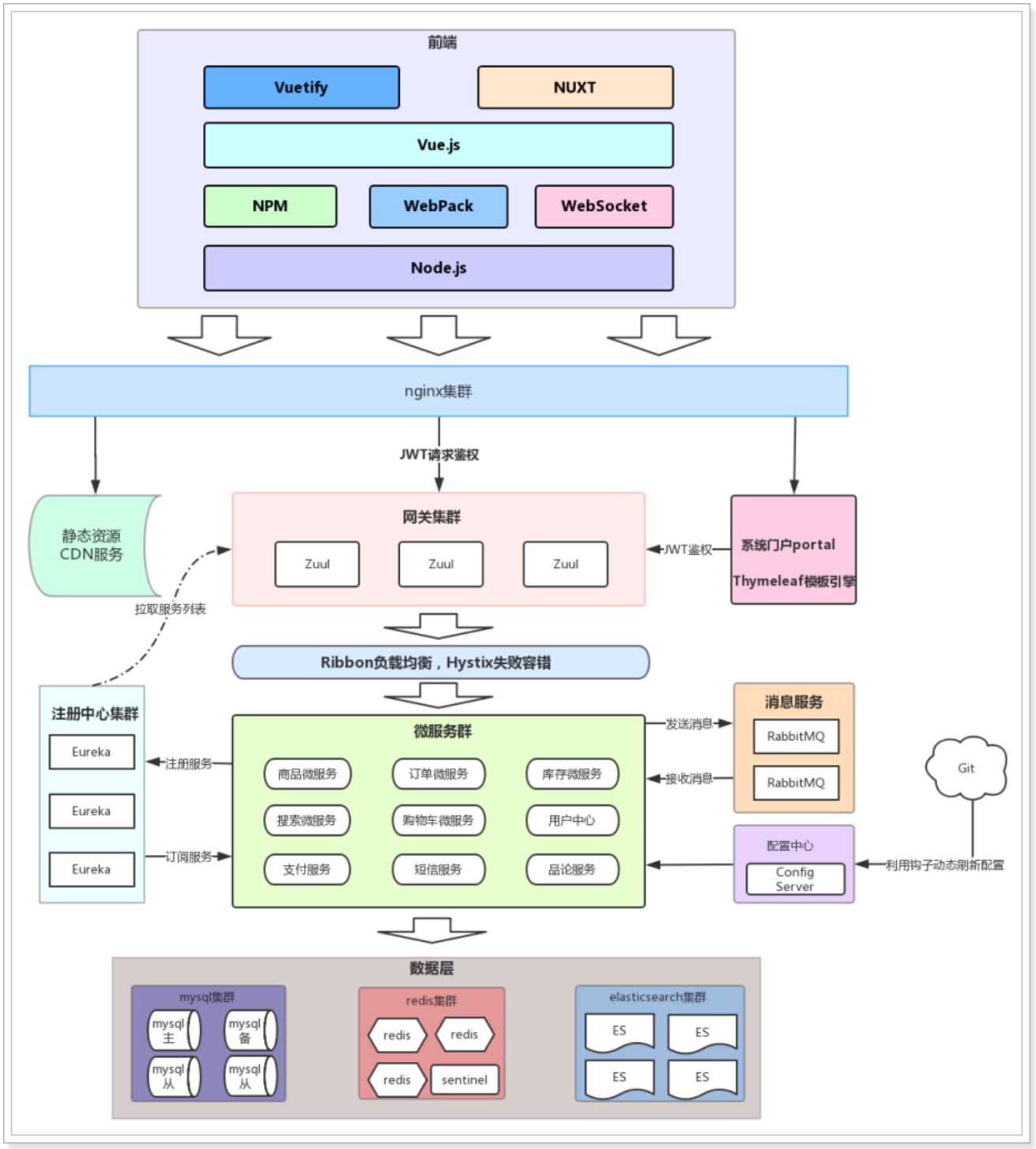
2.1.项目介绍

- 商城是一个全品类的电商购物网站（B2C）。
- 用户可以在线购买商品、加入购物车、下单、秒杀商品
- 可以品论已购买商品
- 管理员可以在后台管理商品的上下架、促销活动
- 管理员可以监控商品销售状况
- 客服可以在后台处理退款操作
- 希望未来3到5年可以支持千万用户的使用

2.2.系统架构

2.2.1.架构图

商城架构缩略图，大图请参考课前资料：



2.2.2.系统架构解读

整个商城可以分为两部分：后台管理系统、前台门户系统。

- 后台管理：
  - 后台系统主要包含以下功能：
    - 商品管理，包括商品分类、品牌、商品规格等信息的管理
    - 销售管理，包括订单统计、订单退款处理、促销活动生成等
    - 用户管理，包括用户控制、冻结、解锁等

        - 权限管理，整个网站的权限控制，采用JWT鉴权方案，对用户及API进行权限控制
        - 统计，各种数据的统计分析展示
    - 后台系统会采用前后端分离开发，而且整个后台管理系统会使用Vue.js框架搭建出单页应用（SPA）。

- 前台门户
    - 前台门户面向的是客户，包含与客户交互的一切功能。例如：
        - 搜索商品
        - 加入购物车
        - 下单
        - 评价商品等等
    - 前台系统我们会使用Thymeleaf模板引擎技术来完成页面开发。出于SEO优化的考虑，我们将不采用单页应用。

无论是前台还是后台系统，都共享相同的微服务集群，包括：

- 商品微服务：商品及商品分类、品牌、库存等的服务
- 搜索微服务：实现搜索功能
- 订单微服务：实现订单相关
- 购物车微服务：实现购物车相关功能
- 用户中心：用户的登录注册等功能
- Eureka注册中心
- Zuul网关服务
- Spring Cloud Config配置中心
- ...

3.项目搭建

3.1.技术选型

前端技术：

- 基础的HTML、CSS、JavaScript（基于ES6标准）
- JQuery
- Vue.js 2.0以及基于Vue的框架：Vuetify
- 前端构建工具：WebPack
- 前端安装包工具：NPM
- Vue脚手架：Vue-cli
- Vue路由：vue-router
- ajax框架：axios
- 基于Vue的富文本框架：quill-editor

后端技术：

- 基础的SpringMVC、Spring 5.0和MyBatis3
- Spring Boot 2.0.1版本

- Spring Cloud 最新版 Finchley.RC1
- Redis-4.0
- RabbitMQ-3.4
- Elasticsearch-5.6.8
- nginx-1.10.2：
- FastDFS - 5.0.8
- MyCat
- Thymeleaf

