谷粒商城

版本：V 1.0

www.atguigu.com

一、课程简介

1、为什么我们要讲电商？

因为就互联网平台来说，电商网站有很多典型的特征：

- 访问量大
- 数据量大
- 涉及的技术多
- 有一定的业务复杂性
- 涉及支付 考虑一定安全性

2、 我们能从这个项目中学到什么？

巩固以前知识，学会应用：

要新掌握的知识

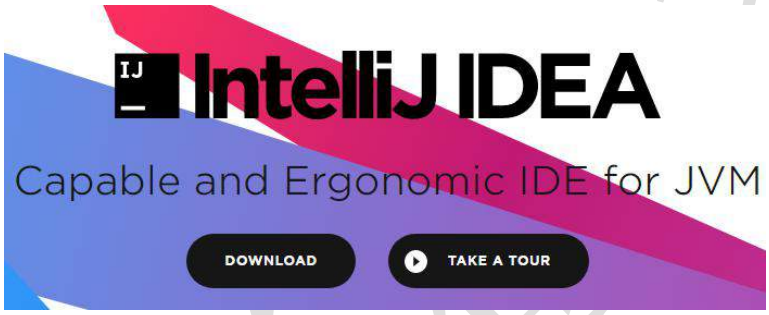
需要掌握的解决方案

课前说明：

二、 Intellij idea

1 介绍

IDEA 全称 IntelliJ IDEA，是 java 语言开发的集成环境，IntelliJ 在业界被公认为最好的 java 开发工具之一，尤其在智能代码助手、代码自动提示、重构、J2EE 支持、各类版本工具(git、svn、github 等)、JUnit、CVS 整合、代码分析、 创新的 GUI 设计等方面的功能可以说是超常的。IDEA 是 JetBrains 公司的产品，这家公司总部位于捷克共和国的首都布拉格，开发人员以严谨著称的东欧程序员为主。它的旗舰版本还支持 HTML，CSS，PHP，MySQL，Python 等。免费版只支持 Java 等少数语言



比起 Eclipse 的好处：

2 安装

解压就可以。

方案一：

前提需要将

0.0.0.0 account.jetbrains.com　　添加到 hosts 文件中

第二种方式 需要有网络的情况下才能注册成功

且在注册成功的情况下,没有网络只能打开第一次,如果打开多次,有可能会需要重新联网注册

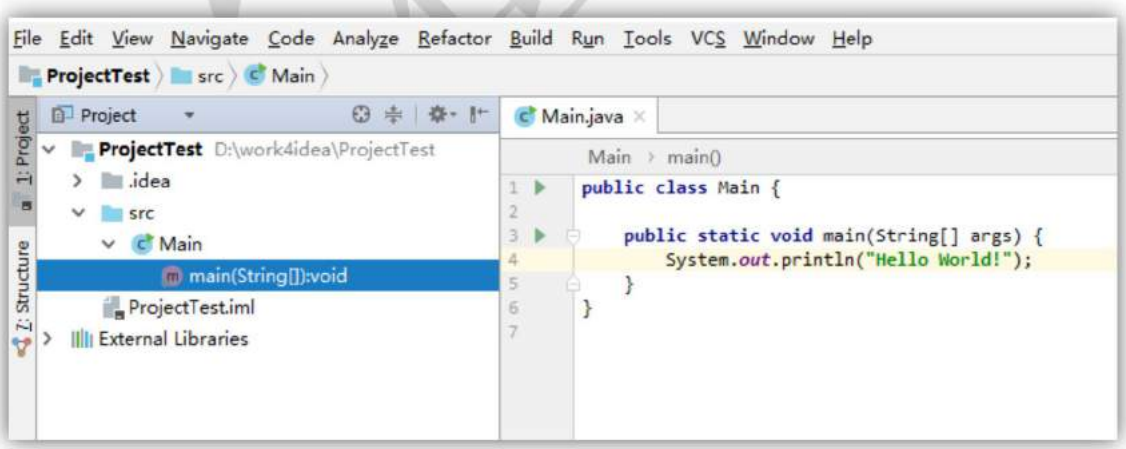
进入 ide 主页面，help-register-license server,然后输入 http://idea.iteblog.com/key.php

3 使用

3.1 Project 与 module

在 idea 中没有 workspace 的概念，每一个窗口只能打开一个 Project。 对于单一工程的项目，直接建一个 Project 在其下面开发就好了。

单一工程的项目：

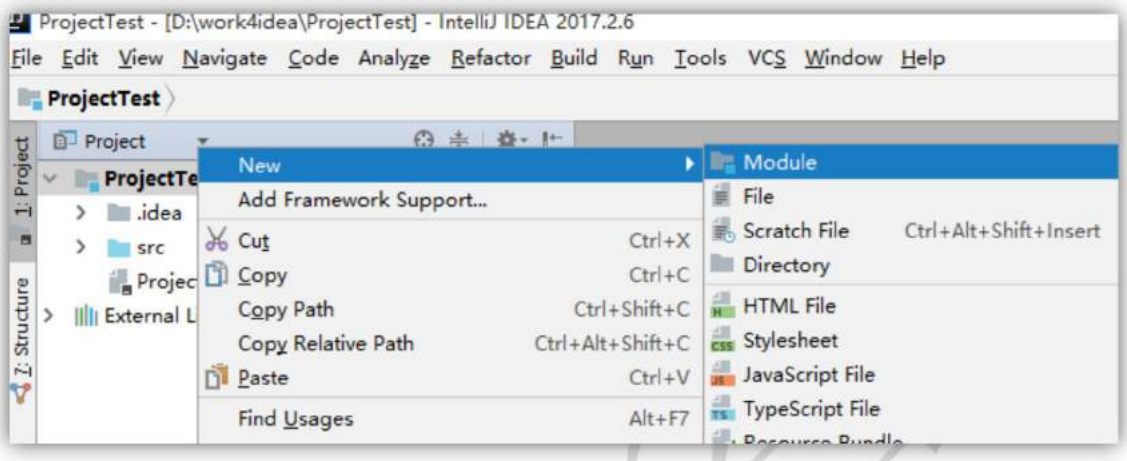


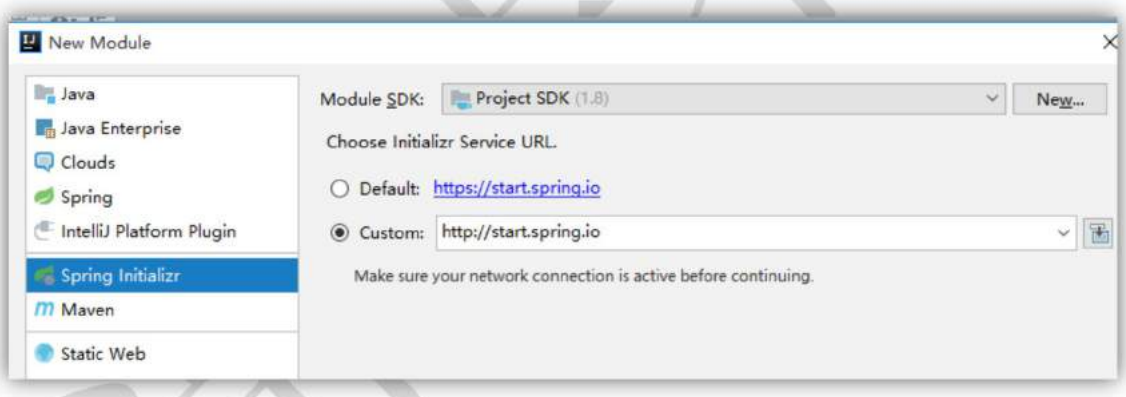
但是我们知道现在稍微大一点的项目都是多项目的分布式部署的，那么岂不是每个子工程都要打开一个窗口？

这时候就需要用到 Module 的概念，Module 是项目的子模块，可以独立运行的工程，当一个多项目组成的系统时，Project 下本身可以不拥有代码，而是作为一种顶级的管理目

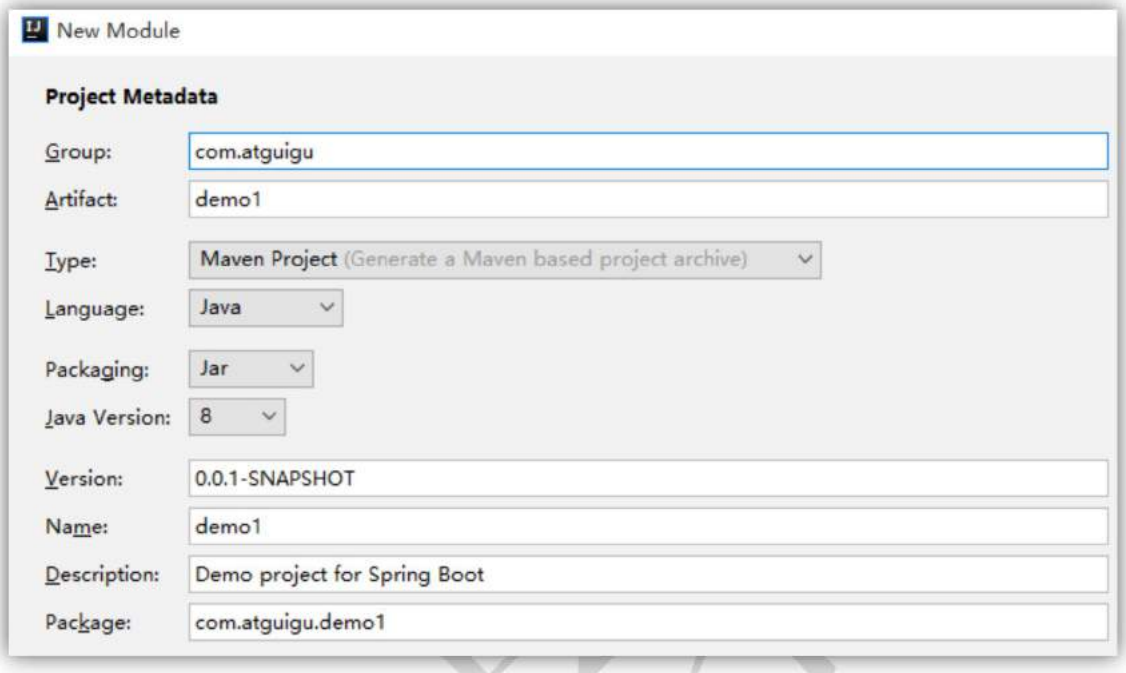
录，所有的代码都放到各个 module 之中。

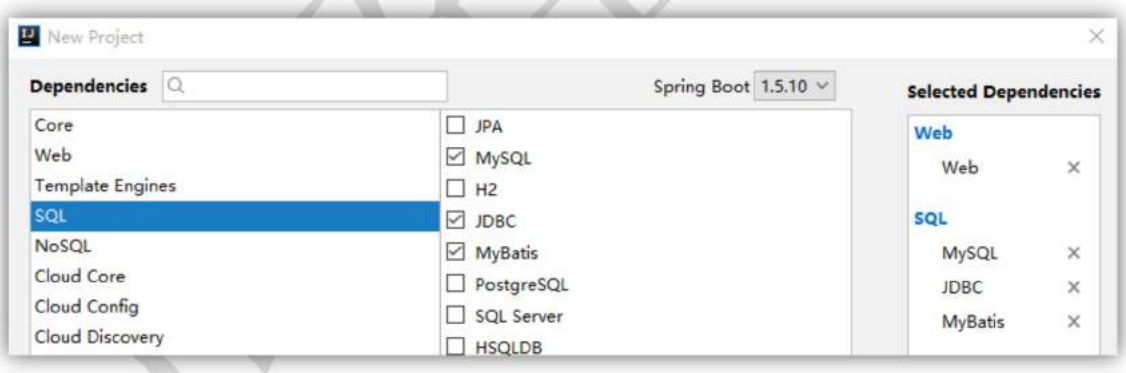
下面我们在这个 Project 下增加 Module，

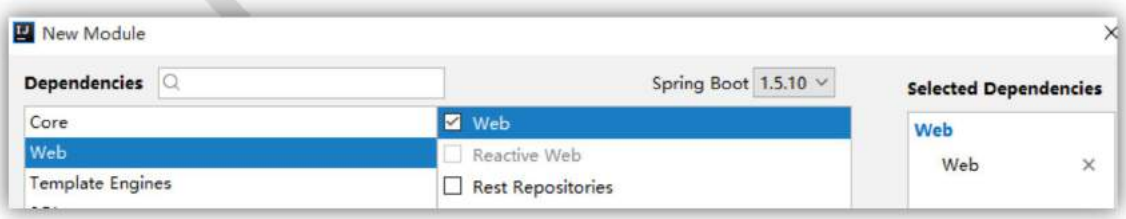


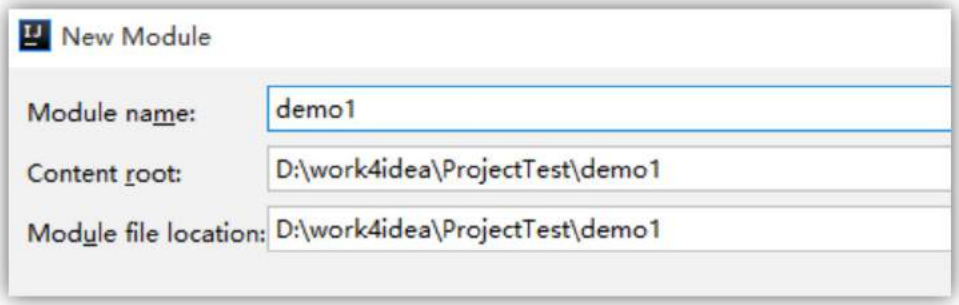


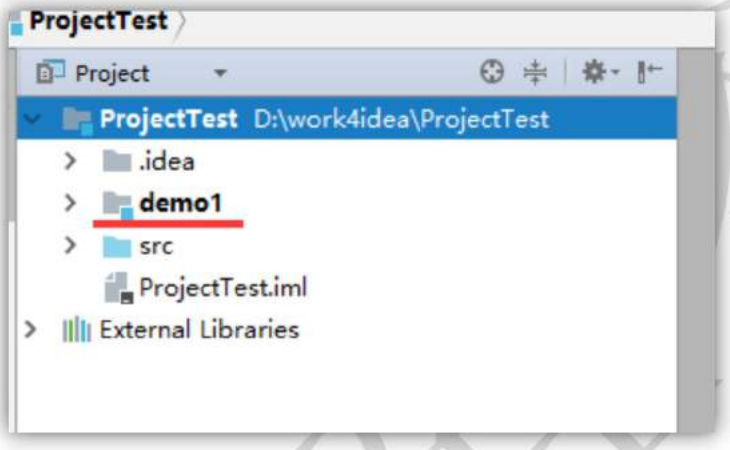
这个时候因为要从网上读取模板所以务必保持联网状态，Spring Initalizr 是 springboot 工程的模板。







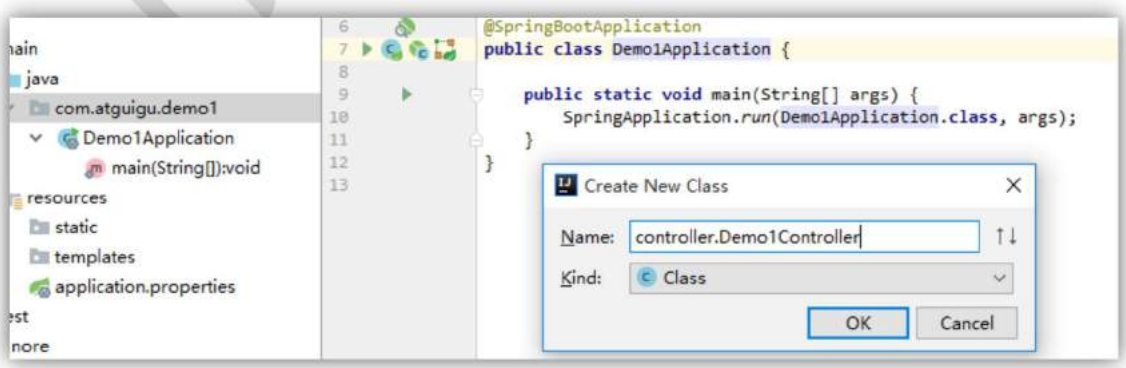




这时候看到 Project 中多了一个 demo1 的 Module 的。

其实这时候 Project 工程下的 src 就没什么用了，可以删掉。

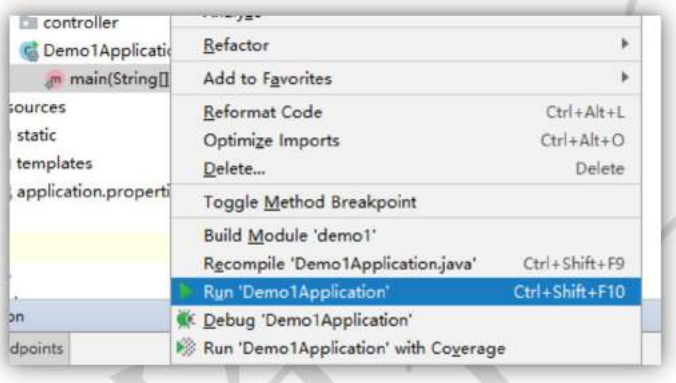
模块建立好了，我们就来用 springmvc 标签建一个 controller 看看好不好使。



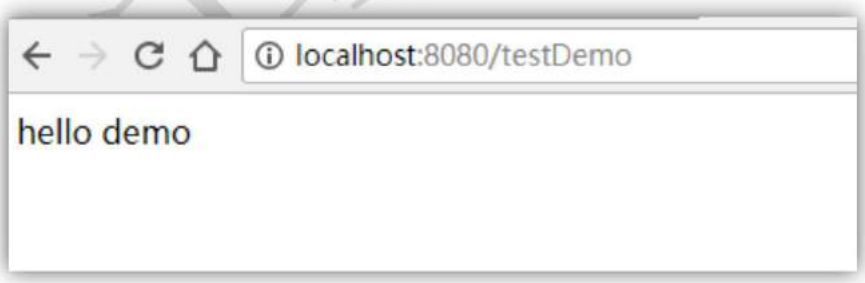
controller 代码

```
@Controller
public class Demo1Controller {
    @ResponseBody
    @RequestMapping("testDemo")
    public String testDemo(){
        return "hello demo";
    }
}
```

运行 Demo1Application 中的 main 方法

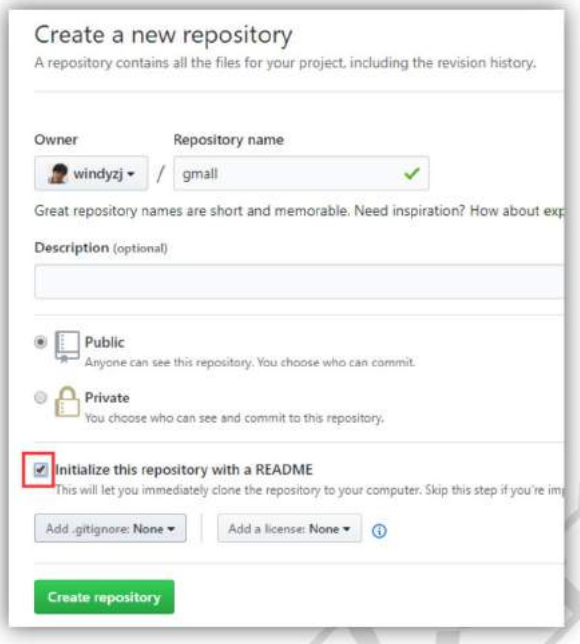


用浏览器测试：

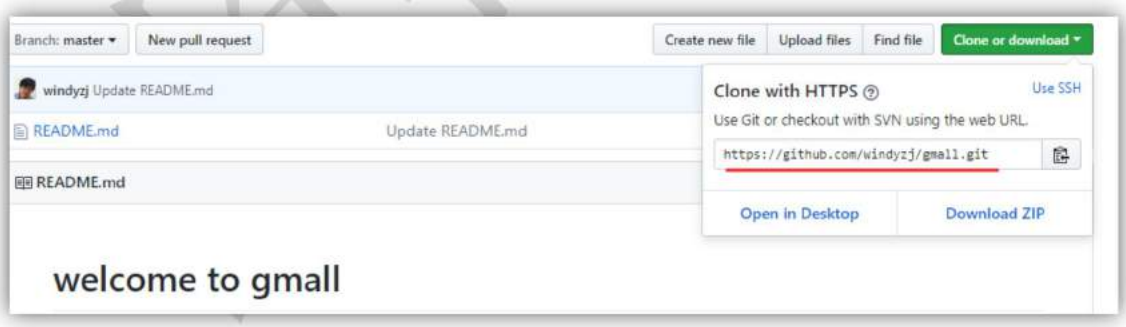


3.2 从 Git 中 clone 项目

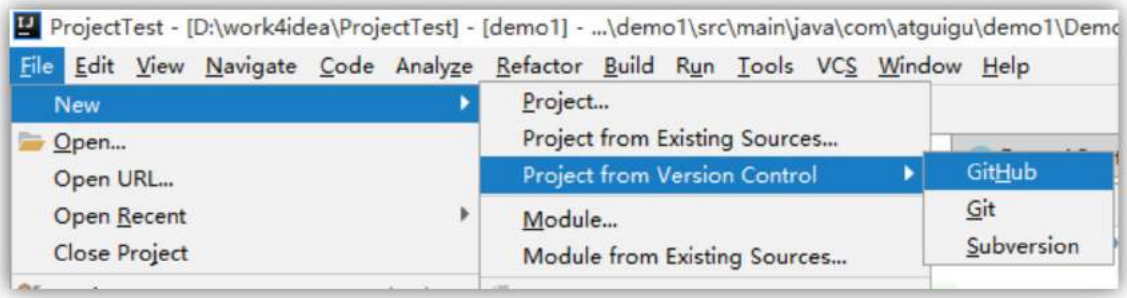
首先要去 GitHub 上创建一个项目

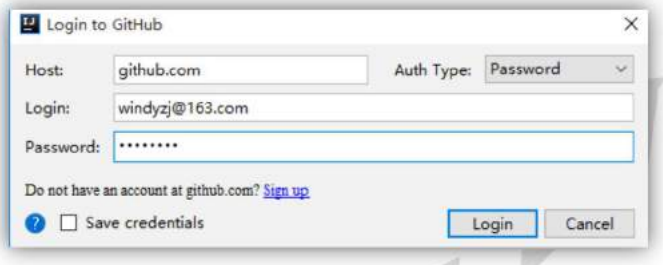


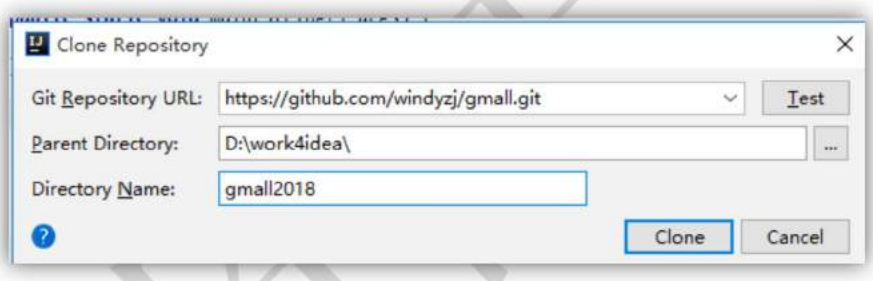
注意的地方是记得加一个 README，这样 clone 下来的工程就不是空的了。

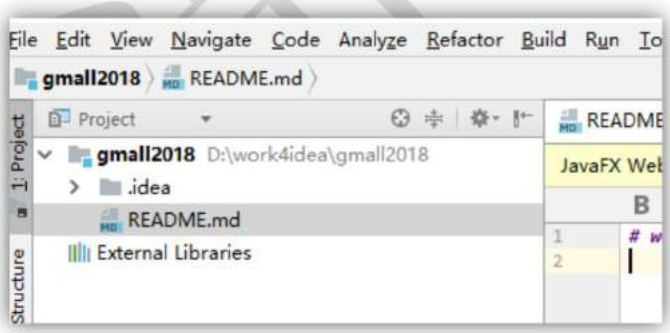


这个就是咱们的仓库地址，咱们来进行第一次复制

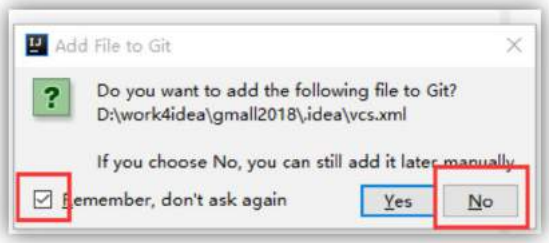






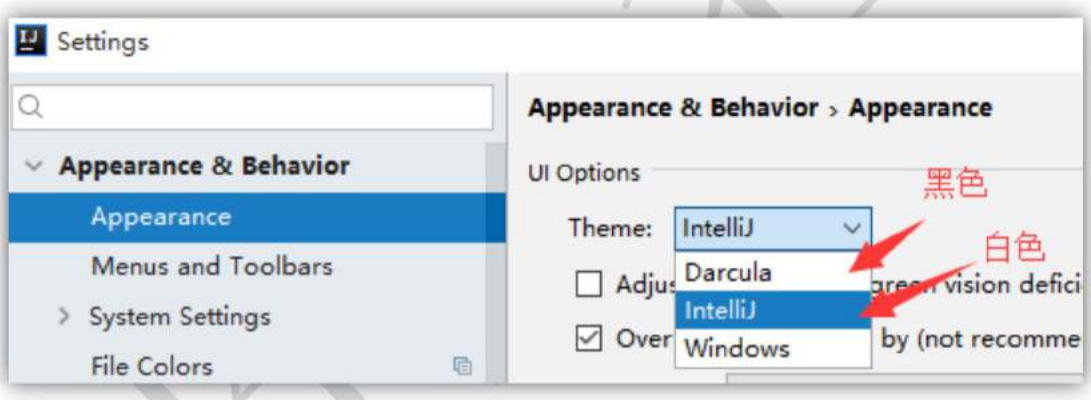


如果弹出提示框如下，问你是否要自动提交某些文件，请一律选 NO,且不再提醒。否则系统会自动提交一些不必要的文件。



4、界面颜色风格

setting->Appearance



5 idea 的快捷键

5.1 常用快捷键

智能补全 引包 alt+Enter

由方法自动生成返回值变量 ctrl+alt+v

跳到方法的实现类 ctrl+alt+b

从实现类跳转到接口 ctrl+u

显示某个接口、抽象类的实现类、子类 ctrl+h

显示最近编辑的文件 ctrl+e

查看方法参数 ctrl+p

查看方法文档 ctrl+q

复制行 ctrl+D

删除行 ctrl+Y

跳转到上一个/下一个位置 ctrl+alt+左右

大小写切换 ctrl+shift+u

5.2 Debug:

F8 执行下一行 (相当于 eclipse 的 F6)
F7 跳入内部 (相当于 eclipse 的 F5)
F9 继续执行 (相当于 eclipse 的 F8)
热部署 ctrl+shift+F9 （仅 debug 模式）

5.3 搜索

全文搜索文本 ctrl+shift+f
全文替换文本 ctrl+shift+r
搜索类 ctrl+n
任何地方搜索 双击 shift

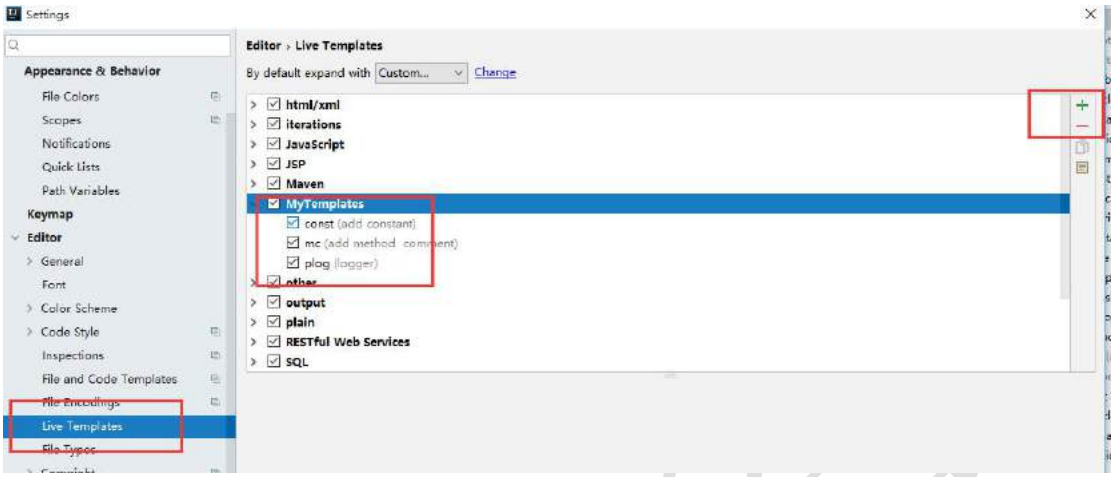
5.4 快速录入

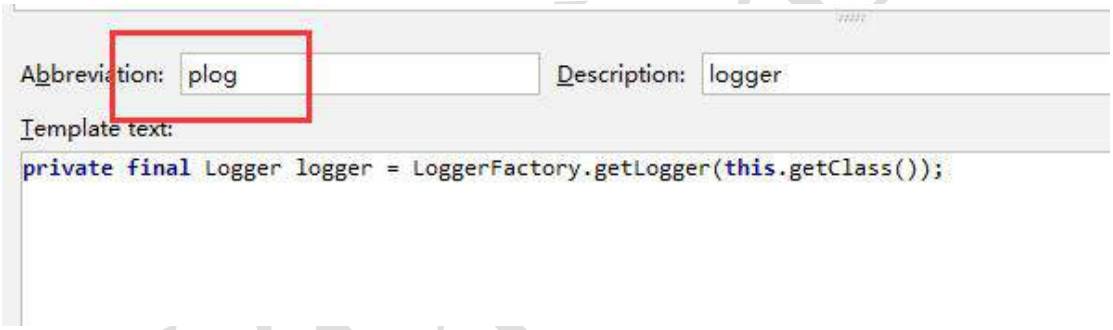
查看快速录入列表 ctrl+j
foreach iter
普通 for 循环 fori
循环数组 itar
迭代器遍历 itco

psvm 主函数
pfs 常量

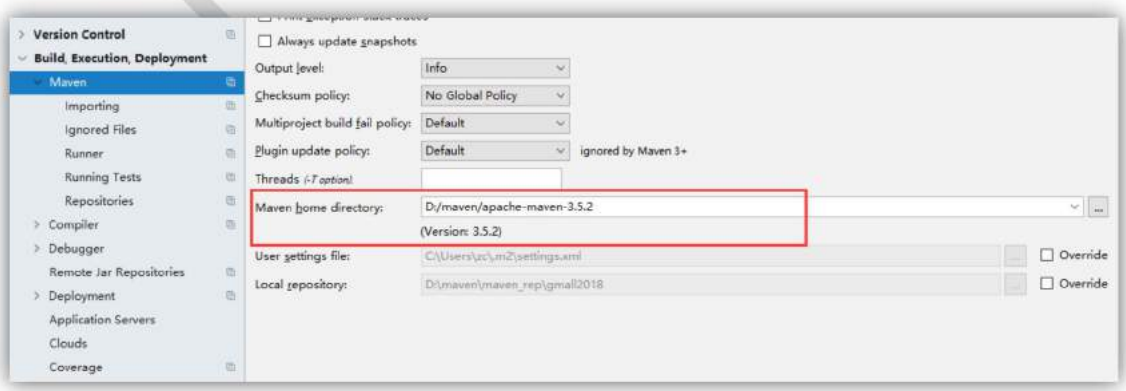
生成代码块： try/ if / for/ while/ synchronized
ctrl+alt+t

6 手工加入快捷键模板





7 配置 maven



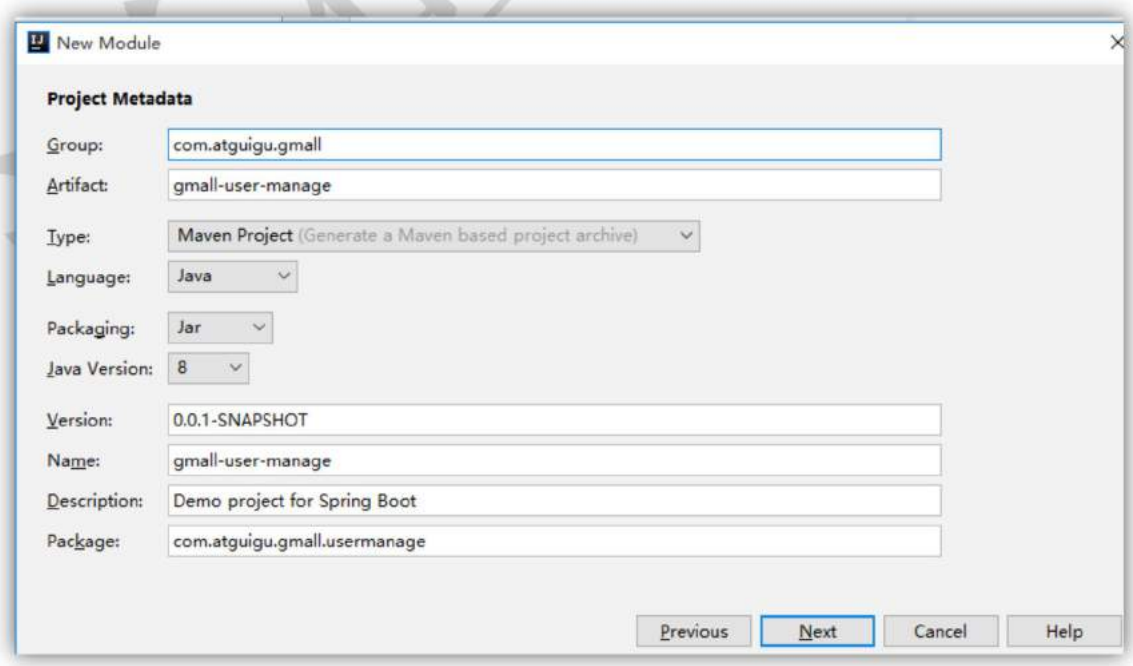
三、EZDML 工具

配置： 工具—>修改 ini 配置

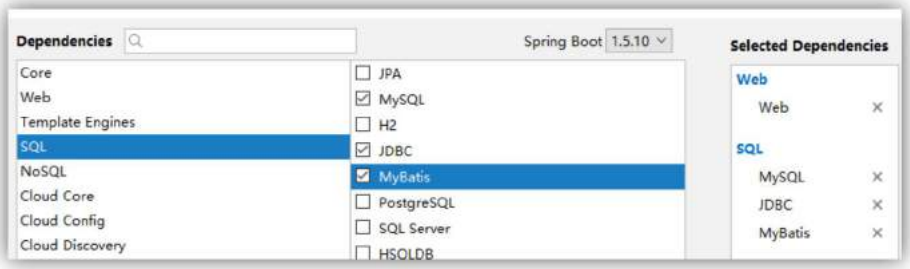
```
[DefaultFieldTypes]
[CustFieldTypes]
1=bigint
2=decimal
[DbConn]
```

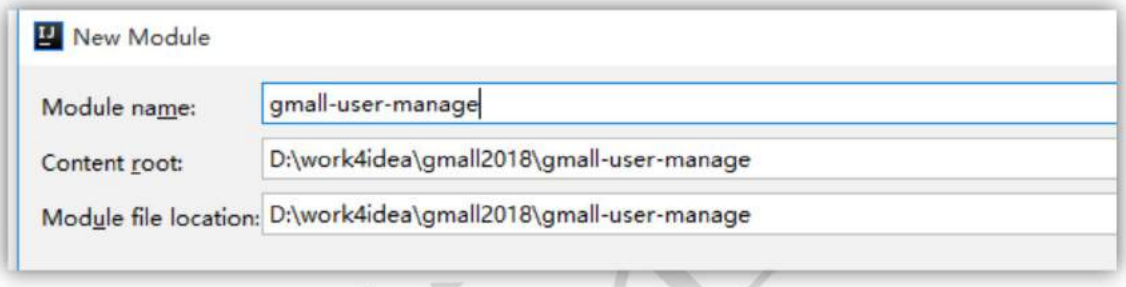
四 通用 Mapper 的使用

1、搭建 module



依赖选 Web 和 Mysql, Jdbc,MyBatis





注意 Module 位置要在 Project 路径下面

2、配置通用 Mapper

在 pom.xml 文件中，加入

```
<!-- 通用mapper -->
<dependency>
    <groupId>tk.mybatis</groupId>
    <artifactId>mapper-spring-boot-starter</artifactId>
    <version>1.2.3</version>
    <exclusions>
        <exclusion>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-jdbc</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

GmallUserManageApplication.java 中增加注解

```
@SpringBootApplication
```

```
@MapperScan(basePackages = "com.atguigu.gmall.usermanage.mapper")
public class GmallOrderServiceApplication {

   public static void main(String[] args) {
      SpringApplication.run(GmallOrderServiceApplication.class, args);
   }
}
```

3、配置数据源

在 application.properties 中

```
spring.datasource.url=jdbc:mysql://localhost:3306/gmall?characterEncoding=UTF-8
spring.datasource.username=root
spring.datasource.password=123123
```

表结构

```
CREATE TABLE `user_info` (
   `id` bigint(20) NOT NULL AUTO_INCREMENT COMMENT '编号',
   `login_name` varchar(200) DEFAULT NULL COMMENT '用户名称',
   `nick_name` varchar(200) DEFAULT NULL COMMENT '用户昵称',
   `passwd` varchar(200) DEFAULT NULL COMMENT '用户密码',
   `name` varchar(200) DEFAULT NULL COMMENT '用户姓名',
   `phone_num` varchar(200) DEFAULT NULL COMMENT '手机号',
   `email` varchar(200) DEFAULT NULL COMMENT '邮箱',
   `head_img` varchar(200) DEFAULT NULL COMMENT '头像',
   `user_level` varchar(200) DEFAULT NULL COMMENT '用户级别',
   PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1000 DEFAULT CHARSET=utf8 COMMENT='用户表'
```

4、代码开发

包	类	说明
controller	UserManageController	web
service	UserManageService	接口
service.impl	UserManageServiceImpl	实现类
bean	UserInfo	实体 bean
mapper	UserInfoMapper	mapper 接口

4.1 bean

```
public class UserInfo implements Serializable{
    @Id
    @Column
    private String id;
    @Column
    private String loginName;
    @Column
    private String nickName;
    @Column
    private String passwd;
    @Column
    private String name;
    @Column
    private String phoneNum;
    @Column
    private String email;
    @Column
    private String headImg;
    @Column
    private String userLevel;
}
```

注意：@Column 和@Id 都是 javax.persistence 包中的

技巧 idea 快捷键：alt+insert 可以快速插入 getter 和 setter

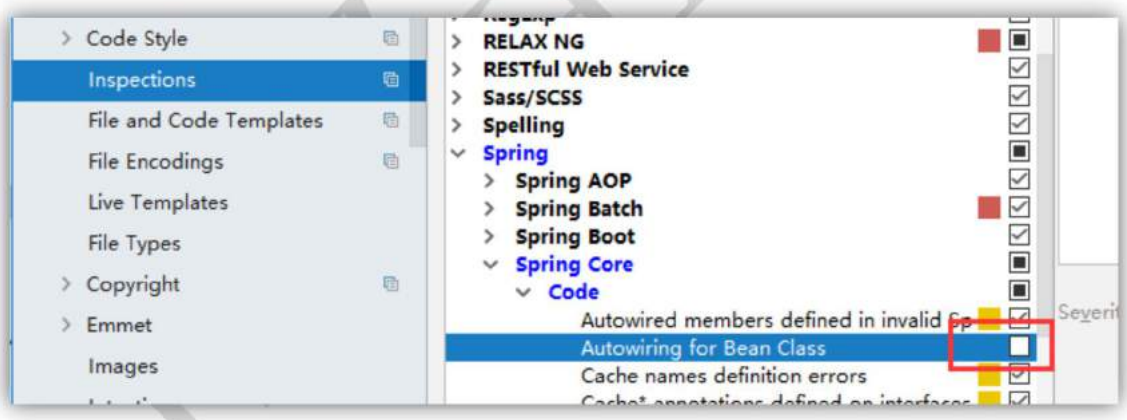
4.2 Mapper

```
public interface UserInfoMapper extends Mapper<UserInfo> {
}
```

注意： Mapper 也是引用 tk.mybatis.mapper.common.Mapper 包中的

Idea 有的时候校验@Autowired 不准 可以把校验关闭

settings -> Inspections -> spring->spring core -> code-> Autowiring for Bean class



4.4 service

```
public interface UserManageService {

    public List<UserInfo> getUserInfoList(UserInfo userInfoQuery);

    public UserInfo getUserInfo(UserInfo userInfoQuery);

    public void delete(UserInfo userInfoQuery);
```

```
    public void addUserInfo(UserInfo userInfo);

    public void updateUserInfo(UserInfo userInfo);

}
```

4.5 ServiceImpl

```
@Service
public class UserManageServiceImpl implements UserManageService {

    @Autowired
    UserInfoMapper userInfoMapper;

    //查询所有
    public List<UserInfo> getUserInfoList(UserInfo userInfoQuery){
        List<UserInfo> userInfos=null;
        //查询所有
        //userInfos = userInfoMapper.selectAll();
        //条件匹配查询
        //userInfos =userInfoMapper.select(userInfoQuery);
        //特殊条件匹配查询 比如：按姓氏匹配
          Example example=new Example(UserInfo.class);

example.createCriteria().andLike("loginName","%"+userInfoQuery.getLoginName()+"%");
          userInfos = userInfoMapper.selectByExample(example);
        return userInfos;
    }

    //查询单表
    public UserInfo getUserInfo(UserInfo userInfoQuery){
        UserInfo userInfo=null;
        //按主键查找
        // userInfo = userInfoMapper.selectByPrimaryKey(userInfoQuery.getId());

