第1章 青橙电商系统工程搭建

1. 走进电商

1.1 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。

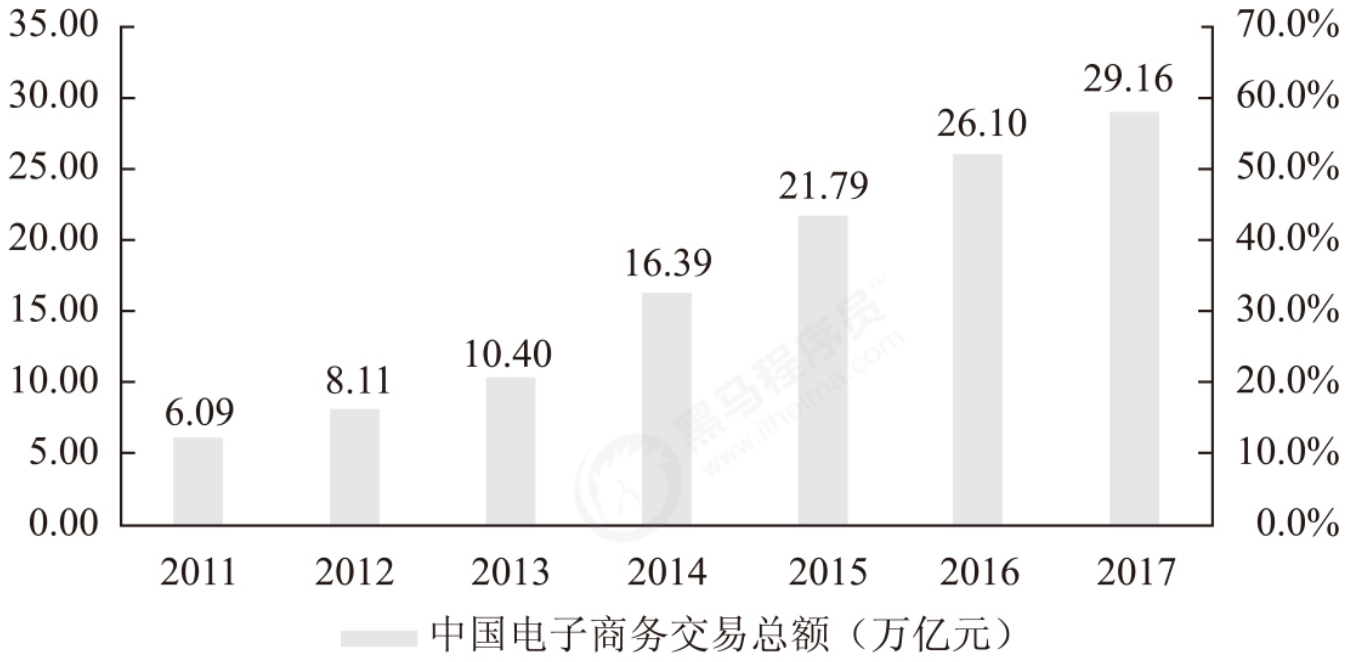


图 1-1　2011—2017 中国电子商务交易总额及增长率


全天成交额纪录
2018年
历年
100亿
2分05秒
2017年3分01秒
191亿
4分20秒
2012年双11全天成交额
362亿
12分14秒
2013年双11全天成交额
500亿
26分03秒
2017年40分12秒
571亿
35分17秒
2014年双11全天成交额
2017年1小时0分49秒
912亿
1小时16分37秒
2015年双11全天成交额
1000亿
1小时47分26秒
2017年9小时0分4秒
1111亿
6小时4分12秒
2017年10小时54分26秒
1207亿
8小时8分52秒
2016年双11全天成交额
2017年13小时9分49秒
1500亿
12小时8分40秒
2017年21小时12分35秒

1.2 电商系统技术特点

技术新

技术范围广

分布式

高并发、集群、负载均衡、高可用

海量数据

业务复杂

系统安全

1.3 主要电商模式

- 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。

  注：我们《青橙电商系统》课程采用**B2C**模式

  案例：唯品会、乐蜂网

- 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。

  案例：京东商城、天猫商城

- 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 SupplyModel）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、饿了吗