3.2.开发环境

为了保证开发环境的统一，希望每个人都按照我的环境来配置：

- IDE：我们使用Idea 2017.3 版本
- JDK：统一使用JDK1.8
- 项目构建：maven3.3.9以上版本即可
- 版本控制工具：git

idea大家可以在我的课前资料中找到。另外，使用帮助大家可以参考课前资料的《idea使用指南.md》

3.3.域名

我们在开发的过程中，为了保证以后的生产、测试环境统一。尽量都采用域名来访问项目。

一级域名：www.hz.com

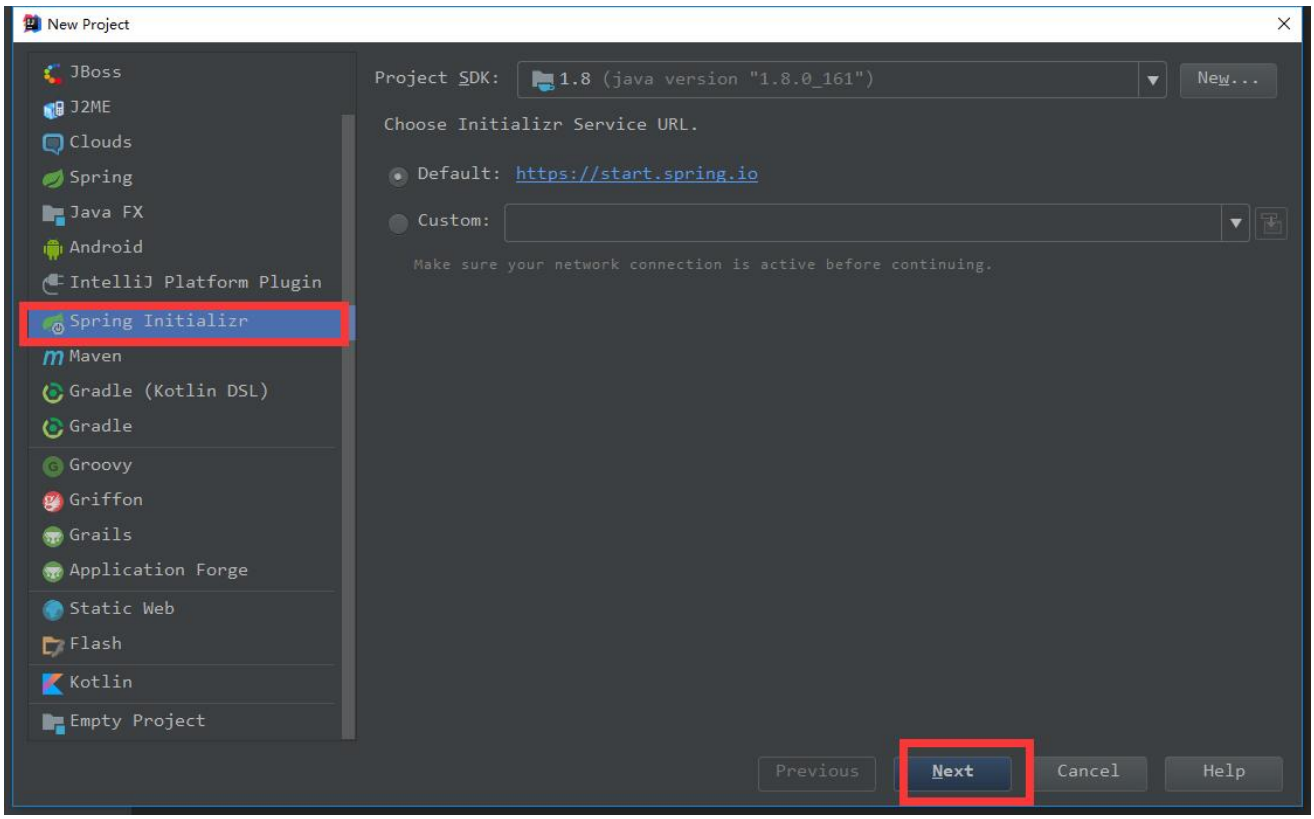
二级域名：manage.hz.com , api.hz.com

我们可以通过switchhost工具来修改自己的host对应的地址，只要把这些域名指向127.0.0.1，那么跟你用localhost的效果是完全一样的。

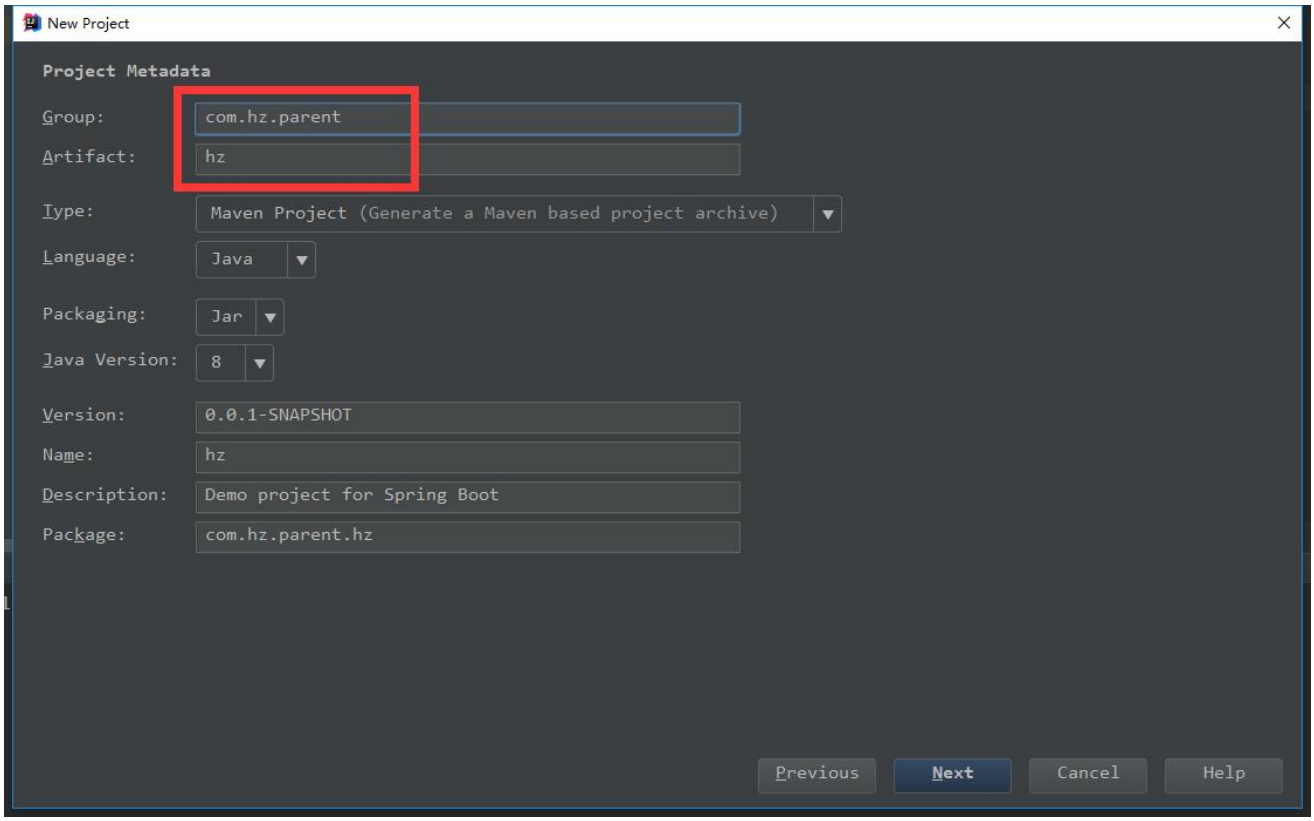
switchhost可以去课前资料寻找。

3.4.创建父工程

创建统一的父工程：hz，用来管理依赖及其版本，注意是创建project，而不是moudle



填写项目信息：



注意：

父工程不需要代码，只是管理依赖，因此我们不选择任何SpringCloud的依赖

跳过依赖选择。

然后将pom文件修改成我这个样子：

```
<?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.hz.parent</groupId>
    <artifactId>hz</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>pom</packaging>

    <name>hz</name>
    <description>Demo project for Spring Boot</description>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.0.1.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
        <java.version>1.8</java.version>
        <spring-cloud.version>Finchley.RC1</spring-cloud.version>
        <mybatis.starter.version>1.3.2</mybatis.starter.version>
        <mapper.starter.version>2.0.2</mapper.starter.version>
        <druid.starter.version>1.1.9</druid.starter.version>
        <mysql.version>5.1.32</mysql.version>
        <pageHelper.starter.version>1.2.3</pageHelper.starter.version>
        <hz.latest.version>1.0.0-SNAPSHOT</hz.latest.version>
        <fastDFS.client.version>1.26.1-RELEASE</fastDFS.client.version>
    </properties>

    <dependencyManagement>
        <dependencies>
            <!-- springCloud -->
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>${spring-cloud.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>

            <!-- mybatis启动器 -->
```

```
            <dependency>
                <groupId>org.mybatis.spring.boot</groupId>
                <artifactId>mybatis-spring-boot-starter</artifactId>
                <version>${mybatis.starter.version}</version>
            </dependency>
            <!-- 通用Mapper启动器 -->
            <dependency>
                <groupId>tk.mybatis</groupId>
                <artifactId>mapper-spring-boot-starter</artifactId>
                <version>${mapper.starter.version}</version>
            </dependency>
            <!-- 分页助手启动器 -->
            <dependency>
                <groupId>com.github.pagehelper</groupId>
                <artifactId>pagehelper-spring-boot-starter</artifactId>
                <version>${pageHelper.starter.version}</version>
            </dependency>
            <!-- mysql驱动 -->
            <dependency>
                <groupId>mysql</groupId>
                <artifactId>mysql-connector-java</artifactId>
                <version>${mysql.version}</version>
            </dependency>
            <!--FastDFS客户端-->
            <dependency>
                <groupId>com.github.tobato</groupId>
                <artifactId>fastdfs-client</artifactId>
                <version>${fastDFS.client.version}</version>
            </dependency>
        </dependencies>
    </dependencyManagement>

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

    <repositories>
        <repository>
            <id>spring-milestones</id>
            <name>Spring Milestones</name>
            <url>https://repo.spring.io/milestone</url>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </repository>
    </repositories>


</project>
```