        //按所有非空值查询   必须只有一行 否则报错
         userInfo = userInfoMapper.selectOne(userInfoQuery );
          return userInfo;
    }
```

```
    //增加用户
    public void addUserInfo(UserInfo userInfo){
        //会覆盖数据默认值
        userInfoMapper.insert(userInfo);
       // 不会覆盖数据库默认值
        userInfoMapper.insertSelective(userInfo);
    }


    public void updateUserInfo(UserInfo userInfo){
        //修改用户  依靠主键去查询 ， 然后更新其他值，如果某个值为空，那么原值被清空
//        userInfoMapper.updateByPrimaryKey(userInfo);
        //修改用户  依靠主键去查询 ， 然后更新其他不为空的值.
//         userInfoMapper.updateByPrimaryKeySelective(userInfo);

        //修改用户  依靠自定义条件去修改
        Example example=new Example(UserInfo.class);

example.createCriteria().andLike("loginName","%"+userInfo.getLoginName()+"%");
        userInfo.setLoginName(null);
//        userInfoMapper.updateByExample( userInfo,example );
        userInfoMapper.updateByExampleSelective( userInfo,example );
        //
    }

    public void delete(UserInfo userInfoQuery){
        userInfoMapper.deleteByPrimaryKey(userInfoQuery.getId());
        //按非空值匹配删除
      //  userInfoMapper.delete(userInfoQuery);
        //按条件匹配删除
       //  userInfoMapper.deleteByExample(new Example(UserInfo.class));
    }
```

4.6 Controller

```
@RestController
public class UserManageController {

    @Autowired
    UserManageService userManageService;

    @RequestMapping("/users")
    public ResponseEntity<List<UserInfo>> getUserList( UserInfo userInfo){
        List<UserInfo> userInfoList =
userManageService.getUserInfoList(userInfo);
        return ResponseEntity.ok().body(userInfoList);
    }


    @RequestMapping(value = "/user" ,method = RequestMethod.POST)
    public    ResponseEntity<Void> add(UserInfo userInfo){ ;

        userManageService.addUserInfo(userInfo);
        return ResponseEntity.ok().build();
    }

    @RequestMapping(value = "/user" ,method = RequestMethod.PUT)
    public    ResponseEntity<Void> update(UserInfo userInfo){
        userManageService.updateUserInfo(userInfo);
        return ResponseEntity.ok().build();
    }

    @RequestMapping(value = "/user" ,method = RequestMethod.DELETE)
    public    ResponseEntity<Void> delete(UserInfo userInfo){
        userManageService.delete(userInfo);
        return ResponseEntity.ok().build();
    }

    @RequestMapping(value = "/user" ,method = RequestMethod.GET)
    public    ResponseEntity<UserInfo> getUserInfo(UserInfo userInfoQuery){
        UserInfo userInfo = userManageService.getUserInfo(userInfoQuery);
        return ResponseEntity.ok().body(userInfo);
    }
```

五　hosts 工具

application.properties

```
spring.datasource.url=jdbc:mysql://mysql.server.com:3306/gmall?characterEncoding=UTF-8
spring.datasource.username=root
spring.datasource.password=123123
```

谷粒商城

版本：V 1.0

www.atguigu.com

一、分布式架构

1 分布式架构的演进

1.1 单一应用架构

适用于小型网站，小型管理系统，将所有功能都部署到一个功能里，简单易用。

缺点： 1、性能扩展比较难

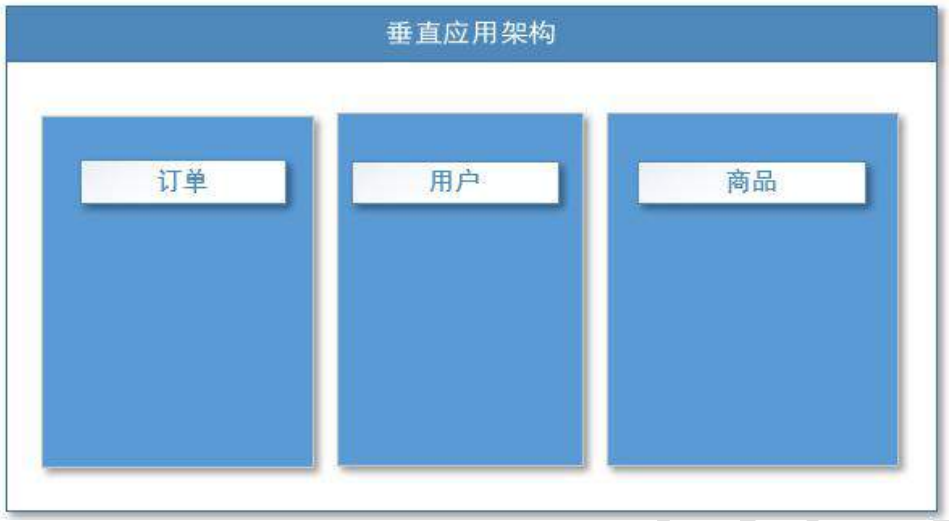
2、协同开发问题

3、不利于升级维护

1.2 垂直应用架构

通过切分业务来实现各个模块独立部署，降低了维护和部署的难度，团队各司其职更易管理，性能扩展也更方便，更有针对性。

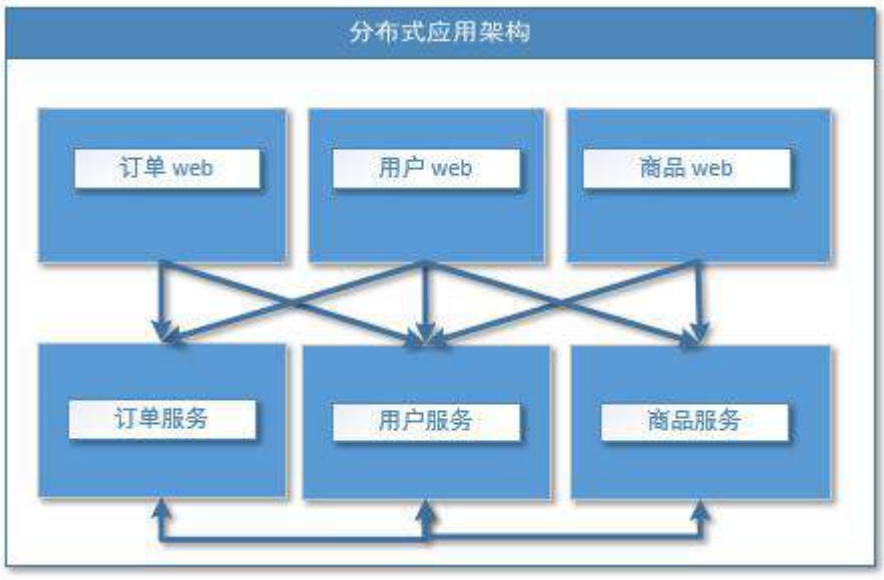
缺点： 公用模块无法重复利用，开发性的浪费



1.3 分布式应用架构

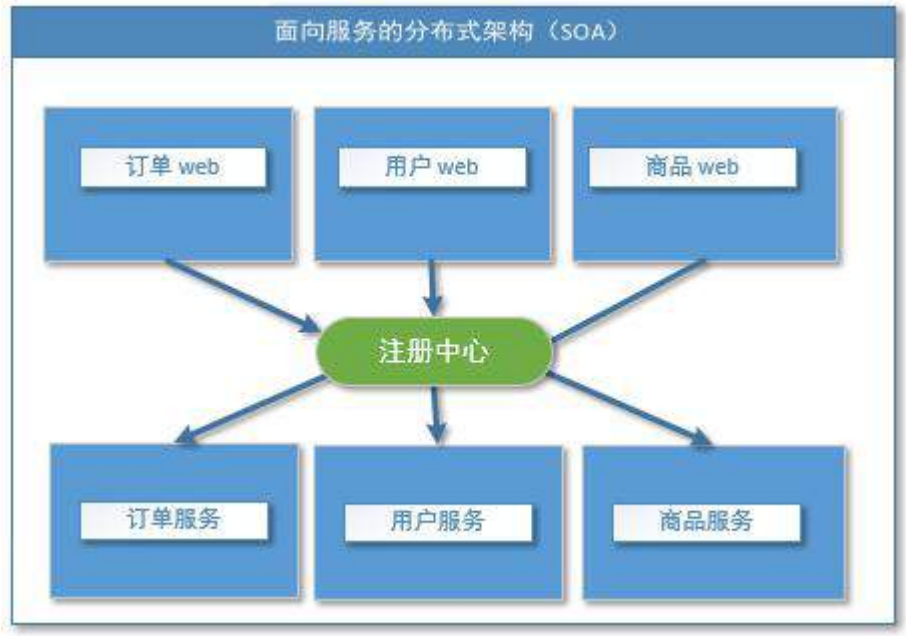
将各个应用通过分层独立出来，可以利用 rpc 实现 web 与 service、service 与 service 的互相调用，提高了代码的复用性。

缺点： 每个调用的模块要存储一份完整的被调用模块的位置和状态，一旦位置和状态发生变化，就要更新所有涉及的配置。



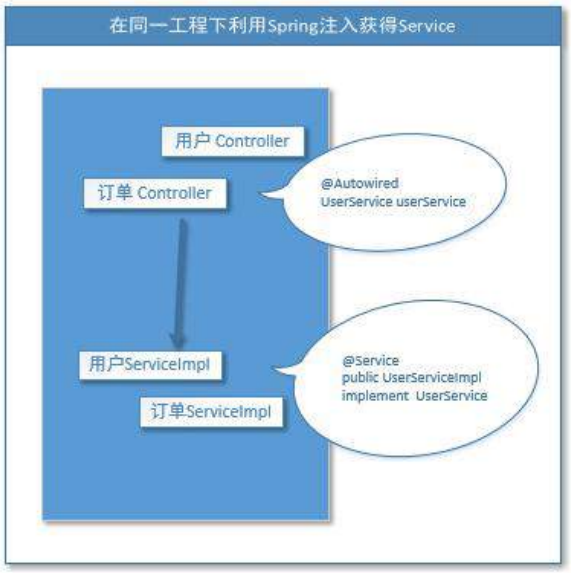
1.4 面向服务的分布式架构

随着架构不断增大，服务节点也越来越多，服务之间的调用和依赖关系也越来越复杂，需要有一个统一的中心来调度、路由、管理所有的服务，基于这个中心构建的这个星型架构就是现在目前最主流的 SOA 分布式架构。

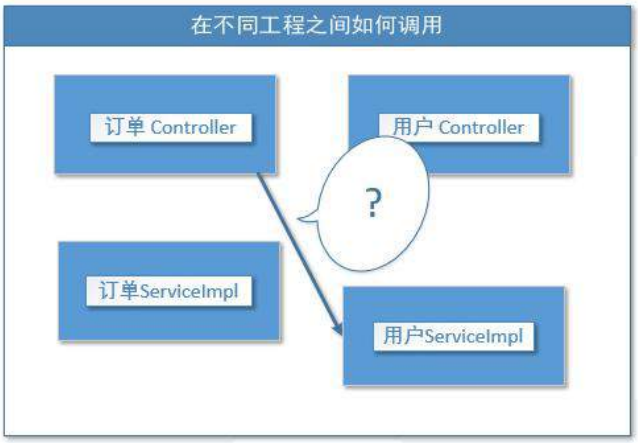


2 如何实现这种 SOA 架构

原来所有的 controller、service 接口、service 实现都在一个工程，通过 Spring 的 ioc 就可以实现互相调用。



那么假如 controller 和 service 实现隶属于不同的应用如何实现调用呢？



3　实现订单的 Web 应用（Controller）调用用户的 Service 应用的用户地址信息功能

3.1 用户地址信息查询

需要开发的类

包	类	说明
service	UserManageService	接口 增加方法

service.impl	UserManageServiceImpl	实现类　增加方法
bean	UserAddress	实体 bean
mapper	UserAddressMapper	mapper 接口

bean

```
public class UserAddress implements Serializable{
    @Column
    @Id
    private String id;
    @Column
    private String userAddress;
    @Column
    private String userId;
    @Column
    private String consignee;
    @Column
    private String phoneNum;
    @Column
    private String isDefault;
}
```

mapper

```
public interface UserAddressMapper extends Mapper<UserAddress> {
}
```

UserManageService 增加方法

```
public List<UserAddress> getUserAddressList(String userId);
```

UserManageServiceImpl 中增加方法

```
public List<UserAddress> getUserAddressList(String userId) {
        List<UserAddress> addressList = null;
        UserAddress userAddress = new UserAddress();
        userAddress.setUserId(userId);
        addressList = userAddressMapper.select(userAddress);
        return addressList;
}
```

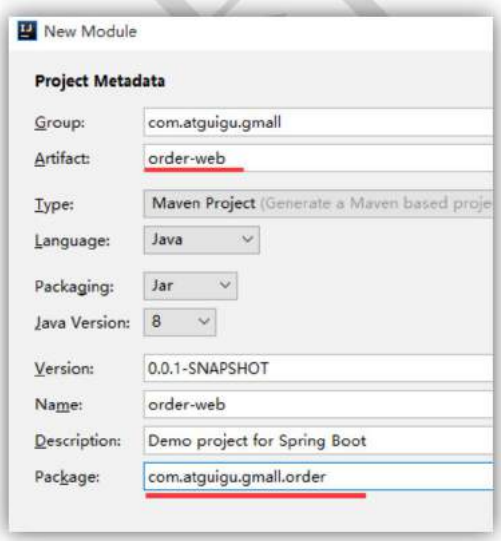
利用测试类 GmallUserManageApplication 测试（选用）

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class GmallUserManageApplicationTests {

    @Autowired
```

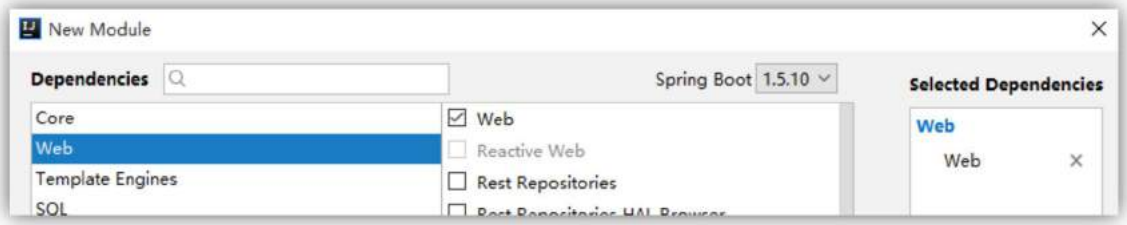
```
    UserManageService userManageService;

    @Test
    public void showAddressList() {
        List<UserAddress> userAddressList = userManageService.getUserAddressList("1");
        for (UserAddress userAddress : userAddressList) {
            System.err.println("userAddress = " + userAddress);
        }

    }
```

二、分布式工程的模块搭建

1 搭建订单的 Web 模块工程



只勾 web 模块就可以了



需要开发的类

包	类	说明
controller	OrderController	web controller

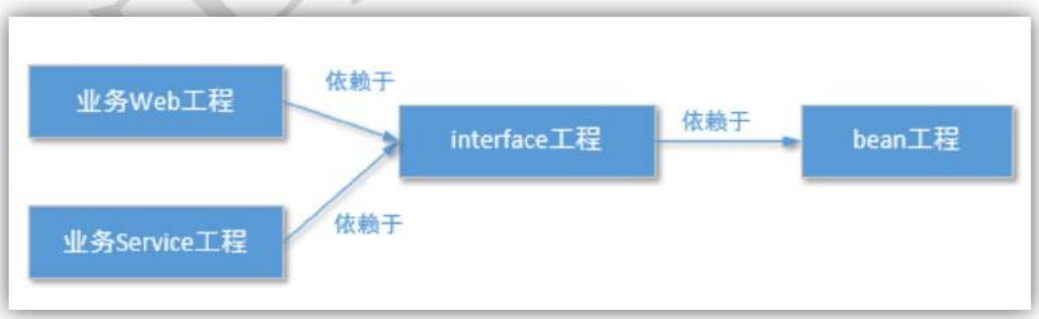
由于需要让订单的 web 应用可以调用用户的 Service 接口，那么必须在订单的工程中也要包含一份 Service 接口。

如果拷贝一个接口到订单工程中，那么如果以后有更多的模块都调用这个接口呢？每个都拷贝一份接口类么？

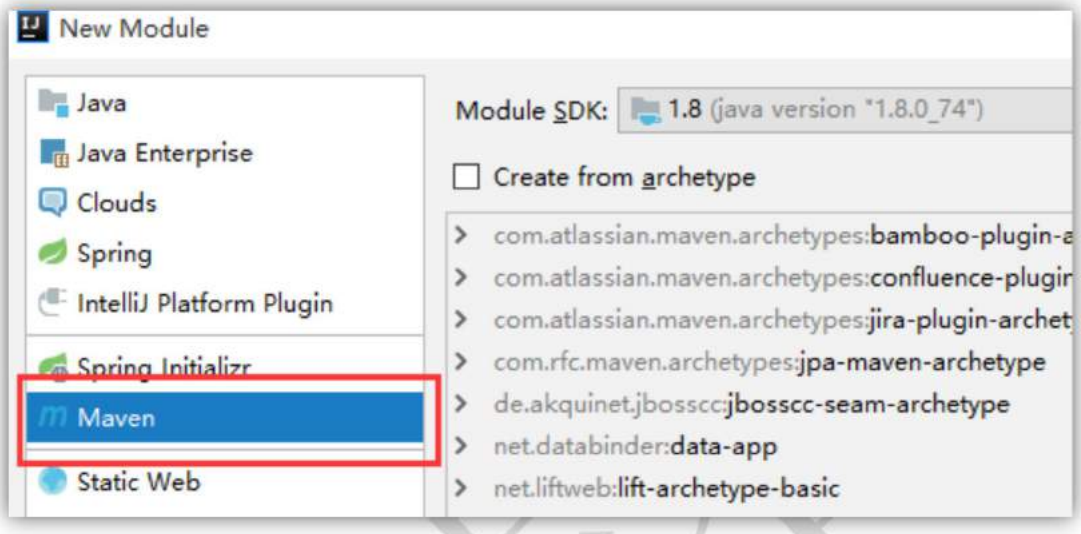
这种情况我们就可以利用 maven 的依赖把这些接口作为公共的包管理起来。

同时接口类种的方法也引用了很多的实体 bean，那么同样的实体 bean 的类我们也统一管理起来。

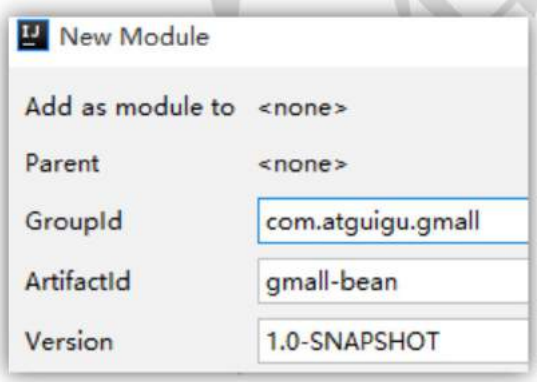
这样我们就有了如下的依赖关系：

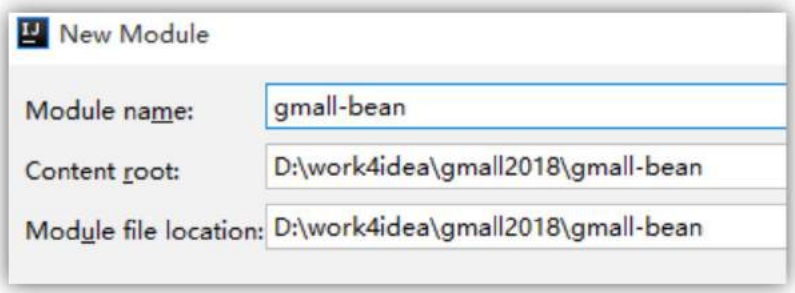


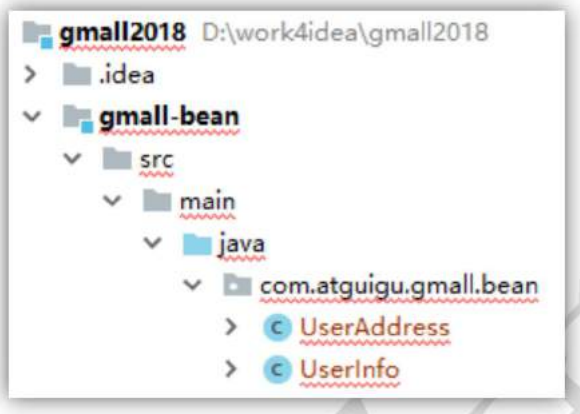
2 创建 bean 模块



同时我们把 UserManage 中的 bean 剪切到 bean 模块中







bean 模块报错是因为其中引用了通用 mapper，所以我们把通用 mapper 的依赖提取出来放到 bean 模块后面，变成如下结构。

bean 模块的 pom.xml

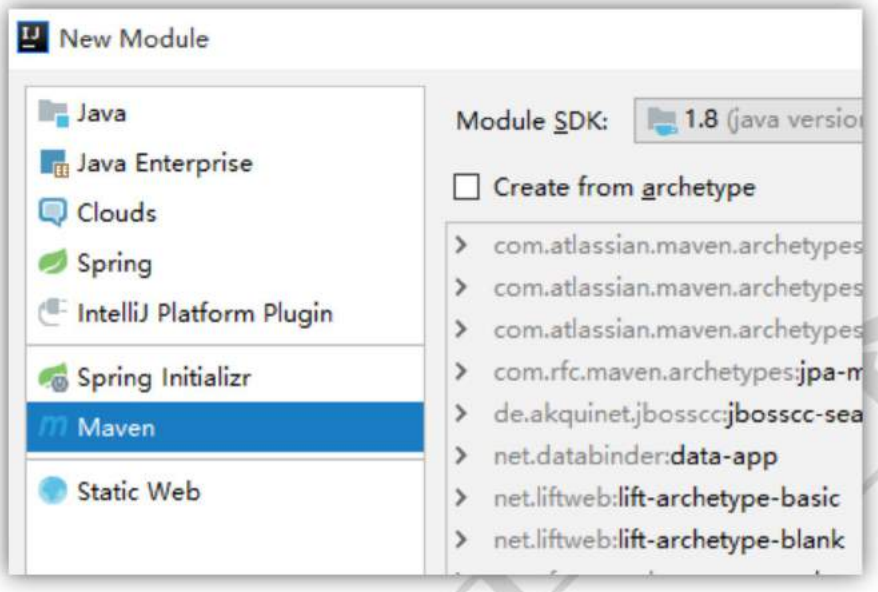
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
```

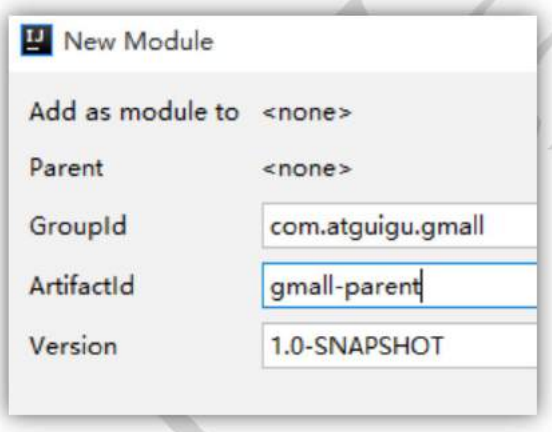
```
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.atguigu.gmall</groupId>
    <artifactId>gmall-bean</artifactId>
    <version>1.0-SNAPSHOT</version>

    <parent>
        <artifactId>gmall-parent</artifactId>
        <groupId>com.atguigu.gmall</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <dependencies>
        <dependency>
            <groupId>tk.mybatis</groupId>
            <artifactId>mapper-spring-boot-starter</artifactId>
            <version>1.2.3</version>
            <exclusions>
                <exclusion>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-starter-jdbc</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
    </dependencies>
</project>
```

由于依赖包分布于多个模块中，最好有一个地方能够把所有依赖的版本通用管理起来。

这就用到了 maven 的<parent>概念。可以让所有的模块都继承这个 parent 模块，由这个 parent 模块来管理版本。

3 搭建 parent 模块





parent 模块的 pom.xml

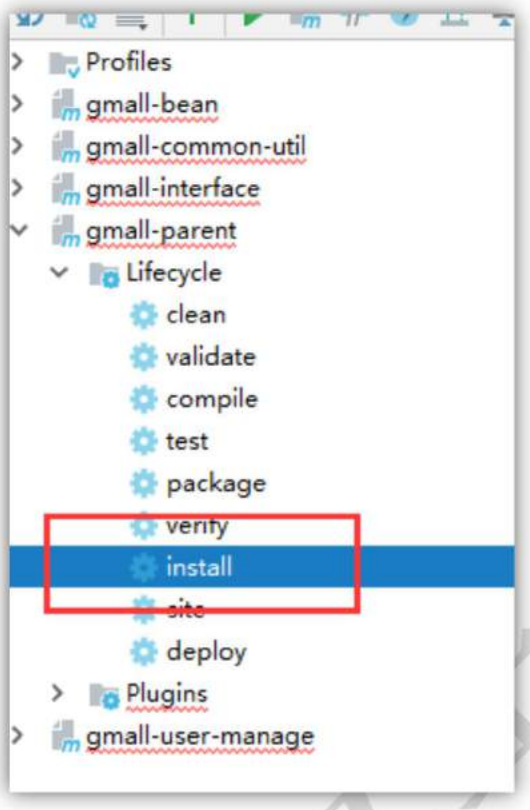
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.atguigu.gmall</groupId>
    <artifactId>gmall-parent</artifactId>
    <version>1.0-SNAPSHOT</version>
```

```
    <packaging>pom</packaging>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
        <java.version>1.8</java.version>

        <fastjson.version>1.2.46</fastjson.version>
        <dubbo-starter.version>1.0.10</dubbo-starter.version>
        <dubbo.version>2.6.0</dubbo.version>
        <zkclient.version>0.10</zkclient.version>
        <mybatis.version>1.3.1</mybatis.version>
        <nekohtml.version>1.9.20</nekohtml.version>
        <xml-apis.version>1.4.01</xml-apis.version>
        <batik-ext.version>1.9.1</batik-ext.version>
        <jsoup.version>1.11.2</jsoup.version>
        <httpclient.version>4.5.5</httpclient.version>
        <commons-lang3.version>3.7</commons-lang3.version>
        <mapper-starter.version>1.2.3</mapper-starter.version>
        <jedis.version>2.9.0</jedis.version>
        <jest.version>5.3.3</jest.version>
        <jna.version>4.5.1</jna.version>
        <beanUtils.version>1.9.3</beanUtils.version>
    </properties>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.5.10.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>com.alibaba</groupId>
                <artifactId>fastjson</artifactId>
                <version>${fastjson.version}</version>
            </dependency>

            <dependency>
                <groupId>com.alibaba</groupId>
                <artifactId>dubbo</artifactId>
                <version>${dubbo.version}</version>
            </dependency>

            <dependency>
                <groupId>com.101tec</groupId>
                <artifactId>zkclient</artifactId>
                <version>${zkclient.version}</version>
            </dependency>

            <dependency>
                <groupId>com.gitee.reger</groupId>
                <artifactId>spring-boot-starter-dubbo</artifactId>
```

```
        <version>${dubbo-starter.version}</version>
    </dependency>

    <dependency>
        <groupId>org.mybatis.spring.boot</groupId>
        <artifactId>mybatis-spring-boot-starter</artifactId>
        <version>${mybatis.version}</version>
    </dependency>


    <dependency>
        <groupId>net.sourceforge.nekohtml</groupId>
        <artifactId>nekohtml</artifactId>
        <version>${nekohtml.version}</version>
    </dependency>

    <dependency>
        <groupId>xml-apis</groupId>
        <artifactId>xml-apis</artifactId>
        <version>${xml-apis.version}</version>
    </dependency>

    <dependency>
        <groupId>org.apache.xmlgraphics</groupId>
        <artifactId>batik-ext</artifactId>
        <version>${batik-ext.version}</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.jsoup/jsoup -->
    <dependency>
        <groupId>org.jsoup</groupId>
        <artifactId>jsoup</artifactId>
        <version>${jsoup.version}</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.apache.httpcomponents/httpclient -->
    <dependency>
        <groupId>org.apache.httpcomponents</groupId>
        <artifactId>httpclient</artifactId>
        <version>${httpclient.version}</version>
    </dependency>

    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-lang3</artifactId>
        <version>${commons-lang3.version}</version>
    </dependency>


    <dependency>
        <groupId>tk.mybatis</groupId>
        <artifactId>mapper-spring-boot-starter</artifactId>
        <version>${mapper-starter.version}</version>
```

```
            </dependency>

            <dependency>
                <groupId>redis.clients</groupId>
                <artifactId>jedis</artifactId>
                <version>${jedis.version}</version>
            </dependency>

            <!-- https://mvnrepository.com/artifact/io.searchbox/jest -->
            <dependency>
                <groupId>io.searchbox</groupId>
                <artifactId>jest</artifactId>
                <version>${jest.version}</version>
            </dependency>

            <!-- https://mvnrepository.com/artifact/net.java.dev.jna/jna -->
            <dependency>
                <groupId>net.java.dev.jna</groupId>
                <artifactId>jna</artifactId>
                <version>${jna.version}</version>
            </dependency>

            <dependency>
                <groupId>commons-beanutils</groupId>
                <artifactId>commons-beanutils</artifactId>
                <version>${beanUtils.version}</version>
            </dependency>


        </dependencies>
    </dependencyManagement>
</project>
```

然后在 idea 右边菜单执行安装



那么除了通用 mapper 以外其他的第三方依赖我们如何放置

4 搭建 util 模块

首先我们可以把所有的第三方依赖包分为四种

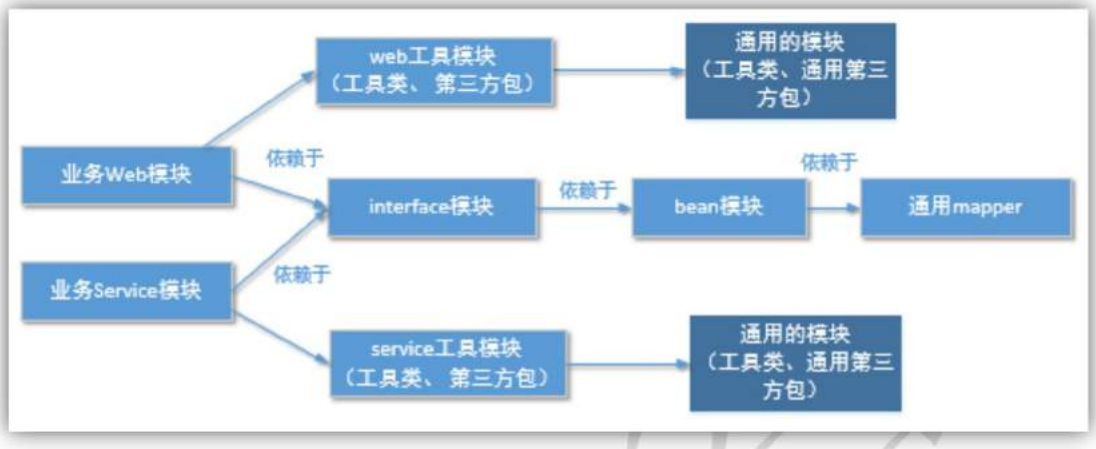
1、 web 业务模块用到的第三方包,比如文件上传客户端、页面渲染工具、操作 cookie 的工具类等等。

2、 service 业务模块用到的第三方包，比如 jdbc、mybatis、jedis、activemq 工具包等等。

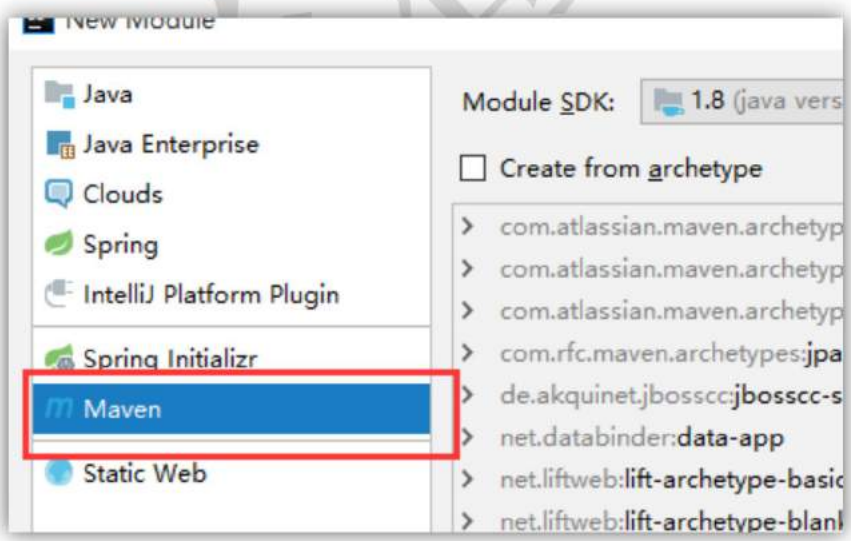
3、 通用型的第三方包，比如 fastjson、httpclient、apache 工具包等等。

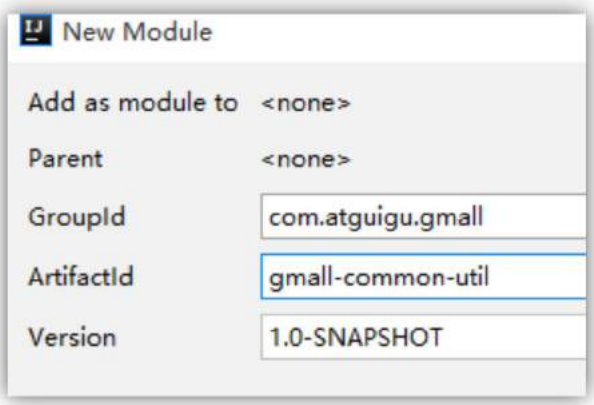
4、 只有本模块用到的 es

基于这四种情况我们可以搭建如下的依赖结构：



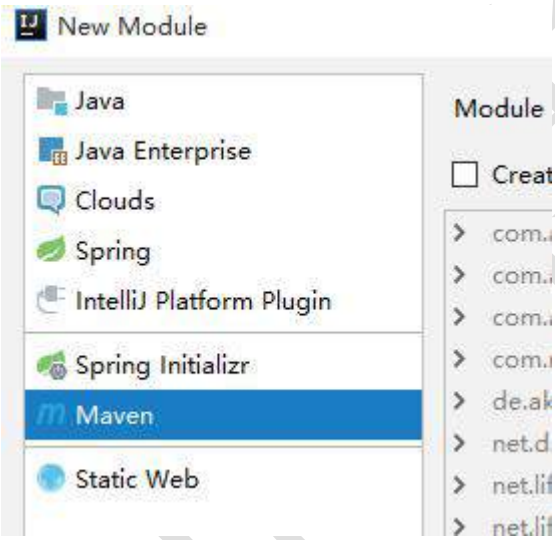
创建 common-util 的模块

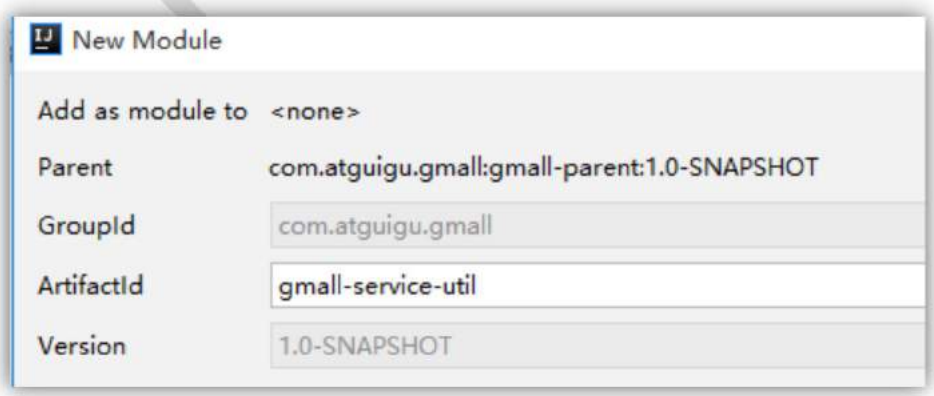




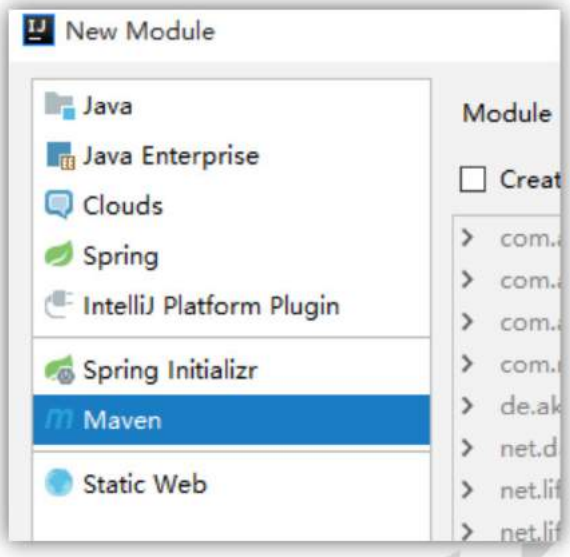
创建 gmall-web-util 和 gmall-service-util

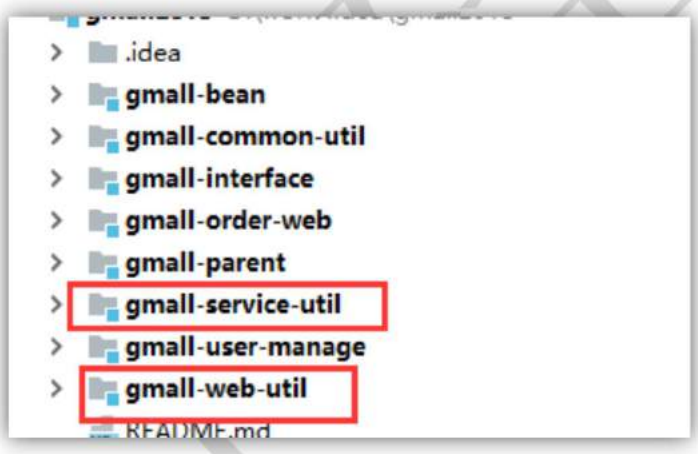
创建 service-util 模块





创建 web-util 模块





pom.xml 文件

首先先分析具体哪些包是通用的

gmall-common-util

spring-boot-starter-test	测试(springboot 有默认版本号)
spring-boot-starter-web	内含 tomcat 容器、HttpSevrletRequest 等 (springboot 有默认版本号)
fastjson	json 工具
commons-lang3	方便好用的 apache 工具库
commons-beanutils	方便好用的 apache 处理实体 bean 工具库
commons-codec	方便好用的 apache 解码工具库
httpclient	restful 调用客户端

gmall-web-util

thymeleaf	springboot 自带页面渲染工具(springboot 有默认版本号)

gmall-service-util

spring-boot-starter-jdbc	数据库驱动(springboot 有默认版本号)
mysql-connector-java	数据库连接器(springboot 有默认版本号)
mybatis-spring-boot-starter	mybatis

gmall-common-util 的 pom.xml

```
<?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.atguigu.gmall</groupId>
    <artifactId>gmall-common-util</artifactId>
```

```
    <version>1.0-SNAPSHOT</version>
    <parent>
        <groupId>com.atguigu.gmall</groupId>
        <artifactId>gmall-parent</artifactId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
        </dependency>

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



        <dependency>
            <groupId>org.apache.httpcomponents</groupId>
            <artifactId>httpclient</artifactId>
        </dependency>

        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-lang3</artifactId>
        </dependency>

        <dependency>
            <groupId>commons-beanutils</groupId>
            <artifactId>commons-beanutils</artifactId>
        </dependency>

        <dependency>
            <groupId>commons-codec</groupId>
            <artifactId>commons-codec</artifactId>
        </dependency>

    </dependencies>
</project>
```

gmall-web-util 的 pom.xml

```
<?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>gmall-parent</artifactId>
        <groupId>com.atguigu.gmall</groupId>
        <version>1.0-SNAPSHOT</version>
```

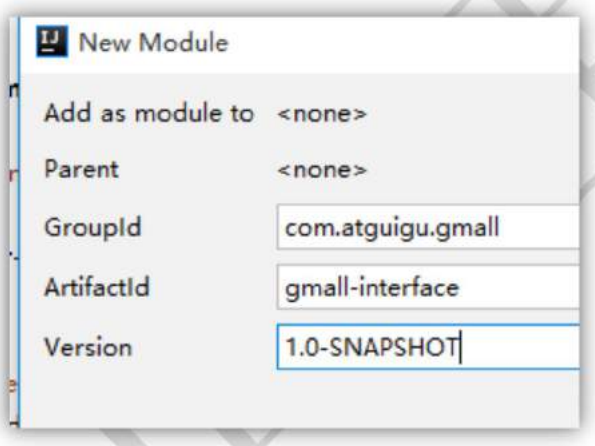
```
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>gmall-web-util</artifactId>
    <groupId>com.atguigu.gmall</groupId>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <dependency>
            <groupId>com.atguigu.gmall</groupId>
            <artifactId>gmall-common-util</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>

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

    </dependencies>

</project>
```

gmall-service-util 的 pom.xml

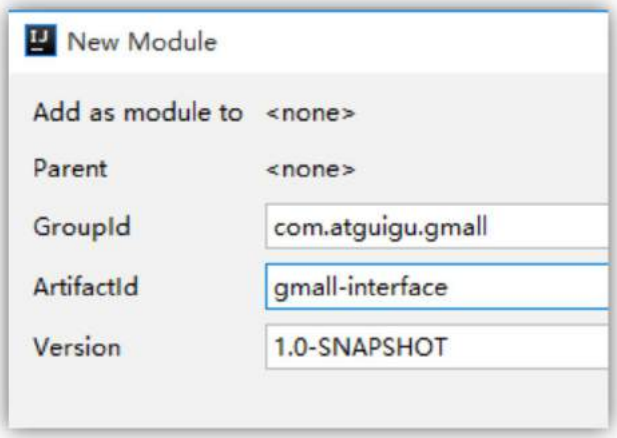
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>gmall-parent</artifactId>
        <groupId>com.atguigu.gmall</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <artifactId>gmall-service-util</artifactId>
    <dependencies>
        <dependency>
            <groupId>com.atguigu.gmall</groupId>
            <artifactId>gmall-common-util</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-jdbc</artifactId>
        </dependency>
        <dependency>
            <groupId>org.mybatis.spring.boot</groupId>
            <artifactId>mybatis-spring-boot-starter</artifactId>
        </dependency>