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

2. 青橙-需求分析与系统设计

2.1 需求分析

《青橙》是一个全品类B2C电商平台，包含网站前台和管理后台两大部分。网站前台包含主站频道（首页、搜索、购物车及支付）、用户中心、秒杀、优惠券等频道。管理后台包含商品、订单、库存、用户、运营、统计、财务、设置等功能。具体功能请看课程提供的静态原型。

2.2 系统设计

2.2.1 数据库分库设计

（1）商品库 qingcheng_goods

（2）订单库 qingcheng_order

（3）基础设置库 qingcheng_config

（4）运营库 qingcheng_business

（5）用户库 qingcheng_user

（6）系统库qingcheng_system

（7）支付库 qingcheng_pay

（8）短信库 qingcheng_sms

2.2.2 技术选型

主框架技术：SSM（通用mapper）+Dubbo

前端技术： 网站后台 Vue.js+ElementUI 网站前台采用Vue.js 和模板技术 thymeleaf

消息中间件技术： RabbitMQ

搜索中间件技术： elasticsearch

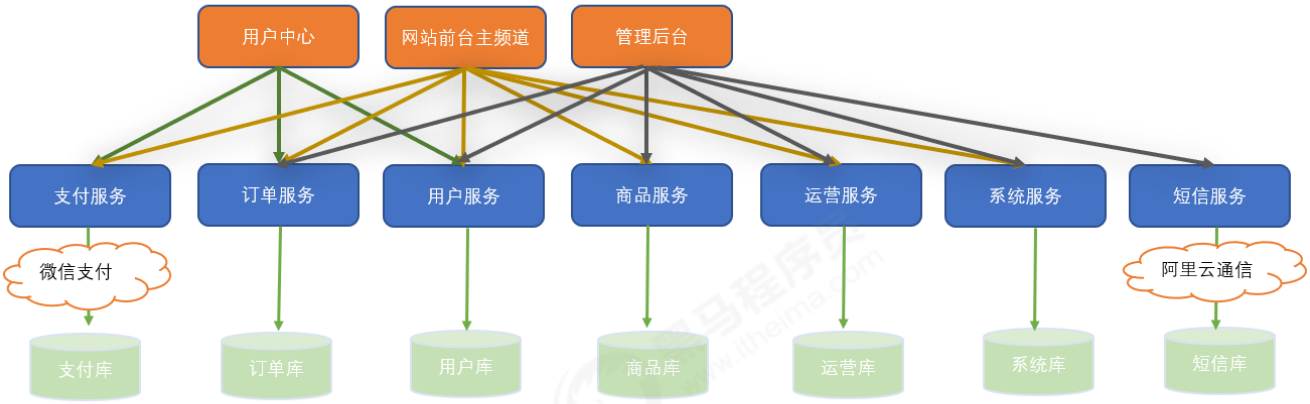
缓存中间件技术： redis

报表插件： echars

安全框架：SpringSecurity

单点登录中间件 ：CAS

2.2.3 系统架构图



3. 通用mapper

3.1 通用mapper简介

通用 Mapper 是一个可以实现任意 MyBatis 通用方法的框架，项目提供了常规的增删改查操作以及Example相关的单表操作。为什么要用通用mapper？我们这里列举一下原生Mybatis的痛点：

1、mapper.xml文件里有大量的sql，当数据库表字段变动，配置文件就要修改

2、需要自己实现sql分页，select * from table where . . . limit 1,3

自己手写分页，除了传参page、pageSize，还需要返回条目总数count。

3、数据库可移植性差：如果项目更换数据库，比如oracle-->mysql，mapper.xml中的sql要重新写，因为Oracle的PLSQL 和mysql 支持的函数是不同的。

4、生成的代码量过大。

5、批量操作，批量插入，批量更新，需要自写。

而这些，通过通用mapper就可以很轻松的解决了。

3.2 通用mapper快速入门

在线官方文档：https://gitee.com/free/Mapper/wikis/Home

3.2.1 通用mapper与Spring集成

官方的文档中介绍了通用mapper的三种使用方式，纯java使用方式、与Spring集成方式、与SpringBoot集成方式。我们这里给大家介绍的是与Spring集成方式。