可以发现，我们在父工程中引入了SpringCloud等很多以后需要用到的依赖，以后创建的子工程就不需要自己引入了。

最后，删除自动生成的hzApplication启动类、测试类以及application.properties文件，我们不需要。

3.5.创建EurekaServer

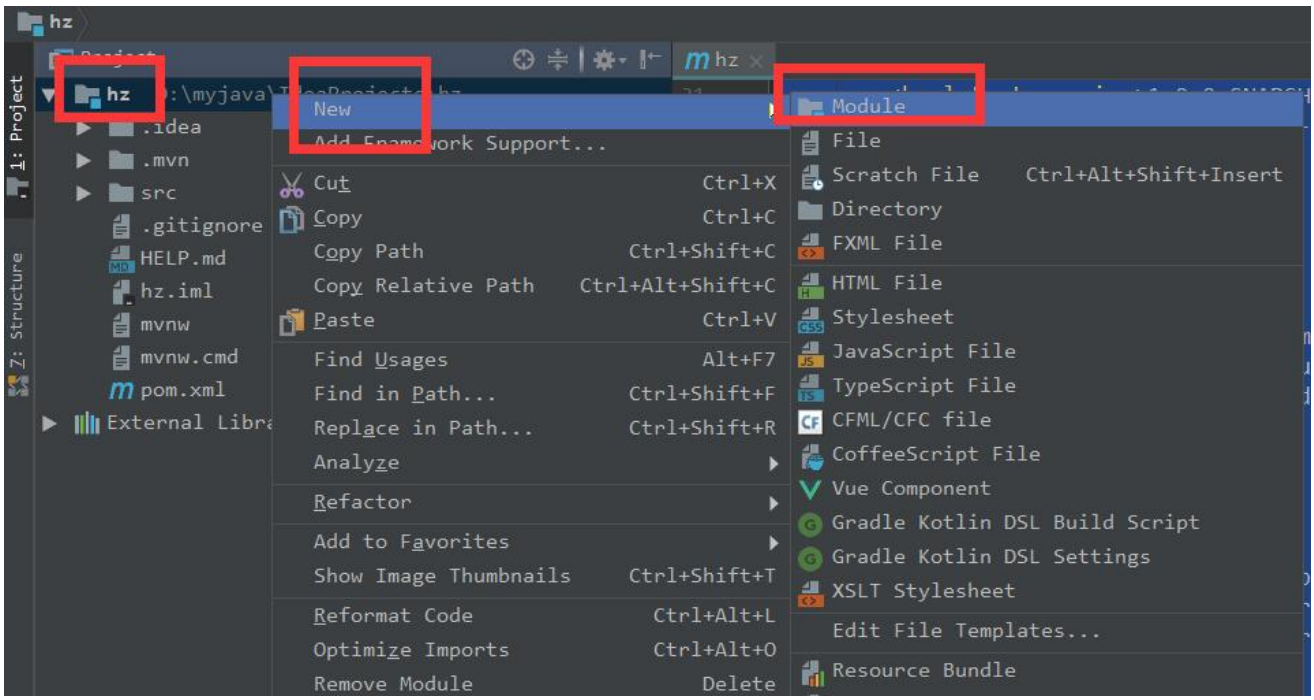
3.5.1.创建工程

这个大家应该比较熟悉了。

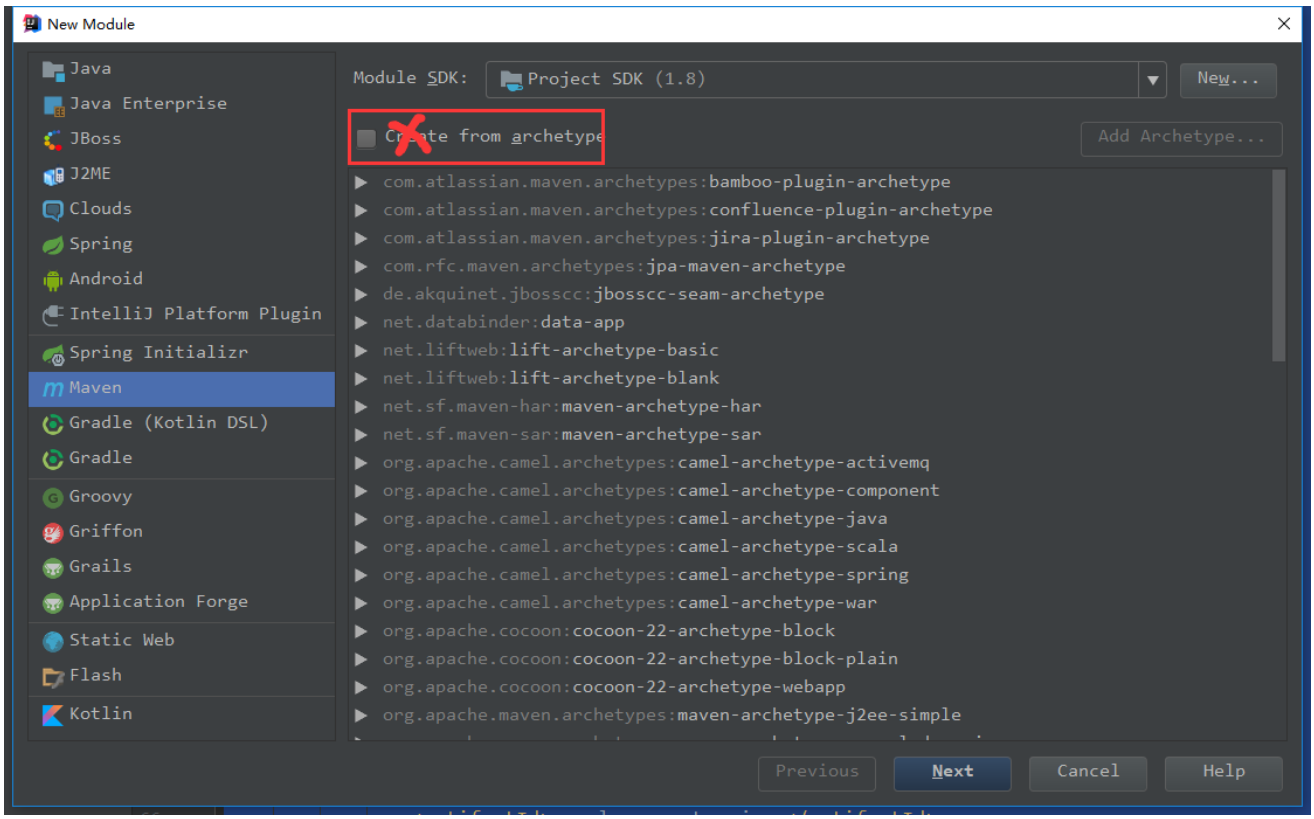
我们的注册中心，起名为：hz-registry

这次我们就不Spring使用提供的脚手架了。直接创建maven项目，自然会继承父类的依赖：

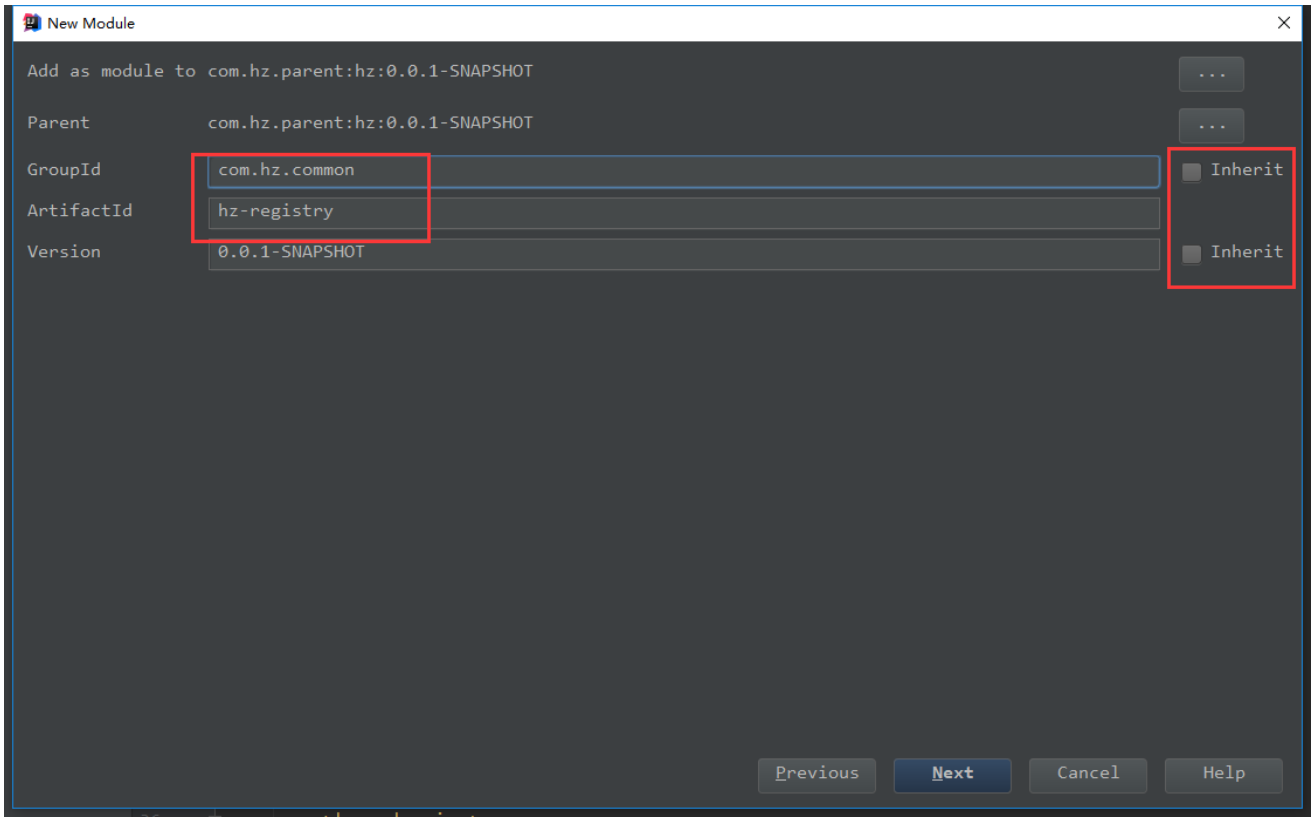
选择新建module：



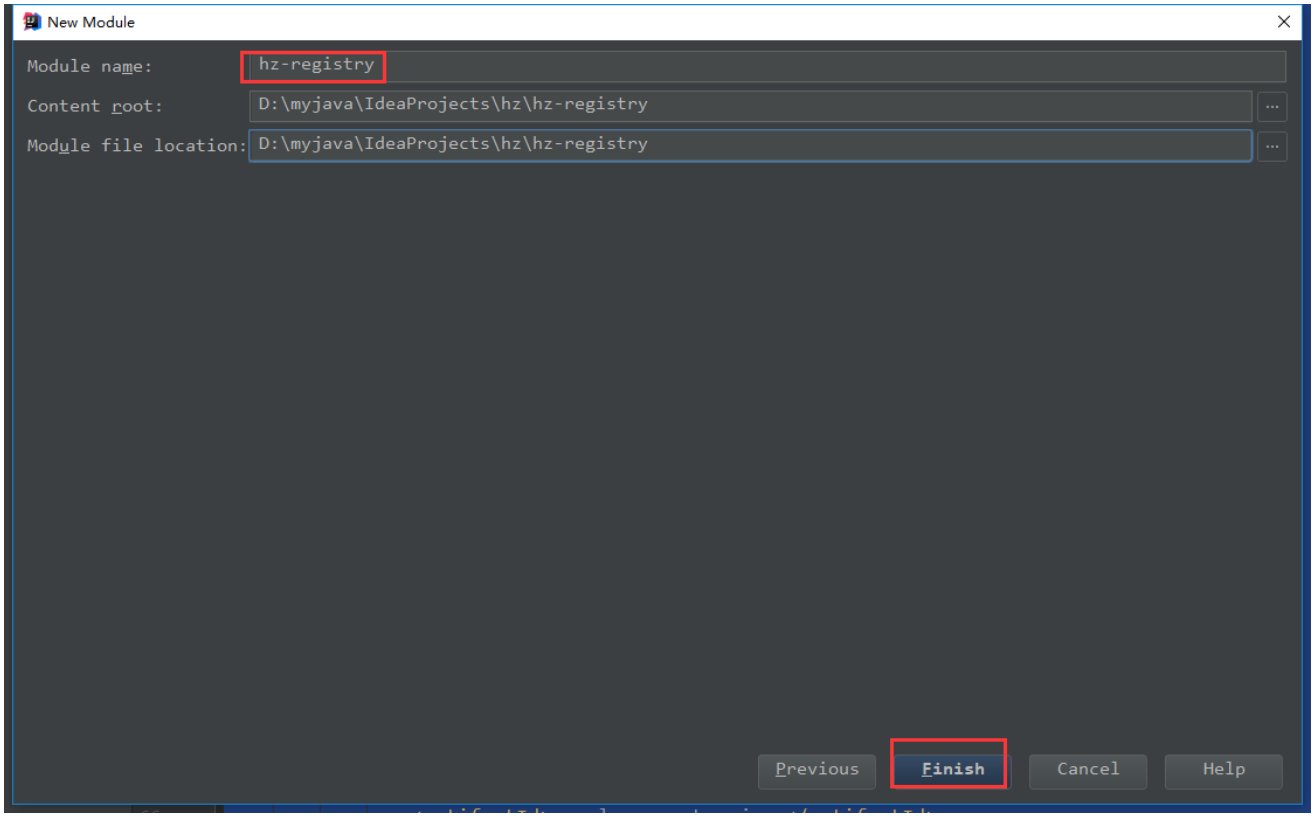
选择maven安装，但是不要选择骨架：



然后填写项目坐标，我们的项目名称为hz-registry:



选择安装目录，因为是聚合项目，目录应该是在父工程hz的下面：



3.5.2.添加依赖

添加EurekaServer的依赖：

```
<?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>hz</artifactId>
        <groupId>com.hz.parent</groupId>
        <version>0.0.1-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.hz.common</groupId>
    <artifactId>hz-registry</artifactId>
    <version>0.0.1-SNAPSHOT</version>


    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
        </dependency>
```

```
    </dependencies>

</project>
```

3.5.3.编写启动类

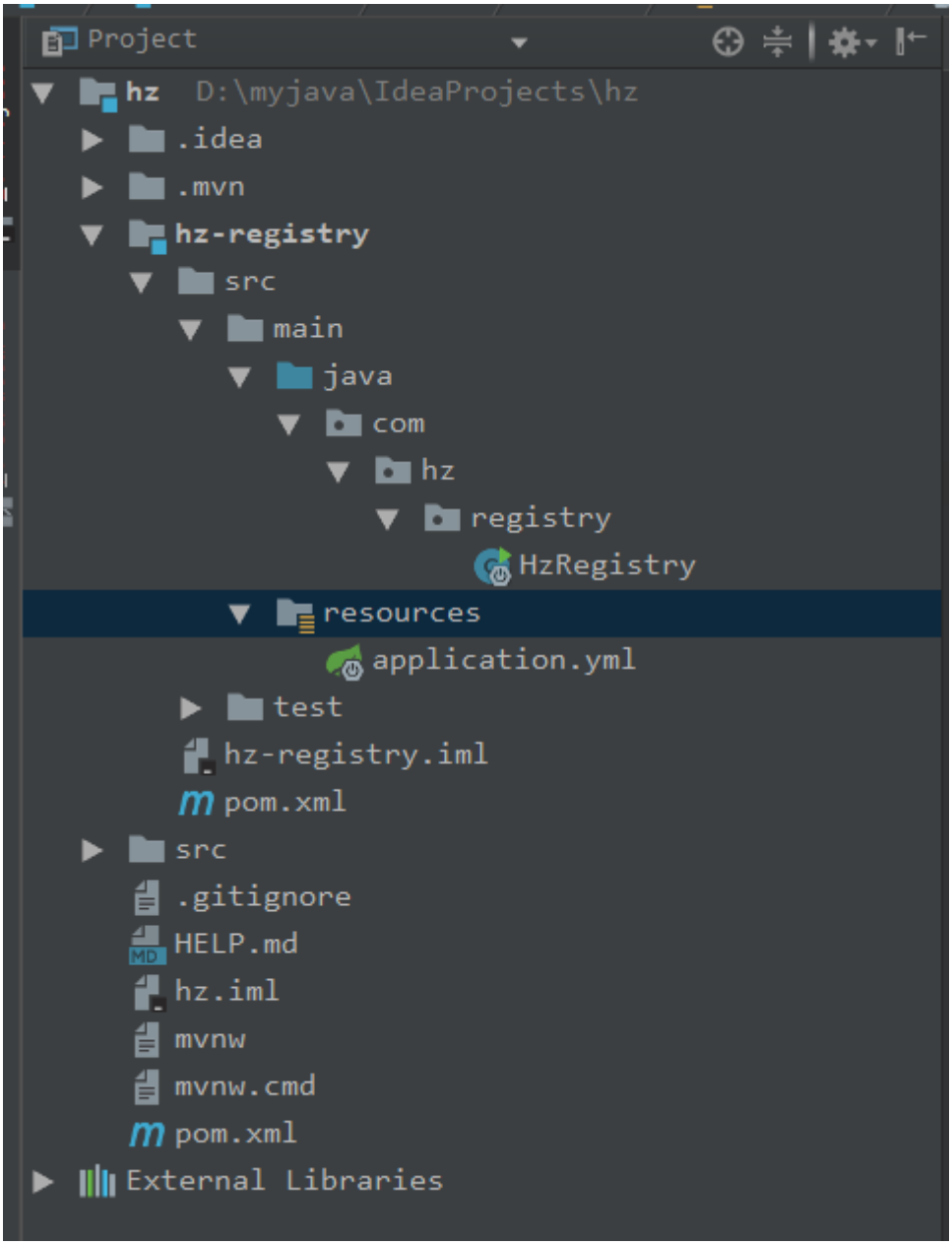
```
@SpringBootApplication
@EnableEurekaServer
public class HzRegistry {
    public static void main(String[] args) {
        SpringApplication.run(HzRegistry.class, args);
    }
}
```