        <dependency>
            <groupId>mysql</groupId>
```

```
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>

        <dependency>
            <groupId>redis.clients</groupId>
            <artifactId>jedis</artifactId>
        </dependency>

    </dependencies>
</project>
```

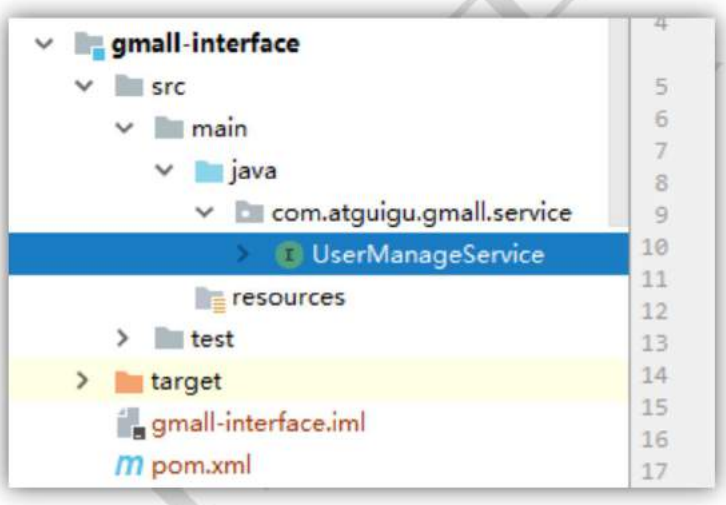
创建 interface 模块



5 搭建 interface 模块



把 UserManageService 接口移动到该模块下

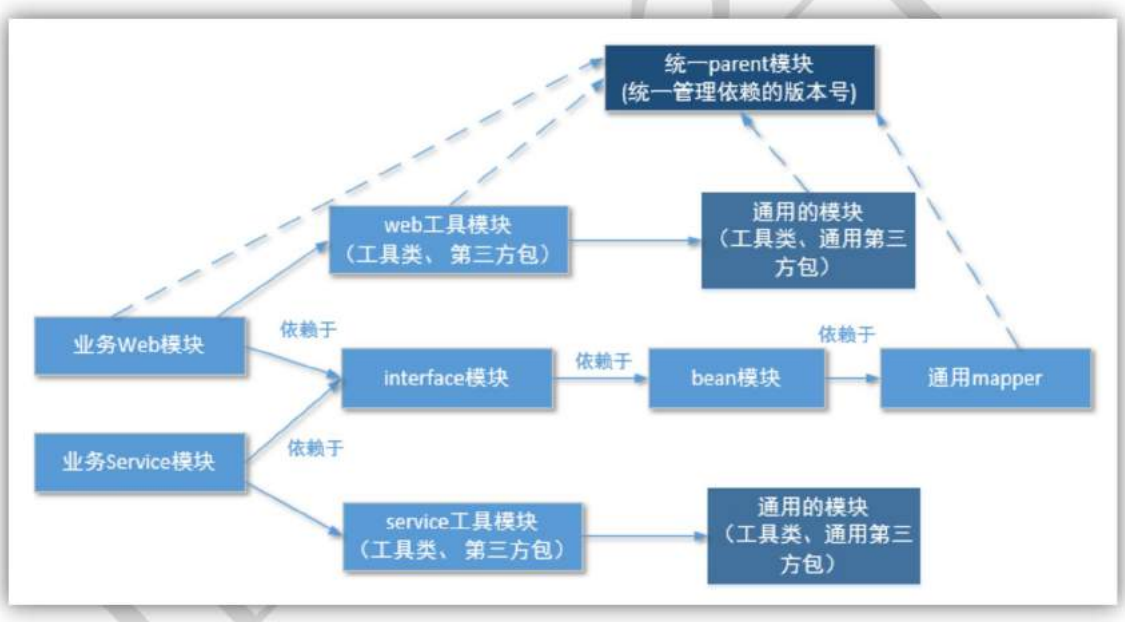


interface 的 pom.xml

```
<?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">

    <groupId>com.atguigu.gmall</groupId>
    <artifactId>gmall-interface</artifactId>
    <version>1.0-SNAPSHOT</version>
    <modelVersion>4.0.0</modelVersion>
```

```
    <packaging>jar</packaging>

    <dependencies>
        <dependency>
            <groupId>com.atguigu.gmall</groupId>
            <artifactId>gmall-bean</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
    </dependencies>

</project>
```

最终的结构图



6 gmall-User-manage 模块

pom.xml

```
<?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.atguigu.gmall</groupId>
    <artifactId>gmall-user-manage</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>gmall-user-manage</name>
    <description>Demo project for Spring Boot</description>

    <parent>
        <groupId>com.atguigu.gmall</groupId>
        <artifactId>gmall-parent</artifactId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <dependencies>

        <dependency>
            <groupId>com.atguigu.gmall</groupId>
            <artifactId>gmall-interface</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>

        <dependency>
            <groupId>com.atguigu.gmall</groupId>
            <artifactId>gmall-service-util</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

其他的类，要重新引一下包

1 同时要修改 bean 的引入

2 同时要修改@Service 和@Autowrited 注解

3 将接口和 bean 转移到 bean 和 interface 项目中

4 原来 user-manage 中的 mapper，service 接口，和实现类中的引用 bean 类的位置需要修改

```
import com.atguigu.gmall.usermanage.bean.UserInfo;
import com.atguigu.gmall.bean.UserInfo;
```

5 重新安装 maven

7 继续开发 gmall-order-web 模块

order-web 的 pom.xml

```
<?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.atguigu.gmall</groupId>
    <artifactId>gmall-order-web</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>gmall-order-web</name>
    <description>Demo project for Spring Boot</description>

    <parent>
        <groupId>com.atguigu.gmall</groupId>
        <artifactId>gmall-parent</artifactId>
        <version>1.0-SNAPSHOT</version>
    </parent>


    <dependencies>

    <dependency>
        <groupId>com.atguigu.gmall</groupId>
        <artifactId>gmall-interface</artifactId>
        <version>1.0-SNAPSHOT</version>
    </dependency>

    <dependency>
        <groupId>com.atguigu.gmall</groupId>
        <artifactId>gmall-web-util</artifactId>
```

```
        <version>1.0-SNAPSHOT</version>
    </dependency>


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


</project>
```

修改 gmall-usermanage　pom.xml

```
<?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.atguigu.gmall1108</groupId>
    <artifactId>gmall-usermanage</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>gmall-usermanage</name>
    <description>Demo project for Spring Boot</description>

    <parent>
        <groupId>com.atguigu.gmall1108</groupId>
        <artifactId>gmall-parent</artifactId>
        <version>1.0-SNAPSHOT</version>
    </parent>

    <dependencies>

        <dependency>
            <groupId>com.atguigu.gmall1108</groupId>
            <artifactId>gmall-interface</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>com.atguigu.gmall1108</groupId>
            <artifactId>gmall-service-util</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
```

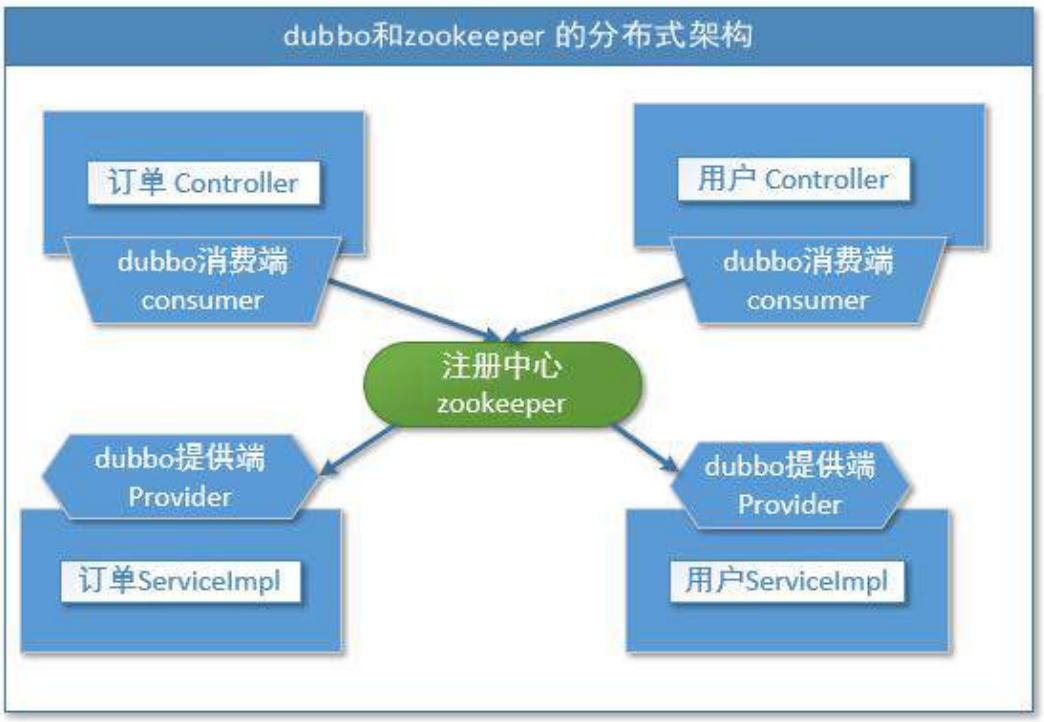
```
    </dependencies>

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


</project>
```

OrderController

```
@Controller
public class OrderController {

    UserManageService userManageService;

    @ResponseBody
    @RequestMapping(value = "initOrder")
    public String initOrder(HttpServletRequest request){
        String userId = request.getParameter("userId");
        List<UserAddress> userAddressList = userManageService.getUserAddressList(userId);
        String jsonString = JSON.toJSONString(userAddressList);
        return   jsonString;

    }
}
```

这样虽然引入的包，可以认出 UserManageService，但是这个接口没有被注入。原来利用 Spring 可以注入，但是现在实现类不在同一个模块如何注入？

三、Dubbo 和 zookeeper



那 dubbo 和 zookeeper 如何引入？

dubbo 其实是一组 jar 包，通过 maven 引入就可以。

zookeeper 是一个开源的服务软件，需要安装到 linux 中。

1 安装 zookeeper

1.1 安装环境：

linux 版本: CentOS 6.8

zookeeper 版本　　zookeeper-3.4.11.tar.gz

拷贝 zookeeper-3.4.11.tar.gz 到/opt 下，并解压缩

```
总用量 221156
drwxr-xr-x.  8   10   143       255 9月   14 17:27 jdk1.8.0_152
-rw-r--r--.  1 root root 189784266 2月   17 13:11 jdk-8u152-linux-x64.tar.gz
drwxr-xr-x. 10  502 games      4096 11月  2 02:52 zookeeper
-rw-r--r--.  1 root root  36668066 2月   16 19:33 zookeeper-3.4.11.tar.gz
[root@localhost opt]#
```

改名叫 zookeeper

```
oot@localhost local]# mv zookeeper-3.4.11 zookeeper
oot@localhost local]# ll
用量 35816
```

1.2 制作开机启动的脚本

```
[root@localhost /]# vim /etc/init.d/zookeeper
```

把如下脚本复制进去

```
#!/bin/bash
#chkconfig:2345 20 90
#description:zookeeper
#processname:zookeeper
ZK_PATH=/opt/zookeeper
export JAVA_HOME=/opt/jdk1.8.0_152
case $1 in
        start) sh   $ZK_PATH/bin/zkServer.sh start;;
        stop)   sh   $ZK_PATH/bin/zkServer.sh stop;;
        status) sh   $ZK_PATH/bin/zkServer.sh status;;
        restart) sh $ZK_PATH/bin/zkServer.sh restart;;
        *)   echo "require start|stop|status|restart"   ;;
esac
```

```
#!/bin/bash
#chkconfig:2345 20 90
#description:zookeeper
#processname:zookeeper
ZK_PATH=/opt/zookeeper
export JAVA_HOME=/opt/jdk1.8.0_152
case $1 in
        start) sh   $ZK_PATH/bin/zkServer.sh start;;
        stop)   sh   $ZK_PATH/bin/zkServer.sh stop;;
        status) sh   $ZK_PATH/bin/zkServer.sh status;;
        restart) sh $ZK_PATH/bin/zkServer.sh restart;;
        *)   echo "require start|stop|status|restart"   ;;
esac
```

然后把脚本注册为 Service

```
[root@localhost /]# chkconfig --add zookeeper
[root@localhost /]# chkconfig --list

注：该输出结果只显示 SysV 服务，并不包含
原生 systemd 服务。SysV 配置数据
可能被原生 systemd 配置覆盖。

      要列出 systemd 服务，请执行 'systemctl list-unit-files'。
      查看在具体 target 启用的服务请执行
      'systemctl list-dependencies [target]'。

netconsole      0:关    1:关    2:关    3:关    4:关    5:关    6:关
network         0:关    1:关    2:开    3:开    4:开    5:开    6:关
zookeeper       0:关    1:关    2:开    3:开    4:开    5:开    6:关
[root@localhost /]#
```

增加权限

```
[root@localhost /]# chmod +x /etc/init.d/zookeeper
[root@localhost /]# ll /etc/init.d/
总用量 44
-rw-r--r--. 1 root root 17500 5月    3 2017 functions
-rwxr-xr-x. 1 root root  4334 5月    3 2017 netconsole
-rwxr-xr-x. 1 root root  7293 5月    3 2017 network
-rw-r--r--. 1 root root  1160 8月    5 2017 README
-rwxr-xr-x. 1 root root   428 2月   17 15:48 zookeeper
[root@localhost /]#
```

1.3 初始化 zookeeper 配置文件

拷贝/opt/zookeeper/conf/zoo_sample.cfg

到同一个目录下改个名字叫 zoo.cfg

```
[root@localhost conf]# ll
总用量 16
-rw-r--r--. 1  502 games  535 11月  2 02:47 configuration.xsl
-rw-r--r--. 1  502 games 2161 11月  2 02:47 log4j.properties
-rw-r--r--. 1 root root   922 2月   17 15:56 zoo.cfg
-rw-r--r--. 1  502 games  922 11月  2 02:47 zoo_sample.cfg
```

然后咱们启动 zookeeper

```
[root@localhost conf]# service zookeeper start
ZooKeeper JMX enabled by default
Using config: /opt/zookeeper/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
[root@localhost conf]#
```

以上状态即为安装成功。

2 dubbo 的使用

dubbo 本身并不是一个服务软件。它其实就是一个 jar 包能够帮你的 java 程序连接到 zookeeper，并利用 zookeeper 消费、提供服务。所以你不用在 Linux 上启动什么 dubbo 服务。但是为了让用户更好的管理监控众多的 dubbo 服务，官方提供了一个可视化的监控程序，不过这个监控即使不装也不影响使用。

2.1 安装监控软件：

材料： tomcat8 + dubbo-admin

拷贝 tomcat8 和 dubbo-admin 到/opt 目录下

```
总用量 261764
-rw-r--r--.  1 root root    9487006 2月  17 17:29 apache-tomcat-8.5.24.tar.gz
-rw-r--r--.  1 root root   32089280 2月  17 17:32 dubbo-admin-2.6.0.war
drwxr-xr-x.  8   10   143       255 9月  14 17:27 jdk1.8.0_152
-rw-r--r--.  1 root root  189784266 2月  17 13:11 jdk-8u152-linux-x64.tar.gz
drwxr-xr-x.  9 root root        160 2月  17 17:30 tomcat4dubbo
drwxr-xr-x. 10  502 games      4096 11月  2 02:52 zookeeper
-rw-r--r--.  1 root root   36668066 2月  16 19:33 zookeeper-3.4.11.tar.gz
```

然后把 dubbo-admin-2.6.0.war 拷贝到 tomcat 的 webapps 目录下

```
[root@localhost tomcat4dubbo]# cp /opt/dubbo-admin-2.6.0.war /opt/tomcat4dubbo/webapps/dubbo.war
[root@localhost tomcat4dubbo]#
```

2.2 设置开机启动 tomcat

```
[root@localhost tomcat4dubbo]# vim /etc/init.d/dubbo-admin
```

复制如下脚本

```
#!/bin/bash
#chkconfig:2345 21 90
#description:dubbo-admin
#processname:dubbo-admin
```

```
CATALANA_HOME=/opt/tomcat4dubbo
export JAVA_HOME=/opt/jdk1.8.0_152
case $1 in
start)
        echo "Starting Tomcat..."
        $CATALANA_HOME/bin/startup.sh
        ;;

stop)
        echo "Stopping Tomcat..."
        $CATALANA_HOME/bin/shutdown.sh
        ;;

restart)
        echo "Stopping Tomcat..."
        $CATALANA_HOME/bin/shutdown.sh
        sleep 2
        echo
        echo "Starting Tomcat..."
        $CATALANA_HOME/bin/startup.sh
        ;;
*)
        echo "Usage: tomcat {start|stop|restart}"
        ;; esac
```

然后同样的注册进入到服务中

```
[root@localhost tomcat4dubbo]# chkconfig --add dubbo-admin
```

加入权限

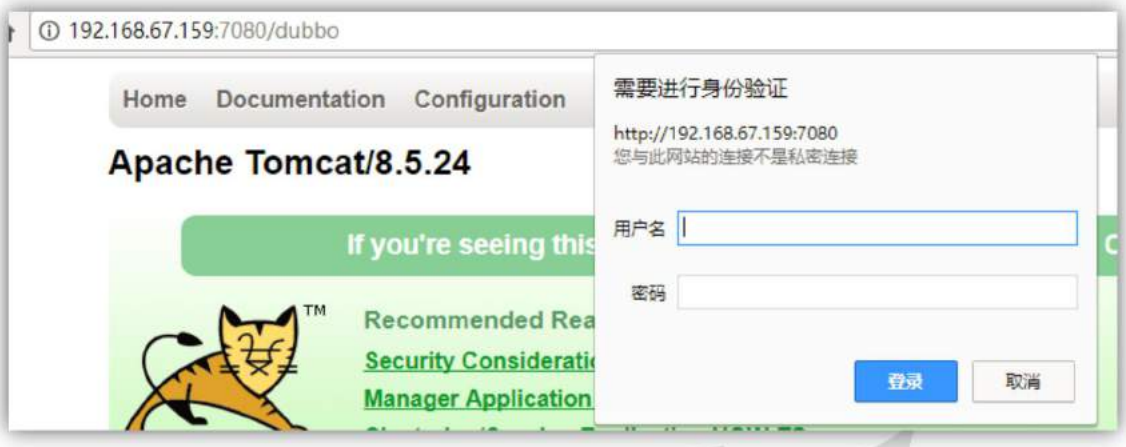
```
[root@localhost tomcat4dubbo]# chmod +x dubbo-admin
```

如果想改变端口号去 tomcat 中的 server.conf 中修改，课件中已改为 7080,然后就可以启动服务了。

2.3 启动服务

```
[root@localhost tomcat4dubbo]# service dubbo-admin start
```

启动后用浏览器访问



可以看到要提示用户名密码，默认是 root/root

(修改的话，可以去)

打开这个界面就说明，dubbo 的监控服务已经启动。但是现在咱们还没有搭建 dubbo 的提供端和消费端。

3 开发功能

3.1 引入 dubbo 的依赖

spring-boot-starter-dubbo

dubbo

zkclient

这个依赖首先要放到 gmall-parent 工程中，用来定义要引入的三个包是什么版本。

```
<?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.atguigu.gmall</groupId>
    <artifactId>gmall-parent</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>pom</packaging>
    <properties>
        <fastjson.version>1.2.46</fastjson.version>
        <mapper.version>3.4.6</mapper.version>
        <dubbo-starter.version>1.0.10</dubbo-starter.version>
        <dubbo.version>2.6.0</dubbo.version>
        <zkclient.version>0.10</zkclient.version>
    </properties>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>com.alibaba</groupId>
                <artifactId>fastjson</artifactId>
                <version>${fastjson.version}</version>
            </dependency>
            <dependency>
                <groupId>tk.mybatis</groupId>
                <artifactId>mapper</artifactId>
                <version>${mapper.version}</version>
            </dependency>

            <dependency>
                <groupId>com.alibaba</groupId>
                <artifactId>dubbo</artifactId>
                <version>${dubbo.version}</version>
            </dependency>
```

```
            <dependency>
                <groupId>com.101tec</groupId>
                <artifactId>zkclient</artifactId>
                <version>${zkclient.version}</version>
            </dependency>

            <dependency>
                <groupId>com.gitee.reger</groupId>
                <artifactId>spring-boot-starter-dubbo</artifactId>
                <version>${dubbo-starter.version}</version>
            </dependency>

        </dependencies>
    </dependencyManagement>
</project>
```

然后加入到 gmall-common-util 模块中

```
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>dubbo</artifactId>
</dependency>

<dependency>
    <groupId>com.101tec</groupId>
    <artifactId>zkclient</artifactId>
    <exclusions>
        <exclusion>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-log4j12</artifactId>
        </exclusion>
    </exclusions>
</dependency>

<dependency>
    <groupId>com.gitee.reger</groupId>
    <artifactId>spring-boot-starter-dubbo</artifactId>
</dependency>
```

这样在所有的业务模块中都可以使用 dubbo 了。

3.2 如何使用：

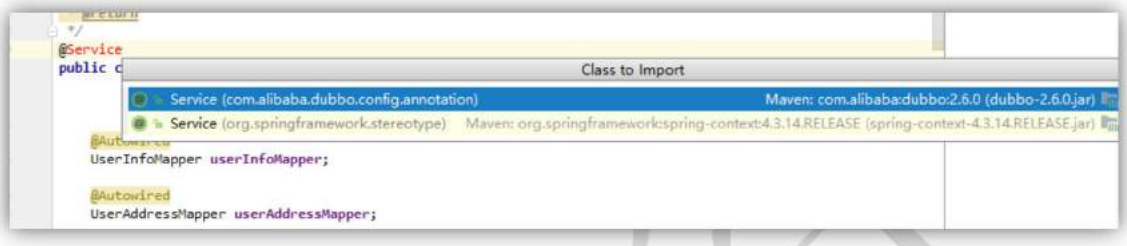
dubbo 的使用分为**提供端**和**消费端**。使用起来非常方便只要记住两个注解@Reference 和 @Service，加上 application.properties 的一段配置就可以了。

3.3 提供端

顾名思义就是提供服务供别人调用的，相当于 spring 中的 Service 的实现类。

使用也很简单，就是一个注解加一份配置

提供端在实现类上增加注解 @Service，和 spring 的是一样的但是引的包是不一样的。如下



在 UsermanageServiceImpl 实现类上重新引包，这次引入 com.alibaba.dubbo.config.annotation 这个包。

在 application.properties 中增加

```
spring.dubbo.application.name=usermanage
spring.dubbo.registry.protocol=zookeeper
spring.dubbo.registry.address=192.168.67.159:2181
spring.dubbo.base-package=com.atguigu.gmall
spring.dubbo.protocol.name=dubbo
```

其中：

application.name 就是服务名，不能跟别的 dubbo 提供端重复

registry.protocol 是指定注册中心协议

registry.address 是注册中心的地址加端口号

protocol.name 是分布式固定是 dubbo,不要改。

base-package 注解方式要扫描的包

port 是服务提供端为 zookeeper 暴露的端口，不能跟别的 dubbo 提供端重复。

3.4 消费端

order-web 模块 **application.properties 配置**

```
spring.dubbo.application.name=order-web
spring.dubbo.registry.protocol=zookeeper
spring.dubbo.registry.address=192.168.67.159:2181
spring.dubbo.base-package=com.atguigu.gmall
spring.dubbo.protocol.name=dubbo
spring.dubbo.consumer.timeout=10000
spring.dubbo.consumer.check=false
```

consumer.timeout 是访问提供端服务的超时时间，默认是 1000 毫秒

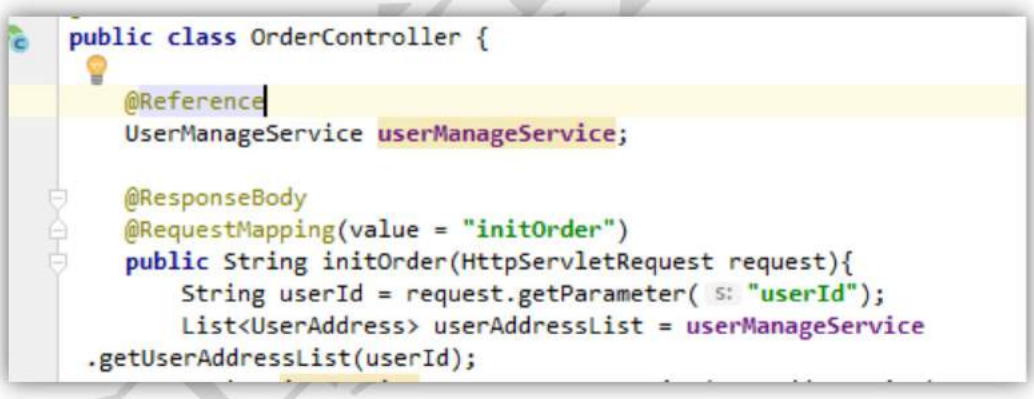
consumer.check 是启动消费端时，是否检查服务端能否正常访问。如果选择 true，那启动消费端时，必须保证提供端服务正常，否则接口无法注入。

消费端代码

使用起来也比较简单，只要把原来@Autowired 改成@Reference 就可以 注意引用的包是

```
com.alibaba.dubbo.config.annotation.Reference
```

不要引用错了



```
public class OrderController {

    @Reference
    UserManageService userManageService;

    @ResponseBody
    @RequestMapping(value = "initOrder")
    public String initOrder(HttpServletRequest request){
        String userId = request.getParameter( s: "userId");
        List<UserAddress> userAddressList = userManageService
.getUserAddressList(userId);
```

4.3.5 启动测试

那么这时候就可以测试消费端和服务端了

分别启动 order-web 模块和 usermanage 模块

然后访问

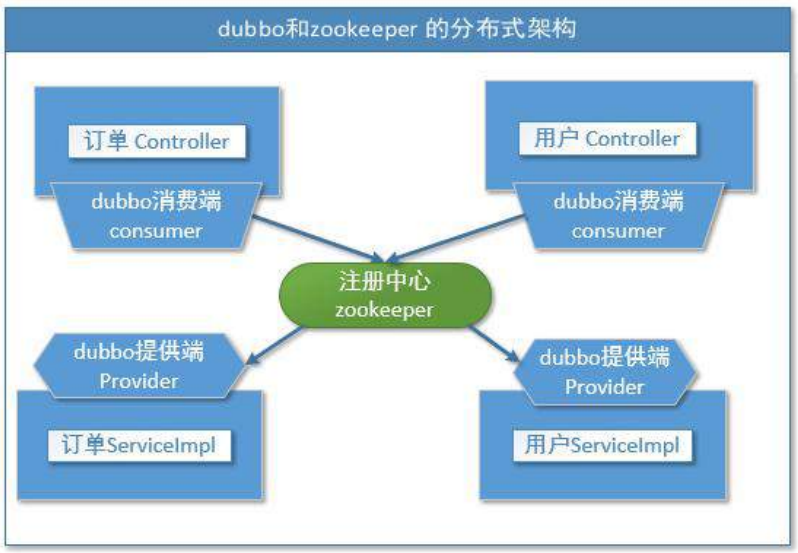
说明 controller 可以通过 dubbo 调用不同模块的 service

我们也可以通过 dubbo-admin 进行观察：

消费端

提供端

那么我们的分布式就可以基于这种方式实现不同模块间的调用。每一个实现服务的消费端和提供端分离。

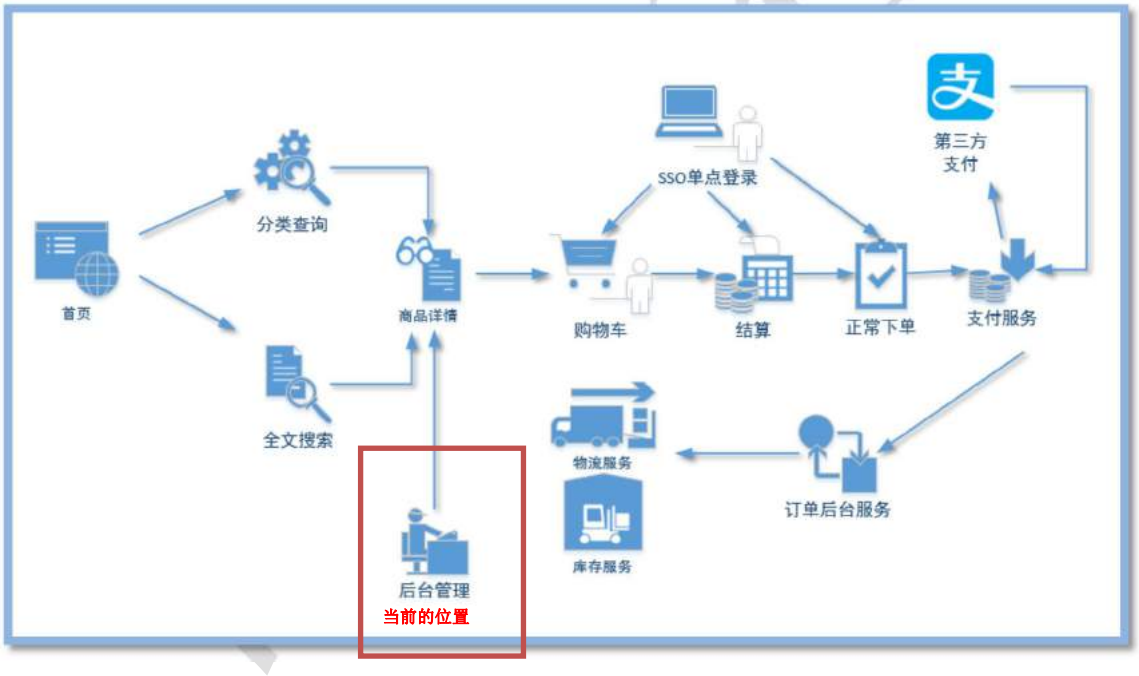


谷粒商城

版本：V 1.0
www.atguigu.com

第一章 电商的业务简介

1 整体业务简介



首页	静态页面，包含了商品分类，搜索栏，商品广告位。
全文搜索	通过搜索栏填入的关键字进行搜索，并列表展示
分类查询	根据首页的商品类目进行查询

商品详情	商品的详细信息展示
购物车	将有购买意向的商品临时存放的地方
单点登录	用户统一登录的管理
结算	将购物车中勾选的商品初始化成要填写的订单
下单	填好的订单提交
支付服务	下单后，用户点击支付，负责对接第三方支付系统。
订单服务	负责确认订单是否付款成功，并对接仓储物流系统。
仓储物流	独立的管理系统，负责商品的库存。（只对接接口不的单独讲解）
后台管理	主要维护类目、商品、库存单元、广告位等信息。

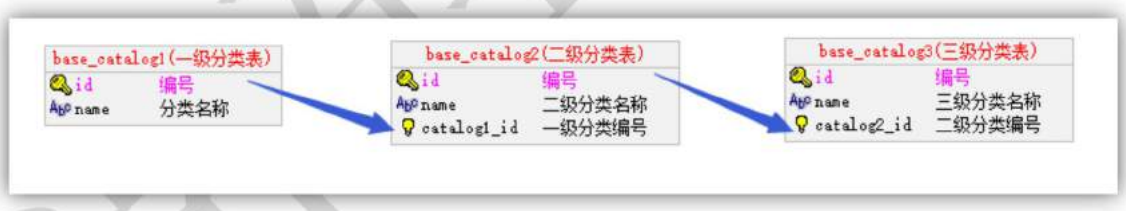
2 商品管理

2.1 基本信息—分类

一般情况可以分为两级或者三级。咱们的项目一共分为三级，即一级分类、二级分类、三级分类。

比如：家用电器是一级分类，电视是二级分类，那么超薄电视就是三级分类。

数据库结构

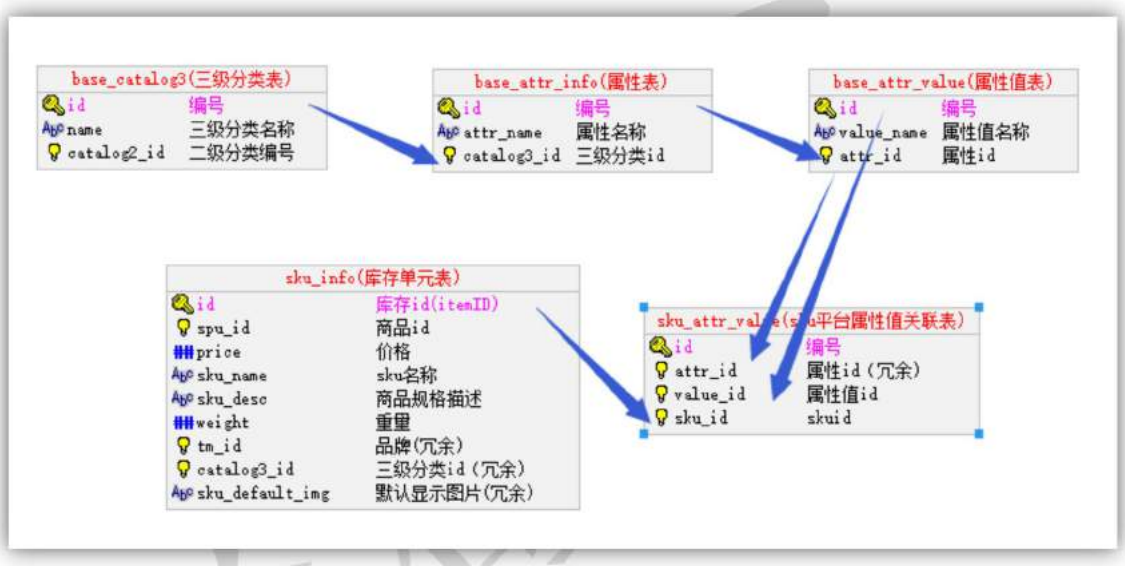


2.2 基本信息—平台属性

平台属性和平台属性值

平台属性和平台属性值主要用于商品的检索，每个三级分类对应的**属性**都不同。

而每个商品对应的每种属性都有对应的**属性值**。

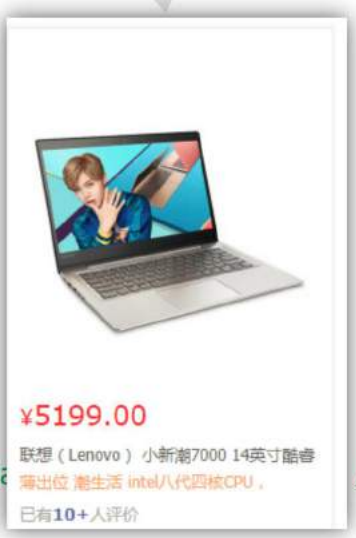


比如

电脑整机的一级分类下，有笔记本、游戏本、台式机、一体机的二级分类。

笔记本这个二级分类又包含了处理器、屏幕尺寸、内存容量、硬盘容量、显卡类别这些属性。

那么针对联想某个型号的笔记本，它作为笔记本这种分类，每个分类属性都有对应的值，cpu(属性)是 i7(属性值)的，内存(属性)是 8G(属性值)的，屏幕尺寸(属性)是 14 寸(属性值)的。



2.3 基本信息—spu 与 sku

SKU=Stock Keeping Unit（库存量单位）。即库存进出计量的基本单元，可以是以件，盒，托盘等为单位。SKU 这是对于大型连锁超市 DC（配送中心）物流管理的一个必要的方法。现在已经被引申为产品统一编号的简称，每种产品均对应有唯一的 SKU 号。

SPU(Standard Product Unit)：标准化产品单元。是商品信息聚合的最小单位，是一组**可复用、易检索**的标准化信息的集合，该集合描述了一个产品的特性。

首先通过检索搜索出来的商品列表中，每个商品都是一个 sku。每个 sku 都有自己独立的库存数。也就是说每一个商品详情展示都是一个 sku。

那 spu 又是干什么的呢？

如上图，一般的电商系统你点击进去以后，都能看到这个商品关联了其他好几个类似的商品，而且这些商品很多的信息都是共用的，比如商品图片，海报、销售属性等。

那么系统是靠什么把这些 sku 识别为一组的呢，那是这些 sku 都有一个公用的 spu 信息。而它们公共的信息，都放在 spu 信息下。

所以，sku 与 spu 的结构如下：

图中有两个图片信息表，其中 spu_image 表示整个 spu 相关下的所有图片信息，而 sku_image 表示这个 spu 下的某个 sku 使用的图片。sku_image 中的图片是从 spu_image 中选取的。

但是由于一个 spu 下的所有 sku 的海报都是一样，所以只存一份 spu_poster 就可以了。

2.4 商品信息—销售属性与平台属性

平台属性， 就是之前分类下面，辅助搜索的，类似于条件的属性。

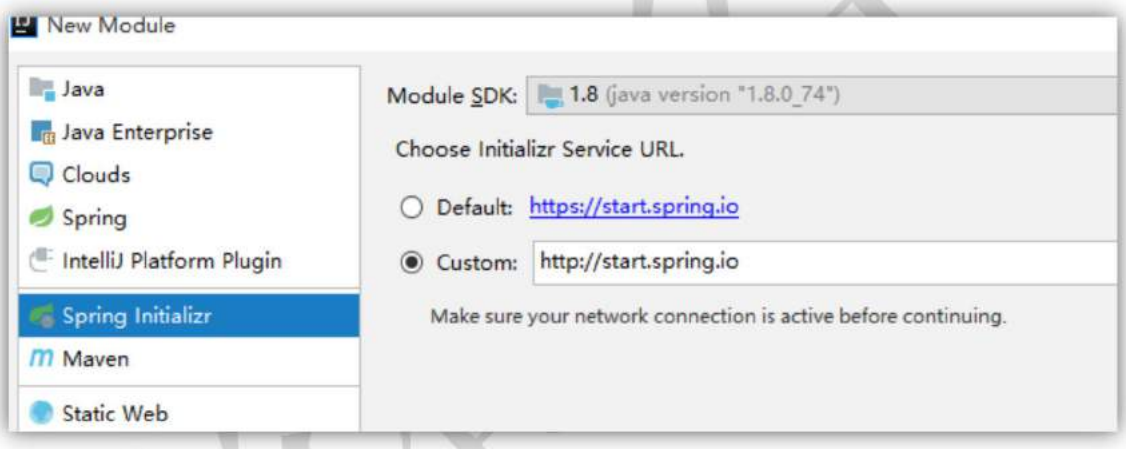
销售属性，就是商品详情页右边，可以通过销售属性来定位一组 spu 下的哪款 sku。可以让当前的商品详情页，跳转到自己的“兄弟”商品。

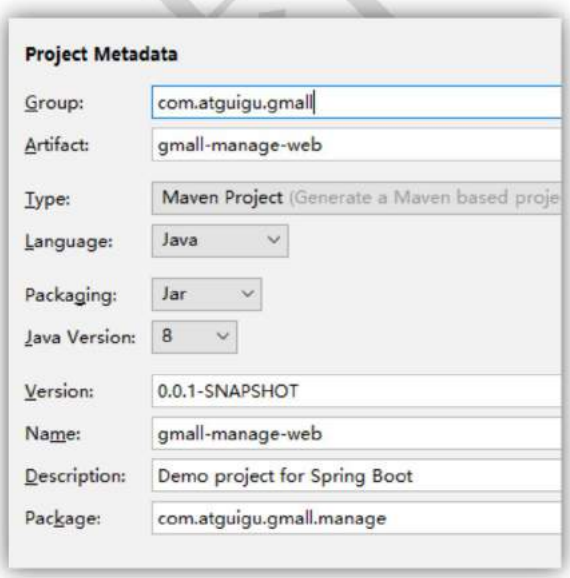
一般每种商品的销售属性不会太多，大约 1-4 种。整个平台的属性种类也不会太多，大概 10 种以内。比如：颜色、尺寸、版本、套装等等。