（1）引入依赖

正常情况下，Spring 和 MyBatis 的集成环境中，应该已经存在下面的依赖：

```
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis</artifactId>
  <version>版本号</version>
</dependency>
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis-spring</artifactId>
  <version>版本号</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <version>版本号</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-tx</artifactId>
  <version>版本号</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-jdbc</artifactId>
  <version>版本号</version>
</dependency>
```

集成通用 Mapper 在上面的基础上添加下面的依赖：

```
<dependency>
    <groupId>tk.mybatis</groupId>
    <artifactId>mapper</artifactId>
    <version>最新版本</version>
</dependency>
```

（2）与spring集成

和通用 Mapper 以前版本一样，可以直接使用 tk.mybatis 提供的 `tk.mybatis.spring.mapper.MapperScannerConfigurer` 进行配置，这个配置和 MyBatis 官方提供的 `org.mybatis.spring.mapper.MapperScannerConfigurer` 区别只是第一层的包名，`tk` 和 `org`。所以使用这种方式时，如果你项目已经使用 `org.` 进行了配置，只需要改成 `tk.` 即可。

```
<bean class="tk.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="扫描包名"/>
</bean>
```

我们项目中采用的是这种xml的配置方式，通用mapper还提供了注解方式的配置，详见文档。

3.2.2 实体类映射

实体类映射类似下列形式

```
@Table(name="tb_brand")
public class Brand implements Serializable{

    @Id
    private Integer id;

    private String name;

    //getter and setter ....
}
```

@Table是指定实体类对应的数据库表 @Id指的是主键映射。经过上面简单的配置后，相当于就有了 MyBatis 中的关系映射了

3.2.3 创建Mapper接口

```
public interface BrandMapper extends Mapper<Brand> {

}
```

这里继承了 `tk.mybatis.mapper.common.Mapper` 接口，在接口上指定了泛型类型 `Brand`。当你继承了 `Mapper` 接口后，此时就已经有了针对 `Brand` 的大量方法，方法如下：

这些方法中和 MBG 生成的大部分方法都一致，还有一部分 MBG 之外的常用方法。

基础接口**select**

```
List<T> select(T record)
根据T对象中的属性名称查询,类似于select * from table where t.name=#{name} and t.password = #{password}
```

```
T selectOne(T record)
根据实体中的属性进行查询，只能有一个返回值，有多个结果是抛出异常，查询条件使用等号
```

```
T selectByPrimaryKey(Object key)
根据主键查询    说明：根据主键字段进行查询，方法参数必须包含完整的主键属性，查询条件使用等号
```

```
int selectCount(T record);
说明：根据实体中的属性查询总数，查询条件使用等号
```

基础接口**insert**

```
int insert(T record);
说明：保存一个实体，null的属性也会保存，不会使用数据库默认值
```

```
int insertSelective(T record);
说明：保存一个实体，null的属性不会保存，会使用数据库默认值
```

基础接口**Update**

```
int updateByPrimaryKey(T record);
说明：根据主键更新实体全部字段，null值会被更新
```

```
int updateByPrimaryKeySelective(T record);
说明：根据主键更新属性不为null的值
```

基础接口**delete**

```
int delete(T record);
说明：根据实体属性作为条件进行删除，查询条件使用等号
```

```
int deleteByPrimaryKey(Object key);
说明：根据主键字段进行删除，方法参数必须包含完整的主键属性
```