3.5.4.配置文件

```
server:
  port: 10086
spring:
  application:
    name: hz-registry
eureka:
  client:
    fetch-registry: false
    register-with-eureka: false
    service-url:
      defaultZone: http://127.0.0.1:${server.port}/eureka
  server:
    enable-self-preservation: false # 关闭自我保护
    eviction-interval-timer-in-ms: 5000 # 每隔5秒进行一次服务列表清理
```

3.5.5.项目的结构：

目前，整个项目的结构如图：



3.6.创建Zuul网关

3.6.1.创建工程

与上面类似，选择maven方式创建Module，然后填写项目名称，我们命名为：hz-api-gateway

3.6.2.添加依赖

这里我们需要添加Zuul和EurekaClient的依赖：

```
<?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>hz</artifactId>
        <groupId>com.hz.parent</groupId>
        <version>0.0.1-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.hz.common</groupId>
    <artifactId>hz-api-gateway</artifactId>
    <version>0.0.1-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-zuul</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
        </dependency>
        <!--是springboot提供的微服务检测接口，默认对外提供几个接口：/info-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-actuator</artifactId>
        </dependency>
    </dependencies>

</project>
```

3.6.3.编写启动类

```
@SpringBootApplication
@EnableDiscoveryClient
@EnableZuulProxy
public class HzApiGateway{
    public static void main(String[] args) {
        SpringApplication.run(HzApiGateway.class, args);
    }
}
```

3.6.4.配置文件

```
server:
  port: 10010
spring:
  application:
    name: api-gateway
```

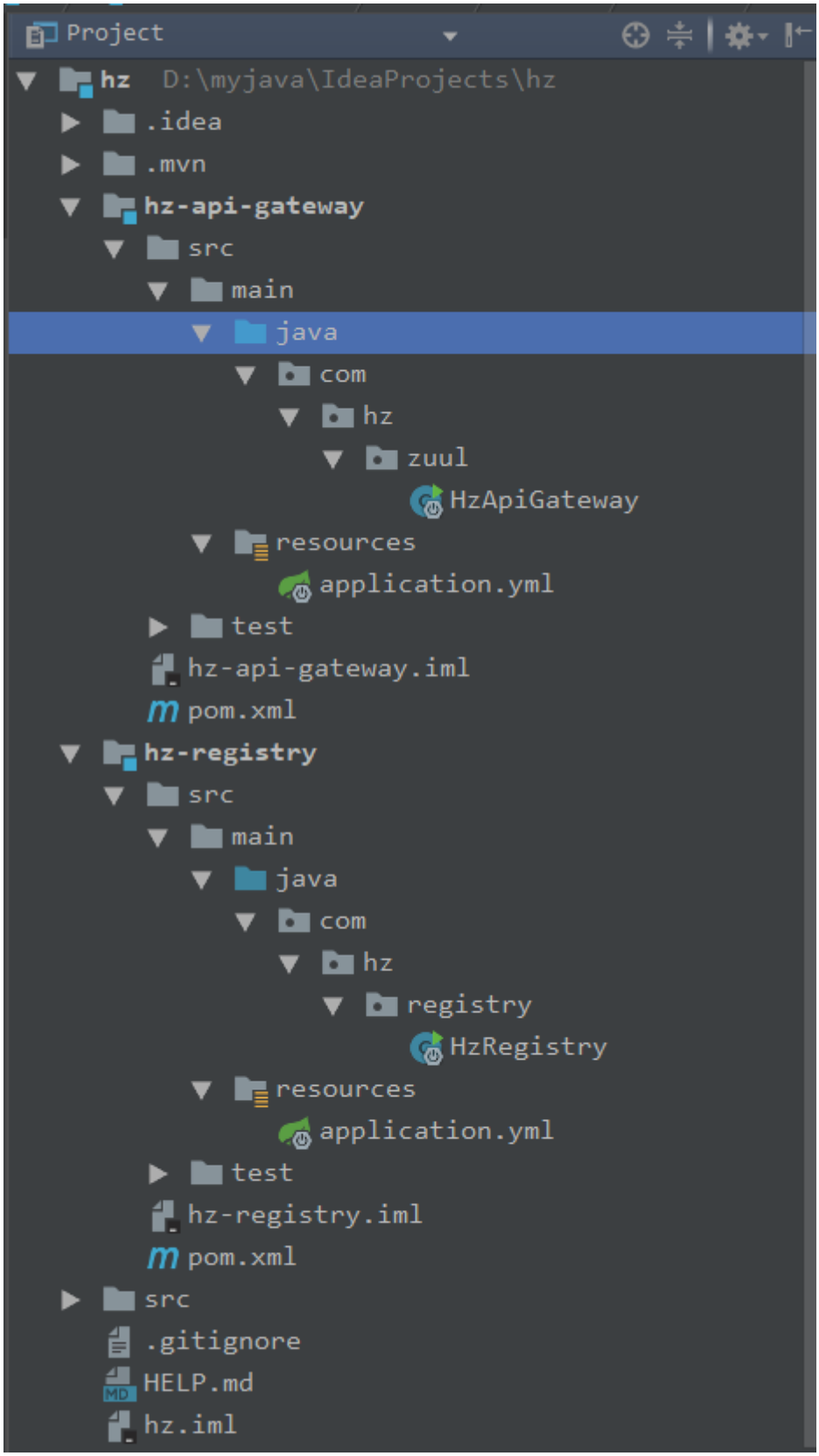
```yaml
eureka:
  client:
    service-url:
      defaultZone: http://127.0.0.1:10086/eureka
    registry-fetch-interval-seconds: 5
  instance:
    prefer-ip-address: true
    ip-address: 127.0.0.1
    instance-id: ${spring.application.name}:${server.port}
zuul:
  prefix: /api # 添加路由前缀
  retryable: true
ribbon:
  ConnectTimeout: 250 # 连接超时时间(ms)
  ReadTimeout: 2000 # 通信超时时间(ms)
  OkToRetryOnAllOperations: true # 是否对所有操作重试
  MaxAutoRetriesNextServer: 1 # 同一服务不同实例的重试次数
  MaxAutoRetries: 1 # 同一实例的重试次数
hystrix:
  command:
    default:
      execution:
        isolation:
          thread:
            timeoutInMillisecond: 10000 # 熔断超时时长：10000ms
```

3.6.5.项目结构

目前，hz下有两个子模块：

- hz-registry：服务的注册中心（EurekaServer）
- hz-api-gateway：服务网关（Zuul）

目前，服务的结构如图所示：



截止到这里，我们已经把基础服务搭建完毕，为了便于开发，统一配置中心（ConfigServer）我们留待以后添加。

3.7.创建商品微服务

既然是一个全品类的电商购物平台，那么核心自然就是商品。因此我们要搭建的第一个服务，就是商品微服务。其中会包含对于商品相关的一系列内容的管理，包括：

- 商品分类管理
- 品牌管理
- 商品规格参数管理
- 商品管理
- 库存管理

我们先完成项目的搭建：

3.7.1.微服务的结构

因为与商品的品类相关，我们的工程命名为`hz-item`.

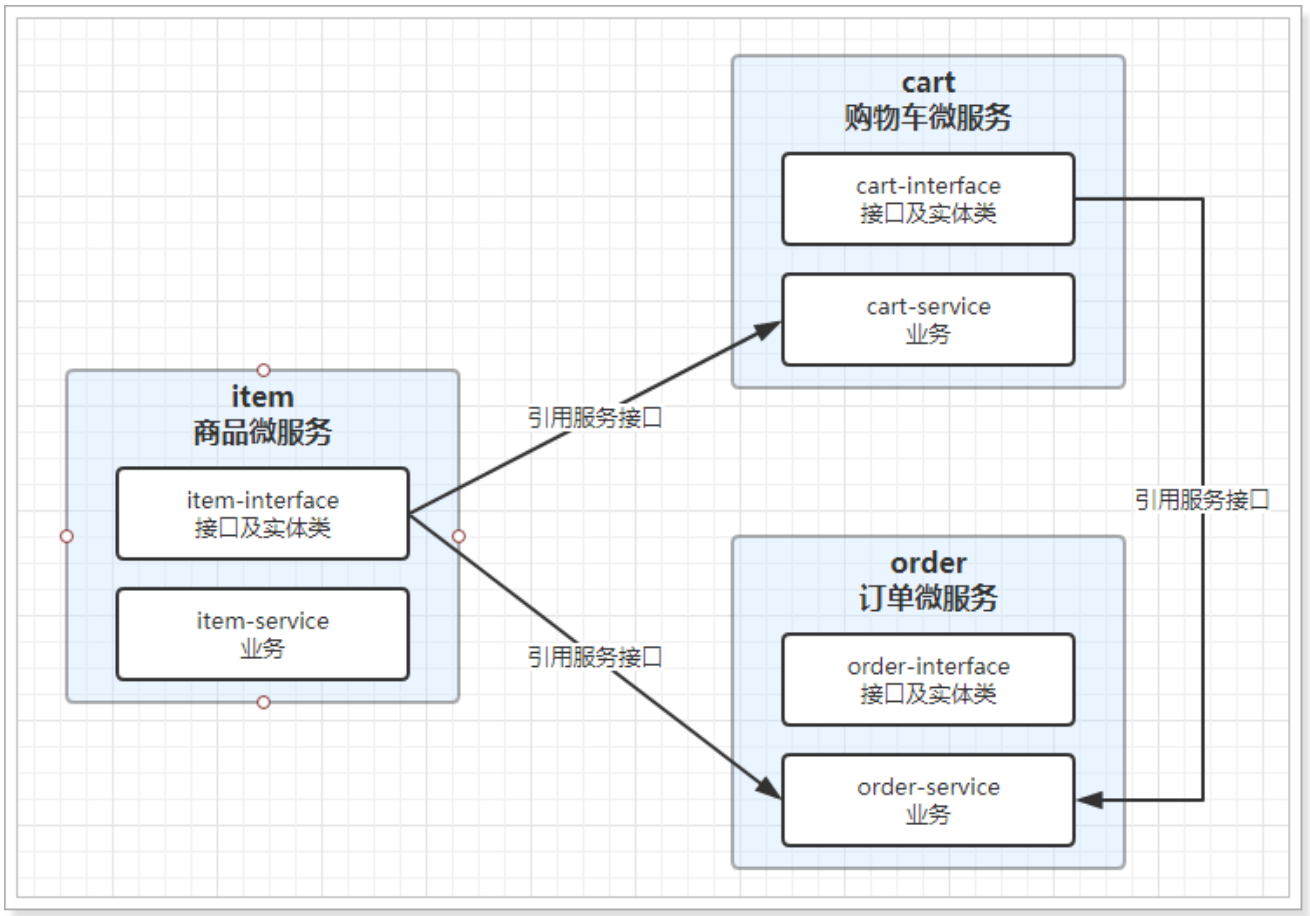
需要注意的是，我们的hz-item是一个微服务，那么将来肯定会有其它系统需要来调用服务中提供的接口，因此肯定也会使用到接口中关联的实体类。