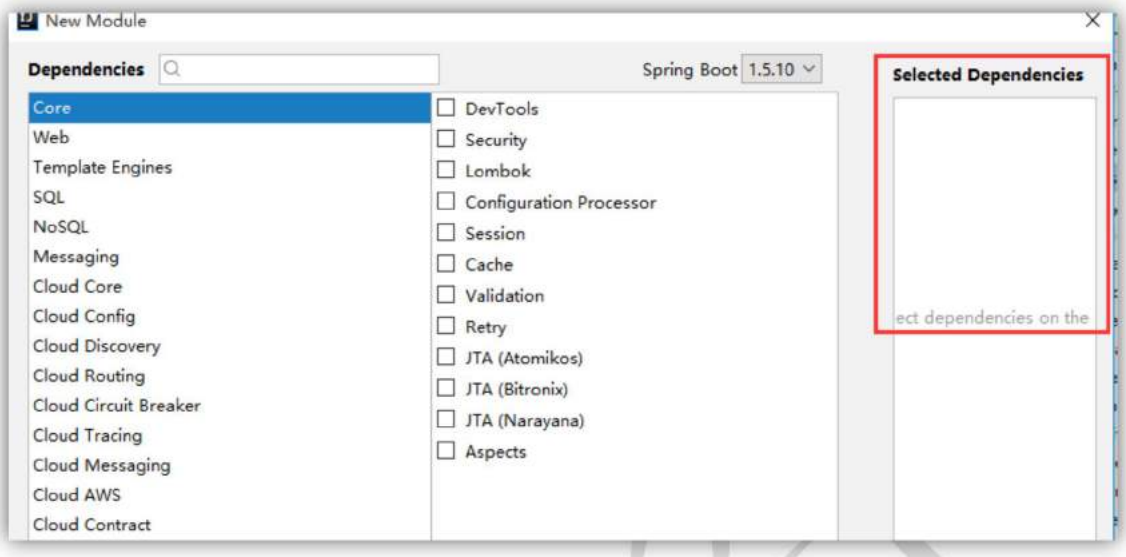
第二章　后台管理模块开发

1 后台的 Web 模块搭建

1.1　建 module







pom.xml

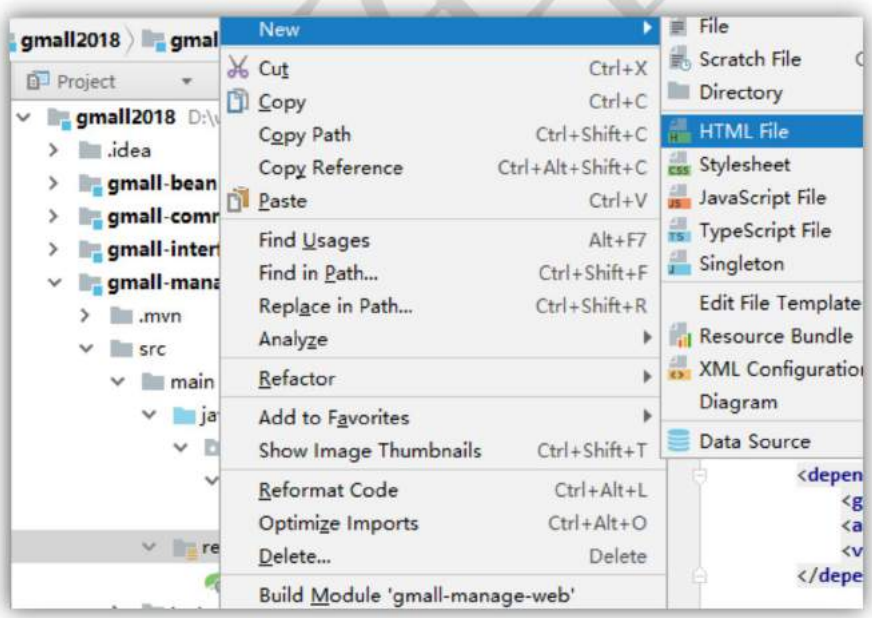
```
<?xml version="1.0" encoding="UTF-8"?>
<project                                    xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0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

    <groupId>com.atguigu.gmall</groupId>
    <artifactId>gmall-manage-web</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>gmall-manage-web</name>
    <description>Demo project for Spring Boot</description>

    <parent>
        <groupId>com.atguigu.gmall</groupId>
        <artifactId>gmall-parent</artifactId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <dependencies>
        <dependency>
            <groupId>com.atguigu.gmall</groupId>
            <artifactId>gmall-interface</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>com.atguigu.gmall</groupId>
            <artifactId>gmall-web-util</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
```

```
    </dependencies>

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

</project>
```

1.2 页面开发

如何在 web 模块中加入网页：

在 resources 中的 templates 文件夹，加入一个 indexhtml 的页面



index.html 代码

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
hello world !
</body>
</html>
```

这时如果你启动服务，然后用浏览器访问其实是访问不到的。index.html 放到 templates 只是一个模板，必须要经过 controller 层渲染，才能被访问。

增加 ManageController

```
@Controller
public class ManageController {
    @RequestMapping(value = "index" )
    public String index(){
        return "index";
    }
}
```

同时 在 web-util 的 pom.xml 文件中增加

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

如果后台出现

这是因为 springboot 默认使用 themeleaf 渲染，而这种模板语言对 html 的标签约束非常严格，所有标签必须有开有闭，比如

 或者
 是可以的，而
 是会报错的。

解决页面松校验的方式

application.properties 中增加

```
spring.thymeleaf.mode=LEGACYHTML5
```

在 gmall-parent 模块中的 pom.xml 增加依赖管理

```
<nekohtml.version>1.9.20</nekohtml.version>
<xml-apis.version>1.4.01</xml-apis.version>
<batik-ext.version>1.9.1</batik-ext.version>

<dependency>
    <groupId>net.sourceforge.nekohtml</groupId>
    <artifactId>nekohtml</artifactId>
    <version>${nekohtml.version}</version>
</dependency>

<dependency>
    <groupId>xml-apis</groupId>
    <artifactId>xml-apis</artifactId>
    <version>${xml-apis.version}</version>
</dependency>

<dependency>
    <groupId>org.apache.xmlgraphics</groupId>
    <artifactId>batik-ext</artifactId>
    <version>${batik-ext.version}</version>
</dependency>
```

在 web-util 模块中的 pom.xml 增加

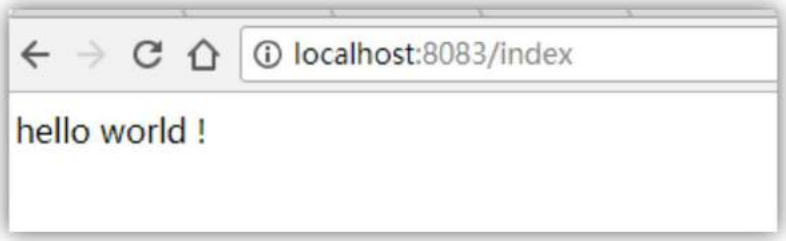
```
<dependency>
    <groupId>net.sourceforge.nekohtml</groupId>
    <artifactId>nekohtml</artifactId>
</dependency>
<dependency>
    <groupId>xml-apis</groupId>
    <artifactId>xml-apis</artifactId>
</dependency>
<dependency>
    <groupId>org.apache.xmlgraphics</groupId>
    <artifactId>batik-ext</artifactId>
</dependency>
```

请在 application.properties 中增加

```
spring.thymeleaf.cache=false
```

这个是关闭了 springboot 的页面缓存，如果不关闭会影响开发调试过程中的热部署，如果系统上线可以再把这个缓存打开。

然后测试页面



2 搭建后台页面

2.1 前置学习：EasyUI

easyui 是 一个前端框架，基于 jquery。它提供一些相应的 js，css 文件。为 web 开发提供了方便，并且有丰富的页面(美观)。通常 easyUI 用户后台的管理系统的开发模板。

通常 easyui 都用来跟 bootstrap 作比较.

共同点：

都有自己的一套完整的 ui 组件，都对各种组价进行了包装美好。但是 bootstrap 更加精致美观，动画特效，阴影，质感更好。

easyui 虽然不及 bootstrap 美观。但是组件与数据的对接更加方便，比如 easyui 中可以直接把从后台获得的一个 json 转化成一个表格直接展示给用户。而 bootstrap 的表格控件就是炫，对数据没有什么支持。

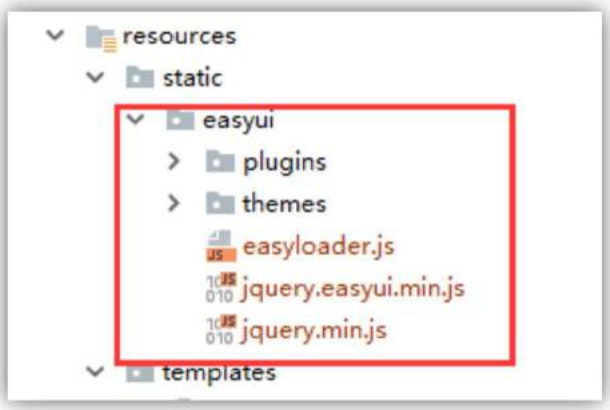
所以 easyui 对于管理系统中大量的表单、表格、插删改查的页面，开发起来更加快捷。而这些页面往往是后台的专业人员（客服、商家、运营）来使用，并不太追求界面多炫。

而 bootstrap 更适合给网上的前台用户使用，增加页面的使用手感，增强用户的体验

导入资源包

版本:easyui 1.5.4.1

把资源包解压后，要把如下图的这个几个文件包和文件考入到 idea 中的 gmall-manage-web 模块。



在页面中引用资源

```
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>主界面</title>
    <script type="text/javascript" src="/easyui/jquery.min.js"></script>
    <script type="text/javascript" src="/easyui/jquery.easyui.min.js"></script>
    <script type="text/javascript" src="/easyui/easyloader.js"></script>
    <link rel="stylesheet" type="text/css" href="/easyui/themes/icon.css">
    <link rel="stylesheet" type="text/css"    href="/easyui/themes/default/easyui.css">
</head>
```

页面布局

```
<body class="easyui-layout">
    <div data-options="region:'north',title:'头部',split:true" style="height:100px;">
            <h1>  顶部  </h1>
    </div>
    <div data-options="region:'south',title:'底部',split:true" style="height:100px;">
            <h1>底部  </h1>
    </div>
    <div data-options="region:'west',title:'左部',split:true" style="width:200px;">
            <h1>左部  </h1>
    </div>
    <div data-options="region:'center',title:'中部'"    >
            <h1>中部  </h1>
    </div>
</body>
```

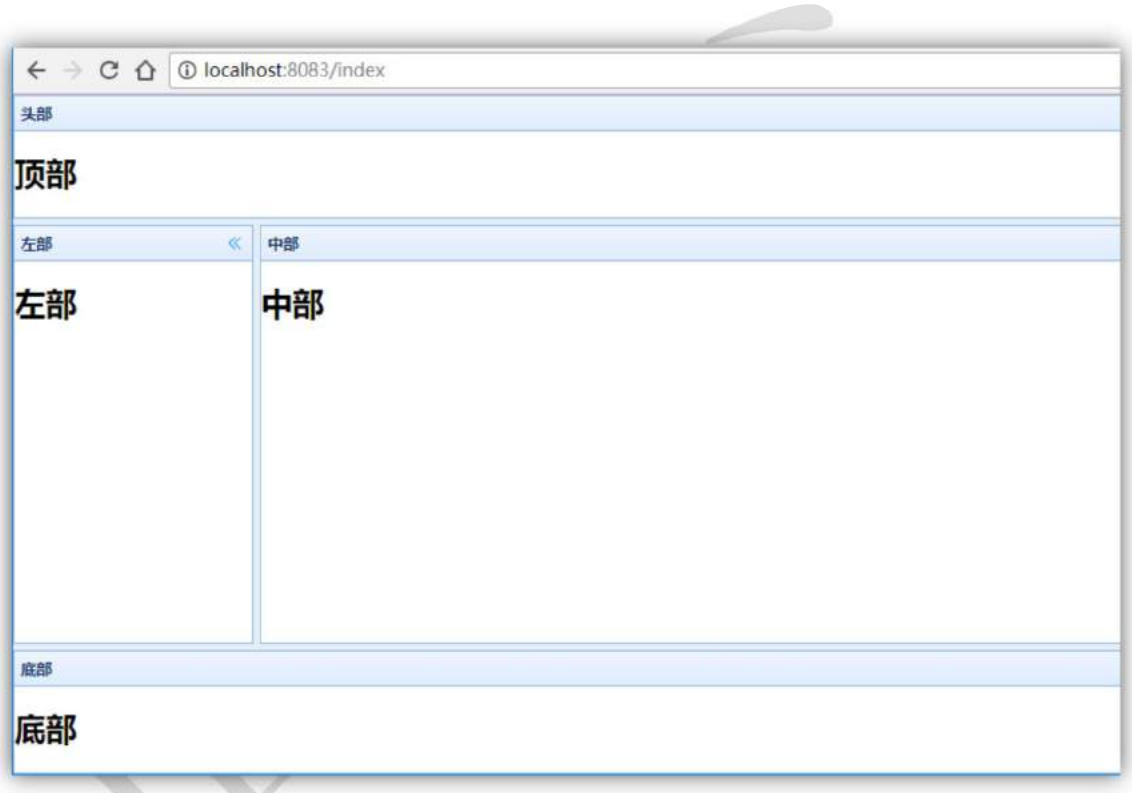
2.2 easyUI 的布局

main.html

主界面分为三部分：top，left，right

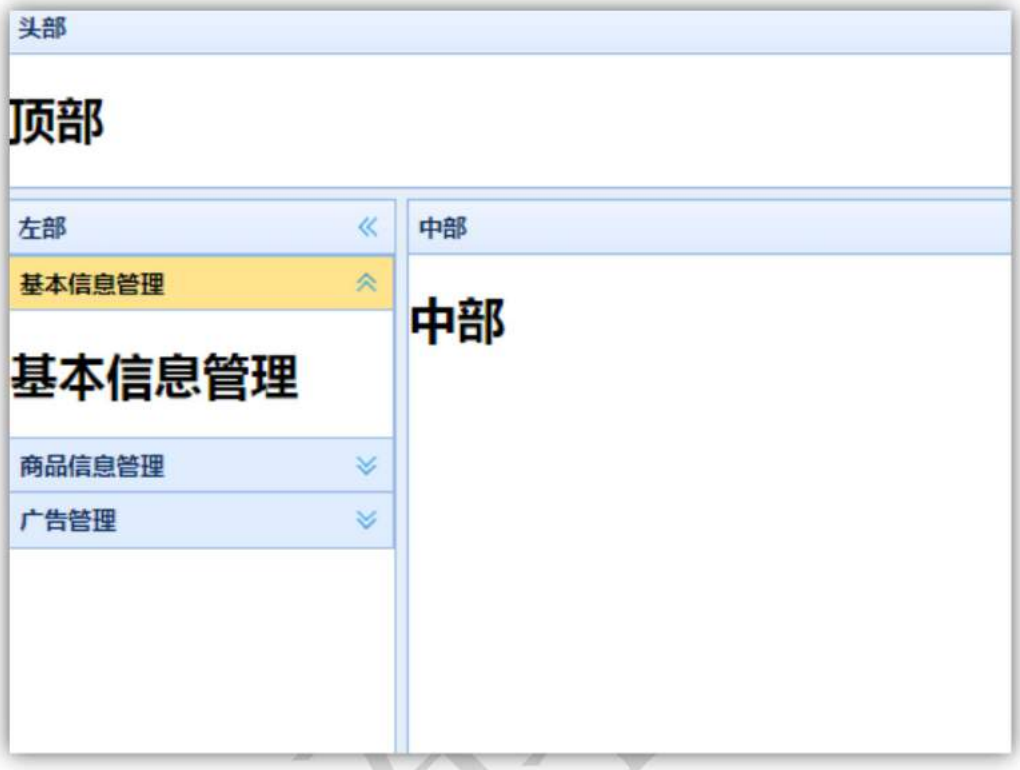
Left 部分有个树形菜单

right 部分显示左侧菜单点击后的详细页面。

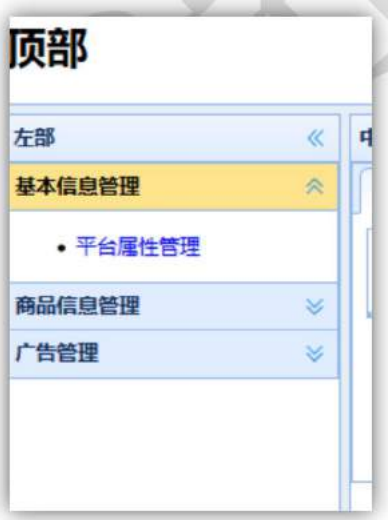


增加手风琴菜单

```
<div data-options="region:'west',title:'左部',split:true" style="width:200px;">
    <div class="easyui-accordion" style="overflow:auto;">
        <div title="基本信息管理">
            <h1>基本信息管理</h1>
        </div>
        <div title="商品信息管理">
            <h1>商品信息管理</h1>
        </div>
        <div title="广告管理">
            <h1>广告管理</h1>
        </div>
    </div>
</div>
```



左侧增加菜单




```
<div title="基本信息管理">
                <ul class="nav">
```

```
                <li><a href="#" style="text-decoration:none;">平台属性管理</a></li>
            </ul>
</div>
```

中部增加 tab 组件

```
<div data-options="region:'center',title:'中部'"  >
    <div id="tt" class="easyui-tabs" style="width:100%;height:100%;" >


    </div>
</div>
```

点击菜单增加右边的标签

```
<div title="基本信息管理">
            <ul class="nav">
                <li><a  href="javascript:add_tab('平台属性管理','attrListPage')"
style="text-decoration:none;">平台属性管理</a></li>
            </ul>
</div>
```

```
<script type="text/javascript">
    function add_tab(title,url){
        if($("#tt").tabs('exists',title)){
            $("#tt").tabs('select',title);
        }else{
            $("#tt").tabs('add',{
                title:title,
                href:url,
                closable:true
            });
        }
    }
</script>
```

左侧的属性列表页面

增加 AttrManageController

```
@Controller
public class AttrManageController {
    @RequestMapping("attrListPage")
    public String getAttrListPage(){
        return "attrListPage";
```

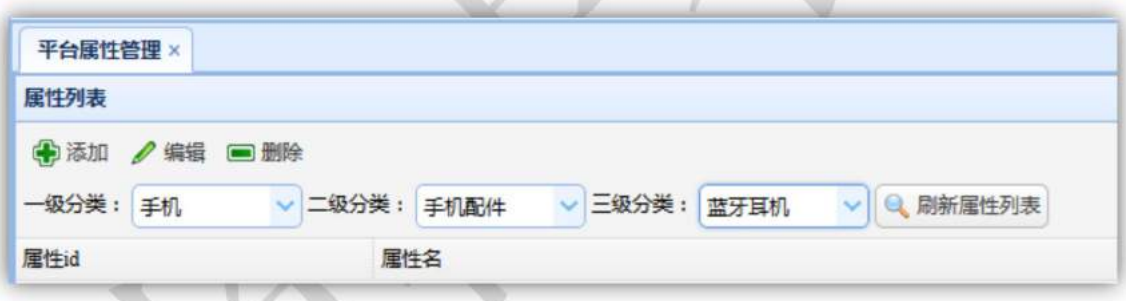
```
    }
}
```

3 属性管理功能

3.1 分类信息及属性的查询

3.1.1 页面

要达到的效果



包含：两个下拉菜单(分类选择)

一个属性列表

一组按钮

属性列表：(属性列要与 bean 的属性名称相同)

```
<table id="dg" class="easyui-datagrid" title="属性列表"
        data-options="singleSelect:true ,method:'get',toolbar:'#tb'">
    <thead>
    <tr>
        <th data-options="field:'id'" width="20%">属性 id </th>
        <th data-options="field:'attrName'" width="80%">属性名</th>
    </tr>
    </thead>
</table>
```

其中 toolbar:'#tb'　是引用了另一个工具栏，这个工具栏要额外定义如下：

工具栏

```
<div id="tb" style="padding:5px;height:auto">
    <div style="margin-bottom:5px">
        <a href="#" class="easyui-linkbutton" iconCls="icon-add" plain="true">添加</a>
        <a href="#" class="easyui-linkbutton" iconCls="icon-edit" plain="true">编辑</a>
        <a href="#" class="easyui-linkbutton" iconCls="icon-remove" plain="true">删除</a>
    </div>
    <div>
        一级分类：
        <select id="ctg1ForAttrList"    class="easyui-combobox" style="width:100px" ></select>
        二级分类：
        <select    name="ctg2ForAttrList"    id="ctg2ForAttrList"        class="easyui-combobox"
style="width:100px" ></select>
        三级分类：
        <select    name="ctg3ForAttrList"    id="ctg3ForAttrList"        class="easyui-combobox"
style="width:100px" ></select>
    </div>
</div>
```

有了页面结构，还要加上交互：

※　**一级菜单的变化，会动态改变二级菜单的内容。**

※　**二级菜单的变化，会动态改变三级菜单的内容。**

※　**三级菜单的变化，会动态改变列表显示的属性。**

还有一个页面初始化的时候，就要执行的方法：

※　**一级菜单内容的加载**

这种动态效果，一般情况下要写 js 实现。　但是利用 easyui 可以直接把这种动态变化绑定到元素上。

给下拉菜单绑定一个 data-options

```
data-options="valueField:'id',textField:'name',url:'/getCatalog1',

    onSelect:function(rec){

            $('#ctg2ForAttrList').combobox('clear');

            $('#ctg2ForAttrList').combobox('reload','getCatalog2?catalog1Id='+rec.id);

    }"
```

data-option 中可以设的内容

url	通过此链接利用 ajax 获得 json 数据
valueField	下拉选项的值要绑定 json 中的元素，要跟传入数据的字段对应
textField	下拉选项的显示内容要绑定 json 中的元素
onSelect	选中某个下拉选项会触发的方法。
.combobox('clear')	清空下拉菜单
.combobox('reload','xxx');	让某个下拉菜单按照路径加载数据。

加载 Datagrid 表格的值

```
$('#dg').datagrid({url:'getAttrList?catalog3Id='+rec.id}
```

最终代码如下：

```
<div>
    一级分类：
    <select id="ctg1ForAttrList" class="easyui-combobox" style="width:100px"
data-options="valueField:'id',textField:'name',url:'getCatalog1',
    onSelect:function(rec){
    $('#ctg2ForAttrList').combobox('clear');
$('#ctg3ForAttrList').combobox('clear');
                $('#ctg2ForAttrList').combobox('reload','getCatalog2?catalog1Id='+rec.id);
        }" ></select>
    二级分类：
    <select name="ctg2ForAttrList" id="ctg2ForAttrList" class="easyui-combobox"
data-options="valueField:'id',textField:'name',
    onSelect:function(rec){
    $('#ctg3ForAttrList').combobox('clear');
                $('#ctg3ForAttrList').combobox('reload','getCatalog3?catalog2Id='+rec.id);
        }" style="width:100px" ></select>
    三级分类：
    <select name="ctg3ForAttrList" id="ctg3ForAttrList" class="easyui-combobox"
data-options="valueField:'id',textField:'name'" style="width:100px" ></select>
    <a href="#" class="easyui-linkbutton" iconCls="icon-search"
onclick="javascript:reloadAttrList()">刷新属性列表</a>
</div>
```

属性列表的刷新按钮增加一个 js 方法

```
<script language="javascript">
    /*<![CDATA[*/
    function reloadAttrList(){
        var ctg3val=$("#ctg3ForAttrList").combobox('getValue');
```

```
            $('#dg').datagrid({url:'getAttrList?catalog3Id='+ctg3val});
        }
        /*]]>*/
</script>
```

其中红色部分，是向后台调用的 get 请求路径。

那么相应的后台要准备四个请求

getCatalog1	获得一级分类
getCatalog2	获得二级分类
getCatalog3	获得三级分类
getAttrList	获得属性列表

3.1.2 后台代码

首先是 bean,以下代码由于节省篇幅没有生产 getter,setter 方法，请自行用 idea 生成。

BaseCatalog1

```
public class BaseCatalog1 implements Serializable {
    @Id
    @Column
    private String id;
    @Column
    private String name;
}
```

BaseCatalog2

```
public class BaseCatalog2 implements Serializable {
    @Id
    @Column
    private String id;
    @Column
    private String name;
    @Column
    private String catalog1Id;
}
```

BaseCatalog3

```
public class BaseCatalog3 implements Serializable {
    @Id
    @Column
    private String id;
    @Column
    private String name;
    @Column
    private String catalog2Id;
}
```

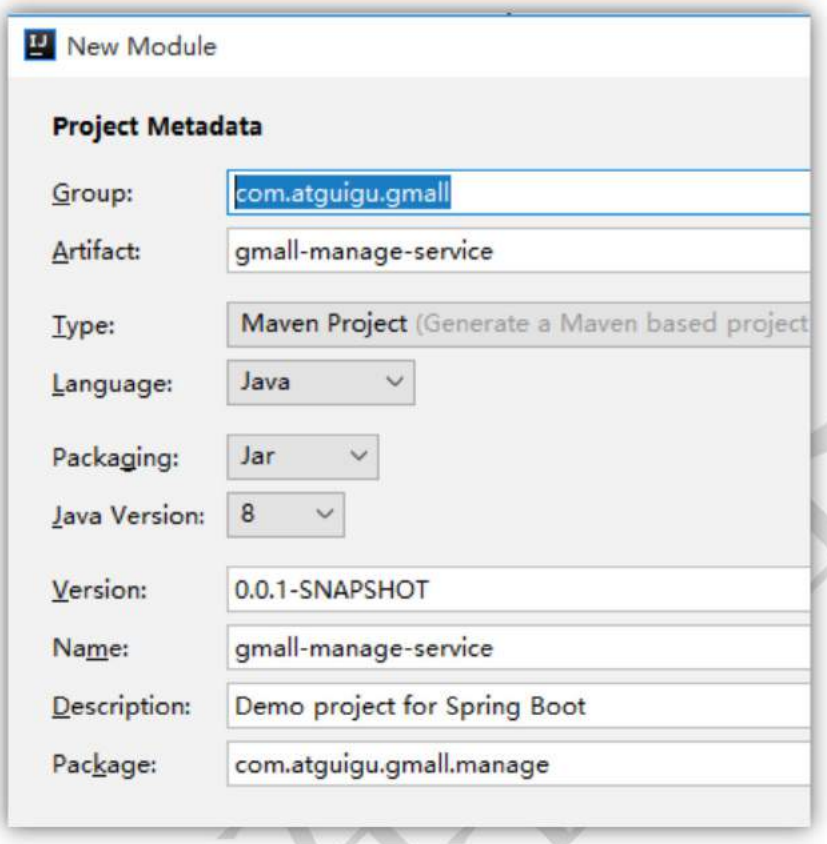
BaseAttrValue

```
public class BaseAttrValue implements Serializable {
	@Id
	@Column
	private String id;
	@Column
	private String valueName;
	@Column
	private String attrId;
	@Column
	private String isEnabled;
}
```

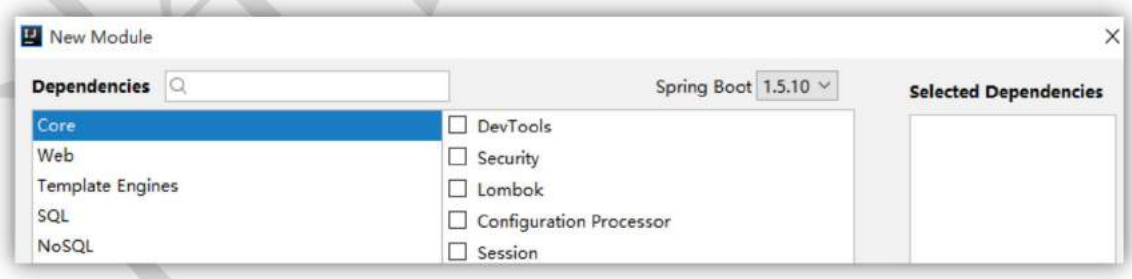
BaseAttrInfo

```
public class BaseAttrInfo implements Serializable {
	@Id
	@Column
	private String id;
	@Column
	private String attrName;
	@Column
	private String catalog3Id;
	@Column
	private String isEnabled;

}
```

创建 manage-service 模块



不用添加任何依赖



pom.xml

```
<?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.atguigu.gmall</groupId>
    <artifactId>gmall-manage-service</artifactId>
```

```
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>gmall-manage-service</name>

    <parent>
        <groupId>com.atguigu.gmall</groupId>
        <artifactId>gmall-parent</artifactId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <dependencies>

        <dependency>
            <groupId>com.atguigu.gmall</groupId>
            <artifactId>gmall-interface</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>

        <dependency>
            <groupId>com.atguigu.gmall</groupId>
            <artifactId>gmall-service-util</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>

    </dependencies>

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

在 manage-service 中 创建 Mapper

BaseCatalog1Mapper

```
public interface BaseCatalog1Mapper extends Mapper<BaseCatalog1> {
}
```

BaseCatalog2Mapper

```
public interface BaseCatalog2Mapper extends Mapper<BaseCatalog2> {
}
```

BaseCatalog3Mapper

```
public interface BaseCatalog3Mapper extends Mapper<BaseCatalog3> {
}
```

BaseAttrInfoMapper

```
public interface BaseAttrInfoMapper extends Mapper<BaseAttrInfo> {
}
```

BaseAttrValueMapper

```
public interface BaseAttrValueMapper extends Mapper<BaseAttrValue> {
}
```

在 interface 中 增加 service 接口

```
public interface ManageService {

public List<BaseCatalog1> getCatalog1();

public List<BaseCatalog2> getCatalog2(String catalog1Id);

public List<BaseCatalog3> getCatalog3(String catalog2Id);

public List<BaseAttrInfo> getAttrList(String catalog3Id);

}
```

增加实现类

```
@com.alibaba.dubbo.config.annotation.Service
public class ManageServiceImpl implements ManageService {

    @Autowired
    BaseAttrInfoMapper baseAttrInfoMapper;

    @Autowired
    BaseAttrValueMapper baseAttrValueMapper;

    @Autowired
    BaseCatalog1Mapper baseCatalog1Mapper;

    @Autowired
    BaseCatalog2Mapper baseCatalog2Mapper;

    @Autowired
    BaseCatalog3Mapper baseCatalog3Mapper;

    @Override
    public List<BaseCatalog1> getCatalog1() {
        List<BaseCatalog1> baseCatalog1List = baseCatalog1Mapper.selectAll();
        return baseCatalog1List;
    }
```

```
    @Override
    public List<BaseCatalog2> getCatalog2(String catalog1Id) {
        BaseCatalog2 baseCatalog2=new BaseCatalog2();
        baseCatalog2.setCatalog1Id(catalog1Id);

        List<BaseCatalog2> baseCatalog2List = baseCatalog2Mapper.select(baseCatalog2);
        return baseCatalog2List;
    }

    @Override
    public List<BaseCatalog3> getCatalog3(String catalog2Id) {
        BaseCatalog3 baseCatalog3=new BaseCatalog3();
        baseCatalog3.setCatalog2Id(catalog2Id);

        List<BaseCatalog3> baseCatalog3List = baseCatalog3Mapper.select(baseCatalog3);
        return baseCatalog3List;
    }

    @Override
    public List<BaseAttrInfo> getAttrList(String catalog3_id) {
        BaseAttrInfo baseAttrInfo = new BaseAttrInfo();
        baseAttrInfo.setCatalog3Id(catalog3_id);

        List<BaseAttrInfo> baseAttrInfoList = baseAttrInfoMapper.select(baseAttrInfo);
        return baseAttrInfoList;

    }
}
```

manage-web 的 controller 中 AttrManageController 中增加方法

```
@Controller
public class AttrManageController {

    @Reference
    ManageService manageService;

    @RequestMapping("attrListPage")
    public String getAttrListPage(){
        return "attrListPage";
    }


    /***
     * 获得一级分类
     * @return
     */
    @RequestMapping("getCatalog1")
    @ResponseBody
    public List<BaseCatalog1> getCatalog1(){
```

```
        List<BaseCatalog1> catalog1List = manageService.getCatalog1();
        return catalog1List;
    }


    /***
     * 获得二级分类
     * @param map
     * @return
     */
    @RequestMapping("getCatalog2")
    @ResponseBody
    public List<BaseCatalog2> getCatalog2(@RequestParam Map<String,String> map){
        String catalog1Id =     map.get("catalog1Id") ;
        List<BaseCatalog2> catalog2List = manageService.getCatalog2(catalog1Id);
        return catalog2List;
    }

    /***
     * 获得三级分类
     * @param map
     * @return
     */
    @RequestMapping("getCatalog3")
    @ResponseBody
    public List<BaseCatalog3> getCatalog3(@RequestParam Map<String,String> map){
        String catalog2Id =     map.get("catalog2Id") ;
        List<BaseCatalog3> catalog3List = manageService.getCatalog3(catalog2Id);
        return catalog3List;
    }

/***
 * 获得属性列表
 * @param map
 * @return
 */
@RequestMapping("getAttrList")
@ResponseBody
public List<BaseAttrInfo>   getAttrList(@RequestParam Map<String,String> map){
    String catalog3Id =     map.get("catalog3Id") ;
    List<BaseAttrInfo> attrList = manageService.getAttrList(catalog3Id);
    return attrList;
}

}
```

配置 manage-web 的 application.properties

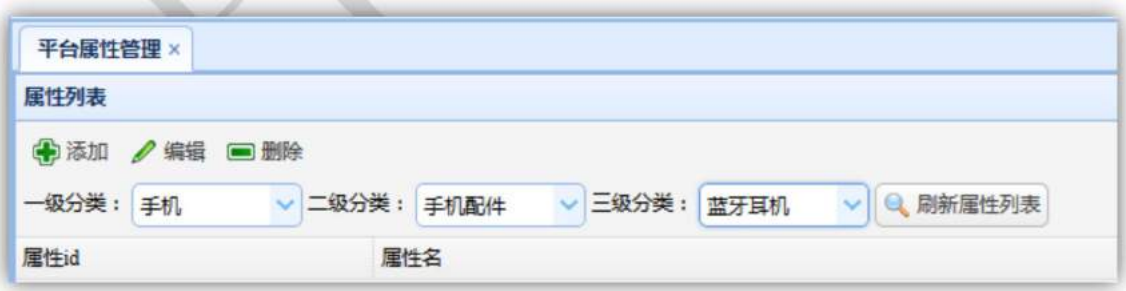
```
server.port=8083

spring.thymeleaf.cache=false

spring.thymeleaf.mode=LEGACYHTML5
```

```
spring.dubbo.application.name=manage-web
spring.dubbo.registry.protocol=zookeeper
spring.dubbo.registry.address=192.168.67.159:2181
spring.dubbo.base-package=com.atguigu.gmall
spring.dubbo.protocol.name=dubbo
spring.dubbo.consumer.timeout=10000
spring.dubbo.consumer.check=false
```

配置 manage-service 的 application.properties

```
server.port=8073

logging.level.root=debug

spring.dubbo.application.name=manage-service
spring.dubbo.registry.protocol=zookeeper
spring.dubbo.registry.address=192.168.67.159:2181
spring.dubbo.base-package=com.atguigu.gmall
spring.dubbo.protocol.name=dubbo
spring.datasource.url=jdbc:mysql://59.110.141.236:3306/gmall?characterEncoding=UTF-8
spring.datasource.username=root
spring.datasource.password=123123
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```

启动服务后



3.2 属性的添加、编辑

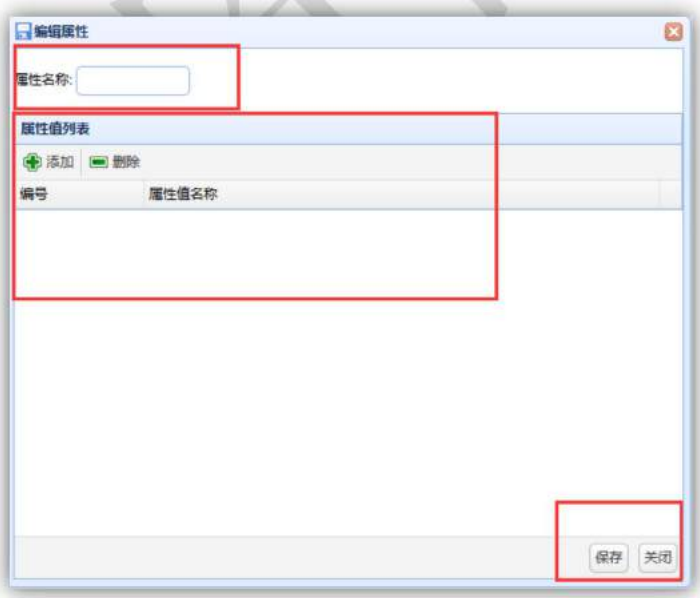
3.2.1 页面

点击增加后弹出编辑框

```
<div style="margin-bottom:5px">
    <a href="#" class="easyui-linkbutton" iconCls="icon-add" plain="true" onclick="addAttrInfo()">添加</a>
    <a href="#" class="easyui-linkbutton" iconCls="icon-edit" plain="true" onclick="editAttrInfo()">编辑</a>
    <a href="#" class="easyui-linkbutton" iconCls="icon-remove" plain="true">删除</a>
</div>
```

制作弹出框

弹出框三部分组成： 属性名称的文本域，属性值列表，保存按钮。



弹出框 html

```
<div id="dlg" class="easyui-dialog" title="编辑属性" style="width:600px;height:500px;"
```

```
      data-options="iconCls:'icon-save',resizable:true,modal:true" buttons="#bb" >
    <form id="attrForm">
        <br/>
        <label>属性名称:</label>
         <input     id="attrName"  name="attrName"  class="easyui-textbox"  data-options=""
style="width:100px"/>
         <input   id="attrId" name="attrId" type="hidden"   />
         <br/><br/>
         <table id="dg_av" class="easyui-datagrid" title="属性值列表"></table>
    </form>
</div>

<div id="bb">
    <a href="#" class="easyui-linkbutton" onclick="saveAttr()">保存</a>
    <a href="#" class="easyui-linkbutton">关闭</a>
</div>
```

其中有一个 hidden 的隐藏域用来，存放当前属性的 id。

这里的 datagrid 没有初始化按钮和列头，所以首先要编写 datagrid 的初始化方法。

```
function initAttrValueDatagrid(){
        $('#dg_av').datagrid('loadData', { total: 0, rows: [] });
        datagrid = $("#dg_av").datagrid({
            columns:[[
                { field:'id',title:'编号',width:'20%'},
                { field:'valueName',title:'属性值名称',width:'80%',
                    editor: {
                        type: 'validatebox', options: { required: true}   //必填项
                    }
                }
            ]],
            toolbar:[{text:'添加',iconCls:'icon-add',
                handler:function () {
                    datagrid.datagrid('appendRow',{id:'',valueName:''});
                }
            },'-',
                {text:'删除',iconCls:'icon-remove',
                    handler:function () {
                        var row = datagrid.datagrid('getSelected');
                        if (row) {
                            var rowIndex = datagrid.datagrid('getRowIndex', row);
                            datagrid.datagrid('deleteRow', rowIndex);
                        }
                    }
                }],
            onDblClickRow: function (rowIndex, rowData) {
                //双击开启编辑行
                    datagrid.datagrid("beginEdit", rowIndex);
                    //设定当失去焦点时,退出编辑状态
                    var valueName = rowData.valueName;

$("input.datagrid-editable-input").val(valueName).bind("blur",function(evt){
                        datagrid.datagrid('endEdit',rowIndex);
```

```
                        });
                }
            });
}
```

然后点击增加后要弹出对话框

```
function addAttrInfo(){
    if(!checkBeforeDialog()){
        return ;
    }
   //进系统前先清空
    $("#attrId").val("");
    $("#attrName").textbox('clear');
    $('#dg_av').datagrid({url:''});
    //  初始化 datagrid
    initAttrValueDatagrid();

    //弹出框
    $("#dlg").dialog("open");
}

function editAttrInfo(){
    if(!checkBeforeDialog()){
        return ;
    }
    //  初始化 datagrid
    initAttrValueDatagrid();
    //进页面前先加载数据
    var attrInfoRow=$("#dg").datagrid('getSelected');
    $('#dg_av').datagrid({url:'getAttrValueList?attrId='+attrInfoRow.id});
    $("#attrId").val(attrInfoRow.id);
    $("#attrName").textbox('setValue',attrInfoRow.attrName);

    //弹出框
    $("#dlg").dialog("open");
}

function checkBeforeDialog(){
    var ctg3val = $("#ctg3ForAttrList").combobox('getValue');
    if(ctg3val==''){
        $.messager.alert('警告','请先选择三级分类','warning');
        return false;
    }
    return true;
}
```

3.2.2 后台代码

点击编辑时调用后台的请求

getAttrValueList	获得属性值列表信息的请求

由于属性和属性值是一对多的关系，所以属性的 bean 中增加一个列表元素

```
public class BaseAttrInfo implements Serializable {

    @Id
    @Column
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private String id;
    @Column
    private String attrName;
    @Column
    private String catalog3Id;

    @Transient
    private List<BaseAttrValue> attrValueList;
}
```

其中@Transient 表示该 Bean 类对应的数据库表中不包含的字段，这个注解必须要加上否则报错。

AttrManageController

```
@RequestMapping(value = "getAttrValueList",method = RequestMethod.POST)
@ResponseBody
public List<BaseAttrValue> getAttrValueList(@RequestParam Map<String,String>
map){
      String attrId= map.get("attrId");
      BaseAttrInfo attrInfo = manageService.getAttrInfo(attrId);
      return attrInfo.getAttrValueList();
}
```

gmall-manage-service 中的实现 ManageServiceImpl

```
@Override
public BaseAttrInfo getAttrInfo(String id) {
```

```
    //查询属性基本信息
    BaseAttrInfo baseAttrInfo = baseAttrInfoMapper.selectByPrimaryKey(id);

    //查询属性对应的属性值
    BaseAttrValue  baseAttrValue4Query =new BaseAttrValue();
    baseAttrValue4Query.setAttrId(baseAttrInfo.getId());
    List<BaseAttrValue> baseAttrValueList =
baseAttrValueMapper.select(baseAttrValue4Query);

    baseAttrInfo.setAttrValueList(baseAttrValueList);
    return baseAttrInfo;
}
```

3.3 属性和属性值的保存

3.3.1 页面 js

```
function saveAttr(){
    var attrJson ={} ;
    //把表格中的数据循环组合成 json
    var attrValueRows= $("#dg_av").datagrid('getRows') ;
    for (var i = 0; i < attrValueRows.length; i++) {
        //技巧：与 bean 中的属性同名可以借助 springmvc 直接注入到实体 bean 中，即使
是 list 也可以。
        attrJson["attrValueList["+i+"].id"]=attrValueRows[i].id;
        attrJson["attrValueList["+i+"].valueName"]=attrValueRows[i].valueName;
    }

    attrJson["attrName"]=$("#attrName").val();
    attrJson["id"]=$("#attrId").val();
    attrJson["catalog3Id"]=$("#ctg3ForAttrList").combobox('getValue');