4. 青橙-工程搭建

4.1 准备工作

（1）配置maven本地仓库 【资料/本地仓库】

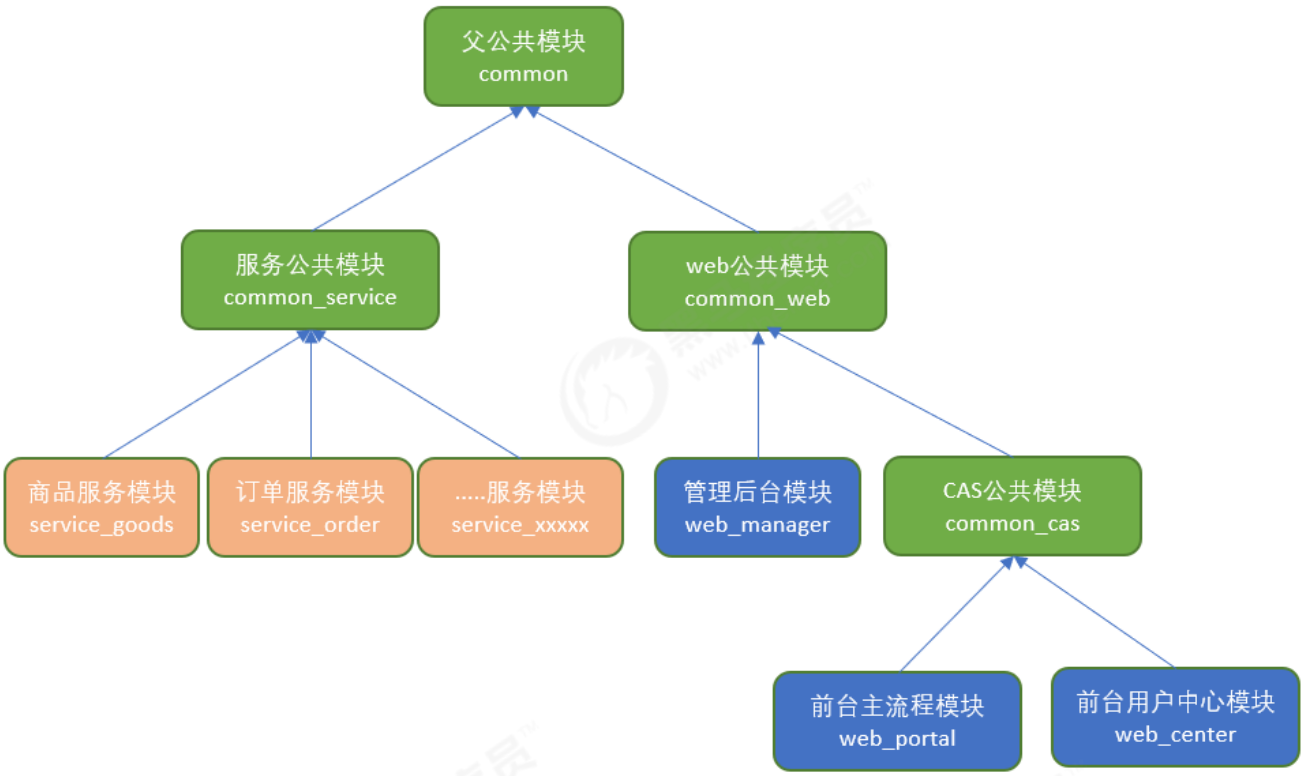
（2）创建数据库表 【资料/建表语句】

（3）注册中心zookeeper 【资料/配套软件 】

4.2 模块依赖关系图

我们的工程有三种模块：（1）公共模块 （2）服务层模块 （3）web层模块。

公共模块主要由公共配置和公共类构成。模块依赖关系入下图：



4.3 工程搭建

4.3.1 父工程与公共模块

（1）创建父工程qingcheng_parent pom.xml内容参见资源/配置文件/主架构/父工程/pom.xml

（2）创建公共模块qingcheng_common，pom.xml内容参见资源/配置文件/主架构/公共模块/父公共模块/pom.xml　resources下创建applicationContext-common.xml

```
<context:property-placeholder location="classpath*:*.properties" />
```

resources下创建dubbo-address.properties

```
dubbo.address=192.168.25.130:2181
```

resources下创建log4j.properties

```
### direct log messages to stdout ###
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.err
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L -
%m%n

### direct messages to file mylog.log ###
log4j.appender.file=org.apache.log4j.FileAppender
log4j.appender.file.File=c:\\mylog.log
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L -
%m%n

### set log levels - for more verbose logging change 'info' to 'debug' ###

log4j.rootLogger=debug, stdout
```