因此这里我们需要使用聚合工程，将要提供的接口及相关实体类放到独立子工程中，以后别人引用的时候，只需要知道坐标即可。

我们会在hz-item中创建两个子工程：

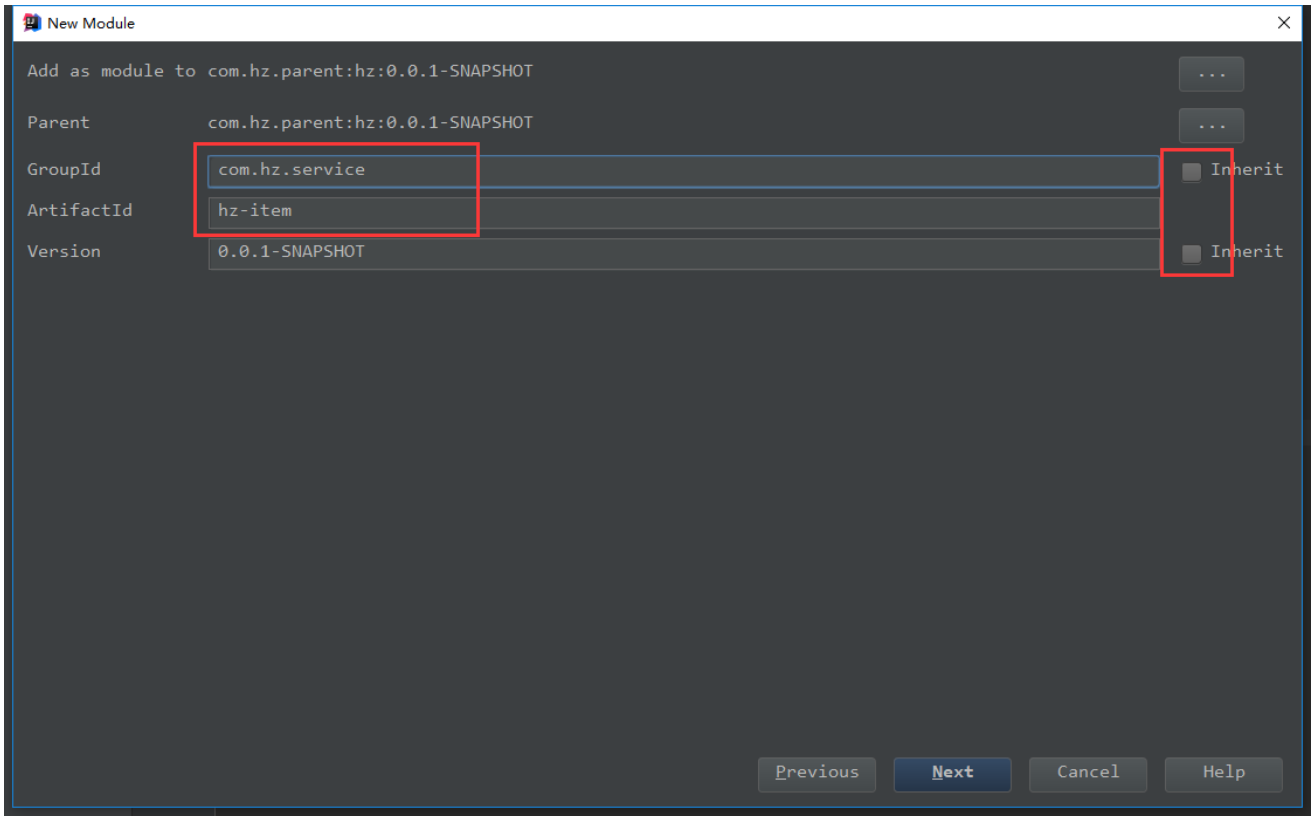
- hz-item-interface：主要是对外暴露的接口及相关实体类
- hz-item-service：所有业务逻辑及内部使用接口

调用关系如图所示：



3.7.2.创建父工程hz-item

依然是使用maven构建：

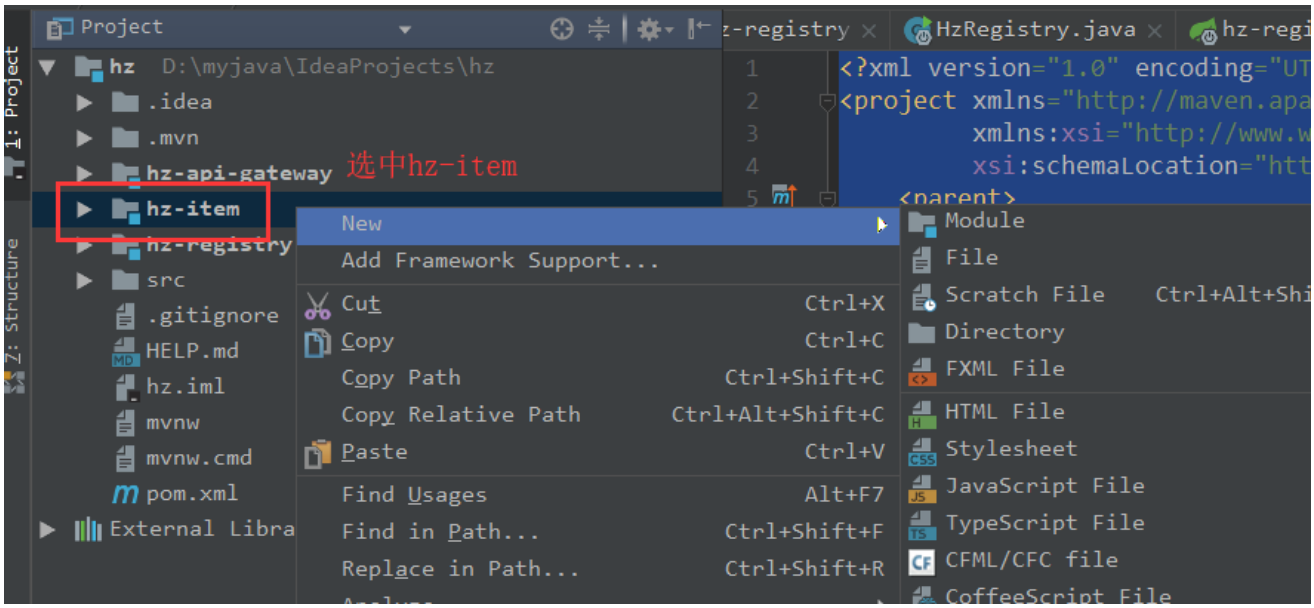


不需要任何依赖，我们可以把项目打包方式设置为pom

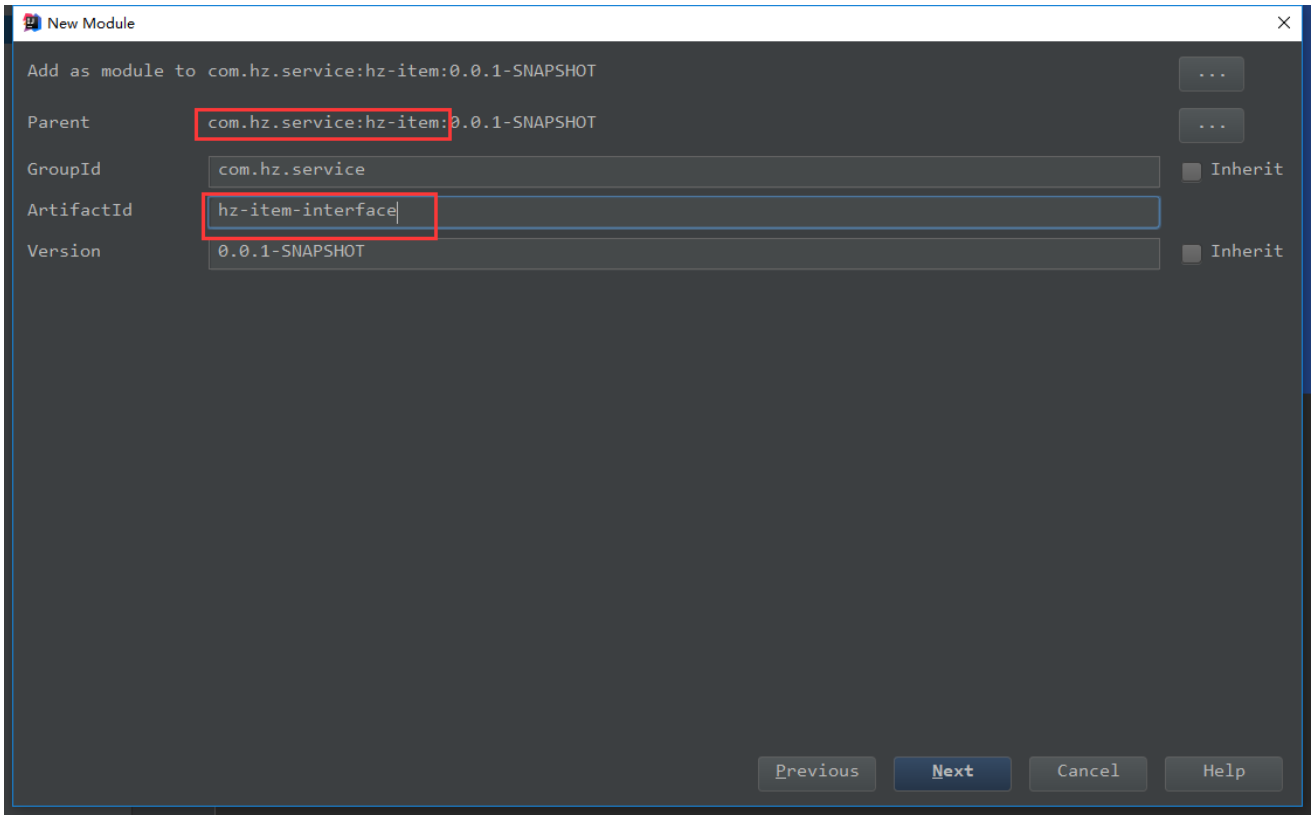
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>hz</artifactId>
        <groupId>com.hz.parent</groupId>
        <version>0.0.1-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.hz.service</groupId>
    <artifactId>hz-item</artifactId>
    <version>0.0.1-SNAPSHOT</version>