    //ajax 保存到后台
    $.post("saveAttrInfo",attrJson,function(data){
        $("#dlg").dialog("close");
        $("#dg").datagrid("reload");
    })
}
```

3.3.2 后台代码

保存时调用后台的请求：

saveAttrInfo	保存属性和属性值的请求

AttrManageController

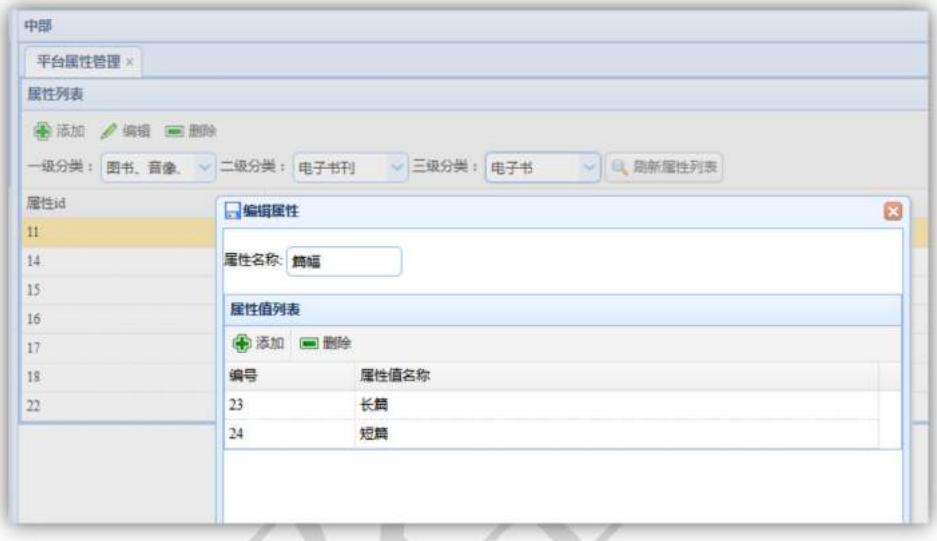
```
@RequestMapping(value = "saveAttrInfo",method = RequestMethod.POST)
@ResponseBody
public String saveAttrInfo(BaseAttrInfo baseAttrInfo){
    manageService.saveAttrInfo(baseAttrInfo);
    return "success";
}
```

gmall-manage-service 中的实现 ManageServiceImpl

```
@Override
public void saveAttrInfo(BaseAttrInfo baseAttrInfo) {

    //如果有主键就进行更新，如果没有就插入
    if(baseAttrInfo.getId()!=null&&baseAttrInfo.getId().length()>0){
        baseAttrInfoMapper.updateByPrimaryKey(baseAttrInfo);
    }else{
        //防止主键被赋上一个空字符串
        if(baseAttrInfo.getId().length()==0){
            baseAttrInfo.setId(null);
        }
        baseAttrInfoMapper.insertSelective(baseAttrInfo);
    }
      //把原属性值全部清空
      BaseAttrValue baseAttrValue4Del = new BaseAttrValue();
      baseAttrValue4Del.setAttrId(baseAttrInfo.getId());
      baseAttrValueMapper.delete(baseAttrValue4Del);

      //重新插入属性
      if(baseAttrInfo.getAttrValueList()!=null&&baseAttrInfo.getAttrValueList().size()>0) {
        for (BaseAttrValue attrValue : baseAttrInfo.getAttrValueList()) {
            //防止主键被赋上一个空字符串
            if(attrValue.getId()!=null&&attrValue.getId().length()==0){
                attrValue.setId(null);
            }
            attrValue.setAttrId(baseAttrInfo.getId());
            baseAttrValueMapper.insertSelective(attrValue);
        }
    }
}
```

节省篇幅，接口类中的代码不再赋上，请自行添加。

3.4 最终效果



谷粒商城

版本：V 1.0

www.atguigu.com

一 业务介绍

1 SPU 与 SKU

SPU(Standard Product Unit)：标准化产品单元。是商品信息聚合的最小单位，是一组可复用、易检索的标准化信息的集合，该集合描述了一个产品的特性。

Stock Keeping Unit（库存量单位）。即库存进出计量的基本单元，可以是以件，盒，托盘等为单位。SKU 这是对于大型连锁超市 DC（配送中心）物流管理的一个必要的方法。现在已经被引申为产品统一编号的简称，每种产品均对应有唯一的 SKU 号。

比如，咱们购买一台 iPhoneX 手机，iPhoneX 手机就是一个 SPU，但是你购买的时候，不可能是以 iPhoneX 手机为单位买的，商家也不可能以 iPhoneX 为单位记录库存。必须要以什么颜色什么版本的 iPhoneX 为单位。比如，你购买的是一台银色、128G 内存的、支持联通网络的 iPhoneX ，商家也会以这个单位来记录库存数。那这个更细致的单位就叫库存单元（SKU）。

销售属性与平台属性

销售属性，就是商品详情页右边，可以通过销售属性来定位一组 spu 下的哪款 sku。可以让当前的商品详情页，跳转到自己的“兄弟”商品。

一般每种商品的销售属性不会太多，大约 1-4 种。整个电商的销售属性种类也不会太多，大概 10 种以内。比如：颜色、尺寸、版本、套装等等。不同销售属性的组合也就构成了一个 spu 下多个 sku 的结构。

平台属性， 就是之前分类下面，辅助搜索的，类似于条件的属性。

销售属性与平台属性各自独立。一个 SPU 会决定一个商品都有哪些销售属性，比如 iPhonx 会有颜色、版本、内存的销售属性，某个 T 恤衫只有尺寸这个销售属性。

而某个商品有什么平台属性，由他的 3 级分类决定。比如笔记本包括：运行内存、cpu、显卡、硬盘、屏幕尺寸等等。

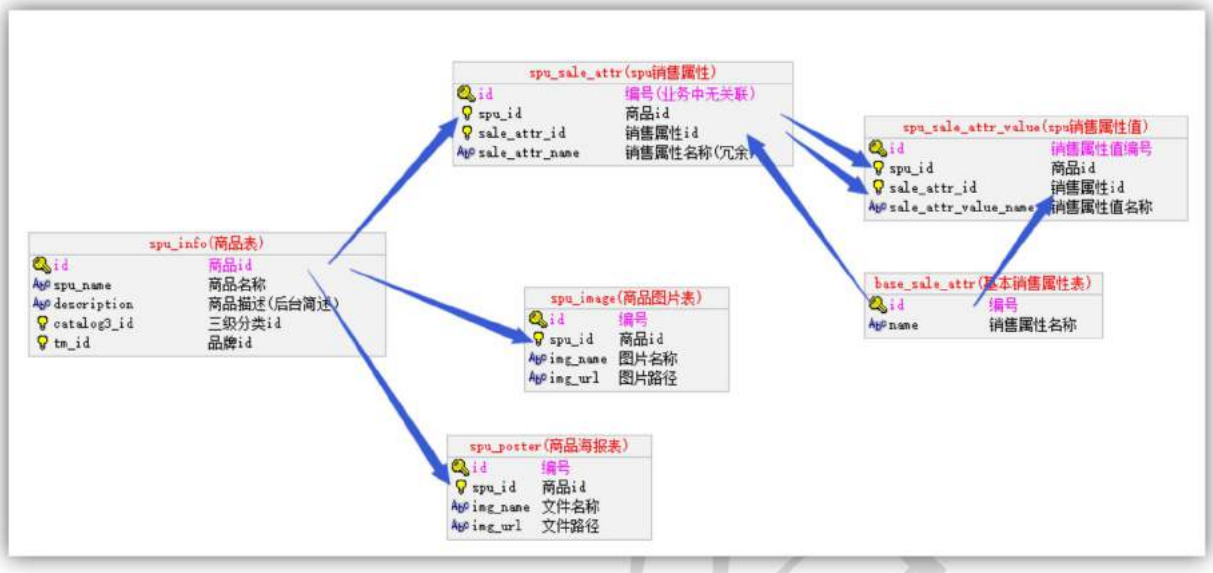
2 SKU 与 SPU 的图片资源

另外同一个 SPU 下的 SKU 可以共用一些资源，比如商品图片，海报等等。毕竟同一种商品，大部分图片都是共用的只有因为颜色尺寸等，很少的差别。那么一般来说商品图片都是在新增 SPU 时上传的，在新增 SKU 时从该 SPU 已上传的图片中选择。

而海报几乎是所有 SPU 下的 SKU 都一样。

3 数据结构图

根据以上的需求，以此将 SPU 关联的数据库表结构设计为如下：



4 数据示例：

spu_info(商品表)

商品编号(spu_id)	spu_name	
6110	iPhoneX	

基本销售属性

id	name
101	颜色
102	尺寸
103	版本
104	内存容量

spu销售属性

id	spu_id	sale_attr_id	sale_attr_name
9901	6110	101	颜色
9902	6110	103	版本
9903	6110	104	内存容量

spu销售属性值

id	spu_id	sale_attr_id	sale_att_value_name
88801	6110	101	太空银
88802	6110	101	土豪金
88803	6110	103	移动
88804	6110	103	联通
88805	6110	104	64G
88806	6110	104	256G

spu_image 图片表

id	spu_id	img_name	img_url
77701	6110	正面图	http://xxxx/xxx/xx.jpg
77702	6110	侧面图	http://xxxx/xxx/xx.jpg
77703	6110	反面图	http://xxxx/xxx/xx.jpg
77704	6110	太空银版图	http://xxxx/xxx/xx.jpg
77705	6110	土豪金版图	http://xxxx/xxx/xx.jpg

spu_poster 海报表

id	spu_id	img_name	img_url
67701	6110	海报1	http://xxxx/xxx/xx.jpg
67702	6110	海报2	http://xxxx/xxx/xx.jpg
67703	6110	海报3	http://xxxx/xxx/xx.jpg
67704	6110	海报4	http://xxxx/xxx/xx.jpg
67705	6110	海报5	http://xxxx/xxx/xx.jpg

二、列表查询功能开发

1 首页菜单 index.html

```
<div title="商品信息管理">
    <ul class="nav">
        <li><a href="javascript:add_tab('商品信息管理','spuListPage')" style="text-decoration:none;">商品SPU管理</a></li>
    </ul>
</div>
```

不要忘记增加 controller 的跳转。

2 SpuManageController

```
@Controller
public class SpuManageController {

@RequestMapping("spuListPage")
    public String getSpuListPage(){
        return "spuListPage";
    }
}
```

3 SPU 列表页面

和属性列表大体相同，都是通过三级分类，获得 spu 信息列表。

spuListPage.html

```
<!DOCTYPE html>
<html lang="en"    xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org">
<head>

    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>spu 列表</title>
    <script type="text/javascript" src="/easyui/jquery.min.js"></script>
    <script type="text/javascript" src="/easyui/jquery.easyui.min.js"></script>
    <script type="text/javascript" src="/easyui/easyloader.js"></script>

    <link rel="stylesheet" type="text/css" href="/easyui/themes/icon.css">
    <link rel="stylesheet" type="text/css"    href="/easyui/themes/default/easyui.css">
```

```
    <!-- 引入图片上传工具 webuploader-->
    <link rel="stylesheet" type="text/css" href="/webuploader/webuploader.css">
    <script type="text/javascript" src="/webuploader/webuploader.js"></script>
</head>
<body>
<div class="easyui-panel" title=""    data-options="border:true">
    <!----------------列表 ---------------------------->
    <table id="spulist_dg" class="easyui-datagrid" title="spu 列表"
           data-options="singleSelect:true,method:'get' ,toolbar:'#spulist_tb'">
        <thead>
        <tr>
            <th data-options="field:'id'" width="10%">商品 id </th>
            <th data-options="field:'spuName'" width="30%">商品名称</th>
            <th data-options="field:'description'" width="60%">商品描述 </th>
        </tr>
        </thead>
    </table>
    <!----------------列表的工具栏---------------------------->
    <div id="spulist_tb" style="padding:5px;height:auto">
        <div style="margin-bottom:5px">
            <a href="#" class="easyui-linkbutton" iconCls="icon-add" plain="true" onclick="addSpuInfo()"> 添
加</a>
            <a href="#" class="easyui-linkbutton" iconCls="icon-edit" plain="true" onclick="editSpuInfo()"> 编
辑</a>
            <a href="#" class="easyui-linkbutton" iconCls="icon-remove" plain="true">删除</a>
            <a href="#" class="easyui-linkbutton" iconCls="icon-add" plain="true" onclick="addSkuInfo()"> 增
加 sku</a>
            <a href="#" class="easyui-linkbutton" iconCls="icon-search" plain="true"
onclick="showSkuInfoList()"> sku 列表</a>
        </div>
        <div>
            一级分类：
            <select id="ctg1ForSpuList" class="easyui-combobox" style="width:100px"
data-options="valueField:'id',textField:'name',url:'getCatalog1',
            onSelect:function(rec){
            $('#ctg2ForSpuList').combobox('clear');
            $('#ctg3ForSpuList').combobox('clear');
                    $('#ctg2ForSpuList').combobox('reload','getCatalog2?catalog1Id='+rec.id);
                }" ></select>
            二级分类：
            <select name="ctg2ForSpuList" id="ctg2ForSpuList" class="easyui-combobox"
data-options="valueField:'id',textField:'name',
            onSelect:function(rec){
            $('#ctg3ForSpuList').combobox('clear');
                    $('#ctg3ForSpuList').combobox('reload','getCatalog3?catalog2Id='+rec.id);
                }" style="width:100px" ></select>
            三级分类：
            <select name="ctg3ForSpuList" id="ctg3ForSpuList" class="easyui-combobox"
data-options="valueField:'id',textField:'name',
            onSelect:function(rec){
                    $('#spulist_dg').datagrid({url:'spuList?catalog3Id='+rec.id});
                }
```

```
"      style="width:100px" ></select>
                <a                href="#"              class="easyui-linkbutton"              iconCls="icon-search"
onclick="javascript:reloadSpuList()"    >刷新列表</a>
            </div>
        </div>
</div>
```

特别注意：咱们现在使用的是 easyui 的单一页面模式，也就是说其实所有的页面最后都会加载到一个页面上， 那么要注意的地方就是中所有打开的标签页的元素 id，都不能重复否则会发生冲突。

4 后台代码

4.1 controller

```
@RequestMapping("spuList")
@ResponseBody
public List<SpuInfo> getSpuInfoList(@RequestParam Map<String,String> map){
      String catalog3Id = map.get("catalog3Id");
      SpuInfo spuInfo =new SpuInfo();
      spuInfo.setCatalog3Id(catalog3Id);
      List<SpuInfo> spuInfoList = manageService.getSpuInfoList(spuInfo);
      return spuInfoList;

}
```

4.2 bean

```
public class SpuInfo implements Serializable {
    private String id;
    private String spuName;
    private String description;
    private  String catalog3Id;
}
```

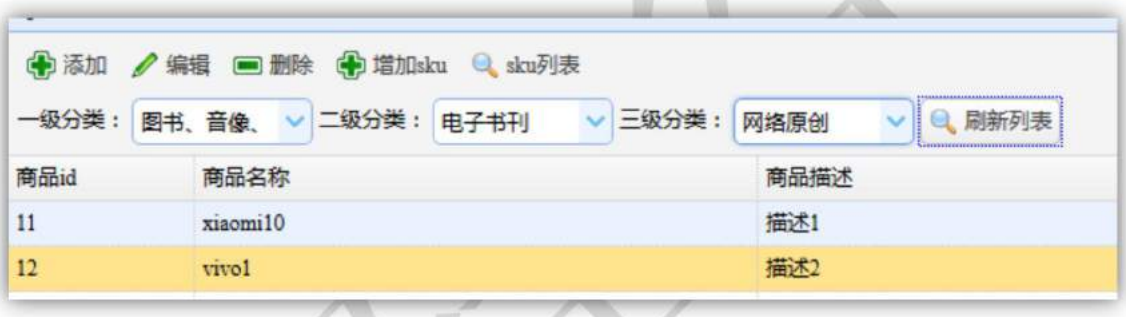
在 gmall-manage-service 工程中增加

4.3 mapper

```
public interface SpuInfoMapper extends Mapper<SpuInfo> {
}
```

实现类 manageServiceImpl 增加方法

```
public List<SpuInfo> getSpuInfoList(SpuInfo spuInfo){
    List<SpuInfo> spuInfoList = spuInfoMapper.select(spuInfo);
    return    spuInfoList;
}
```



三、 spu 的保存功能

1 点击新增弹出录入框

首先由列表页点击【增加】出现弹出框

```
function addSpuInfo(){
    initSpuInfoDlg();   //在 spuInfoPage.html 中
}
```

由于这次弹出框是新的窗体，虽然整个后台系统是一个单一页面，但是如果把所有代码都集中到一个 html 的话，维护起来非常不方便，可读性也差。所以咱们利用页面的渲染工具 themeleaf 的一个标签，把一段代码提取到另外一个 html 文件中。

```
<div th:include="spuInfoPage"></div>
```

这个有点类似于 Jsp 的<jsp:include page="">标签。

这样我们跟这个新增弹出框窗体有关系的都可以放到 spuInfoPage.html 这个文件中了。

1.1 创建 spuInfoPage.html（商品 spu 的详情页）

代码见思维导图

2 难点分析

两个难点：

1、 涉及上传图片附件，而且要上传多个图片附件，每个图片都要可以管理分配，因为每张图片最后都要分配给不同的 sku 。 同时上传成功后用户可以预览。

2、 一个 spu 对应多个销售属性，但是每个销售属性又对应了多个属性值。

3 图片上传预览解决方案：

※ WebUploader (客户端图片上传控件)

※ FastDFS(服务器端文件管理软件)

※ datagridview (easyui 预览控件)

3.1 首先先说 WebUploader

官方网站 ：‘

WebUploader 是由 Baidu WebFE(FEX)团队开发的一个简单的以 HTML5 为主，FLASH 为辅的现代文件上传组件。

如何使用，可以参考官网上给的例子。

简单归纳：要安装约定完成控件需要的几个动作。

3.2 webuploader 的实现步骤

控件的创建	var spuImgUploader = WebUploader.create({
	auto:false,
	// swf 文件路径
	swf: '/webuploader/Uploader.swf',
	// 文件接收路径
	server: '/fileUpload',
	// 选择文件的按钮。
	pick: '#spuImgAdd',
	// 不压缩 image，默认如果是 jpeg，文件上传前会压缩一把再上传!
	resize: false,
	//设定文件大小上限 2M
	fileSingleSizeLimit:2*1024*1024,
	//可接受的文件类型
	accept: {
	title: 'Images',
	extensions: 'gif,jpg,jpeg,bmp,png',
	mimeTypes: 'image/*'
	}
	});

文件被选择后	spuImgUploader.on('fileQueued',function (file) { console.log("用户增加文件:"+file.id+'||'+file.name); });
上传过程中，会反复触发的事件，每次 percentage 会变化。	spuImgUploader.on('uploadProgress', function(file, percentage) { console.log("用户增加文件:"+file.id+'||'+file.name+'||'+ percentage); });
上传成功后触发的事件	spuImgUploader.on('uploadSuccess', function(file ,response) { console.log("上传完成： "+file.id+"||"+response._raw); });

点击上传后，要执行的动作	$('#spuImgUploadBtn').click(function(){
	console.log("开始上传");
	if(spuImgUploader.getFiles().length<=0){
	$.messager().alert('警告','没有需要上传的文件','warning');
	return;
	}
	spuImgUploader.upload();
	});

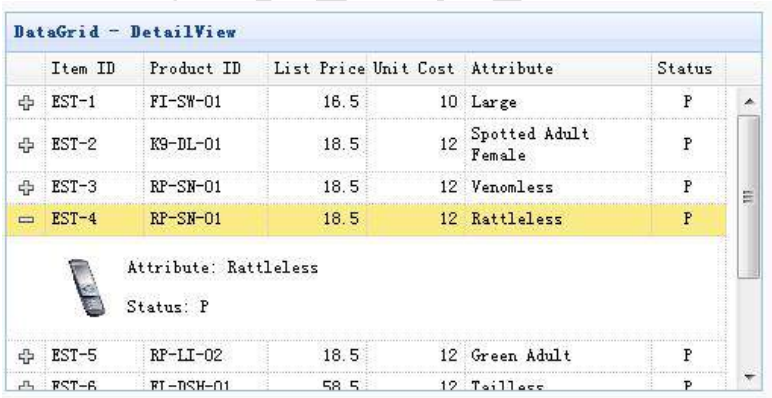
后台代码（临时测试）

```
@RestController
public class FileUploadController {

    @RequestMapping(value = "fileUpload",method = RequestMethod.POST)
    public String fileUpload(@RequestParam("file") MultipartFile[] files){
        if(files.length!=0){
            System.out.println("multipartFile = " + files[0].getName()+"|"+files[0].getSize());
        }
        return
"https://m.360buyimg.com/babel/jfs/t5137/20/1794970752/352145/d56e4e94/591417dcN4fe5ef33.jpg";
    }
}
```

实际业务代码见脑图

3.3 解决上传后的预览，通过伸缩行实现

利用 datagridview 。 这是 easyui 的一个扩展空间。



使用方式，把 datagridview.js 拷贝到，并且在表格中加入如下标红部分

```
$(function(){
                $('#tt').datagrid({
                    title:'DataGrid - DetailView',
                    width:500,
                    height:250,
                    remoteSort:false,
                    singleSelect:true,
                    nowrap:false,
```

```
                fitColumns:true,
                url:'./datagrid_data.json',
                columns:[[
                    {field:'itemid',title:'Item ID',width:80},
                    {field:'productid',title:'Product ID',width:100,sortable:true},
                    {field:'listprice',title:'List Price',width:80,align:'right',sortable:true},
                    {field:'unitcost',title:'Unit Cost',width:80,align:'right',sortable:true},
                    {field:'attr1',title:'Attribute',width:150,sortable:true},
                    {field:'status',title:'Status',width:60,align:'center'}
                ]],
                view: detailview,
                detailFormatter: function(rowIndex, rowData){
                    return '<table><tr>' +
                            '<td rowspan=2 style="border:0"><img src="images/' + rowData.itemid +
'.png" style="height:50px;"></td>' +
                            '<td style="border:0">' +
                            '<p>Attribute: ' + rowData.attr1 + '</p>' +
                            '<p>Status: ' + rowData.status + '</p>' +
                            '</td>' +
                            '</tr></table>';
                }
            });
        });
```

4、文件服务器

现在咱们实现了文件从客户端提交，并展示的功能。服务器端要做的就是接收文件流，保存起来，并且返回给客户端文件的访问地址。

传统的用 io 流保存到 web 服务器本地的方式，可以直接用当前 web 服务的路径+图片名称来访问。

但是类似于商品图片这种海量级文件，光靠 web 服务器的硬盘是无法满足的。

另外如果，web 服务器是集群的那么 A 服务器是没法访问 B 服务器的本地文件的。

所以需要把文件服务单独管理起来，成为文件服务器。

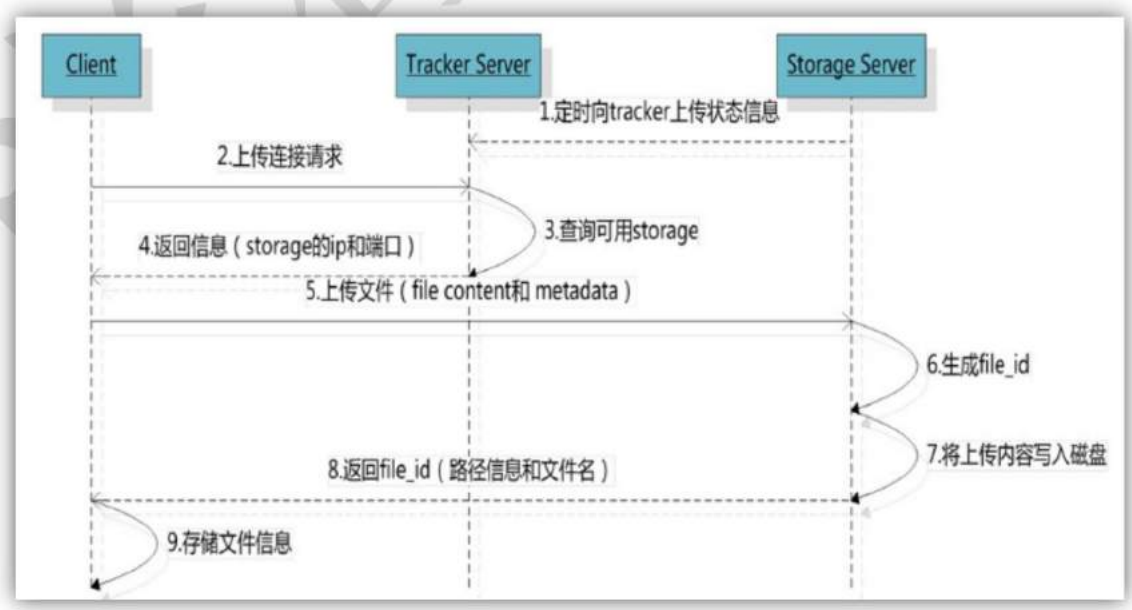
实现方式就是 nginx+FastDFS

4.1 FastDFS 介绍

FastDFS 是一个由 C 语言实现的开源轻量级分布式文件系统，作者余庆(happyfish100)，支持 Linux、FreeBSD、AID 等 Unix 系统，解决了大数据存储和读写负载均衡等问题，适合存储 4KB~500MB 之间的小文件，如图片网站、短视频网站、文档、app 下载站等，UC、京东、支付宝、迅雷、酷狗等都有使用。

该软件作者是阿里巴巴大牛、chinaUnix 版主**余庆**个人独立开发的。

4.2 FastDFS 上传下载的流程



4.3 安装步骤参见《FastDFS 安装说明》

4.4 利用 Java 客户端调用 FastDFS

服务器安装完毕后，咱们通过 Java 调用 fastdfs

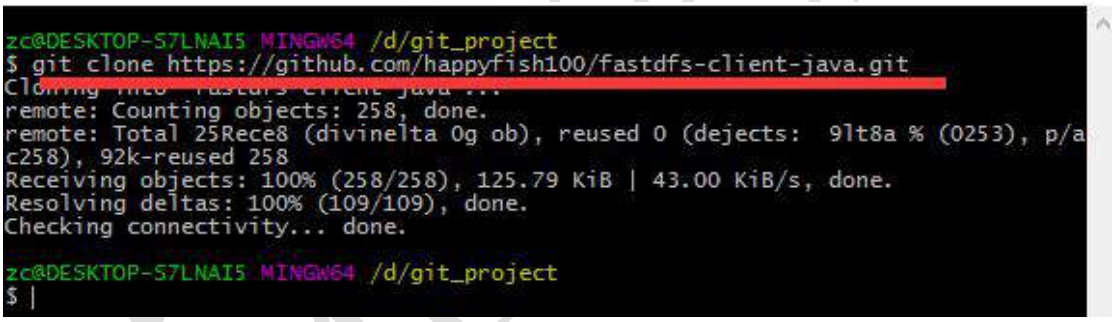
加载 Maven 依赖

fastdfs 没有在中心仓库中提供获取的依赖坐标。

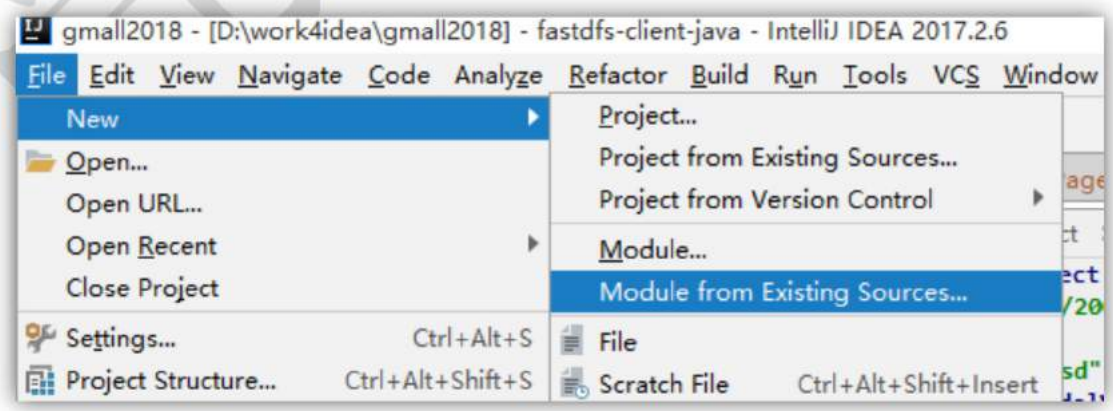
只能自己通过源码方式编译，打好 jar 包，安装到本地仓库。

官方仓库地址：

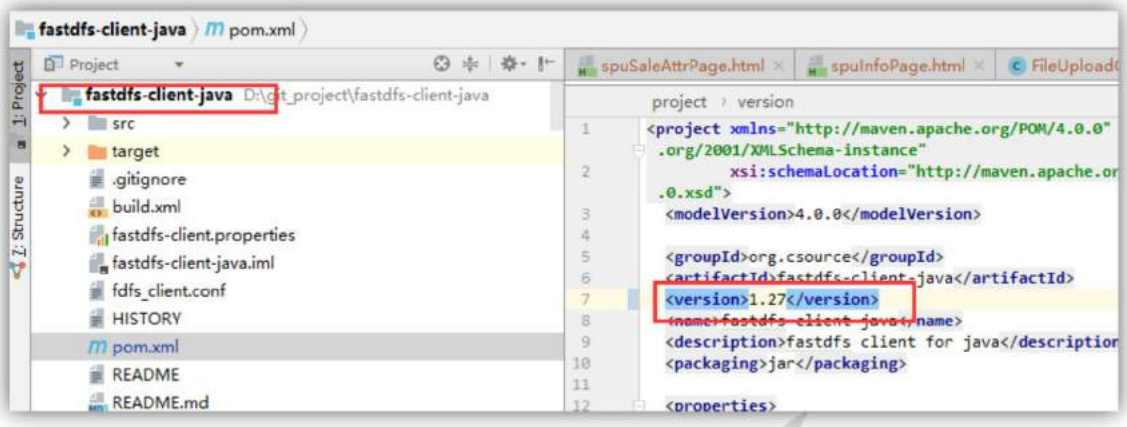
https://github.com/happyfish100/fastdfs-client-java



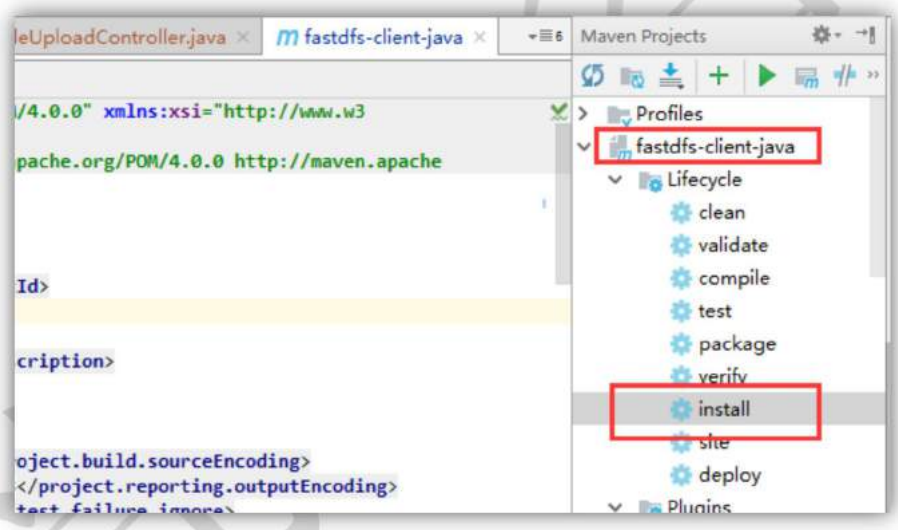
直接用 idea 直接把这个源码作为模块导入工程



别的不用改，只把 pom.xml 中的版本改成 1.27。



然后右边 执行 install 就好了



安装好了 ， 别的模块就可以直接使用这个坐标了。

```
<groupId>org.csource</groupId>
<artifactId>fastdfs-client-java</artifactId>
<version>1.27</version>
```

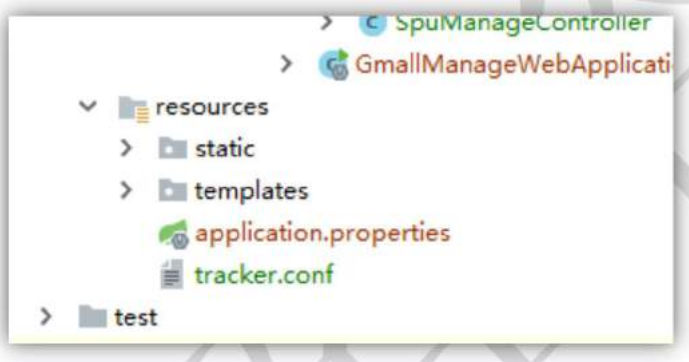
而这个 fastdfs-client-java 模块可以从 idea 中删除。

然后可以进行一下**上传的测试**

```
@Test
public void textFileUpload() throws IOException, MyException {
    String file = this.getClass().getResource("/tracker.conf").getFile();
    ClientGlobal.init(file);
    TrackerClient trackerClient=new TrackerClient();
```

```
        TrackerServer trackerServer=trackerClient.getConnection();
        StorageClient storageClient=new StorageClient(trackerServer,null);
        String orginalFilename="e://victor.jpg";
        String[] upload_file = storageClient.upload_file(orginalFilename, "jpg", null);
        for (int i = 0; i < upload_file.length; i++) {
            String s = upload_file[i];
            System.out.println("s = " + s);
        }

}
```

加入 tracker.conf 文件




```
tracker_server=file.server.com:22122

# 连接超时时间，针对 socket 套接字函数 connect，默认为 30 秒
connect_timeout=30000

# 网络通讯超时时间，默认是 60 秒
network_timeout=60000
```

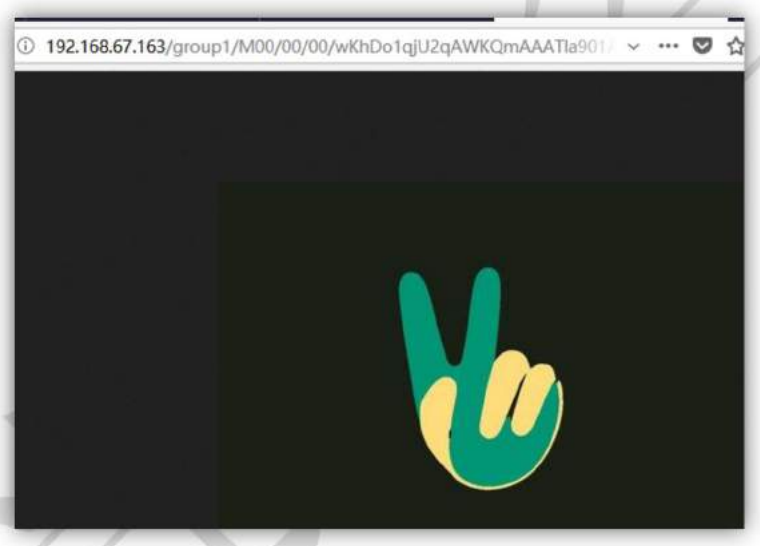
打印结果

```
"C:\Program Files\Java\jdk1.8.0_74\bin\java" ...
s = group1
s = M00/00/00/wKhDo1qjU2qAWKQmAAATla901AQ534.jpg

Process finished with exit code 0
```

这个打印结果实际上就是我们访问的路径，加上服务器地址我们可以拼接成一个字符串

```
http://192.168.67.163/group1/M00/00/00/wKhDo1qjU2qAWKQmAAATla901AQ534.jpg
```

直接放到浏览器去访问



上传成功！

对接到业务模块中

在修改 FileUploadController 的方法

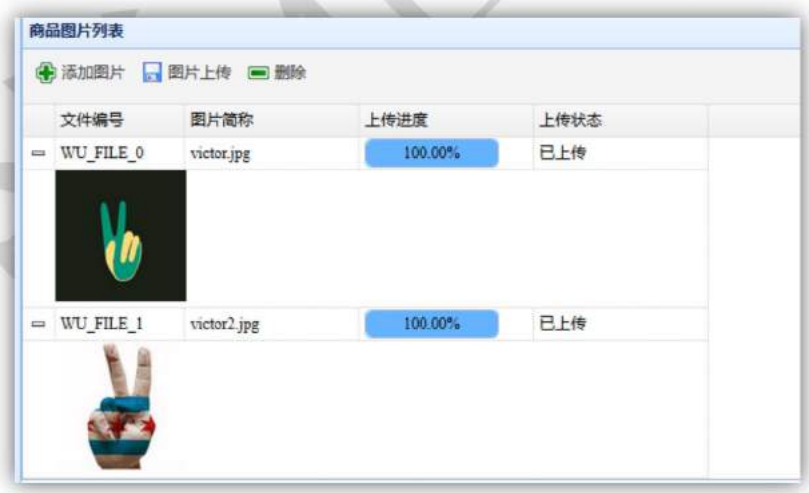
```
@Value("${fileServer.url}")
String fileUrl;



@RequestMapping(value = "fileUpload",method = RequestMethod.POST)
public String fileUpload(@RequestParam("file") MultipartFile file) throws IOException, MyException {
    String imgUrl=fileUrl;
    if(file!=null){
```

```
            System.out.println("multipartFile = " + file.getName()+"|"+file.getSize());

                String configFile = this.getClass().getResource("/tracker.conf").getFile();
                ClientGlobal.init(configFile);
                TrackerClient trackerClient=new TrackerClient();
                TrackerServer trackerServer=trackerClient.getConnection();
                StorageClient storageClient=new StorageClient(trackerServer,null);
                String filename=       file.getOriginalFilename();
                String extName = StringUtils.substringAfterLast(filename, ".");

                String[] upload_file = storageClient.upload_file(file.getBytes(), extName, null);
                imgUrl=fileUrl ;
                for (int i = 0; i < upload_file.length; i++) {
                    String path = upload_file[i];
                    imgUrl+="/"+path;
                }

            }

        return imgUrl;
    }
```

利用@Value 标签可以引用 application.properties 中的值

```
fileServer.url=http://file.server.com
```

测试结果：



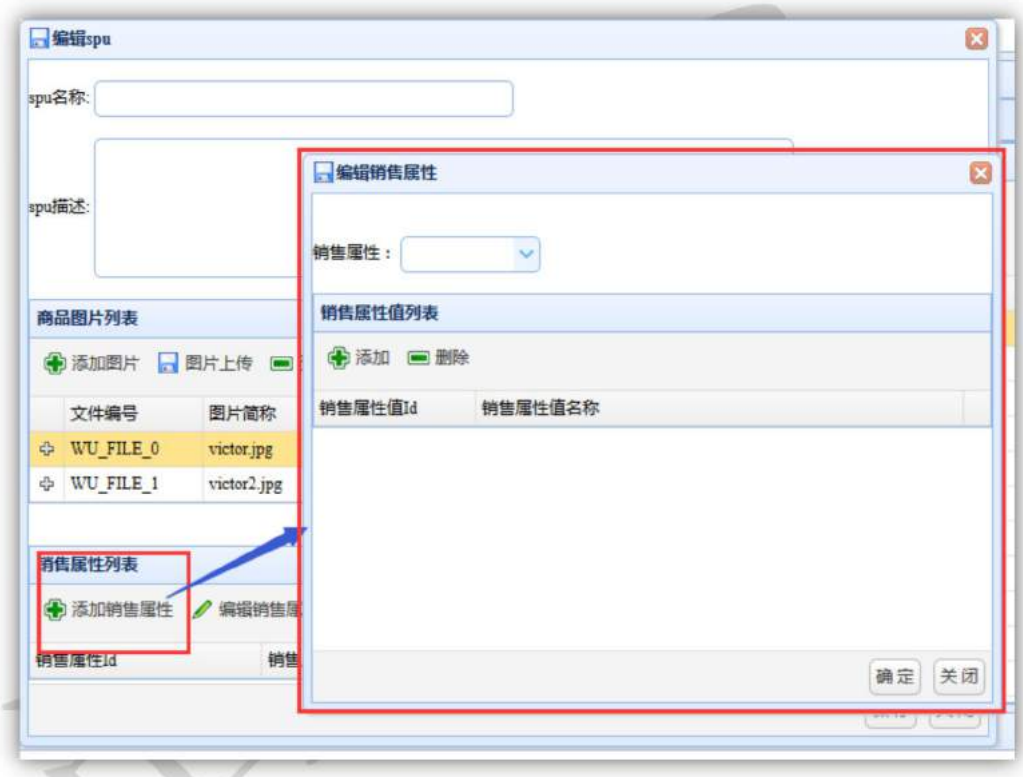
至此我们解决了文件上传的功能。

5 销售属性 ----两层多对多的关系

一个 spu 对应多个销售属性，但是每个销售属性又对应了多个属性值。

在 spu 的详情页面销售属性已经通过表格方式展现了一对多的关系，但是每一行代表的属性又有多个属性值。

只能针对每一行再弹出一个表格来增加多个属性。



但是跟之前弹出框保存有所不同的是，弹出框打开，软后录入数值，点击右下角确定的时候，不能够将数据直接保存到数据库中而是要把这个子弹出框的数据保存到销售属性表格控件中。只有当整个 spu 都录入完成，点击保存时才真的去写入数据库。

录入属性值信息确认后保存

```
function saveSpuSaleAttr() {

    var spuSaleAttrValueJson= $('#spuSaleAttrValueDg').datagrid('getData');
    var saleAttrId=$('#saleAttrSelect').combobox("getValue");
    var saleAttrName=$('#saleAttrSelect').combobox("getText");

     //
     var rowIndex = $("#spuSaleAttrDg").datagrid("getRowIndex",saleAttrId);
```

```
    console.log("delete rowIndex:"+rowIndex);
    if(rowIndex>=0){
        $("#spuSaleAttrDg").datagrid("deleteRow",rowIndex);
    }

$("#spuSaleAttrDg").datagrid("appendRow",{saleAttrId:saleAttrId,saleAttrName:saleAttrName,spuSaleAttrValueJson:spuSaleAttrValueJson});

    $("#spuSaleAttr_dlg").dialog("close");

}
```

其中 datagrid('getData') 是把整个 datagrid 的数据提取成一个 Json。

```
var spuSaleAttrValueJson= $('#spuSaleAttrValueDg').datagrid('getData');
```

这个 Json 的格式如下