（3）创建qingcheng_common_service ， pom.xml 参见资源/配置文件/主架构/公共模块/服务公共模块/pom.xml

resources下创建applicationContext-dao.xml

```
    <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource"
        destroy-method="close">
        <property name="url" value="${jdbc.url}" />
        <property name="username" value="${jdbc.username}" />
        <property name="password" value="${jdbc.password}" />
        <property name="driverClassName" value="${jdbc.driver}" />
        <property name="maxActive" value="10" />
        <property name="minIdle" value="5" />
    </bean>
    <bean id="sqlSessionFactory"
class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="dataSource" />
        <property name="plugins">
            <array>
                <bean class="com.github.pagehelper.PageHelper">
                    <property name="properties">
                        <value>
                            dialect=mysql
                        </value>
                    </property>
                </bean>
            </array>
        </property>
    </bean>
    <bean class="tk.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="basePackage" value="com.qingcheng.dao" />
    </bean>
    <!-- 事务管理器  -->
    <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource" />
    </bean>
    <tx:annotation-driven transaction-manager="transactionManager" proxy-
target-class="true"/>
```

resources下创建applicationContext-dubbo.xml

```
    <dubbo:protocol name="dubbo" port="${dubbo.port}"/>
    <dubbo:application name="${dubbo.application}" />
    <dubbo:registry protocol="zookeeper" address="192.168.25.130:2181" />
    <dubbo:annotation package="com.qingcheng.service" />
```

（4）创建qingcheng_common_web模块，pom.xml

```
        <dependency>
            <groupId>com.qingcheng</groupId>
            <artifactId>qingcheng_common</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>commons-fileupload</groupId>
            <artifactId>commons-fileupload</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-config</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-taglibs</artifactId>
        </dependency>
```

resources下创建applicationContext-json.xml

```
    <mvc:annotation-driven>
      <mvc:message-converters register-defaults="true">
        <bean
class="com.alibaba.fastjson.support.spring.FastJsonHttpMessageConverter">
          <property name="supportedMediaTypes" value="application/json"/>
          <property name="features">
            <list>
              <value>WriteMapNullValue</value>
              <value>WriteDateUseDateFormat</value>
            </list>
          </property>
        </bean>
      </mvc:message-converters>
    </mvc:annotation-driven>
```

resources下创建applicationContext-dubbo.xml

```
    <!-- 引用dubbo 服务 -->
    <dubbo:application name="${dubbo.application}" />
    <dubbo:registry protocol="zookeeper" address="${dubbo.address}"/>
    <dubbo:annotation package="com.qingcheng.controller" />
```

（5）创建实体层模块qingcheng_pojo，pom.xml

```
        <dependency>
            <groupId>javax.persistence</groupId>
            <artifactId>persistence-api</artifactId>
            <version>1.0</version>
            <scope>compile</scope>
        </dependency>
```

（6）创建服务接口层模块 qingcheng_interface ， pom.xml

```
        <dependency>
            <groupId>com.qingcheng</groupId>
            <artifactId>qingcheng_pojo</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
```

4.3.2 服务层模块（商品）

（1）创建qingcheng_service_goods模块，pom.xml

```
    <dependencies>
        <dependency>
            <groupId>com.qingcheng</groupId>
            <artifactId>qingcheng_interface</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>com.qingcheng</groupId>
            <artifactId>qingcheng_common_service</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.tomcat.maven</groupId>
                <artifactId>tomcat7-maven-plugin</artifactId>
                <configuration>
                    <!-- 指定端口 -->
                    <port>9001</port>
                    <!-- 请求路径 -->
                    <path>/</path>
                </configuration>
            </plugin>
        </plugins>
    </build>
```

（2）创建webapp/WEB-INF/web.xml

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >
<web-app>
  <display-name>Archetype Created Web Application</display-name>
  <!-- 加载spring容器 -->
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath*:applicationContext*.xml</param-value>
  </context-param>
  <listener>
    <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-
class>
  </listener>
</web-app>
```

（3）resources下创建dubbo.properties

```
dubbo.port=20881
dubbo.application=goods
```

（4）resources下创建db.properties

```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/qingcheng_goods?
characterEncoding=utf-8
jdbc.username=root
jdbc.password=123456
```

4.3.3 web层（管理后台）

（1）创建qingcheng_web_manager模块，pom.xml