    <!-- 打包方式为pom -->
    <packaging>pom</packaging>


</project>
```

3.7.3.创建hz-item-interface

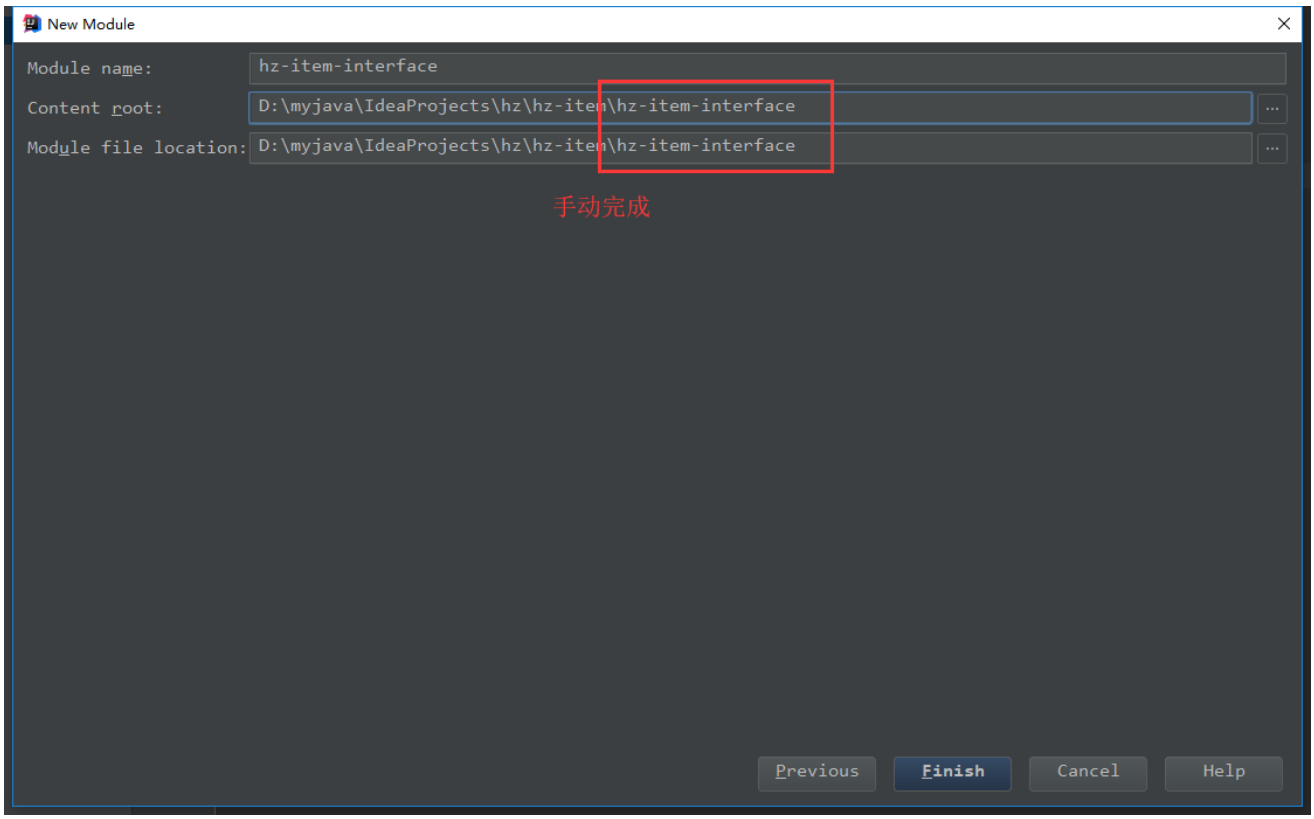
在hz-item工程上点击右键，选择new > module:



依然是使用maven构建，注意父工程是hz-item：

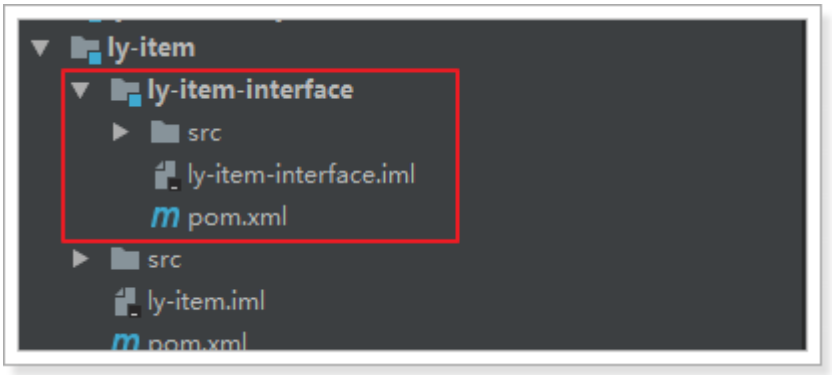


注意：接下来填写的目录结构需要自己手动完成，保存到`hz-item`下的`hz-item-interface`目录中：



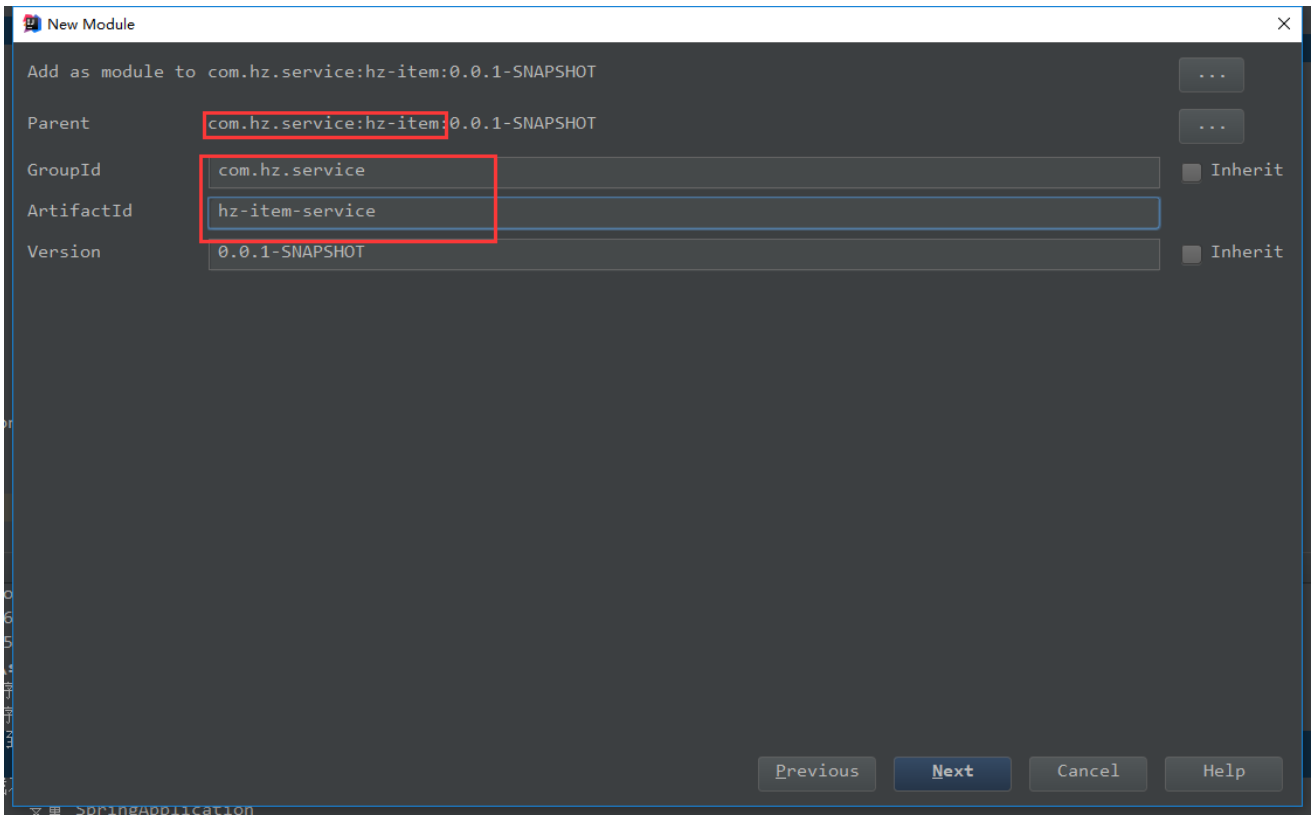
点击Finish完成。

此时的项目结构：

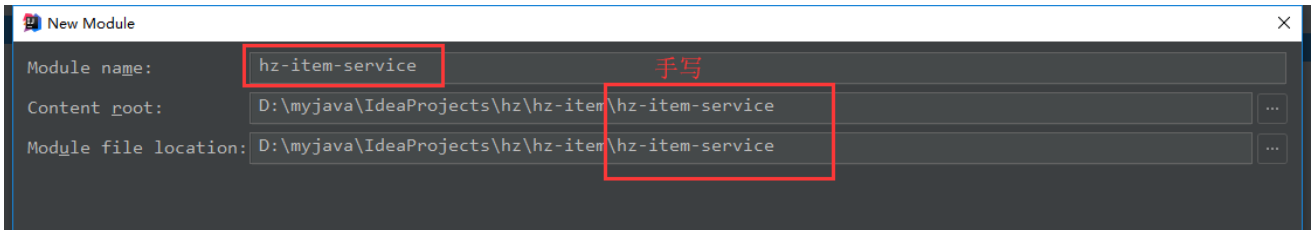


3.7.4.创建hz-item-service

与`hz-item-interface`类似，我们选择在`hz-item`上右键，新建module，然后填写项目信息：



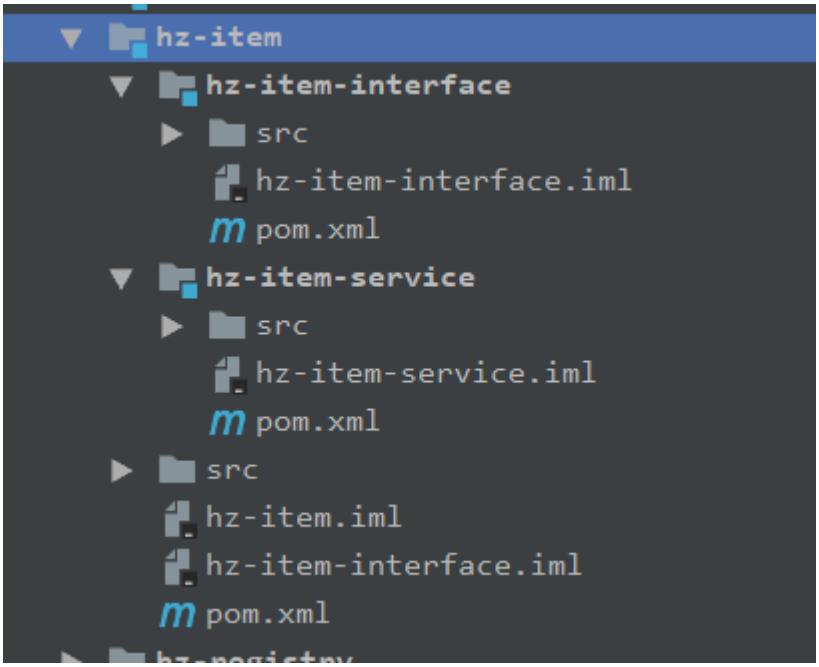
填写存储位置，是在`/hz-item/hz-item-service`目录



点击Finish完成。

3.7.5.整个微服务结构

如图所示：



我们打开hz-item的pom查看，会发现hz-item-interface和hz-item-service都已经称为module了：