```
{ total:2,

  row:[ {id:xxx,name:xxxx} ,  {id:xxx,name:xxxxx},.....  ]

}
```

但是这是一个 Json 对象，不是一个 Json 字符串。

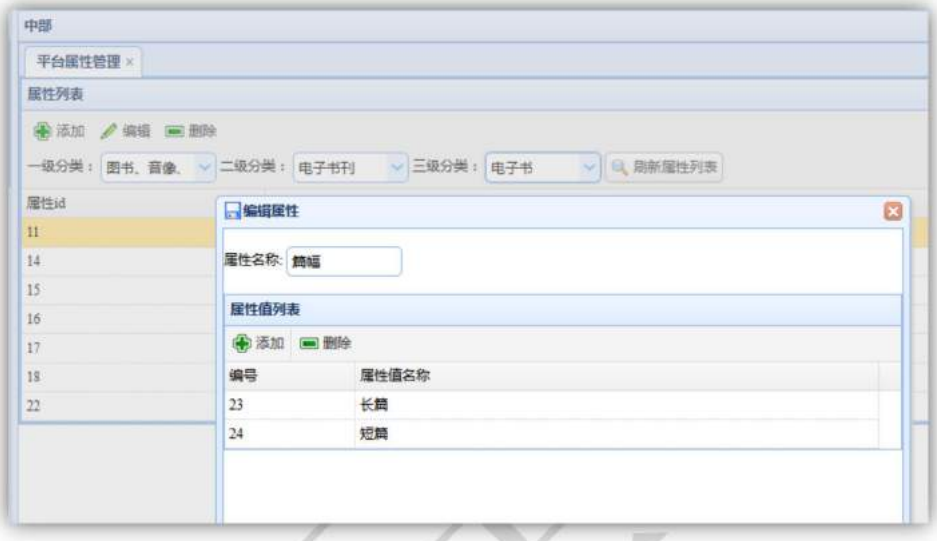
那么如果保存到了控件的一个暂存列中。

```
function initSpuSaleAttrListDatagrid(spuInfo){
    console.log("初始化销售属性表格:"+spuInfo);
    var spuSaleAttrDg=$('#spuSaleAttrDg').datagrid({url:""});
     spuSaleAttrDg =$('#spuSaleAttrDg').datagrid('loadData', { total: 0, rows: [] });

    spuSaleAttrDg.datagrid({
        idField: 'saleAttrId',
        columns:[[
            { field:'id',title:'id',hidden:true },
            { field:'saleAttrId',title:'销售属性Id',width:'35%'},
            { field:'saleAttrName',title:'销售属性名称',width:'45%'},
            { field:'spuSaleAttrValueJson',title:'销售属性值暂存',hidden:true, width:'20%'}
        ]]
    });

    //有初始数据说明是编辑状态，要加载数据
    if(spuInfo){
        spuSaleAttrDg.datagrid({url:"spuSaleAttrList?spuId="+spuInfo.id});
    }
}

</script>
```

那么当以后，想要编辑该属性的时候，利用 loadData 把原数据加载回来就可以了。

```
console.log("spuSaleAttr:"+ JSON.stringify(spuSaleAttr.saleAttrValue)) ;
 if(spuSaleAttr&&spuSaleAttr.spuSaleAttrValueJson&&spuSaleAttr.spuSaleAttrValueJson!=''){
      console.log("加载暂存");
```

```
        spuSaleAttrValueDg.datagrid("loadData",spuSaleAttr.spuSaleAttrValueJson);
}
```



6 页面的整体保存

详见脑图中的代码

谷粒商城

版本：V 1.0

www.atguigu.com

一、业务介绍

1 SPU 与 SKU

SPU(Standard Product Unit)：标准化产品单元。是商品信息聚合的最小单位，是一组可复用、易检索的标准化信息的集合，该集合描述了一个产品的特性。

SKU=Stock Keeping Unit（库存量单位）。即库存进出计量的基本单元，可以是以件，盒，托盘等为单位。SKU 这是对于大型连锁超市 DC（配送中心）物流管理的一个必要的方法。现在已经被引申为产品统一编号的简称，每种产品均对应有唯一的 SKU 号。

比如，咱们购买一台 iPhoneX 手机，iPhoneX 手机就是一个 SPU，但是你购买的时候，不可能是以 iPhoneX 手机为单位买的，商家也不可能以 iPhoneX 为单位记录库存。必须要以什么颜色什么版本的 iPhoneX 为单位。比如，你购买的是一台银色、128G 内存的、支持联通网络的 iPhoneX ，商家也会以这个单位来记录库存数。那这个更细致的单位就叫库存单元（SKU）。

销售属性与平台属性

销售属性，就是商品详情页右边，可以通过销售属性来定位一组 spu 下的哪款 sku。可以让当前的商品详情页，跳转到自己的“兄弟”商品。

一般每种商品的销售属性不会太多，大约 1-4 种。整个电商的销售属性种类也不会太多，大概 10 种以内。比如：颜色、尺寸、版本、套装等等。不同销售属性的组合也就构成了一个 spu 下多个 sku 的结构。

平台属性， 就是之前分类下面，辅助搜索的，类似于条件的属性。

销售属性与平台属性各自独立。一个 SPU 会决定一个商品都有哪些销售属性，比如 iPhonx 会有颜色、版本、内存的销售属性，某个 T 恤衫只有尺寸这个销售属性。

而某个商品有什么平台属性，由他的 3 级分类决定。比如笔记本包括：运行内存、cpu、显卡、硬盘、屏幕尺寸等等。

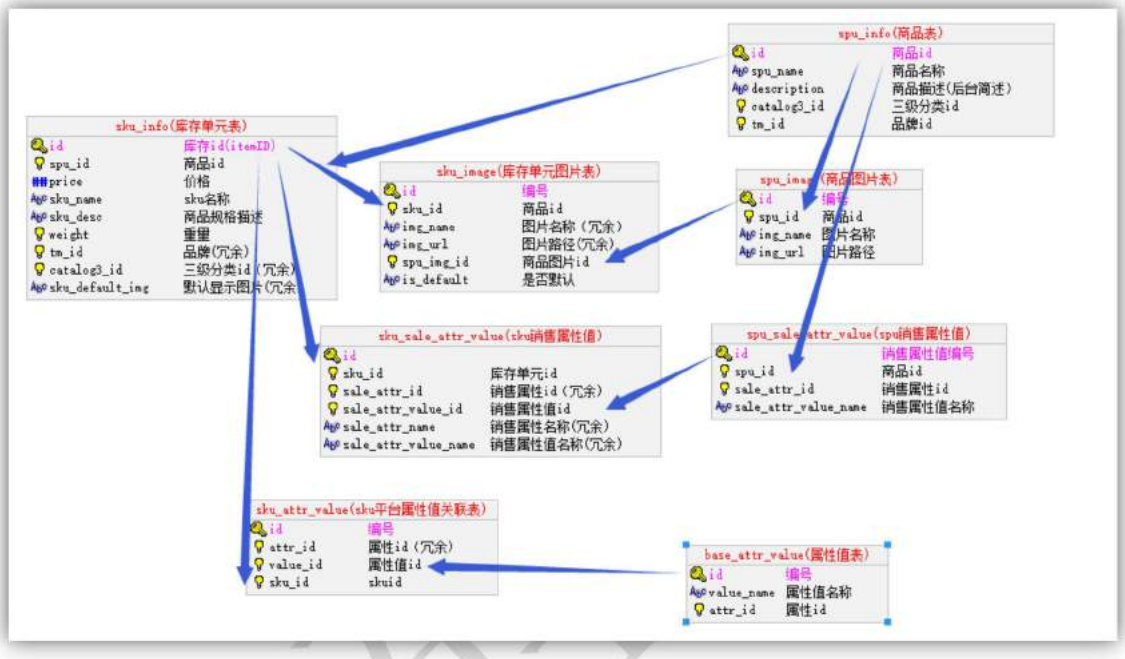
SKU 与 SPU 的图片资源

另外同一个 SPU 下的 SKU 可以共用一些资源，比如商品图片，海报等等。毕竟同一种商品，大部分图片都是共用的只有因为颜色尺寸等，很少的差别。那么一般来说商品图片都是在新增 SPU 时上传的，在新增 SKU 时从该 SPU 已上传的图片中选择。

而海报几乎是所有 SPU 下的 SKU 都一样。

2 数据库表结构

根据以上的需求，以此将 SPU 关联的数据库表结构设计为如下：



数据示例：

sku_info(商品表)

库存单元编号(sku_id)	sku_name	价格
33001	iPhoneX 土豪金 移动版 256G	7988
33002	iPhoneX 土豪金 联通版 256G	7888
33003	iPhoneX 太空银 联通版 64G	6888
33004	iPhoneX 太空银 移动版 64G	6988

spu销售属性值

id	spu_id	sale_attr_id	sale_att_value_name
88801	6110	101	太空银
88802	6110	101	土豪金
88803	6110	103	移动
88804	6110	103	联通
88805	6110	104	64G
88806	6110	104	256G

sku销售属性值

id	skuid	sale_attr_value_id	spu_id	sale_attr_id	sale_att_value_name
880001	33001	88802	6110	101	土豪金
880002	33001	88803	6110	103	移动
880003	33001	88806	6110	104	256G
880004	33002	88802	6110	101	土豪金
880005	33002	88804	6110	103	联通
880006	33002	88806	6110	104	256G
880007	33003	88801	6110	101	太空银
880008	33003	88804	6110	103	联通
880009	33003	88805	6110	104	64G
880010	33004	88801	6110	101	太空银
880011	33004	88803	6110	103	移动
880012	33004	88805	6110	104	64G

sku_info(商品表)

库存单元编号(sku_id)	sku_name	价格
33001	iPhoneX 土豪金 移动版 256G	7988
33002	iPhoneX 土豪金 联通版 256G	7888
33003	iPhoneX 太空银 联通版 64G	6888
33004	iPhoneX 太空银 移动版 64G	6988

平台属性表

id	attr_name	catalog3_id
301	内存	30
302	屏幕尺寸	30

平台属性值表

id	value_name	attr_id
3301	32	301
3302	64	301
3303	128	301
3304	256	301
3305	4寸以下	302
3306	4.5-5.0寸	302
3307	5.1-5.5寸	302
3308	5.6以上	302

sku平台属性值

id	skuid	attr_id	value_id
6501	33001	301	3304
6502	33002	301	3304
6503	33003	301	3302
6504	33004	301	3302
6505	33001	302	3307
6506	33002	302	3307
6507	33003	302	3307
6508	33004	302	3307

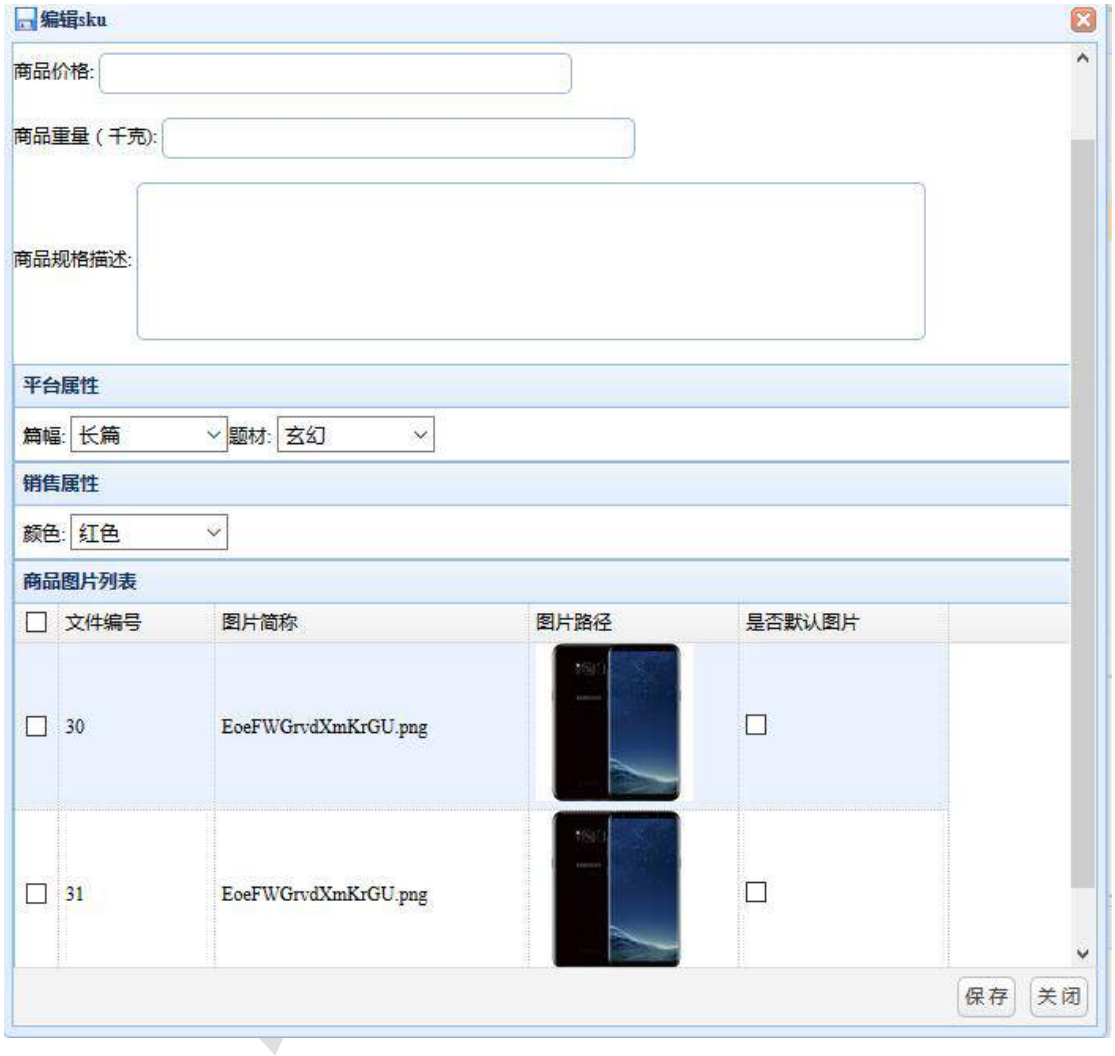
sku_info(商品表)			
库存单元编号(sku_id)	sku_name	价格	默认图片
33001	iPhoneX 土豪金 移动版 256G	7988	http://xxxx/xxx/AA.jpg
33002	iPhoneX 土豪金 联通版 256G	7888	http://xxxx/xxx/AA.jpg
33003	iPhoneX 太空银 联通版 64G	6888	http://xxxx/xxx/CC.jpg
33004	iPhoneX 太空银 移动版 64G	6988	http://xxxx/xxx/CC.jpg

spu_image 图片表			
id	spu_id	img_name	img_url
77701	6110	太空银正面	http://xxxx/xxx/AA.jpg
77702	6110	太空银侧面	http://xxxx/xxx/BB.jpg
77703	6110	土豪金正面	http://xxxx/xxx/CC.jpg
77704	6110	土豪金反面	http://xxxx/xxx/DD.jpg

sku_image 图片表				
id	sku_id	spu_img_id	img_name	img_url
7600001	33001	77703	土豪金正面图	http://xxxx/xxx/xx.jpg
7600002	33001	77704	土豪金反面图	http://xxxx/xxx/xx.jpg
7600003	33002	77703	土豪金正面图	http://xxxx/xxx/xx.jpg
7600004	33002	77704	土豪金反面图	http://xxxx/xxx/xx.jpg
7600005	33003	77701	太空银正面图	http://xxxx/xxx/xx.jpg
7600006	33003	77702	太空银侧面图	http://xxxx/xxx/xx.jpg
7600007	33004	77701	太空银正面图	http://xxxx/xxx/xx.jpg
7600008	33004	77702	太空银侧面图	http://xxxx/xxx/xx.jpg

二 开发增加功能

1 页面



2 难点分析

1、要动态生成平台属性和销售属性的下拉菜单

2、要生成图片列表以供用户选择

3 解决动态生成下拉菜单：

用 ajax 根据 spuid 查询销售属性及销售属性值的列表数据。 然后通过 js 把数据展开成 html，变为多下拉菜单。

4 解决生成图片列表

从后台查询数据库绑定表格控件。

注意在表格控件中的两种 checkbox 的使用。

代码见思维导图

5 保存

没有技术难点，细致的工作，利用 jquery 找到页面各个元素组合成一个大 Json，利用 ajax 提交到后台。

商品详情页

版本：V 1.0

www.atguigu.com

一　业务介绍

商品详情页，简单说就是以购物者的角度展现一个 sku 的详情信息。

这个页面不同于传统的 crud 的详情页，使用者并不是管理员，需要对信息进行查删改查，取而代之的是点击购买、放入购物车、切换颜色等等。

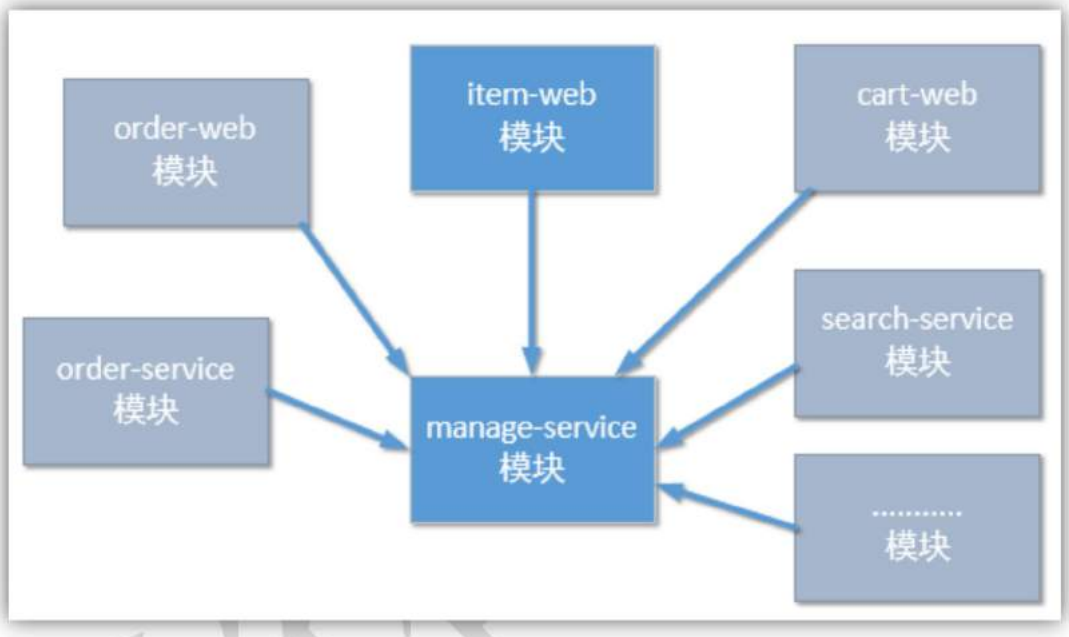
另外一个特点就是该页面的高访问量，虽然只是一个查询操作，但是由于频繁的访问所以我们必须对其性能进行最大程度的优化。

二 难点分析

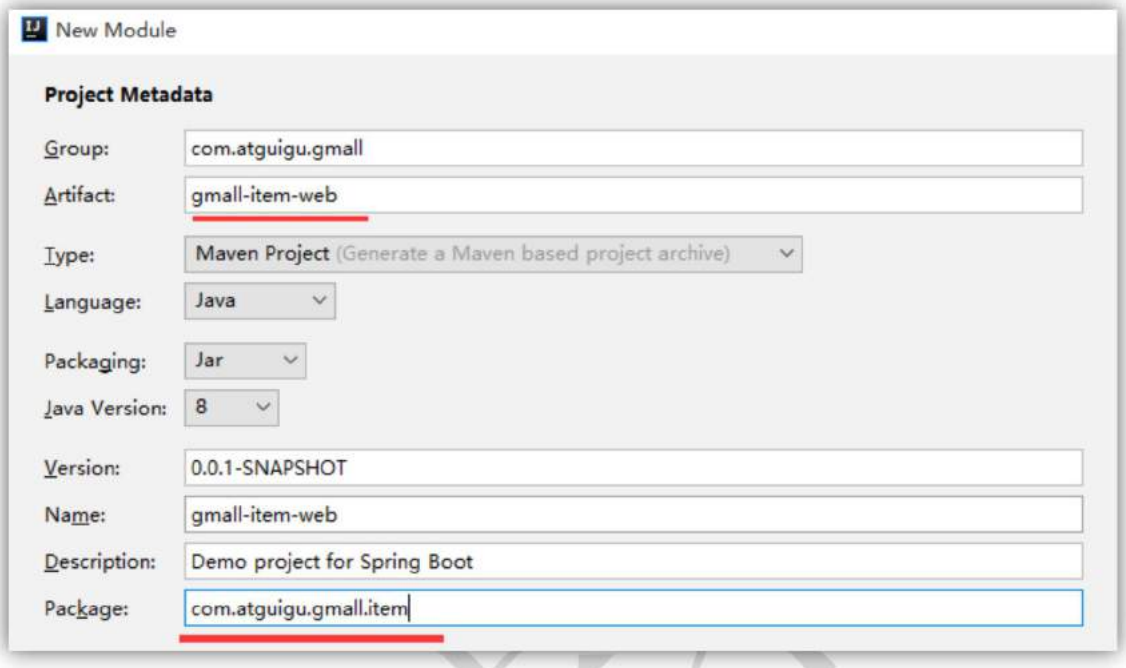
1　光从功能角度上来说，并没有太多难点，唯一实现起来麻烦的就是用户对于不同销售属性的切换操作。

2 从性能角度来看，需要最大程度的提升页面的访问速度。

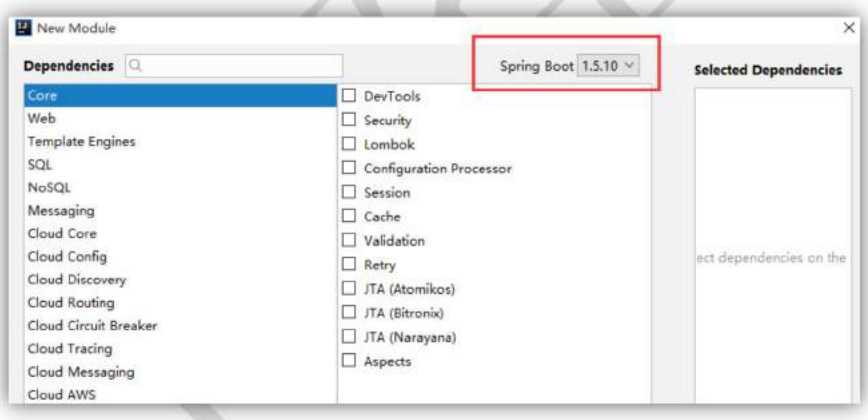
三 功能开发

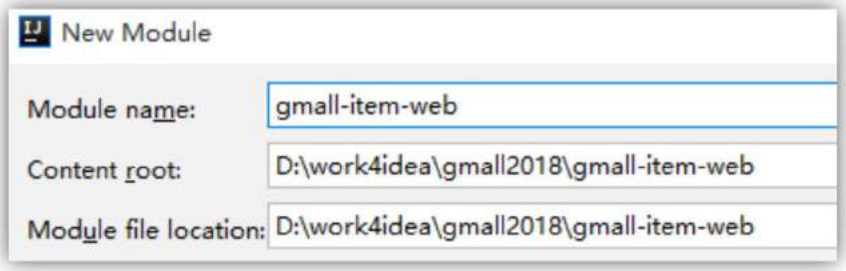


详情页功能，只增加一个 web 模块，后台调用商品管理的模块 manage-service item-web 模块负责前端的页面渲染和控制层(controller)。



依赖包全都不选，SpringBoot 版本选 1.5.10



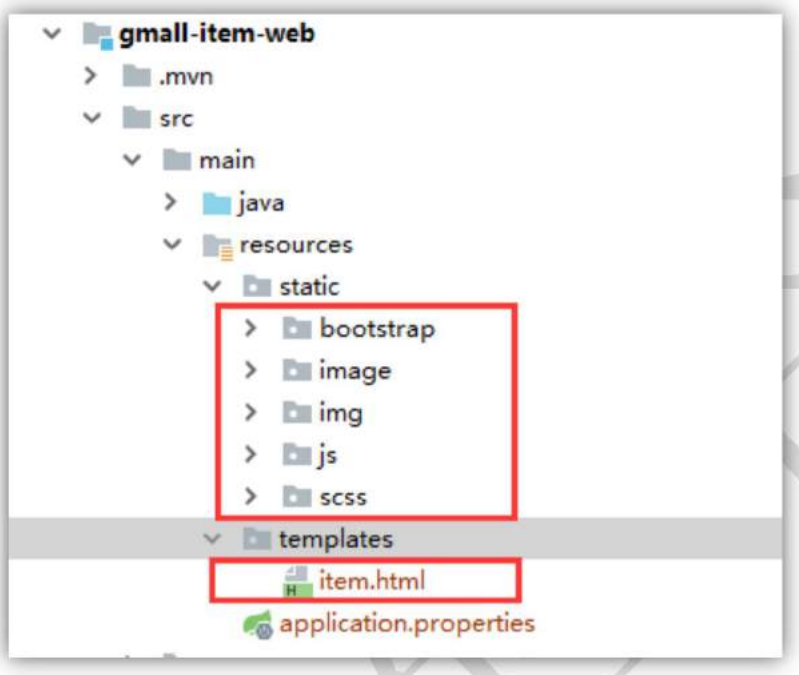


pom.xml

```
<?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.atguigu.gmall</groupId>
    <artifactId>gmall-item-web</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>gmall-item-web</name>
    <description>Demo project for Spring Boot</description>

    <parent>
        <groupId>com.atguigu.gmall</groupId>
        <artifactId>gmall-parent</artifactId>
        <version>1.0-SNAPSHOT</version>
    </parent>


    <dependencies>

    <dependency>
        <groupId>com.atguigu.gmall</groupId>
        <artifactId>gmall-interface</artifactId>
        <version>1.0-SNAPSHOT</version>
    </dependency>

    <dependency>
        <groupId>com.atguigu.gmall</groupId>
        <artifactId>gmall-web-util</artifactId>
        <version>1.0-SNAPSHOT</version>
    </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>


</project>
```

搭建完成后。

首先导入前端页面



静态页资源全部拷贝到 static 目录中，如果没有该目录请手工创建

动态的 html 文件拷贝到 templates 目录中

编写 Controller 类（入口方法）

```
@Controller
public class ItemController {

    @RequestMapping("/{skuId}.html")
    public String getSkuInfo(@PathVariable("skuId") String skuId){
        return "item";
    }

}
```

application.properties

```
server.port=8084

spring.thymeleaf.cache=false

spring.thymeleaf.mode=LEGACYHTML5
```

修改 item.html 改为绝对路径

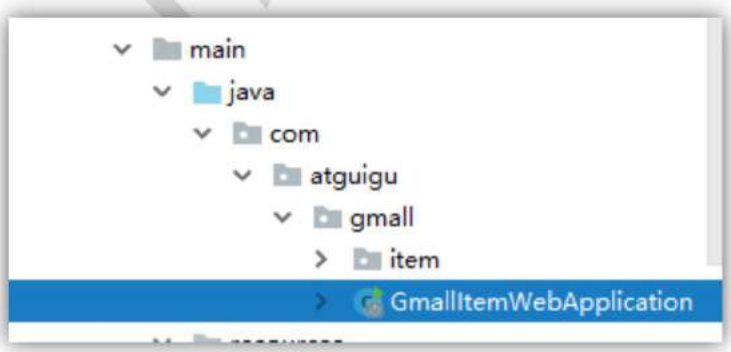
把 item.html 所有 src=”./ 替换成 sr=”/

<link>标签的路径也改为/开头

```
<head>
   <meta charset="UTF-8">
   <title></title>
   <link rel="stylesheet" type="text/css" href="/scss/shop.css" />
   <link rel="stylesheet" type="text/css" href="/scss/main.css"/>
   <link rel="stylesheet" href="/scss/header.css" />
   <link rel="stylesheet" type="text/css"
href="/bootstrap/css/bootstrap.css"/>
</head>
```

把启动类 GmallItemWebApplication 提到和 item 平级的目录中。

或者增加@ComponentScan(basePackages = "com.atguigu.gmall")



然后启动服务

启动测试

可以看到商品详情页的静态页面。

后台实现在 gmall-item-web 模块中

增加 ItemController

```
@RequestMapping("/{skuId}.html")
public String getSkuInfo(@PathVariable("skuId") String skuId, Model model){
   SkuInfo skuInfo = manageService.getSkuInfo(skuId);
   model.addAttribute("skuInfo",skuInfo);
}
```

gmall-manage-service 中增加

后台实现类

```
public SkuInfo getSkuInfo(String skuId){

      SkuInfo skuInfo = skuInfoMapper.selectByPrimaryKey(skuId);
      if(skuInfo==null){
            return null;
      }
      SkuImage skuImage=new SkuImage();
      skuImage.setSkuId(skuId);
      List<SkuImage> skuImageList = skuImageMapper.select(skuImage);
      skuInfo.setSkuImageList(skuImageList);
```

```
        SkuSaleAttrValue skuSaleAttrValue=new SkuSaleAttrValue();
        skuSaleAttrValue.setSkuId(skuId);
        List<SkuSaleAttrValue> skuSaleAttrValueList = skuSaleAttrValueMapper.select(skuSaleAttrValue);
        skuInfo.setSkuSaleAttrValueList(skuSaleAttrValueList);

        return    skuInfo;
}
```

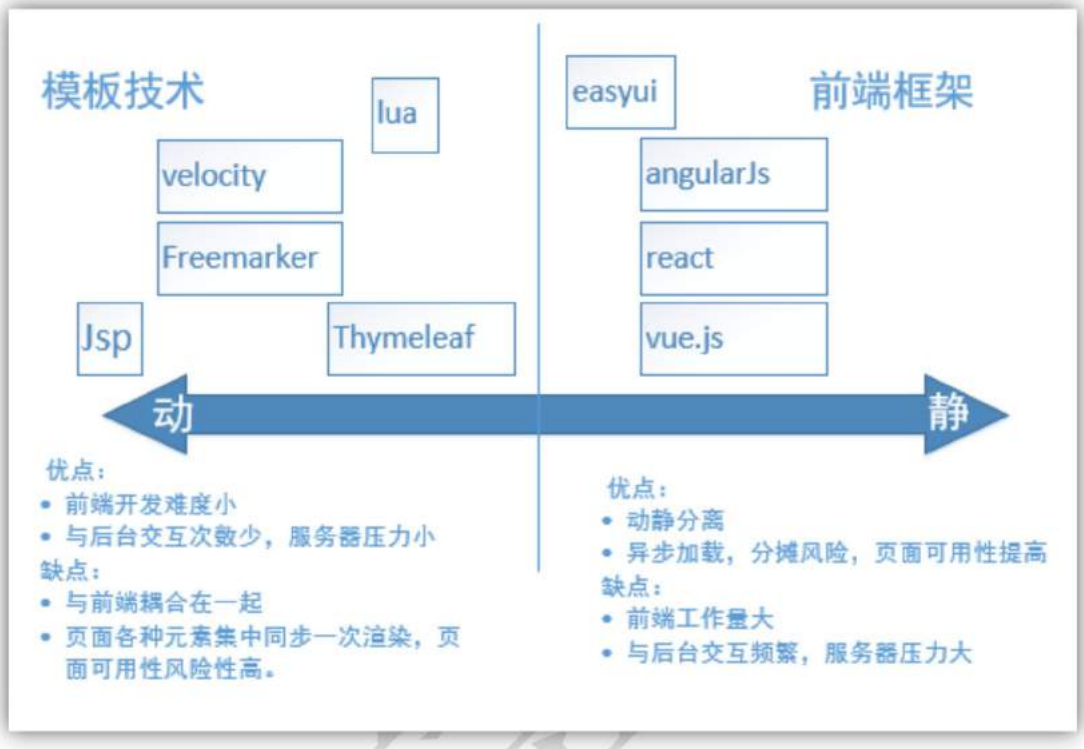
四 Thymeleaf

1 模板技术

把页面中的静态数据替换成从后台数据库中的数据。这种操作用 jsp 就可以实现。但是 Springboot 的架构不推荐使用 Jsp，而且支持也不好，所以如果你是用 springboot 的话，一般使用 Freemarker 或者 Thymeleaf。

而官方也是推荐使用 Thymeleaf。

关于与前端有关的技术的比较



2 Themeleaf 简介

Thymeleaf 的主要目标是提供一个优雅和高度可维护的创建模板的方式。为了实现这一点，它建立在*自然*模板的概念上，将其逻辑注入到模板文件中，不会影响模板被用作设计原型。这改善了设计的沟通，弥合了设计和开发团队之间的差距。

比 Jsp 和 Freemarker 的优势，一般的模板技术都会在页面加各种表达式、标签甚至是 java 代码，而这些都必须要经过后台服务器的渲染才能打开。

但如果前端开发人员做页面调整，双击打开某个 jsp 或者 ftl 来查看效果，基本上是打不开的。

那么 Thymeleaf 的优势就出来了，因为 Thymeleaf 没有使用自定义的标签或语法，所有的模板语言都是扩展了标准 H5 标签的属性

比如

```
<div  th:text="${item.skuName} "></div>
```

它的效果和 Jsp 中的

```
<div>${item.skuName}</div>
```

渲染后效果一样，但是如果你直接用浏览器打开页面文件，H5 会把 th:text 这种不认识的属性忽略掉。效果就和 没有区别，所以对于前端调页面影响更新。以上只是举了一个例子，如果是循环、分支的判断效果更明显。

3 快速入门：

3.1 所有头文件

就行 Jsp 的<%@Page %>一样 ，Thymeleaf 的也要引入标签规范。不加这个虽然不影响程序运行，但是你的 idea 会认不出标签，不方便开发。

```
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:th="http://www.thymeleaf.org">
```

3.2 取出请求域中的值，即取得 request.attribute 中的值

```
<p th:text="${hello}">打底值</p>
```

3.3 循环

```
<table border="1">
    <tr th:each="skuImage:${skuInfo.skuImageList}">
        <td th:text="${skuImage.id}">
        </td>
        <td th:text="${skuImage.imgName}">
        </td>
    </tr>
```

```
</table>
```

3.4 判断

```
<td th:text="(${skuImage.id}=='10')?'是 10':'不是 10'">
</td>
<td th:if="${skuImage.id}=='10'" >123456</td>
```

3.5 取 session 中的属性

```
<div th:text="${session.userName}"> </div>
```

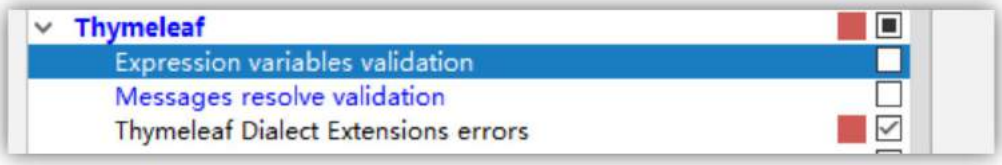
3.6 引用内嵌页

```
<div th:include="itemInner"/>
```

如果 idea 编辑器质疑你页面中的元素是否存在

```
<body>
<div th:text="${skuInfo.skuName}"></div>
<div th:text="${session.skuName}"> </div>
```

通常不准确，可以去掉验证，在 settings->editor->Inspections 中关掉验证。



五　开发详情页功能

名称：

```
<div class="box-name" th:text="${skuInfo.skuName}">
   华为 HUAWEI Mate 10 6GB+128GB 亮黑色 移动联通电信 4G 手机 双卡双待
</div>
```

价格：

```
<span th:text="${#numbers.formatDecimal(skuInfo.price,1,2)}">4499.00</span>
```

重量：

```
<li  th:text="${#numbers.formatDecimal(skuInfo.weight,1,2)}+' kg'"></li>
```

图片

```
<div class="box-lh-one">
   <ul>
      <li th:each="skuImage:${skuInfo.skuImageList}">
            <img th:src="${skuImage.imgUrl}"/>
      </li>
   </ul>
</div>
```

```
numbers.formatDecimal(<值>,<小数点左边的占位>,<小数点右边的保留位>)
```

六　销售属性的处理

1　思路：

1、查出该商品的 spu 的所有销售属性和属性值

2、标识出本商品对应的销售属性

3、点击其他销售属性值的组合，跳转到另外的 sku 页面

2 查询出 sku 对应 spu 的销售属性

第 1、2 条通过此 sql 实现

```
SELECT sa.id ,sa.spu_id, sa.sale_attr_name,sa.sale_attr_id,
sv.id sale_attr_value_id,
sv.sale_attr_value_name,
skv.sku_id,
IF(skv.sku_id IS NOT NULL,1,0)  is_check
FROM spu_sale_attr sa
INNER JOIN spu_sale_attr_value  sv ON  sa.spu_id=sv.spu_id AND sa.sale_attr_id=sv.sale_attr_id
LEFT JOIN sku_sale_attr_value skv ON skv.sale_attr_id= sa.sale_attr_id AND skv.sale_attr_value_id=sv.id
AND skv.sku_id=10
WHERE  sa.spu_id=24
ORDER BY sv.sale_attr_id,sv.id
```

此 sql 列出所有该 spu 的销售属性和属性值，并关联某 skuid 如果能关联上 is_check 设为 1，否则设为 0。

页面开发部分

```
<div class="box-attr-2 clear"  th:each="spuSaleAttr:${spuSaleAttrListCheckBySku}">
    <dl>
        <dt th:text="${spuSaleAttr.saleAttrName}">选择属性</dt>
        <dd                                          th:class="(${saleAttrValue.isCheck}=='1')?'redborder':''"
th:each="saleAttrValue:${spuSaleAttr.spuSaleAttrValueList}">
            <div th:value="${saleAttrValue.id}" th:text="${saleAttrValue.saleAttrValueName}">
                属性值
            </div>
        </dd>
    </dl>
</div>
```

其中 th:class 的设置，redborder 是一个自定义的样式类，主要是边框设红。如果 isCheck=1 标识当前这个 sku 的所拥有的属性值，所以锁定为红边框。

需要增加修改的代码文件清单

增 加 SpuSaleAttrMapper.xml 的方法	selectSpuSaleAttrListCheckBySku 注意双参数的处理
增加 SpuSaleAttrMapper 的方法	
增加 ManageServiceImpl 的方法	getSpuSaleAttrListCheckBySku
增加 ManageService 的方法	getSpuSaleAttrListCheckBySku
修改 SpuSaleAttrValue	增加 isCheck 属性

代码见代码清单

增加 ManageServiceImpl 的方法	getSpuSaleAttrListCheckBySku

```
@Override
public    List<SpuSaleAttr> getSpuSaleAttrListCheckBySku(String skuId,String spuId){

      List<SpuSaleAttr> spuSaleAttrList =
spuSaleAttrMapper.selectSpuSaleAttrListCheckBySku(Long.parseLong(skuId),Long.parseLong(spuId));
      return spuSaleAttrList;

}
```

增加 SpuSaleAttrMapper 的方法
public List<SpuSaleAttr>   selectSpuSaleAttrListCheckBySku(long skuId,long spuId);

增 加 SpuSaleAttrMapper.xml 的方法	selectSpuSaleAttrListCheckBySku 注意双参数的处理

```
<select    id ="selectSpuSaleAttrListCheckBySku"      resultMap="spuSaleAttrMap">
      SELECT sa.id ,sa.spu_id, sa.sale_attr_name,sa.sale_attr_id,
      sv.id sale_attr_value_id,
      sv.sale_attr_value_name,
      skv.sku_id,
      IF(skv.sku_id IS NOT NULL,1,0)    is_check
      FROM spu_sale_attr sa
      INNER JOIN spu_sale_attr_value    sv ON    sa.spu_id=sv.spu_id AND sa.sale_attr_id=sv.sale_attr_id
      LEFT JOIN sku_sale_attr_value skv ON skv.sale_attr_id= sa.sale_attr_id AND skv.sale_attr_value_id=sv.id AND
skv.sku_id=#{arg0}
      WHERE    sa.spu_id=#{arg1}
      ORDER BY sv.sale_attr_id,sv.id
</select>
```

修改 SpuSaleAttrValue	增加 isCheck 属性

修改 itemController	调用 getSpuSaleAttrListCheckBySku

```
List<SpuSaleAttr> spuSaleAttrListCheckBySku =
manageService.getSpuSaleAttrListCheckBySku(skuInfo.getId(),
skuInfo.getSpuId());
model.addAttribute("spuSaleAttrListCheckBySku",spuSaleAttrListCheckBySku);
```

修改 item.html	增加渲染的代码

```
<div class="box-attr-2 clear"    th:each="spuSaleAttr:${spuSaleAttrListCheckBySku}">
    <dl>
        <dt th:text="${spuSaleAttr.saleAttrName}">选择颜色</dt>
        <dd                                                th:class="(${saleAttrValue.isCheck}=='1')?'redborder':''"
th:each="saleAttrValue:${spuSaleAttr.spuSaleAttrValueList}">
            <div th:value="${saleAttrValue.id}" th:text="${saleAttrValue.saleAttrValueName}">
                摩卡金
            </div>
        </dd>
    </dl>
</div>
```

3 点击其他销售属性值的组合，跳转到另外的 sku 页面

实现思路：

1 、从页面中获得得所有选中的销售属性进行组合比如：

“属性值 1|属性值 2|属性值 3” 用这个字符串匹配一个对照表，来获得 skuId。并进行跳转，或者告知无货。

2、后台要生成一个“属性值 1|属性值 2|属性值 3：skuId”的一个 json 串以提供页面进行匹配。如 `valuesSku:{"46|50":"10","47|50":"13","48|49":"12","47|49":"11"}`

3、需要从后台数据库查询出该 spu 下的所有 skuId 和属性值关联关系。然后加工成如上的

Json 串。

实现：

skuSaleAttrValueMapper.xml 中 sql

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper SYSTEM "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.atguigu.gmall.manage.mapper.SkuSaleAttrValueMapper">
    <select  id ="selectSkuSaleAttrValueListBySpu" parameterType="long" resultMap="skuSaleAttrValueMap">
        SELECT  skv.*
        FROM sku_sale_attr_value skv
        INNER JOIN  sku_info  sk  ON skv.sku_id =sk.id
        where  sk.spu_id=#{spuId}
        ORDER BY skv.sku_id , skv.sale_attr_id
    </select>
    <resultMap  id="skuSaleAttrValueMap"  type="com.atguigu.gmall.bean.SkuSaleAttrValue" autoMapping="true">
        <result property="id" column="id" ></result>
    </resultMap>
</mapper>
```

要注意排序，方便后面整理。

实现类很简单：

```
public List<SkuSaleAttrValue> getSkuSaleAttrValueListBySpu(String spuId){
    List<SkuSaleAttrValue>  skuSaleAttrValueList  =
skuSaleAttrValueMapper.selectSkuSaleAttrValueListBySpu(Long.parseLong(spuId));
    return skuSaleAttrValueList;
}
```

难点是整理成咱们要求的 json 串，这个写在 controller 类。

```
List<SkuSaleAttrValue>  skuSaleAttrValueListBySpu  =
manageService.getSkuSaleAttrValueListBySpu(skuInfo.getSpuId());


//把列表变换成 valueid1|valueid2|valueid3 ：skuId  的 哈希表 用于在页面中定位查询
String valueIdsKey="";

Map<String,String> valuesSkuMap=new HashMap<>();

for (int i = 0; i < skuSaleAttrValueListBySpu.size(); i++) {
    SkuSaleAttrValue skuSaleAttrValue = skuSaleAttrValueListBySpu.get(i);
    if(valueIdsKey.length()!=0){
        valueIdsKey= valueIdsKey+"|";
    }
    valueIdsKey=valueIdsKey+skuSaleAttrValue.getSaleAttrValueId();

    if((i+1)==
```

```
skuSaleAttrValueListBySpu.size()||!skuSaleAttrValue.getSkuId().equals(skuSaleAttrValueListBySpu.get(i+1).getSkuId())  ){

            valuesSkuMap.put(valueIdsKey,skuSaleAttrValue.getSkuId());
            valueIdsKey="";
      }

}

//把 map 变成 json 串
String valuesSkuJson = JSON.toJSONString(valuesSkuMap);

model.addAttribute("valuesSkuJson",valuesSkuJson);
```

在 item 中增加隐藏域

```
<input id="valuesSku" type="hidden"  th:value= " ${valuesSkuJson}"  />
<input id="skuId" type="hidden"  th:value= " ${skuInfo.id}"  />
```

valuesSku 存储属性值与 skuid 的对照 json

skuId 存放当前商品的 skuId.