```
    <dependencies>
        <dependency>
            <groupId>com.qingcheng</groupId>
            <artifactId>qingcheng_interface</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>com.qingcheng</groupId>
            <artifactId>qingcheng_common_web</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.tomcat.maven</groupId>
                <artifactId>tomcat7-maven-plugin</artifactId>
                <configuration>
                    <!-- 指定端口 -->
                    <port>9101</port>
                    <!-- 请求路径 -->
                    <path>/</path>
                </configuration>
            </plugin>
        </plugins>
    </build>
```

（2）创建webapp/WEB-INF/web.xml

```
<web-app>
  <display-name>Archetype Created Web Application</display-name>
  <!-- 解决post乱码 -->
  <filter>
    <filter-name>CharacterEncodingFilter</filter-name>
    <filter-
class>org.springframework.web.filter.CharacterEncodingFilter</filter-
class>
    <init-param>
      <param-name>encoding</param-name>
      <param-value>utf-8</param-value>
    </init-param>
    <init-param>
      <param-name>forceEncoding</param-name>
      <param-value>true</param-value>
    </init-param>
  </filter>
  <filter-mapping>
    <filter-name>CharacterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

  <servlet>
    <servlet-name>springmvc</servlet-name>
    <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <!-- 指定加载的配置文件 ，通过参数contextConfigLocation加载 -->
    <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>*.do</url-pattern>
  </servlet-mapping>
</web-app>
```

（3）resources下创建dubbo.properties

```
dubbo.application=manager
```

5. 青橙管理后台-品牌管理后端

5.1 需求分析

实现对品牌的基本操作（增删改查），只完成后端代码部分，并通过浏览器等工具完成测试。

5.2 表结构分析

tb_brand　品牌表

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	100	
image	品牌图片地址	VARCHAR	1000	
letter	品牌的首字母	CHAR	1	
seq	排序	INT		

5.3 代码实现

5.3.1 品牌列表

url

> /brand/findAll.do

http请求方式

> GET

返回格式

```
[{
    "id": 品牌id,
    "name": 品牌名称,
    "image": 品牌图片地址,
    "letter": 品牌的首字母,
    "seq": 排序,

},
.......
]
```

代码实现：

（1）在qingcheng_pojo工程创建com.qingcheng.pojo包，包下创建实体类

```
@Table(name="tb_brand")
public class Brand implements Serializable{

    @Id
    private Integer id;//品牌id
    private String name;//品牌名称
    private String image;//品牌图片地址
    private String letter;//品牌的首字母
    private Integer seq;//排序

    // getter and setter ......

}
```

（2）qingcheng_service_goods工程创建com.qingcheng.dao，包下创建数据访问层接口

```
public interface BrandMapper extends Mapper<Brand> {

}
```

（3）qingcheng_interface工程创建com.qingcheng.service.goods包，包下创建业务接口

```
/**
 * 品牌业务逻辑层
 */
public interface BrandService {
    public List<Brand> findAll();
}
```

（4）qingcheng_service_goods工程创建com.qingcheng.service.impl包，包下创建类

```
@Service
public class BrandServiceImpl implements BrandService {

    @Autowired
    private BrandMapper brandMapper;

    public List<Brand> findAll() {
        return brandMapper.selectAll();
    }
}
```

（5）qingcheng_web_manager工程创建com.qingcheng.controller.goods 包，包下创建类

```
@RestController
@RequestMapping("/brand")
public class BrandController {

    @Reference
    private BrandService brandService;


    @GetMapping("/findAll")
    public List<Brand> findAll(){
        return brandService.findAll();
    }

}
```

启动工程，浏览器测试：http://localhost:9101/brand/findAll.do

5.3.2 品牌分页列表

接口定义：

url

> /brand/findPage.do

http请求方式

> GET

请求参数

参数	必选	类型	说明
page	true	int	页码
size	true	int	每页记录数

例子：

```
GET /brand/findPage.do?page=1&size=10
```

返回格式

```
{rows:[{
        "id": 品牌id,
        "name": 品牌名称,
        "image": 品牌图片地址,
        "letter": 品牌的首字母,
        "seq": 排序,

},
.......
],
total:100}
```

代码实现：

（1）qingcheng_pojo创建com.qingcheng.entity包，包下创建类