```
<modules>
   <module>hz-item-interface</module>
   <module>hz-item-service</module>
</modules>
```

3.7.6.添加依赖

接下来我们给`hz-item-service`中添加依赖：

思考一下我们需要什么？

- Eureka客户端
- web启动器
- mybatis启动器
- 通用mapper启动器
- 分页助手启动器
- 连接池，我们用默认的Hykira
- mysql驱动
- 千万不能忘了，我们自己也需要`hz-item-interface`中的实体类

这些依赖，我们在顶级父工程：hz中已经添加好了。所以直接引入即可：

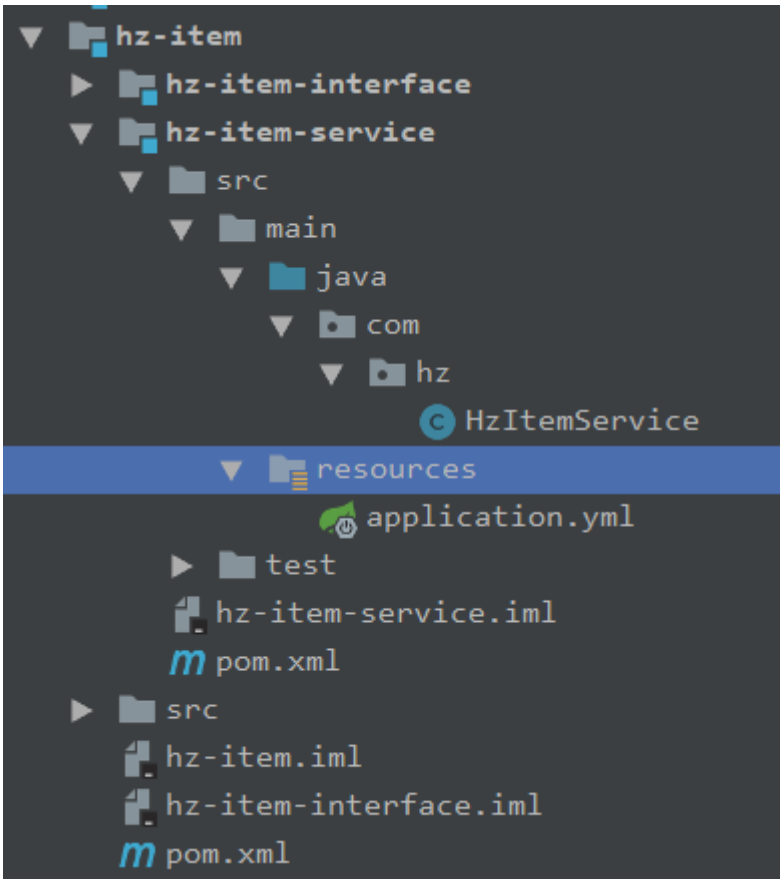
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>hz-item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

```
        <groupId>com.hz.service</groupId>
        <version>0.0.1-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.hz.service</groupId>
    <artifactId>hz-item-service</artifactId>
    <version>0.0.1-SNAPSHOT</version>

    <dependencies>
        <!--Eureka客户端-->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
        </dependency>
        <!--web启动器-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <!-- mybatis启动器 -->
        <dependency>
            <groupId>org.mybatis.spring.boot</groupId>
            <artifactId>mybatis-spring-boot-starter</artifactId>
            <version>${mybatis.starter.version}</version>
        </dependency>
        <!-- 通用Mapper启动器 -->
        <dependency>
            <groupId>tk.mybatis</groupId>
            <artifactId>mapper-spring-boot-starter</artifactId>
            <version>${mapper.starter.version}</version>
        </dependency>
        <!-- 分页助手启动器 -->
        <dependency>
            <groupId>com.github.pagehelper</groupId>
            <artifactId>pagehelper-spring-boot-starter</artifactId>
            <version>${pageHelper.starter.version}</version>
        </dependency>
        <!-- mysql驱动 -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>${mysql.version}</version>
        </dependency>
        <dependency>
            <groupId>com.hz.service</groupId>
            <artifactId>hz-item-interface</artifactId>
            <version>0.0.1-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-actuator</artifactId>

        </dependency>
```

```
    </dependencies>


</project>
```

hz-item-interface中需要什么我们暂时不清楚，所以先不管。

整个结构：



3.7.7.编写启动和配置

在整个`hz-item工程`中，只有`hz-item-service`是需要启动的。因此在其中编写启动类即可：

```
@SpringBootApplication
@EnableDiscoveryClient
public class HzItemService {
    public static void main(String[] args) {
        SpringApplication.run(HzItemService.class, args);
    }
}
```

然后是全局属性文件：

```
server:
  port: 8081
spring:
  application:
    name: item-service
  datasource:
    url: jdbc:mysql://localhost:3306/hz
    username: root
    password: root
    hikari:
      maximum-pool-size: 30
      minimum-idle: 10
eureka:
  client:
    service-url:
      defaultZone: http://127.0.0.1:10086/eureka
  instance:
    lease-renewal-interval-in-seconds: 5 # 每隔5秒发送一次心跳
    lease-expiration-duration-in-seconds: 10 # 10秒不发送就过期
    prefer-ip-address: true
    ip-address: 127.0.0.1
    instance-id: ${spring.application.name}:${server.port}
```

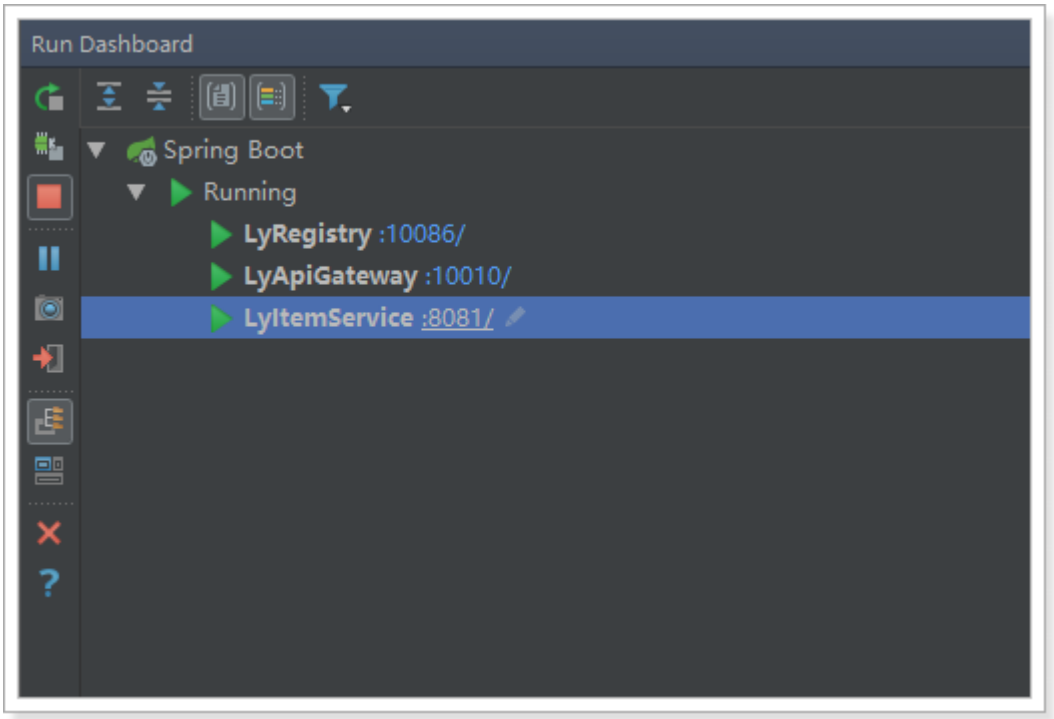
3.8.添加商品微服务的路由规则

既然商品微服务已经创建，接下来肯定要添加路由规则到Zuul中，我们不使用默认的路由规则。

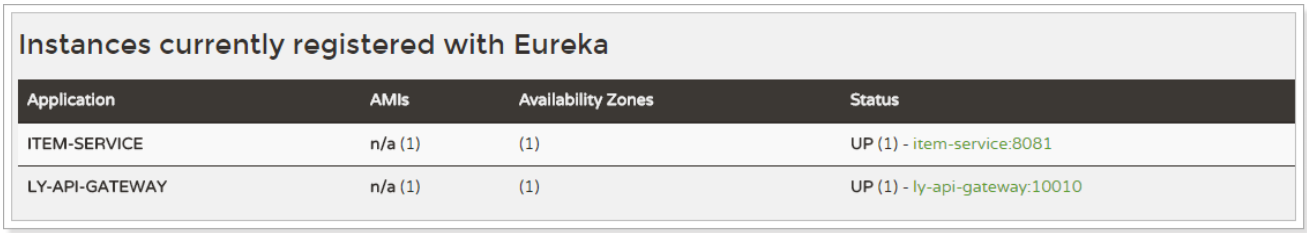
```
zuul:
  prefix: /api # 添加路由前缀
  retryable: true
  routes:
    item-service: /item/** # 将商品微服务映射到/item/**
```

3.9.启动测试

我们分别启动：hz-registry，hz-api-gateway，hz-item-service



查看Eureka面板：



Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
ITEM-SERVICE	n/a (1)	(1)	UP (1) - item-service:8081
LY-API-GATEWAY	n/a (1)	(1)	UP (1) - ly-api-gateway:10010

3.10.测试路由规则

为了测试路由规则是否畅通，我们是不是需要在item-service中编写一个controller接口呢？

其实不需要，Spring提供了一个依赖：actuator

只要我们添加了actuator的依赖，它就会为我们生成一系列的访问接口：

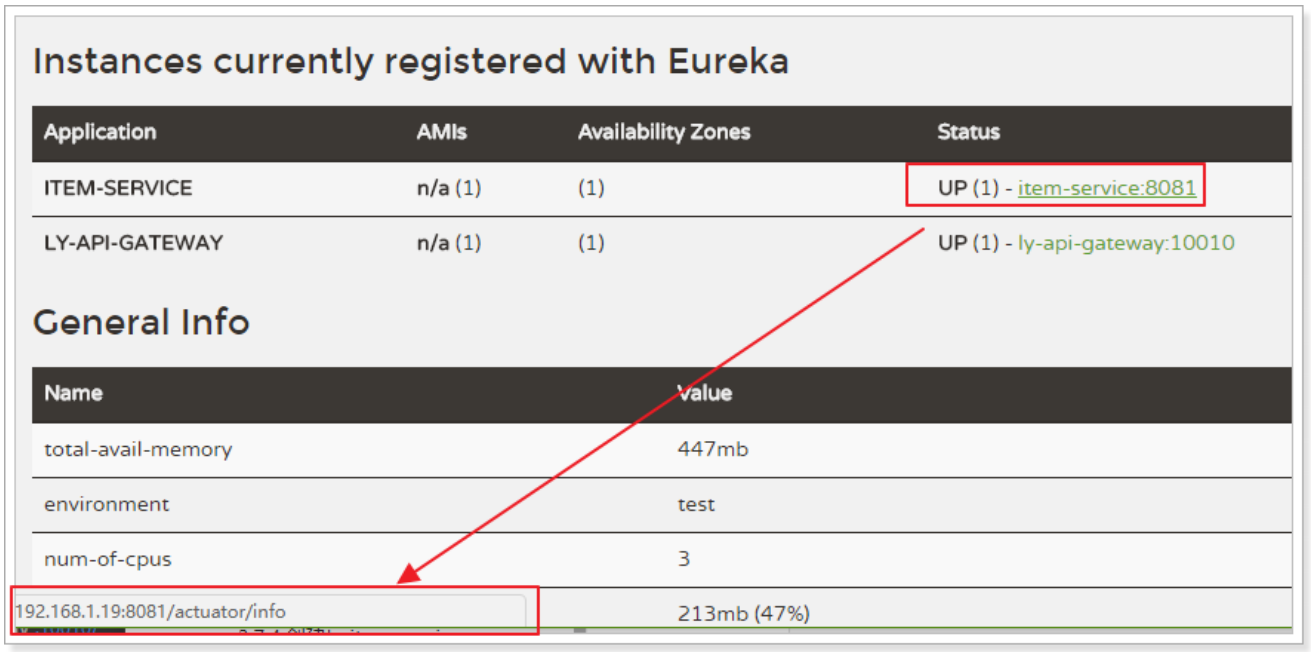
- /info
- /health
- /refresh
- ...

添加依赖：

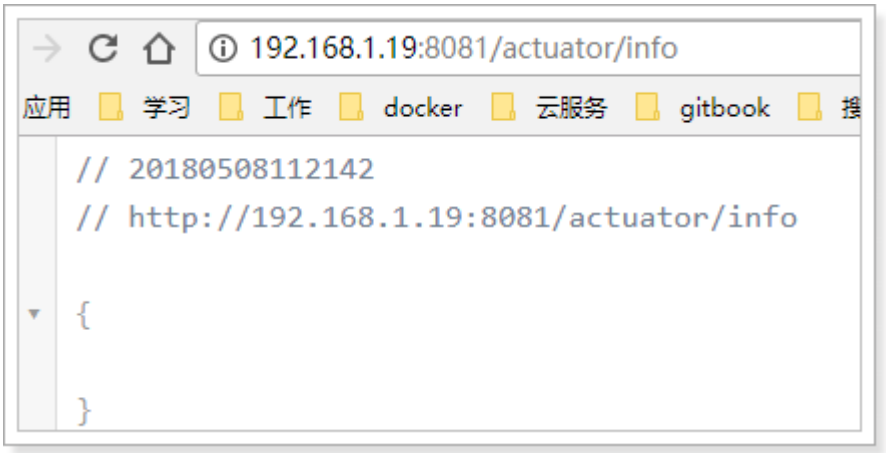
```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