修改页面的 js

```
      //红边框
$(".box-attr-2 dd").click(function() {
    $(this).addClass("redborder").siblings("dd").removeClass("redborder");
            switchSkuId();
})
```

增加点击销售属性时触发切换 sku 方法

```
function switchSkuId() {
var redborderDivs = $(".redborder div");
var valueIdkeys="";
for(i=0;i<redborderDivs.length;i++){
                    var redborderDiv= redborderDivs.eq(i);
                    var attrValueId = redborderDiv.attr("value");
                    if(i>0){
                          valueIdkeys+="|";
    }
                    valueIdkeys+=attrValueId;
}
console.log("valueIdkeys:"+valueIdkeys);
             var valueIdSkuJson= $("#valueIdSkuJson").val();
             console.log("valueIdSkuJson:"+valueIdSkuJson);
            var skuSelfId= $("#skuId").val();
```

```
            var valueIdSku = JSON.parse(valueIdSkuJson);
            var skuIdTarget = valueIdSku[valueIdkeys];
            if(!skuIdTarget){
               $("#cartBtn").attr("class","box-btns-two-off");
                  $("#cartBtn").attr("canClick",'0');
                  $("#cartBtn").css("cursor",'not-allowed') ;
}else{
               if(skuSelfId!=skuIdTarget){
                     window.location.href="/"+skuIdTarget+".html";
                  }else{
                        $("#cartBtn").attr("class","box-btns-two");
                        $("#cartBtn").attr("canClick",'1');
                        $("#cartBtn").css("cursor",'pointer') ;
    }
 }

    }
 }
```

最后测试：

七 性能优化

1 思路：

虽然咱们实现了页面需要的功能，但是考虑到该页面是被用户高频访问的，所以性能必须进行尽可能的优化。

一般一个系统最大的性能瓶颈，就是数据库的 io 操作。从数据库入手也是调优性价比最高的切入点。

一般分为两个层面，一是提高数据库 sql 本身的性能，二是尽量避免直接查询数据库。

提高数据库本身的性能首先是优化 sql，包括：使用索引，减少不必要的大表关联次数，控制查询字段的行数和列数。另外当数据量巨大是可以考虑分库分表，以减轻单点压力。

这部分知识在 mysql 高级已有讲解，这里大家可以以详情页中的 sql 作为练习，尝试进行优化，这里不做赘述。

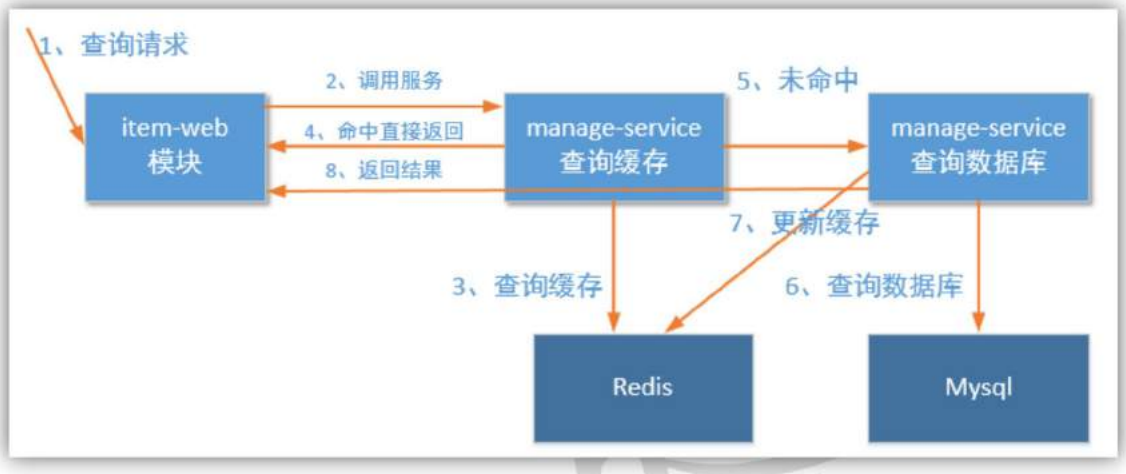
重点要讲的是另外一个层面：尽量避免直接查询数据库。

解决办法就是：缓存

缓存可以理解是数据库的一道保护伞，任何请求只要能在缓存中命中，都不会直接访问数据库。而缓存的处理性能是数据库 10-100 倍。

咱们就用 Redis 作为缓存系统进行优化。

结构图：



安装 Redis ： 略

2 整合 redis 到大工程中。

由于 redis 作为缓存数据库，要被多个项目使用，所以要制作一个通用的工具类，方便工程中的各个模块使用。

而主要使用 redis 的模块，都是后台服务的模块，xxx-service 工程。所以咱们把 redis 的工具类放到 service-util 模块中，这样所有的后台服务模块都可以使用 redis。

首先引入依赖包

```
<!-- https://mvnrepository.com/artifact/redis.clients/jedis -->
<dependency>
    <groupId>redis.clients</groupId>
    <artifactId>jedis</artifactId>
    <version>2.9.0</version>
</dependency>
```

分别按照之前的方式放到 parent 模块和 service-util 的 pom 文件中。

然后在 service-util 中创建两个类 RedisConfig 和 RedisUtil

RedisConfig 负责在 spring 容器启动时自动注入，而 RedisUtil 就是被注入的工具类以供其他模块调用。

RedisUtil

```
public class RedisUtil {

    private    JedisPool jedisPool;

    public void initPool(String host,int port ,int database){
        JedisPoolConfig poolConfig = new JedisPoolConfig();
        poolConfig.setMaxTotal(200);
        poolConfig.setMaxIdle(30);
        poolConfig.setBlockWhenExhausted(true);
        poolConfig.setMaxWaitMillis(10*1000);
        poolConfig.setTestOnBorrow(true);
        jedisPool=new JedisPool(poolConfig,host,port,20*1000);
    }

    public Jedis getJedis(){
        Jedis jedis = jedisPool.getResource();
        return jedis;
    }

}
```

RedisConfig

```
@Configuration
public class RedisConfig {

    //读取配置文件中的 redis 的 ip 地址
    @Value("${spring.redis.host:disabled}")
    private String host;

    @Value("${spring.redis.port:0}")
    private int port;

    @Value("${spring.redis.database:0}")
    private int database;

    @Bean
    public RedisUtil getRedisUtil(){
        if(host.equals("disabled")){
            return null;
        }
        RedisUtil redisUtil=new RedisUtil();
        redisUtil.initPool(host,port,database);
```

```
        return redisUtil;
    }

}
```

同时，任何模块想要调用 redis 都必须在 application.properties 配置，否则不会进行注入。

```
spring.redis.host=redis.server.com
spring.redis.port=6379
spring.redis.database=0
```

现在可以在 manage-service 中的 getSkuInfo()方法测试一下

```
try {
    Jedis jedis = redisUtil.getJedis();
    jedis.get("test","text_value" );
}catch (JedisConnectionException e){
    e.printStackTrace();
}
```

3 使用 redis 进行业务开发

开始开发先说明 redis key 的命名规范，由于 Redis 不像数据库表那样有结构，其所有的数据全靠 key 进行索引，所以 redis 数据的可读性，全依靠 key。

企业中最常用的方式就是：object:id:field

比如：sku:1314:info

user:1092:password

重构 getSkuInfo 方法

```
public SkuInfo getSkuInfo(String skuId){

        Jedis jedis = redisUtil.getJedis();
        String skuKey= RedisConst.sku_prefix+skuId+RedisConst.skuInfo_suffix;
        String skuInfoJson = jedis.get(skuKey);
        if(skuInfoJson!=null ){
```

```
            System.err.println( Thread.currentThread().getName()+"：命中缓存
"  );
            SkuInfo skuInfo = JSON.parseObject(skuInfoJson, SkuInfo.class);
            jedis.close();
            return skuInfo;
        }else{
              System.err.println( Thread.currentThread().getName()+"：未命中
缓存"  );

               System.err.println( Thread.currentThread().getName()+"：  查询
数据###################### ##" );
               SkuInfo skuInfoDB = getSkuInfoDB(skuId);
               String skuInfoJsonStr = JSON.toJSONString(skuInfoDB);

jedis.setex(skuKey,RedisConst.skuinfo_exp_sec,skuInfoJsonStr);
               System.err.println( Thread.currentThread().getName()+"：数据库
更新完毕################ #####" );
               jedis.close();
               return skuInfoDB;
        }
}
```

以上基本实现使用缓存的方案。

4 高并发时可能会出现的问题：

但在高并发环境下还有如下三个问题。

1、 如果 redis 宕机了，或者链接不上，怎么办？

2、 如果 redis 缓存在高峰期到期失效，在这个时刻请求会向雪崩一样，直接访问数据库如何处理？

3、 如果用户不停地查询一条不存在的数据，缓存没有，数据库也没有，那么会出现什么

情况，如何处理？

```
public SkuInfo getSkuInfo(String skuId){
  SkuInfo skuInfo = null;
  try {
        Jedis jedis = redisUtil.getJedis();
        String skuInfoKey = ManageConst.SKUKEY_PREFIX + skuId + ManageConst.SKUKEY_SUFFIX;
        String skuInfoJson = jedis.get(skuInfoKey);

        if (skuInfoJson == null || skuInfoJson.length() == 0) {
              System.err.println(Thread.currentThread().getName()+"缓存未命中！ ");
              String skuLockKey = ManageConst.SKUKEY_PREFIX + skuId + ManageConst.SKULOCK_SUFFIX;
              String lock = jedis.set(skuLockKey, "OK", "NX", "PX", ManageConst.SKULOCK_EXPIRE_PX);

              if ("OK".equals(lock) ){
                    System.err.println(Thread.currentThread().getName()+"获得分布式锁！ ");
                    skuInfo = getSkuInfoFromDB(skuId);
                    if(skuInfo==null){
                          jedis.setex(skuInfoKey, ManageConst.SKUKEY_TIMEOUT, "empty");
                          return null;
                    }


                    String skuInfoJsonNew = JSON.toJSONString(skuInfo);
                    jedis.setex(skuInfoKey, ManageConst.SKUKEY_TIMEOUT, skuInfoJsonNew);
                    jedis.close();
                    return skuInfo;
              }else{
                    System.err.println(Thread.currentThread().getName()+"未获得分布式锁，开始自旋！ ");
                    Thread.sleep(1000);
                    jedis.close();
                    return     getSkuInfo(    skuId);
              }

        } else if(skuInfoJson.equals("empty")){
              return null;
        } else {
              System.err.println(Thread.currentThread().getName()+"缓存已命中!!!!!!!!!!!!!!!!!!! ");
              skuInfo = JSON.parseObject(skuInfoJson, SkuInfo.class);
              jedis.close();
              return skuInfo;
        }

  }catch (JedisConnectionException e){
        e.printStackTrace();
  } catch (InterruptedException e) {
        e.printStackTrace();
  }

  return getSkuInfoFromDB(skuId);
}
```

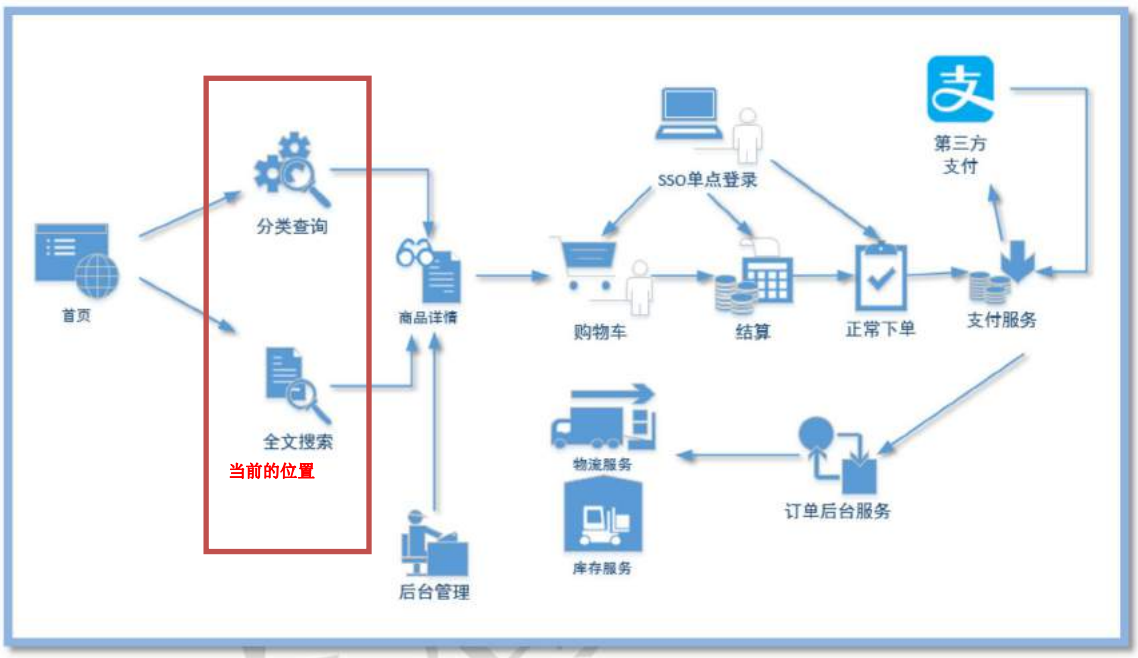
作业： 依照 getSkuInfo 的方法补全，getSpuSaleAttrListCheckBySku 方法和 getSkuSaleAttrValueListBySpu，实现详情页面全部信息都从缓存中查询。

skuInfo 保存后，清除原缓存中数据。

全文搜索

版本：V 1.0

www.atguigu.com



一、搜索

什么是搜索， 计算机根据用户输入的关键词进行匹配，从已有的数据库中摘录出相关的记录反馈给用户。

常见的全网搜索引擎，像百度、谷歌这样的。但是除此以外，搜索技术在垂直领域也有广泛的使用，比如淘宝、京东搜索商品，万芳、知网搜索期刊，csdn 中搜索问题贴。也都是基于海量数据的搜索。

1 如何处理搜索

1.1 用传统关系性数据库

弊端：

1、对于传统的关系性数据库对于关键词的查询，只能逐字逐行的匹配，性能非常差。

2、匹配方式不合理，比如搜索“小密手机”，如果用 like 进行匹配，根本匹配不到。但是考虑使用者的用户体验的话，除了完全匹配的记录，还应该显示一部分近似匹配的记录，至少应该匹配到“手机”。

1.2 专业全文索引是怎么处理的

全文搜索引擎目前主流的索引技术就是倒排索引的方式。

传统的保存数据的方式都是

记录→单词

而倒排索引的保存数据的方式是

单词→记录

例如

搜索“红海行动”

但是数据库中保存的数据如图：

id	标题
1	红海行动影评
2	红海事件始末
3	湄公河行动导演
4	.....

那么搜索引擎是如何能将两者匹配上的呢？

基于分词技术构建倒排索引：

首先每个记录保存数据时，都不会直接存入数据库。系统先会对数据进行分词，然后以倒排索引结构保存。如下：

分词	ids
红海	1,2
行动	1,3
影评	1
事件	2
始末	2
湄公河	3
导演	3

然后等到用户搜索的时候，会把搜索的关键词也进行分词，会把“红海行动”分词分成：红海和行动两个词。

这样的话，先用红海进行匹配，得到 id=1 和 id=2 的记录编号，再用行动匹配可以迅速

定位 id 为 1,3 的记录。

那么全文索引通常，还会根据匹配程度进行打分，显然 1 号记录能匹配的次数更多。所以显示的时候以评分进行排序的话，1 号记录会排到最前面。而 2、3 号记录也可以匹配到。

二　全文检索工具 elasticsearch

1　lucene 与 elasticsearch

咱们之前讲的处理分词，构建倒排索引，等等，都是这个叫 lucene 的做的。那么能不能说这个 lucene 就是搜索引擎呢？

还不能。lucene 只是一个提供全文搜索功能类库的核心工具包，而真正使用它还需要一个完善的服务框架搭建起来的应用。

好比 lucene 是类似于 jdk，而搜索引擎软件就是 tomcat 的。

目前市面上流行的搜索引擎软件，主流的就两款，elasticsearch 和 solr,这两款都是基于 lucene 的搭建的，可以独立部署启动的搜索引擎服务软件。由于内核相同，所以两者除了服务器安装、部署、管理、集群以外，对于数据的操作，修改、添加、保存、查询等等都十分类似。就好像都是支持 sql 语言的两种数据库软件。只要学会其中一个另一个很容易上手。

从实际企业使用情况来看，elasticSearch 的市场份额逐步在取代 solr，国内百度、京东、新浪都是基于 elasticSearch 实现的搜索功能。国外就更多了 像维基百科、GitHub、Stack Overflow 等等也都是基于 ES 的

2　elasticSearch 的使用场景

1、 为用户提供按关键字查询的全文搜索功能。

2、 著名的 ELK 框架(ElasticSearch,Logstash,Kibana)，实现企业海量日志的处理分析的解决方案。大数据领域的重要一份子。

3 elasticSearch 的安装

详见《elasticSearch 的安装手册》

4 elasticsearch 的基本概念

cluster	整个 elasticsearch 默认就是集群状态，整个集群是一份完整、互备的数据。
node	集群中的一个节点，一般只一个进程就是一个 node
shard	分片，即使是一个节点中的数据也会通过 hash 算法，分成多个片存放，默认是 5 片。
index	相当于 rdbms 的 database，对于用户来说是一个逻辑数据库，虽然物理上会被分多个 shard 存放，也可能存放在多个 node 中。
type	类似于 rdbms 的 table，但是与其说像 table，其实更像面向对象中的 class，同一 Json 的格式的数据集合。
document	类似于 rdbms 的 row、面向对象里的 object
field	相当于字段、属性

5 利用 kibana 学习 elasticsearch restful api (DSL)

5.1 es 中保存的数据结构

```
public class    Movie {
        String id;
        String name;
```

```
        Double doubanScore;
        List<Actor> actorList;
}

public class Actor{
        String id;
        String name;
}
```

这两个对象如果放在关系型数据库保存，会被拆成 2 张表，但是 elasticsearch 是用一个 json 来表示一个 document。

所以他保存到 es 中应该是：

```
{
    “id”:”1”,
    “name”:”operation red sea”,
    “doubanScore”:”8.5”,
    “actorList”:[
        {“id”:”1”,”name”:”zhangyi”},
        {“id”:”2”,”name”:”haiqing”},
        {“id”:”3”,”name”:”zhanghanyu”}
    ]
}
```

5.2 对数据的操作

5.2.1 查看 es 中有哪些索引

```
GET /_cat/indices?v
```

es 中会默认存在一个名为.kibana 的索引

表头的含义

health	green(集群完整) yellow(单点正常、集群不完整) red(单点不正常)
status	是否能使用
index	索引名
uuid	索引统一编号
pri	主节点几个
rep	从节点几个

docs.count	文档数
docs.deleted	文档被删了多少
store.size	整体占空间大小
pri.store.size	主节点占

5.2.2 增加一个索引

```
PUT /movie_index
```

5.2.3 删除一个索引

ES 是不删除也不修改任何数据

```
DELETE /movie_index
```

5.2.4 新增文档

1、 格式 PUT /index/type/id

```
PUT /movie_index/movie/1
{ "id":1,
  "name":"operation red sea",
  "doubanScore":8.5,
  "actorList":[
    {"id":1,"name":"zhang yi"},
    {"id":2,"name":"hai qing"},
    {"id":3,"name":"zhang han yu"}
  ]
}
PUT /movie_index/movie/2
{
  "id":2,
  "name":"operation meigong river",
  "doubanScore":8.0,
  "actorList":[
    {"id":3,"name":"zhang han yu"}
    ]
}

PUT /movie_index/movie/3
{
```

```
    "id":3,
    "name":"incident red sea",
    "doubanScore":5.0,
    "actorList":[
       {"id":4,"name":"zhang chen"}
    ]
}
```

如果之前没建过 index 或者 type，es 会自动创建。

5.2.5 直接用 id 查找

```
GET movie_index/movie/1
```

5.2.6 修改—整体替换

和新增没有区别

```
PUT /movie_index/movie/3
{
    "id":"3",
    "name":"incident red sea",
    "doubanScore":"5.0",
    "actorList":[
      {"id":"1","name":"zhang chen"}
    ]
}
```

5.2.7 修改—某个字段

```
POST movie_index/movie/3/_update
{
   "doc": {
      "doubanScore":"7.0"
   }
}
```

5.2.8 删除一个 document

```
DELETE movie_index/movie/3
```

5.2.9 搜索 type 全部数据

```
GET movie_index/movie/_search
```

结果

```
{
  "took": 2,      //耗费时间 毫秒
  "timed_out": false, //是否超时
  "_shards": {
    "total": 5,     //发送给全部 5 个分片
    "successful": 5,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": 3,   //命中 3 条数据
    "max_score": 1,     //最大评分
    "hits": [   // 结果
      {
        "_index": "movie_index",
        "_type": "movie",
        "_id": 2,
        "_score": 1,
        "_source": {
          "id": "2",
          "name": "operation meigong river",
          "doubanScore": 8.0,
          "actorList": [
            {
              "id": "1",
              "name": "zhang han yu"
            }
          ]
        }
          。。。。。。。。
          。。。。。。。。
      }
```

5.2.10 按条件查询(全部)

```
GET movie_index/movie/_search
{
   "query":{
      "match_all": {}
   }
}
```

5.2.11 按分词查询

```
GET movie_index/movie/_search
{
   "query":{
      "match": {"name":"red"}
   }
}
```

注意结果的评分

5.2.12 按分词子属性查询

```
GET movie_index/movie/_search
{
   "query":{
      "match": {"actorList.name":"zhang"}
   }
}
```

5.2.13 match phrase

```
GET movie_index/movie/_search
{
      "query":{
         "match_phrase": {"name":"operation red"}
      }
}
```

按短语查询，不再利用分词技术，直接用短语在原始数据中匹配

5.2.14 fuzzy 查询

```
GET movie_index/movie/_search
{
      "query":{
         "fuzzy": {"name":"rad"}
      }
}
```

校正匹配分词，当一个单词都无法准确匹配，es 通过一种算法对非常接近的单词也给与一定的评分，能够查询出来，但是消耗更多的性能。

5.2.15 过滤--查询后过滤

```
GET movie_index/movie/_search
{
      "query":{
         "match": {"name":"red"}
      },
      "post_filter":{
         "term": {
            "actorList.id": 3
         }
      }
}
```

5.2.16 过滤--查询前过滤（推荐）

```
GET movie_index/movie/_search
{
      "query":{
            "bool":{
               "filter":[ {"term": {    "actorList.id": "1"    }},
                                 {"term": {    "actorList.id": "3"    }}
                 ],
                 "must":{"match":{"name":"red"}}
             }
      }
}
```

5.2.17 过滤--按范围过滤

```
GET movie_index/movie/_search
{
    "query": {
      "bool": {
        "filter": {
          "range": {
              "doubanScore": {"gte": 8}
          }
        }
      }
    }
}
```

关于范围操作符：

gt	大于
lt	小于
gte	大于等于
lte	小于等于

5.2.18 排序

```
GET movie_index/movie/_search
{
  "query":{
    "match": {"name":"red sea"}
  }
  , "sort": [
    {
      "doubanScore": {
        "order": "desc"
      }
    }
  ]
}
```

5.2.19 分页查询

```
GET movie_index/movie/_search
{
   "query": { "match_all": {} },
   "from": 1,
   "size": 1
}
```

5.2.20 指定查询的字段

```
GET movie_index/movie/_search
{
   "query": { "match_all": {} },
   "_source": ["name", "doubanScore"]
}
```

5.2.21 高亮

```
GET movie_index/movie/_search
{
      "query":{
         "match": {"name":"red sea"}
      },
      "highlight": {
         "fields": {"name":{} }
      }

}
```

5.2.22 聚合

取出每个演员共参演了多少部电影

```
GET movie_index/movie/_search
{
   "aggs": {
      "groupby_actor": {
         "terms": {
            "field": "actorList.name.keyword"
```

```
        }
      }
    }
}
```

每个演员参演电影的平均分是多少，并按评分排序

```
GET movie_index/movie/_search
{
  "aggs": {
    "groupby_actor_id": {
      "terms": {
        "field": "actorList.name.keyword" ,
        "order": {
          "avg_score": "desc"
          }
      },
      "aggs": {
        "avg_score":{
          "avg": {
            "field": "doubanScore"
          }
        }
       }
    }
  }
}
```

5.3 关于 mapping

之前说 type 可以理解为 table，那每个字段的数据类型是如何定义的呢

查看看 mapping

```
GET movie_index/_mapping/movie
```

实际上每个 type 中的字段是什么数据类型，由 mapping 定义。

但是如果没有设定 mapping 系统会自动，根据一条数据的格式来推断出应该的数据格式。

- true/false → boolean
- 1020 → long
- 20.1 → double

- “2018-02-01” → date
- “hello world” → text +keyword

默认只有 text 会进行分词，keyword 是不会分词的字符串。

mapping 除了自动定义，还可以手动定义，但是只能对新加的、没有数据的字段进行定义。一旦有了数据就无法再做修改了。

注意：虽然每个 Field 的数据放在不同的 type 下,但是同一个名字的 Field 在一个 index 下只能有一种 mapping 定义。

5.4 中文分词

elasticsearch本身自带的中文分词，就是单纯把中文一个字一个字的分开，根本没有词汇的概念。但是实际应用中，用户都是以词汇为条件，进行查询匹配的，如果能够把文章以词汇为单位切分开，那么与用户的查询条件能够更贴切的匹配上，查询速度也更加快速。

分词器下载网址：https://github.com/medcl/elasticsearch-analysis-ik

5.4.1 安装

下载好的 zip 包，请解压后放到 /usr/share/elasticsearch/plugins/ik

```
总用量 0
drwxrwxrwx. 4 root root 223 3月  18 01:12 ik
[root@jack plugins]# pwd
/usr/share/elasticsearch/plugins
[root@jack plugins]#
```

然后重启 es

5.4.2 测试使用

使用默认

```
GET movie_index/_analyze
{
```

```
  "text": "我是中国人"
}
```

请观察结果

使用分词器

```
GET movie_index/_analyze
{   "analyzer": "ik_smart",
   "text": "我是中国人"
}
```

请观察结果

另外一个分词器

ik_max_word

```
GET movie_index/_analyze
{   "analyzer": "ik_max_word",
   "text": "我是中国人"
}
```

请观察结果

能够看出不同的分词器，分词有明显的区别，所以以后定义一个 type 不能再使用默认的 mapping 了，要手工建立 mapping，因为要选择分词器。

5.4.3 基于中文分词搭建索引

1、建立 mapping

```
PUT movie_chn
{
  "mappings": {
    "movie":{
      "properties": {
        "id":{
          "type": "long"
```

```
            },
            "name":{
               "type": "text"
               , "analyzer": "ik_smart"
            },
            "doubanScore":{
               "type": "double"
            },
            "actorList":{
               "properties": {
                  "id":{
                     "type":"long"
                  },
                  "name":{
                     "type":"keyword"
                  }
               }
            }
         }
      }
   }
}
```

插入数据

```
PUT /movie_chn/movie/1
{ "id":1,
   "name":"红海行动",
   "doubanScore":8.5,
   "actorList":[
   {"id":1,"name":"张译"},
   {"id":2,"name":"海清"},
   {"id":3,"name":"张涵予"}
  ]
}
PUT /movie_chn/movie/2
{
   "id":2,
   "name":"湄公河行动",
   "doubanScore":8.0,
   "actorList":[
{"id":3,"name":"张涵予"}
]
}

PUT /movie_chn/movie/3
{
   "id":3,
   "name":"红海事件",
   "doubanScore":5.0,
   "actorList":[
{"id":4,"name":"张晨"}
```

```
]
}
```

查询测试

```
GET /movie_chn/movie/_search
{
   "query": {
      "match": {
         "name": "红海战役"
      }
   }
}

GET /movie_chn/movie/_search
{
   "query": {
      "term": {
         "actorList.name": "张译"
      }
   }
}
```

5.4.4 自定义词库

修改/usr/share/elasticsearch/plugins/ik/config/中的 IKAnalyzer.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
        <comment>IK Analyzer 扩展配置</comment>
        <!--用户可以在这里配置自己的扩展字典 -->
        <entry key="ext_dict"></entry>
         <!--用户可以在这里配置自己的扩展停止词字典-->
        <entry key="ext_stopwords"></entry>
        <!--用户可以在这里配置远程扩展字典 -->
         <entry key="remote_ext_dict">http://192.168.67.163/fenci/myword.txt</entry>
        <!--用户可以在这里配置远程扩展停止词字典-->
        <!-- <entry key="remote_ext_stopwords">words_location</entry> -->
</properties>
```

按照标红的路径利用 nginx 发布静态资源

在 nginx.conf 中配置

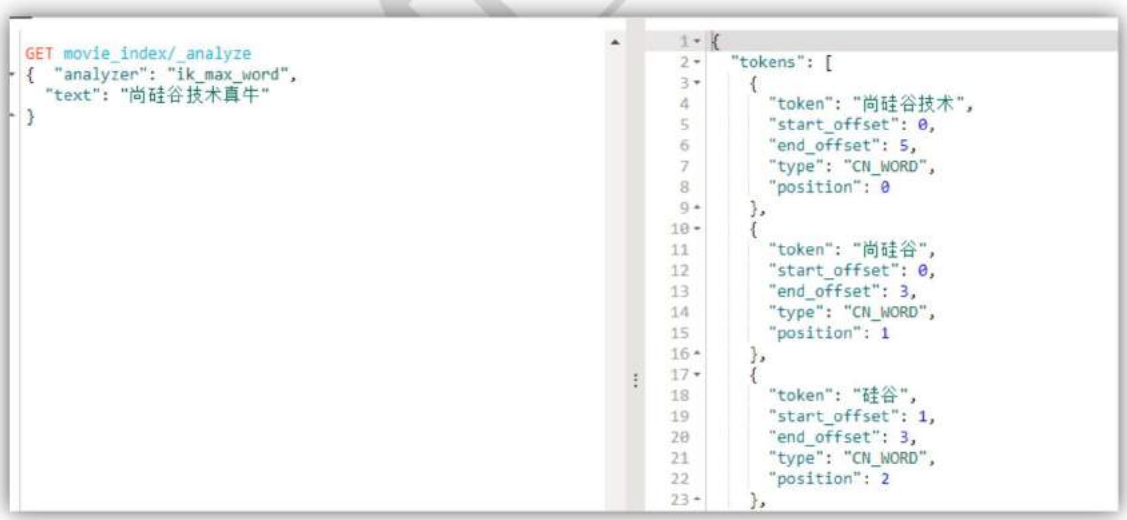
```
server {
        listen    80;
        server_name    192.168.67.163;
        location /fenci/ {
            root es;
    }
  }
```

并且在/usr/local/nginx/下建/es/fenci/目录，目录下加 myword.txt

myword.txt 中编写关键词，每一行代表一个词。

然后重启 es 服务器，重启 nginx。

在 kibana 中测试分词效果

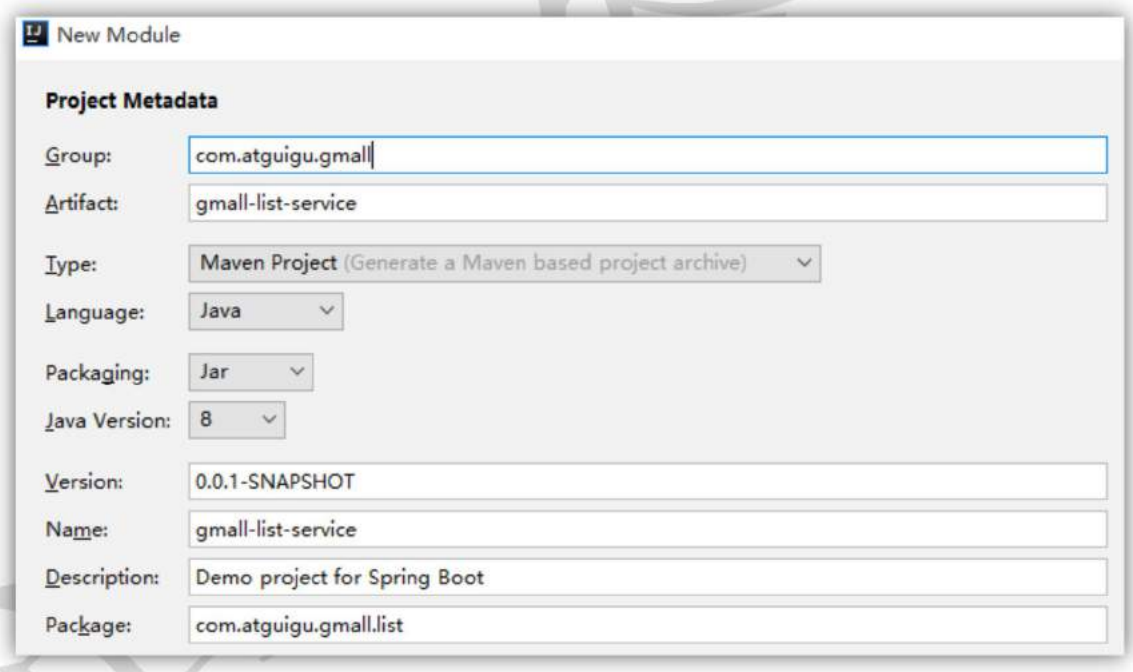


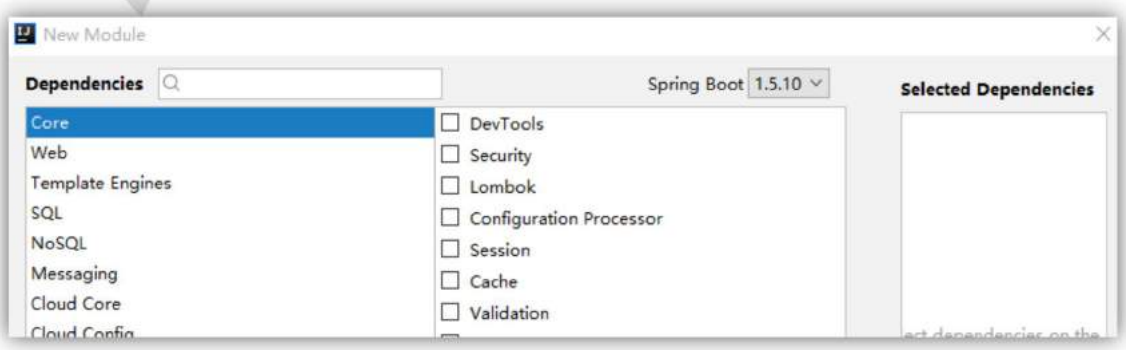
更新完成后，es 只会对新增的数据用新词分词。历史数据是不会重新分词的。如果想要历史数据重新分词。需要执行：

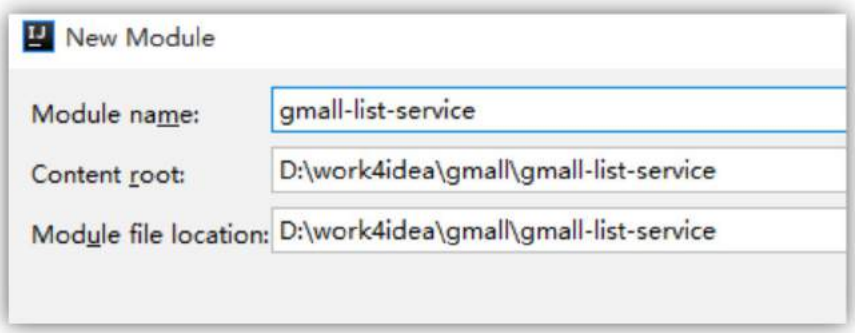
```
POST movies_index_chn/_update_by_query?conflicts=proceed
```

三　Java 程序中的应用

1 、搭建模块







pom.xml

```
<?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.atguigu.gmall</groupId>
    <artifactId>gmall-list-service</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>gmall-list-service</name>
    <description>Demo project for Spring Boot</description>

    <parent>
        <groupId>com.atguigu.gmall</groupId>
        <artifactId>gmall-parent</artifactId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <dependencies>

        <dependency>
            <groupId>com.atguigu.gmall</groupId>
            <artifactId>gmall-interface</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>

        <dependency>
            <groupId>com.atguigu.gmall</groupId>
            <artifactId>gmall-service-util</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>

    </dependencies>

    <build>
        <plugins>
```

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


</project>
```

elasticsearch 的 版本号，请提取到 gmall-parent 中

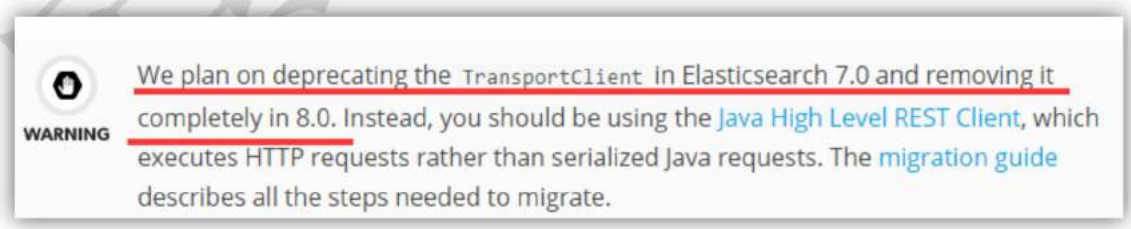
2、 关于 es 的 java 客户端的选择

目前市面上有两类客户端

一类是 TransportClient 为代表的 ES 原生客户端，不能执行原生 dsl 语句必须使用它的 Java api 方法。

另外一种是以 Rest Api 为主的 missing client，最典型的就是 jest。 这种客户端可以直接使用 dsl 语句拼成的字符串，直接传给服务端，然后返回 json 字符串再解析。

两种方式各有优劣，但是最近 elasticsearch 官网，宣布计划在 7.0 以后的版本中废除 TransportClient。以 RestClient 为主。



所以在官方的 RestClient 基础上，进行了简单包装的 Jest 客户端，就成了首选，而且该客户端也与 springboot 完美集成。

3 、导入 Jest 依赖

```
<dependency>
```

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

<!-- https://mvnrepository.com/artifact/io.searchbox/jest -->
<dependency>
    <groupId>io.searchbox</groupId>
    <artifactId>jest</artifactId>
    <version>5.3.3</version>
</dependency>


<!-- https://mvnrepository.com/artifact/net.java.dev.jna/jna -->
<dependency>
    <groupId>net.java.dev.jna</groupId>
    <artifactId>jna</artifactId>
    <version>4.5.1</version>
</dependency>
```

其中 jest 和 jna 请将版本号，部分纳入 gmall-parent 中管理。spring-boot-starter-data-elasticsearch 不用管理版本号，其版本跟随 springboot 的 1.5.10 大版本号。

4 、在测试类中测试 ES

application.properties 中加入

```
spring.elasticsearch.jest.uris=http://192.168.67.163:9200
```

测试类

```
@Autowired
JestClient jestClient;

@Test
public void testEs() throws IOException {
    String query="{\n" +
            "    \"query\": {\n" +
            "      \"match\": {\n" +
            "        \"actorList.name\": \"张译\"\n" +
            "      }\n" +
            "    }\n" +
            "}";
    Search search = new Search.Builder(query).addIndex("movie_chn").addType("movie").build();

    SearchResult result = jestClient.execute(search);
```

```
        List<SearchResult.Hit<HashMap, Void>> hits = result.getHits(HashMap.class);

        for (SearchResult.Hit<HashMap, Void> hit : hits) {
            HashMap source = hit.source;
            System.err.println("source = " + source);