```java
/**
 * 分页结果
 * @param <T>
 */
public class PageResult<T> implements Serializable {

    private Long total;//记录数
    private List<T> rows;//结果集

    public PageResult(Long total, List<T> rows) {
        this.total = total;
        this.rows = rows;
    }

    public PageResult() {
    }

    public Long getTotal() {
        return total;
    }

    public List<T> getRows() {
        return rows;
    }

    public void setRows(List<T> rows) {
        this.rows = rows;
    }

    public void setTotal(Long total) {
        this.total = total;
    }

}
```

（2）qingcheng_interface工程BrandService接口新增方法

```java
    public PageResult<Brand> findPage(int page, int size);
```

（3）qingcheng_service_goods工程BrandServiceImpl新增方法

```
    /**
     * 分页查询
     * @param page 页码
     * @param size 每页记录数
     * @return 分页结果
     */
    public PageResult<Brand> findPage(int page, int size) {
        PageHelper.startPage(page,size);
        Page<Brand> brands = (Page<Brand>) brandMapper.selectAll();
        return new PageResult<Brand>
(brands.getTotal(),brands.getResult());
    }
```

（4）qingcheng_web_manager工程BrandController新增方法

```
    @GetMapping("/findPage")
    public PageResult<Brand> findPage(int page, int size){
        return brandService.findPage(page, size);
    }
```

启动工程，浏览器测试：http://localhost:9101/brand/findPage.do?page=1&size=10

5.3.3 品牌条件查询

url

> /brand/findList.do

http请求方式

> POST

请求参数

参数	必选	类型	说明
searchMap	true	Map	条件对象，格式如实体对象

例子：

```
POST /brand/findList.do
{
        "name": 品牌名称,
        "letter": 品牌的首字母
}
```

返回格式

```
[{
        "id": 品牌id,
        "name": 品牌名称,
        "image": 品牌图片地址,
        "letter": 品牌的首字母,
        "seq": 排序,
},
.......
]
```

代码实现：

（1）qingcheng_interface工程BrandService接口新增方法

```
public List<Brand> findList(Map<String, Object> searchMap);
```

（2）qingcheng_service_goods工程BrandServiceImpl新增方法

```
    /**
     * 条件查询
     * @param searchMap 查询条件
     * @return
     */
    public List<Brand> findList(Map<String, Object> searchMap) {
        Example example = createExample(searchMap);
        return brandMapper.selectByExample(example);
    }

    /**
     * 构建查询条件
     * @param searchMap
     * @return
     */
    private Example createExample(Map<String, Object> searchMap){
        Example example=new Example(Brand.class);
        Example.Criteria criteria = example.createCriteria();
        if(searchMap!=null){
            //名称条件
            if(searchMap.get("name")!=null &&
!"".equals(searchMap.get("name"))){
                criteria.andLike("name","%"+
(String)searchMap.get("name")+"%");
            }
            //首字母
            if(searchMap.get("letter")!=null &&
!"".equals(searchMap.get("letter"))){
                criteria.andEqualTo("letter",
(String)searchMap.get("letter"));
            }
        }
        return example;
    }
```

（3）qingcheng_web_manager工程BrandController新增方法

```
    @PostMapping("/findList")
    public List<Brand> findList(@RequestBody Map<String,Object>
searchMap){
        return brandService.findList(searchMap);
    }
```

5.3.4 品牌条件+分页查询

url

> /brand/findPage.do

http请求方式

> POST

请求参数

参数	必选	类型	说明
searchMap	true	Map	条件对象，格式如实体对象
page	true	int	页码（GET）
size	true	int	每页记录数（GET）

例子：

```
POST /brand/findPage.do?page=1&size=10
{
        "name": 品牌名称,
        "letter": 品牌的首字母
}
```

返回格式：

```
{rows:[{
        "id": 品牌id,
        "name": 品牌名称,
        "image": 品牌图片地址,
        "letter": 品牌的首字母,
        "seq": 排序
},
.......
],
total:100}
```

代码实现：

（1）qingcheng_interface工程BrandService接口新增方法