重启后访问Eureka控制台：

鼠标悬停在item-service上，会显示一个地址：



这就是actuator提供的接口，我们点击访问：



因为我们没有添加信息，所以是一个空的json，但是可以肯定的是：我们能够访问到item-service了。

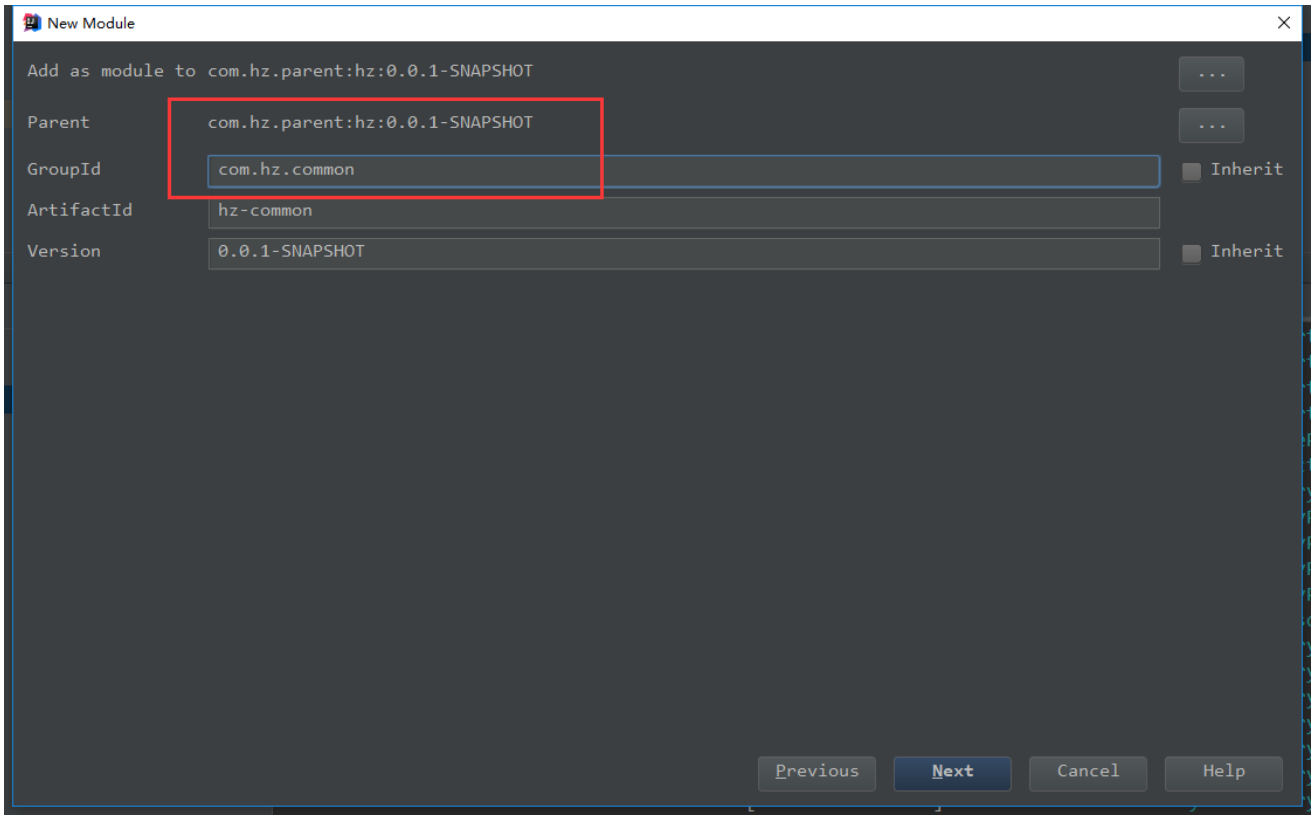
接下来我们通过路由访问试试，根据路由规则，我们需要访问的地址是：

http://127.0.0.1:10010/api/item/actuator/info

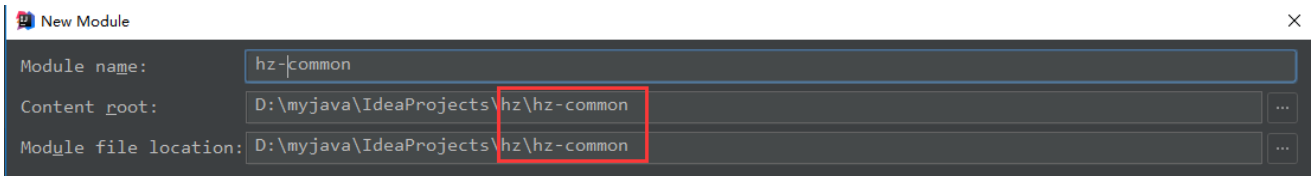
3.11.通用工具模块

有些工具或通用的约定内容，我们希望各个服务共享，因此需要创建一个工具模块：`hz-common`

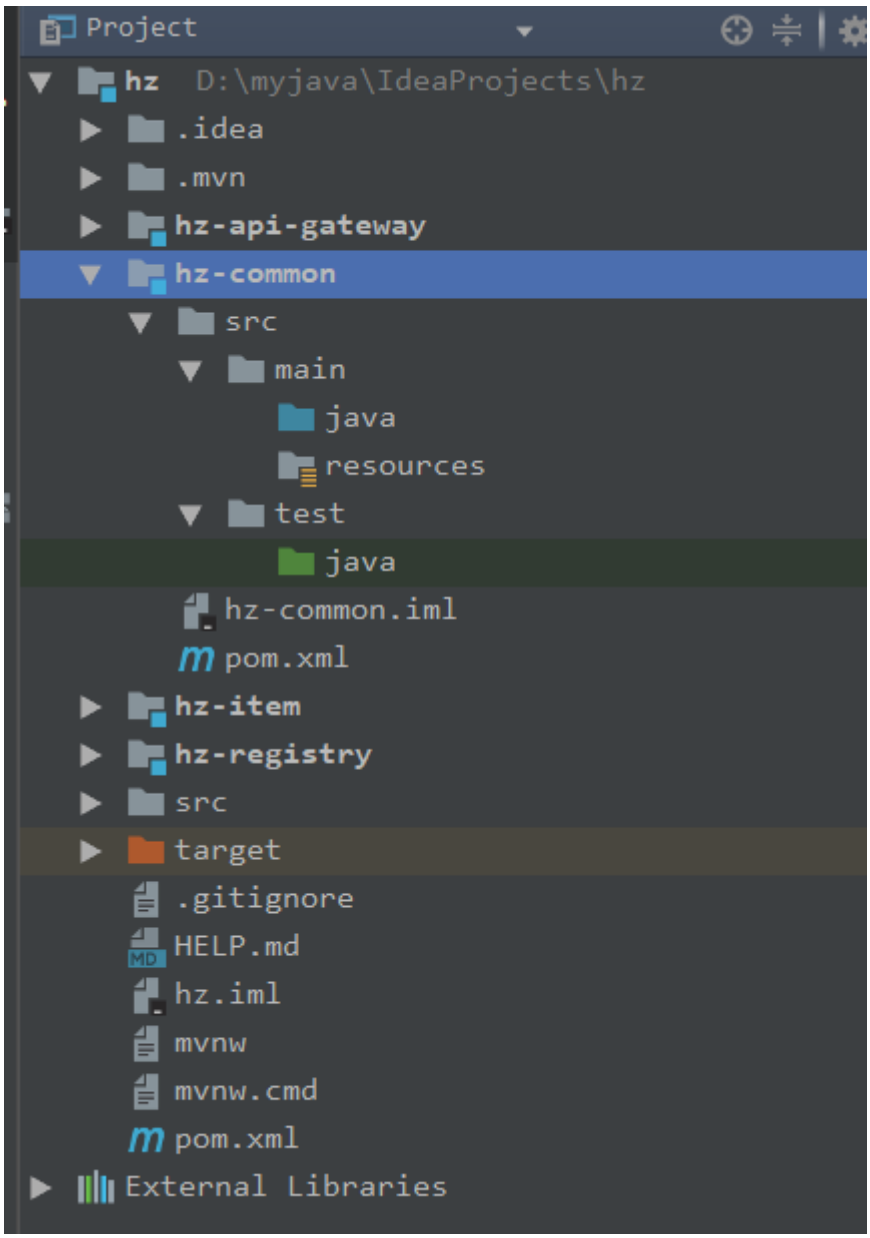
使用maven来构建module：



位置信息：



结构：



目前还不需要编码。

4.搭建后台管理前端

4.1.导入已有资源

后台项目相对复杂，为了有利于教学，我们不再从0搭建项目，而是直接使用课前资料中给大家准备好的源码：

1525955154460

我们解压缩，放到工作目录中：

1525955615381

然后在eclipse中导入新的工程：

1525955644681

选中我们的工程：

1525955709334

这正是一个用vue-cli构建的webpack工程，是不是与昨天的一样：

1525955811416

4.2.安装依赖

你应该注意到，这里并没有node_modules文件夹，方便给大家下发，已经把依赖都删除了。不过package.json中依然定义了我们所需的一切依赖：

1525956016683

我们只需要通过命令来安装所需依赖即可。打开终端，进入项目目录，输入：`npm install`

1525957503521

大概需要1分钟。

4.3.运行一下看看

输入命令：

```
npm run dev
```

1525957604219

发现默认的端口是9001。访问：http://localhost:9001

会自动进行跳转：

1525958950616