```
public PageResult<Brand> findPage(Map<String,Object> searchMap,int page,
int size);
```

（2）qingcheng_service_goods工程BrandServiceImpl新增方法

```
    /**
     * 分页+条件查询
     * @param searchMap
     * @param page
     * @param size
     * @return
     */
    public PageResult<Brand> findPage(Map<String, Object> searchMap, int
page, int size)     {
        PageHelper.startPage(page,size);
        Example example = createExample(searchMap);
        Page<Brand> brands = (Page<Brand>)
brandMapper.selectByExample(example);
        return new PageResult<Brand>
(brands.getTotal(),brands.getResult());
    }
```

（3）qingcheng_web_manager工程BrandController新增方法

```
    @PostMapping("/findPage")
    public PageResult<Brand> findPage(@RequestBody Map<String,Object>
searchMap,int page, int size){
        return brandService.findPage(searchMap,page,size);
    }
```

5.3.5 根据ID查询品牌

url

> /brand/findById.do

http请求方式

> GET

请求参数

参数	必选	类型	说明
id	true	int	主键

代码实现：

（1）qingcheng_interface工程BrandService接口新增方法

```
public Brand findById(Integer id);
```

（2）qingcheng_service_goods工程BrandServiceImpl新增方法

```
    /**
     * 根据Id查询
     * @param id
     * @return
     */
    public Brand findById(Integer id) {
        return brandMapper.selectByPrimaryKey(id);
    }
```

（3）qingcheng_web_manager工程BrandController新增方法

```
    @GetMapping("/findById")
    public Brand findById(Integer id){
        return brandService.findById(id);
    }
```

5.3.6 品牌新增

（1）qingcheng_pojo 新增类

```java
/**
 * 返回前端的消息封装
 */
public class Result implements Serializable {

    private Integer code;//返回的业务码  0：成功执行  1：发生错误
    private String message;//信息

    public Result(Integer code, String message) {
        this.code = code;
        this.message = message;
    }

    public Result() {
        this.code=0;
        this.message = "执行成功";
    }

    public Integer getCode() {
        return code;
    }

    public void setCode(Integer code) {
        this.code = code;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```

（2）qingcheng_interface工程BrandService接口新增方法

```java
public void add(Brand brand);
```

（3）qingcheng_service_goods工程BrandServiceImpl新增方法

```
    /**
     * 新增
     * @param brand
     */
    public void add(Brand brand) {
        brandMapper.insert(brand);
    }
```

（4）qingcheng_web_manager工程BrandController新增方法

```
    @PostMapping("/add")
    public Result add(@RequestBody Brand brand){
        brandService.add(brand);
        return new Result();
    }
```

5.3.7 品牌修改

（1）qingcheng_interface工程BrandService接口新增方法

```
public void update(Brand brand);
```

（2）qingcheng_service_goods工程BrandServiceImpl新增方法

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

（3）qingcheng_web_manager工程BrandController新增方法

```
    @PostMapping("/update")
    public Result update(@RequestBody Brand brand){
        brandService.update(brand);
        return new Result();
    }
```

5.3.8 品牌删除

url

> /brand/delete.do

http请求方式

> GET

请求参数

参数	必选	类型	说明
id	true	int	主键

例子：

```
GET /brand/delete.do?id=1
```

返回格式：

```
{
  code:0,
  message:""
}
```

code为0表示成功，为1表示失败

代码实现：

（1）qingcheng_interface工程BrandService接口新增方法

```
public void delete(Integer id);
```

（2）qingcheng_service_goods工程BrandServiceImpl新增方法

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

（3）qingcheng_web_manager工程BrandController新增方法

```
    @GetMapping("/delete")
    public Result delete(Integer id){
        brandService.delete(id);
        return new Result();
    }
```

6. 公共异常处理

qingcheng_common_web工程创建com.qingcheng.controller包，包下创建类

```
/**
 * 统一异常处理类
 */
@ControllerAdvice
public class BaseExceptionHandler {

    @ExceptionHandler(Exception.class)
    @ResponseBody
    public Result error(Exception e) {
        e.printStackTrace();
        System.out.println("调用了公共异常处理类");
        return new Result(1,e.getMessage());
    }
}
```