        }

}
```

打印结果：

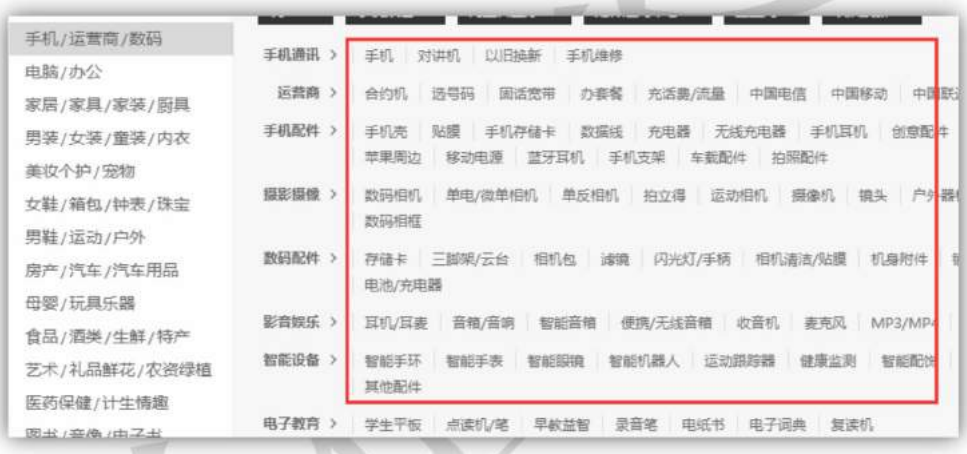
以上技术方面的准备就做好了。下面回到咱们电商的业务

三、利用 elasticSearch 开发电商的搜索列表功能

1、功能简介

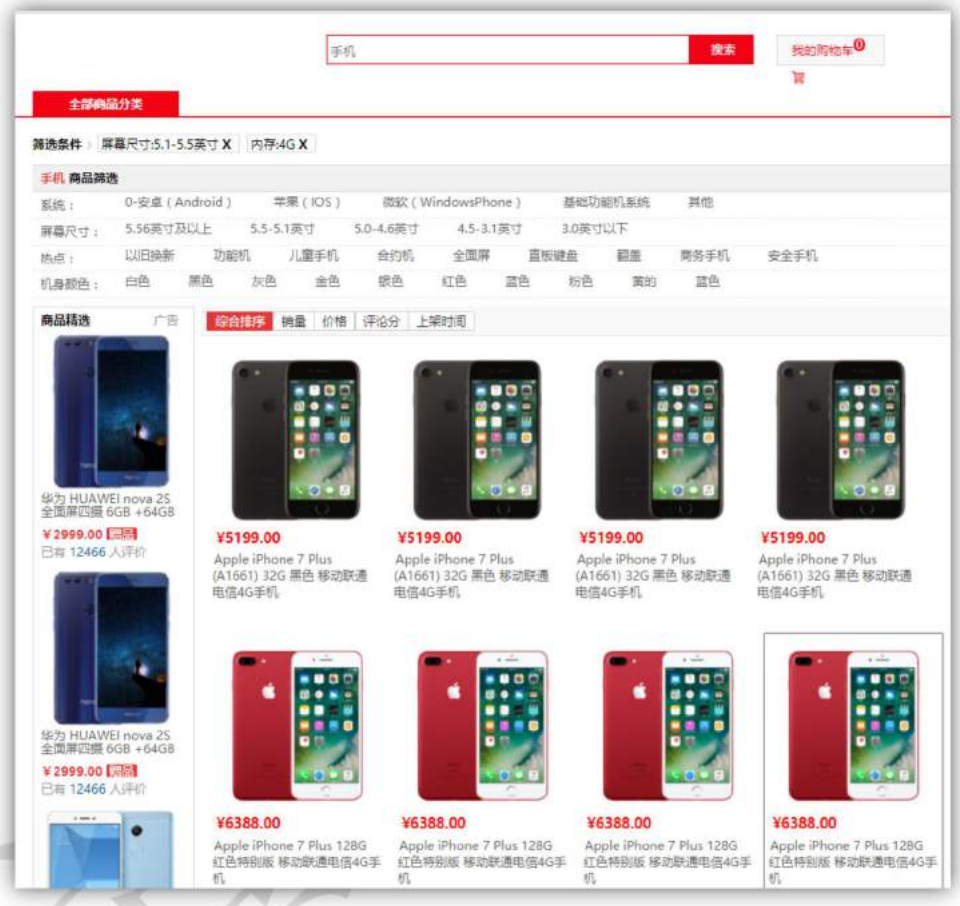
1.1 入口： 两个

首页的分类



搜索栏

列表展示页面



2 根据业务搭建数据结构

这时我们要思考三个问题：

1、 哪些字段需要分词

2、 我们用哪些字段进行过滤

3、 哪些字段我们需要通过搜索显示出来。

需要分词的字段	sku 名称　sku 描述	分词、定义分词器
有可能用于过滤的字段	平台属性、三级分类、价格	要索引
其他需要显示的字段	skuId　图片路径	不索引

根据以上制定出如下结构：

```
PUT gmall
{
  "mappings": {
    "SkuInfo":{
      "properties": {
        "id":{
          "type": "keyword"
          , "index": false
        },
        "price":{
          "type": "double"
        },
         "skuName":{
          "type": "text",
          "analyzer": "ik_max_word"
        },
        "skuDesc":{
          "type": "text",
          "analyzer": "ik_smart"
        },
        "catalog3Id":{
          "type": "keyword"
        },
        "skuDefaultImg":{
          "type": "keyword",
          "index": false
        },
        "skuAttrValueList":{
          "properties": {
            "valueId":{
              "type":"keyword"
            }
          }
        }
      }
    }
  }
```

```
}
```

3 sku 数据保存到 ES

思路：

回顾一下，es 数据保存的 dsl

```
PUT /movie_index/movie/1
{ "id":1,
   "name":"operation red sea",
   "doubanScore":8.5,
   "actorList":[
   {"id":1,"name":"zhang yi"},
   {"id":2,"name":"hai qing"},
   {"id":3,"name":"zhang han yu"}
  ]
}
```

es 存储数据是以 json 格式保存的，那么如果一个 javabean 的结构刚好跟要求的 json 格式吻合，我们就可以直接把 javaBean 序列化为 json 保持到 es 中，所以我们要制作一个与 es 中 json 格式一致的 javabean.

3.1 JavaBean

skuLsInfo

```
public class SkuLsInfo implements Serializable {


    String id;

    BigDecimal price;

    String skuName;

    String skuDesc;

    String catalog3Id;
```

```
    String skuDefaultImg;

    Long hotScore;

    List<SkuLsAttrValue> skuAttrValueList;
}
```

SkuLsAttrValue

```
public class SkuLsAttrValue implements Serializable {

    String valueId;
}
```

3.2 保存 sku 数据的业务实现类

在 gmall-list-service 模块中增加业务实现类 listServiceImpl

```
public static final String index_name_gmall="gmall";

public static final String type_name_gmall="SkuInfo";

public void saveSkuInfo(SkuLsInfo skuLsInfo){
    Index index= new Index.Builder(skuLsInfo).index(index_name_gmall).type(type_name_gmall).id(skuLsInfo.getId()).build();
    try {
        jestClient.execute(index);
    } catch (IOException e) {
        e.printStackTrace();
    }

}
```

自行添加 gmall-inteface 中增加接口方法

3.3 在后台管理的 sku 保存中，调用该方法

```
private void sendSkuToList(SkuInfo skuInfo){

    SkuLsInfo skuLsInfo=new SkuLsInfo();

    try {
        BeanUtils.copyProperties(skuLsInfo, skuInfo);
    } catch (IllegalAccessException e) {
        e.printStackTrace();
    } catch (InvocationTargetException e) {
        e.printStackTrace();
    }
    listService.saveSkuInfo(skuLsInfo);

}


public void saveSkuInfo(SkuInfo skuInfo){

……

sendSkuToList(   skuInfo);    //在保存 sku 最后保存到 es
}
```

这时在界面中保存 sku 是可以，自动向 es 发送数据的。

4 查询数据的后台方法

4.1 分析

首先先观察功能页面，咱们一共要用什么查询条件，查询出什么数据？

查询条件：

1、 关键字

2、 可以通过**分类**进入列表页面

3、 属性值

4、 分页页码

查询结果：

1 sku 的列表(关键字高亮显示)

2 这些 sku 涉及了哪些属性和属性值

3 命中个数，用于分页

基于以上

4.2 编写 DSL 语句：

```
GET gmall/SkuInfo/_search
{
  "query": {
    "bool": {
      "filter": [
            {"terms":{ "skuAttrValueList.valueId": ["46","45"]}},
            {"term":{"catalog3Id":"61"}}
        ],
        "must":
            { "match": { "skuName": "128" }   }
    }
  }
  , "highlight": {
    "fields": {"skuName":{}}
  },
  "from": 3,
  "size": 1,
  "sort":{"hotScore":{"order":"desc"}},
  "aggs": {
    "groupby_attr": {
      "terms": {
        "field": "skuAttrValueList.valueId"
      }
    }

  }
}
```

4.3 制作传入参数的类

```
public class SkuLsParams implements Serializable {
```

```
    String    keyword;

    String catalog3Id;

    String[] valueId;

    int pageNo=1;

    int pageSize=20;
}
```

4.4 返回结果的类

```
public class SkuLsResult implements Serializable {

    List<SkuLsInfo> skuLsInfoList;

    long total;

    long totalPages;

    List<String> attrValueIdList;
}
```

4.5 基于这个 DSL 查询编写 Java 代码

```
public SkuLsResult search(SkuLsParams skuLsParams){

    String query=makeQueryStringForSearch(skuLsParams);

    Search                                    search=                                    new
Search.Builder(query).addIndex(index_name_gmall).addType(type_name_gmall).build();
    SearchResult searchResult=null;
 try {
        searchResult = jestClient.execute(search);
    } catch (IOException e) {
        e.printStackTrace();
    }

    SkuLsResult skuLsResult = makeResultForSearch(skuLsParams, searchResult);

    return skuLsResult;

}
```

4.5.1 构造查询 DSL

查询的过程很简单，但是要构造查询的 query 这个字符串有点麻烦，主要是这个 Json 串中的数据都是动态的。要拼接这个字符串，需要各种循环判断，处理标点符号等等。操作麻烦，可读性差。

但是 jest 这个客户端包，提供了一组 builder 工具。这个工具可以比较方便的帮程序员组合复杂的查询 Json。

```
private String makeQueryStringForSearch(SkuLsParams skuLsParams){
    SearchSourceBuilder searchSourceBuilder=new SearchSourceBuilder();

    BoolQueryBuilder boolQueryBuilder = QueryBuilders.boolQuery();
    if(skuLsParams.getKeyword()!=null){
        MatchQueryBuilder matchQueryBuilder=new
MatchQueryBuilder("skuName",skuLsParams.getKeyword());
boolQueryBuilder.must(matchQueryBuilder);

        HighlightBuilder highlightBuilder=new HighlightBuilder();
        highlightBuilder.field("skuName");
        highlightBuilder.preTags("<span style='color:red'>");
        highlightBuilder.postTags("</span>");
        searchSourceBuilder.highlight(highlightBuilder);

        TermsBuilder groupby_attr =
AggregationBuilders.terms("groupby_attr").field("skuAttrValueList.valueId");
        searchSourceBuilder.aggregation(groupby_attr);
    }
    if(skuLsParams.getCatalog3Id()!=null){
        QueryBuilder termQueryBuilder=new
TermQueryBuilder("catalog3Id",skuLsParams.getCatalog3Id());
        boolQueryBuilder.filter(termQueryBuilder);
    }
    if(skuLsParams.getValueId()!=null&&skuLsParams.getValueId().length>=0){

        for (int i = 0; i < skuLsParams.getValueId().length; i++) {
                String valueId = skuLsParams.getValueId()[i];
                QueryBuilder termQueryBuilder=new
TermsQueryBuilder("skuAttrValueList.valueId",valueId);
                boolQueryBuilder.filter(termQueryBuilder);
         }

    }
    searchSourceBuilder.query(boolQueryBuilder);
```

```
        int from =(skuLsParams.getPageNo()-1)*skuLsParams.getPageSize();
        searchSourceBuilder.from(from);
        searchSourceBuilder.size(skuLsParams.getPageSize());

        searchSourceBuilder.sort("hotScore",SortOrder.DESC);

        String query = searchSourceBuilder.toString();

        System.err.println("query = " + query);
        return query;
}
```

4.5.2 处理返回值

思路：所有的返回值其实都在这个 searchResult 中

```
searchResult = jestClient.execute(search);
```

它的结构其实可以在 kibana 中观察一下：

命中的结果

高亮显示

分组统计结果：

针对这三个部分来解析 searchResult

```
private SkuLsResult makeResultForSearch(SkuLsParams skuLsParams,SearchResult searchResult){
    SkuLsResult skuLsResult=new SkuLsResult();
    List<SkuLsInfo> skuLsInfoList=new ArrayList<>(skuLsParams.getPageSize());

    //获取 sku 列表
    List<SearchResult.Hit<SkuLsInfo, Void>> hits = searchResult.getHits(SkuLsInfo.class);
    for (SearchResult.Hit<SkuLsInfo, Void> hit : hits) {
        SkuLsInfo skuLsInfo = hit.source;
        if(hit.highlight!=null&&hit.highlight.size()>0){
            List<String> list = hit.highlight.get("skuName");
            //把带有高亮标签的字符串替换 skuName
            String skuNameHl = list.get(0);
            skuLsInfo.setSkuName(skuNameHl);
        }
        skuLsInfoList.add(skuLsInfo);
```

```
        }
        skuLsResult.setSkuLsInfoList(skuLsInfoList);
        skuLsResult.setTotal(searchResult.getTotal());

        //取记录个数并计算出总页数
        long totalPage= (searchResult.getTotal() + skuLsParams.getPageSize() -1) / skuLsParams.getPageSize();
        skuLsResult.setTotalPages(  totalPage);

        //取出涉及的属性值 id
        List<String> attrValueIdList=new ArrayList<>();
        MetricAggregation aggregations = searchResult.getAggregations();
        TermsAggregation groupby_attr = aggregations.getTermsAggregation("groupby_attr");
        if(groupby_attr!=null){
            List<TermsAggregation.Entry> buckets = groupby_attr.getBuckets();
            for (TermsAggregation.Entry bucket : buckets) {
                attrValueIdList.add( bucket.getKey()) ;
            }
            skuLsResult.setAttrValueIdList(attrValueIdList);
        }


        return skuLsResult;
    }
```

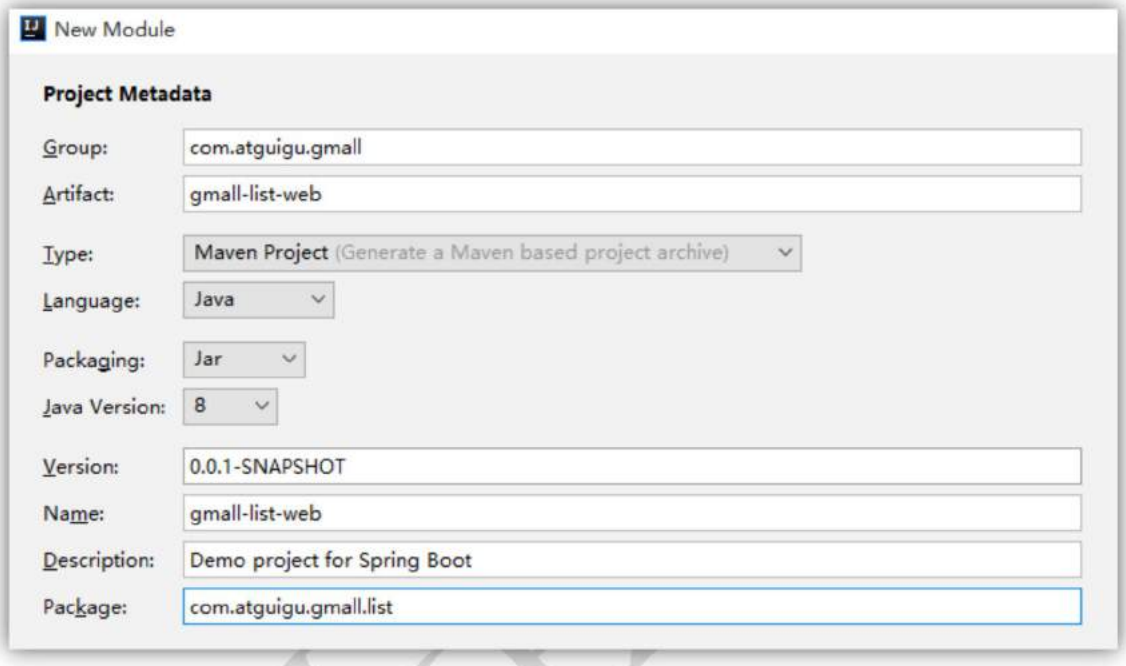
测试后台程序…

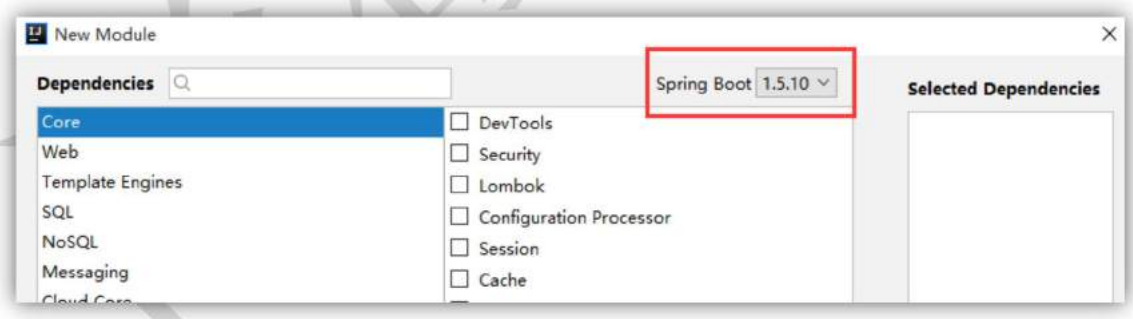
完成了 sku 列表数据获取的方法。回到页面功能上来。

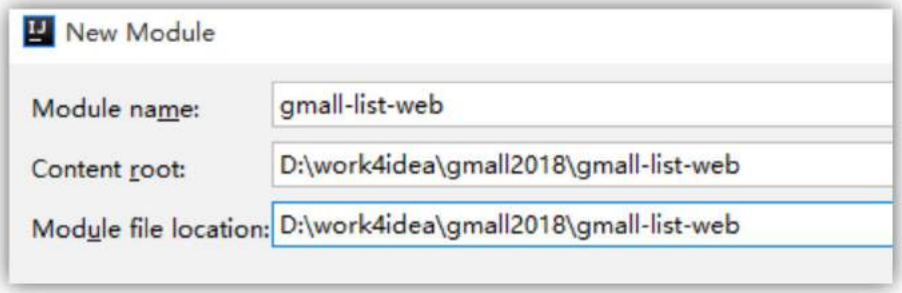
5 检索的页面

检索功能

5.1 创建 gmall-list-web 模块





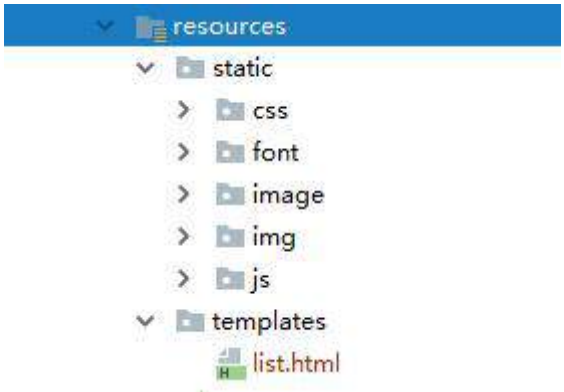


5.1.1 pom.xml

```
<parent>
   <groupId>com.atguigu.gmall</groupId>
   <artifactId>gmall-parent</artifactId>
   <version>1.0-SNAPSHOT</version>
</parent>


<dependencies>

<dependency>
   <groupId>com.atguigu.gmall</groupId>
   <artifactId>gmall-interface</artifactId>
   <version>1.0-SNAPSHOT</version>
</dependency>

<dependency>
   <groupId>com.atguigu.gmall</groupId>
   <artifactId>gmall-web-util</artifactId>
   <version>1.0-SNAPSHOT</version>
</dependency>
</dependencies>
```

5.1.2 静态网页及资源文件

拷贝静态文件到 resources 目录下，手工建立 static 和 templates 目录



手工替换一下目录

把 list.html 的路径中"../static/" 替换成 "/"

5.1.3 application.properties

```
server.port=8085

spring.thymeleaf.cache=false

spring.thymeleaf.mode=LEGACYHTML5
```

5.1.4 nginx 配置

host 文件

```
# gmall
192.168.67.163    item.gmall.com list.gmall.com manage.gmall.com      resource.gmall.com
```

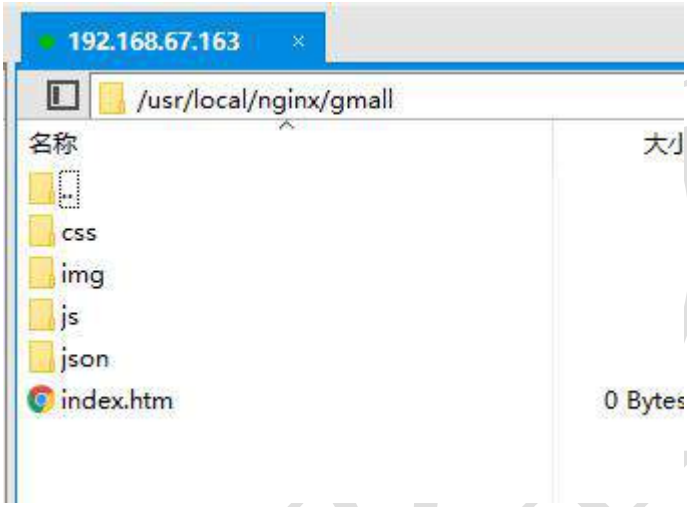
nginx.conf

```
    upstream list.gmall.com{
        server 192.168.67.1:8085;
    }
   server {
      listen 80;
      server_name list.gmall.com;
      location / {
          proxy_pass http:// list.gmall.com;
          proxy_set_header X-forwarded-for $proxy_add_x_forwarded_for;
```

```
    }
}
```

5.1.5 放置 nginx 首页静态资源

用 xftp 拷贝首页资源到 nginx/gmall 目录下



修改 nginx.conf 配置文件

```
server {
        listen          80;
        server_name    www.gmall.com;
         location / {
              root     gmall;
              index    index.html index.htm;
    }
}
```

host 文件

```
# gmall
192.168.67.163          item.gmall.com    list.gmall.com    manage.gmall.com    www.gmall.com

resource.gmall.com
```

5.2 sku 列表功能

首先是根据关键字、属性值、分类 Id、页码查询 sku 列表。

5.2.1 ListController

```
@RequestMapping("list.html")
public String getList(  SkuLsParams skuLsParams, Model model){

    skuLsParams.setPageSize(4);
    //根据参数返回 sku 列表
    SkuLsResult skuLsResult = listService.search(skuLsParams);
    model.addAttribute("skuLsInfoList",skuLsResult.getSkuLsInfoList());
     return "list";
}
```

5.2.2 页面 html 渲染

```
<div style="width:215px" th:each="skuLsInfo:${skuLsInfoList}" >
      <p class="da">
          <a href="#"
th:onclick="'javascript:item(\''+${skuLsInfo.id}+'\')'">
```

```
            <img th:src="${skuLsInfo.skuDefaultImg}"
src="img/57d0d400Nfd249af4.jpg" class="dim">
        </a>
    </p>

    <p class="tab_R">
        <span th:text="'
￥'+${#numbers.formatDecimal(skuLsInfo.price,1,2)}">¥5199.00</span>
    </p>

        <a href="#" title=""
th:onclick="'javascript:item(\''+${skuLsInfo.id}+'\')'" class="tab_JE"
th:utext="${skuLsInfo.skuName}" >
            Apple iPhone 7 Plus (A1661) 32G 黑色 移动联通电信 4G 手机
        </a>
  </div>
```

要注意的是其中 skuName 中因为关键字标签所以必须要用 utext 否则标签会被转义。

5.2.3 搜索栏相关 html

```
<!--搜索导航-->
<div class="header_sous">
    <div class="logo">
        <a href="#"><img src="/image/logo1.jpg" alt=""></a>
    </div>
    <div class="header_form">
        <input id="keyword" name="keyword" type="text" placeholder="手机" />
        <a href="#" onclick="searchList()">搜索</a>
    </div>
    <div class="header_ico">
        <div class="header_gw">
            <span><a href="#">我的购物车</a></span>
            <img src="/image/settleup-@1x.png" />
            <span>0</span>
        </div>
        <div class="header_ko">
            <p>购物车中还没有商品，赶紧选购吧！</p>
```

```
        </div>
    </div>
```

5.2.4 js 代码

```
function searchList(){
    var keyword = $("#keyword").val();
    window.location.href="/list.html?keyword="+keyword;
}

function item(skuid) {
    window.location.href="http://item.gmall.com/"+skuid+".html";
}
```

这时可以看到列表效果了。

5.3 页面功能—提供可供选择的属性列表

5.3.1 思路：

这个列表有两种情况

1、 如果是通过首页的 3 级分类点击进入的，要按照分类 Id 查询对应的属性和属性值列表。

2、 如果是直接用搜索栏输入文字进入的，要根据 sku 的查询结果涉及的属性值(好在我们已经通过 es 的聚合取出来了)，再去查询数据库把文字列表显示出来。

5.3.2 ListController

getList 方法中添加

```
//根据查询的结果返回属性和属性值列表
List<BaseAttrInfo> attrList=null;
if(skuLsParams.getCatalog3Id()!=null){
     attrList =
manageService.getAttrList(skuLsParams.getCatalog3Id());
}else {
    List<String> attrValueIdList = skuLsResult.getAttrValueIdList();
    if(attrValueIdList!=null&&attrValueIdList.size()>0){
        attrList = manageService.getAttrList(attrValueIdList);
    }

}
model.addAttribute("attrList",attrList);
```

其中 manageService.getAttrList(String catalog3Id)这个方法我们已经有现成的了。

但是 manageService.getAttrList(attrValueIdList) 这个方法我们要新添加。

5.3.3 在 ManageServiceImpl 中增加方法

```
@Override
public List<BaseAttrInfo> getAttrList(List<String> attrValueIdList) {
      String attrValueIds = StringUtils.join(attrValueIdList.toArray(), ",");
      List<BaseAttrInfo>                                baseAttrInfoList                                =
baseAttrInfoMapper.selectAttrInfoListByIds(attrValueIds);
      return baseAttrInfoList;

}
```

5.3.4 BaseAttrInfoMapper.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE mapper SYSTEM "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.atguigu.gmall.manage.mapper.BaseAttrInfoMapper">
    <select  id ="selectAttrInfoList" parameterType="long" resultMap="attrInfoMap">
        SELECT ba.id,ba.attr_name,ba.catalog3_id,
        bv.id value_id ,bv.value_name, bv.attr_id FROM
        base_attr_info ba INNER JOIN  base_attr_value bv  ON ba.id =bv.attr_id
        where  ba.catalog3_id=#{catalog3Id}
    </select>
    <resultMap          id="attrInfoMap"          type="com.atguigu.gmall.bean.BaseAttrInfo"
autoMapping="true">
        <result property="id" column="id" ></result>
        <collection  property="attrValueList"  ofType="com.atguigu.gmall.bean.BaseAttrValue"
autoMapping="true">
            <result property="id" column="value_id" ></result>
        </collection>
    </resultMap>


    <select  id ="selectAttrInfoListByIds"  resultMap="attrInfoMap">
        SELECT ba.id,ba.attr_name,ba.catalog3_id,
        bv.id value_id ,bv.value_name, bv.attr_id FROM
        base_attr_info ba INNER JOIN  base_attr_value bv  ON ba.id =bv.attr_id
        where  bv.id in (${attrValueIds})
    </select>
</mapper>
```

注意这里面没有用#{}是因为 attrValueIds 是两个数字用逗号分开的，所以不能整体套上单引，所以使用${}。

5.3.5 BaseAttrInfoMapper.class

```
public interface BaseAttrInfoMapper extends Mapper<BaseAttrInfo> {

    public List<BaseAttrInfo> selectAttrInfoList(long catalog3Id);

    public List<BaseAttrInfo> selectAttrInfoListByIds(@Param("attrValueIds")
String attrValueIds);
}
```

此处必须要用@Param 注解否则${ }无法识别。

5.3.6 点击属性值的链接

getAttrList(List<String> attrValueIdList)方法实现后，还有一个问题就是，点击属性时，要把上次查询的内容也带上，即带上历史参数。

```
private String makeUrlParam(String keyword,String catalog3Id, String[] valueIds,String excludeValueId ){
        String url="";
        List<String> paramList=new ArrayList<>();
        if(keyword!=null&&keyword.length()>0){
                paramList.add("keyword="+keyword);
        }
        if(catalog3Id!=null){
                paramList.add("catalog3Id="+catalog3Id);
        }

        if(valueIds!=null) {
                for (int i = 0; i < valueIds.length; i++) {
                        String valueId = valueIds[i];
                        if (!excludeValueId.equals(valueId)) {
                                paramList.add("valueId=" + valueId) ;
                        }
                }
        }
          url = StringUtils.join(paramList, "&");
        return url;
}
```

getList 方法中增加

```
//历史参数
String[] valueIds=skuLsParams.getValueId();
String catalog3Id = skuLsParams.getCatalog3Id();
String keyword=skuLsParams.getKeyword();

String urlParam = makeUrlParam(keyword,catalog3Id, valueIds, "");
model.addAttribute("urlParam",urlParam);
```

java 部分的代码就完成了。

5.3.7 生成属性列表的 html 部分

```
<div class="GM_selector">
        <!--手机商品筛选-->
        <div class="title">
                <h3><em>商品筛选</em></h3>
```

```
        </div>
        <div class="GM_nav_logo">

            <div class="GM_pre"    th:each="attrInfo:${attrList}">
                <div class="sl  key">
                    <span th:text="${attrInfo.attrName}+':'">属性：</span>
                </div>
                <div class="sl_value">
                    <ul>
                        <li                    th:each="attrValue:${attrInfo.attrValueList}"><a
th:href="'/list.html?'+${urlParam}+'&valueId='+${attrValue.id}"
th:text="${attrValue.valueName}">属性值</a></li>
                    </ul>
                </div>
            </div>
        </div>
</div>
```

完成后

5.4 页面功能--面包屑

面包屑导航是为了能够让用户清楚的知道当前页面的所在位置和筛选条件的功能。但是这个小的人性化功能却有点麻烦。

功能点： 1、点击某个属性值的时候对应的那行属性要消失掉不能再次选择。

2、列在上面的属性面包屑，要可以取消掉，恢复到没选择之前。

5.4.1 思路：

1 把本应显示的列表与用户已选择的属性值列表用循环交叉判断，如果匹配把本应显示的那个属性去掉。

2 已选择的属性值列表，要携带点击跳转的路径，这个路径参数就是咱们上边讲的那个“历史参数”，但是要把自己本身的属性值去掉。

5.4.2 扩展类

增加 BaseAttrValueExt 类 ，这个类是扩展了原 BaseAttrValue 类为了方便携带参数。

```
public class BaseAttrValueExt extends BaseAttrValue {


    String attrName ;

    String cancelUrlParam;
}
```

5.4.3 controller 中的 getList 方法增加

代码如下

```
//去掉已选择的属性值
if(skuLsParams.getValueId()!=null&&attrList!=null) {
    for (int i = 0; valueIds != null && i < valueIds.length; i++) {
        String selectedValueId = valueIds[i];

        for (Iterator<BaseAttrInfo> iterator = attrList.iterator(); iterator.hasNext(); ) {
            BaseAttrInfo baseAttrInfo = iterator.next();
            List<BaseAttrValue> attrValueList = baseAttrInfo.getAttrValueList();
            for (BaseAttrValue baseAttrValue : attrValueList) {
                if (selectedValueId.equals(baseAttrValue.getId())) {
                    BaseAttrValueExt baseAttrValueExt = new BaseAttrValueExt();
                    baseAttrValueExt.setAttrName(baseAttrInfo.getAttrName());
                    baseAttrValueExt.setId(baseAttrValue.getId());

baseAttrValueExt.setValueName(baseAttrValue.getValueName());
                    baseAttrValueExt.setAttrId(baseAttrInfo.getId());
                    String cancelUrlParam = makeUrlParam(keyword,catalog3Id,
valueIds, selectedValueId);
                    baseAttrValueExt.setCancelUrlParam(cancelUrlParam);

                    selectedValuelist.add(baseAttrValueExt);

                    iterator.remove();// 如果选中了该属性 就从属性列表中去
掉
                }
```

```
                }
            }
        }
    }

    model.addAttribute("selectedValueList", selectedValuelist);
    model.addAttribute("keyword",keyword);
    model.addAttribute("attrList",attrList);
```

这块代码看似多层循环嵌套性能隐患，其实因为单次循环基本不会超过五次，循环中没有网络或者 io 访问，完全在虚拟机中运行，所以即使多层循环嵌套压力也不会太大。

5.4.4 页面代码

```
<div class="GM_ipone">
    <div class="GM_ipone_bar">
        <div class="GM_ipone_one a">
            筛选条件
        </div>
        <i><img src="/image/right-@1x.png" alt=""></i>
        <span th:if="${keyword}!=null" th:text="'&quot;'+${keyword}+'&quot;'"></span>
        <a class="select-attr" th:each="selectedValue:${selectedValueList}"
th:href="'/list.html?'+${selectedValue.cancelUrlParam}"
th:utext="${selectedValue.attrName}+':'+${selectedValue.valueName} +'<b> X </b>'" > 屏幕尺寸:5.1-5.5 英寸
        </a>

    </div>
</div>
```

5.5 页面功能--分页

由于咱们在 es 中查询数据时已经完成了后台分页，这里主要就是把分页结果丢给前台页面展示就好了。

注意每个页面都要带上历史参数。

5.5.1 controller 中的 getList 里加入

```
long totalPages = skuLsResult.getTotalPages();
model.addAttribute("totalPages",totalPages);
model.addAttribute("pageNo",skuLsParams.getPageNo());
```

至此 ListController 就全部完成了

5.5.2 ListController 代码

```
@Controller
public class ListController {

    @Reference
    ManageService manageService;

    @Reference
    ListService listService;


    @RequestMapping("list.html")
    public String getList(    SkuLsParams skuLsParams, Model model){


        skuLsParams.setPageSize(4);
        //根据参数返回 sku 列表
        SkuLsResult skuLsResult = listService.search(skuLsParams);
        model.addAttribute("skuLsInfoList",skuLsResult.getSkuLsInfoList());

        //根据查询的结果返回属性和属性值列表
        List<BaseAttrInfo> attrList=null;
        if(skuLsParams.getCatalog3Id()!=null){
            attrList = manageService.getAttrList(skuLsParams.getCatalog3Id());
        }else {
            List<String> attrValueIdList = skuLsResult.getAttrValueIdList();
            if(attrValueIdList!=null&&attrValueIdList.size()>0){
                attrList = manageService.getAttrList(attrValueIdList);
            }

        }
        model.addAttribute("attrList",attrList);

        //历史参数
        String[] valueIds=skuLsParams.getValueId();
        String catalog3Id = skuLsParams.getCatalog3Id();
        String keyword=skuLsParams.getKeyword();
```

```
        String urlParam = makeUrlParam(keyword,catalog3Id, valueIds, "");
        model.addAttribute("urlParam",urlParam);

        List<BaseAttrValueExt> selectedValuelist =new ArrayList<>();



        //去掉已选择的属性值
        if(skuLsParams.getValueId()!=null&&attrList!=null) {
            for (int i = 0; valueIds != null && i < valueIds.length; i++) {
                String selectedValueId = valueIds[i];

                for (Iterator<BaseAttrInfo> iterator = attrList.iterator(); iterator.hasNext(); ) {
                    BaseAttrInfo baseAttrInfo = iterator.next();
                    List<BaseAttrValue> attrValueList = baseAttrInfo.getAttrValueList();
                    for (BaseAttrValue baseAttrValue : attrValueList) {
                        if (selectedValueId.equals(baseAttrValue.getId())) {
                            BaseAttrValueExt baseAttrValueExt = new BaseAttrValueExt();
                            baseAttrValueExt.setAttrName(baseAttrInfo.getAttrName());
                            baseAttrValueExt.setId(baseAttrValue.getId());

baseAttrValueExt.setValueName(baseAttrValue.getValueName());
                            baseAttrValueExt.setAttrId(baseAttrInfo.getId());
                            String cancelUrlParam = makeUrlParam(keyword,catalog3Id,
valueIds, selectedValueId);
                            baseAttrValueExt.setCancelUrlParam(cancelUrlParam);

                            selectedValuelist.add(baseAttrValueExt);

                            iterator.remove();// 如果选中了该属性 就从属性列表中去
掉
                        }
                    }
                }
            }
        }

        model.addAttribute("selectedValueList", selectedValuelist);
        model.addAttribute("keyword",keyword);
        model.addAttribute("attrList",attrList);

        long totalPages = skuLsResult.getTotalPages();
        model.addAttribute("totalPages",totalPages);
        model.addAttribute("pageNo",skuLsParams.getPageNo());

        //以下是查询商品信息


        return "list";
    }
```

```
    private String    makeUrlParam(String keyword,String catalog3Id,      String[] valueIds,String
excludeValueId ){
        String url="";
        List<String> paramList=new ArrayList<>();
        if(keyword!=null&&keyword.length()>0){
            paramList.add("keyword="+keyword);
        }
        if(catalog3Id!=null){
            paramList.add("catalog3Id="+catalog3Id);
        }

        if(valueIds!=null) {
            for (int i = 0; i < valueIds.length; i++) {
                String valueId = valueIds[i];
                if (!excludeValueId.equals(valueId)) {
                    paramList.add("valueId=" + valueId) ;
                }
            }
        }
         url = StringUtils.join(paramList, "&");
        return url;
    }
}
```

5.5.3 分页的页面部分代码

```
<script th:inline="javascript">
    /*<![CDATA[*/

    $(function(){
        $(".page_span1").empty();
        var totalPage = [[${totalPages}]];
        var pageNo = [[${pageNo}]];
        var urlParam = [[${urlParam}]];

        //上一页
        var $lastPage =    '<a href="/list.html?'+urlParam+'&pageNo='+(pageNo-1)+'"><  上一
页</a>';
        if(pageNo == 1){
            $lastPage = '<a disabled="true"><  上一页</a>';
        }
        $(".page_span1").append($lastPage);
        for(var i = 1; i <= totalPage; i++) {
            if (i==pageNo){
```

```
                $(".page_span1").append('<a style="border: 0;font-size: 20px;color: red;background: #fff">'+i+'</a>');
            }else if(i==1 || i==totalPage||(i>=(pageNo-2)&&i<=(pageNo+2))){
                $(".page_span1").append('<a href="/list.html?'+urlParam+'&pageNo='+i+'">'+i+'</a>');
            } else if((i==(pageNo+3) && i<totalPage)||(i==(pageNo-3) && i>1)){
                $(".page_span1").append('<a style="border: 0;font-size: 20px;color: #999;background: #fff">...</a>');
            }
        }
        //下一页
        var $nextPage =  '<a href="/list.html?'+urlParam+'&pageNo='+(pageNo+1)+'">下一页 ></a>';
        if(pageNo == totalPage||totalPage==0){
            $nextPage = '<a disabled="true">下一页 ></a>';
        }
        $(".page_span1").append($nextPage);

    })
    /*]]>*/
</script>
```

分页栏由 6 部分组成，通过判断页面决定显示效果。

6 排序

页面结构完成了，考虑一下如何排序，es 查询的 dsl 语句中我们是用了 hotScore 来进行排序的。

但是 hotScore 从何而来，根据业务去定义，也可以扩展更多类型的评分，让用户去选择如何排序。

这里的 hotScore 我们假定以点击量来决定热度。

那么我们每次用户点击，将这个评分+1,不就可以了么。

6.1 问题：

1、 es 大量的写操作会影响 es 性能，因为 es 需要更新索引，而且 es 不是内存数据库，会做相应的 io 操作。

2、而且修改某一个值，在高并发情况下会有冲突，造成更新丢失，需要加锁，而 es 的乐观锁会恶化性能问题。

从业务角度出发，其实我们为商品进行排序所需要的热度评分，并不需要非常精确，大致能比出个高下就可以了。

利用这个特点我们可以稀释掉大量写操作。

6.2 解决思路：

用 redis 做精确计数器，redis 是内存数据库读写性能都非常快，利用 redis 的原子性的自增可以解决并发写操作。

redis 每计 100 次数（可以被 100 整除）我们就更新一次 es ，这样写操作就被稀释了 100 倍，这个倍数可以根据业务情况灵活设定。

代码 在 listServiceImpl 中增加更新操作

6.3 代码：更新 es

```
private void updateHotScore(String skuId,Long hotScore){
     String updateJson="{\n" +
             "   \"doc\":{\n" +
             "     \"hotScore\":"+hotScore+"\n" +
             "   }\n" +
             "}";

    Update update = new
Update.Builder(updateJson).index("gmall").type("SkuInfo").id(skuId).build()
;
    try {
        jestClient.execute(update);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

6.4 更新 redis 计数器

```
//更新热度评分
@Override
public void incrHotScore(String skuId){
    Jedis jedis = redisUtil.getJedis();
    int timesToEs=100;
    Double hotScore = jedis.zincrby("hotScore", 1, "skuId:" + skuId);
    if(hotScore%timesToEs==0){
        updateHotScore(  skuId,  Math.round(hotScore)   );
    }

}
```

6.5 详情页调用

这个 incrHotScore 方法可以在进入详情页面的时候调用。

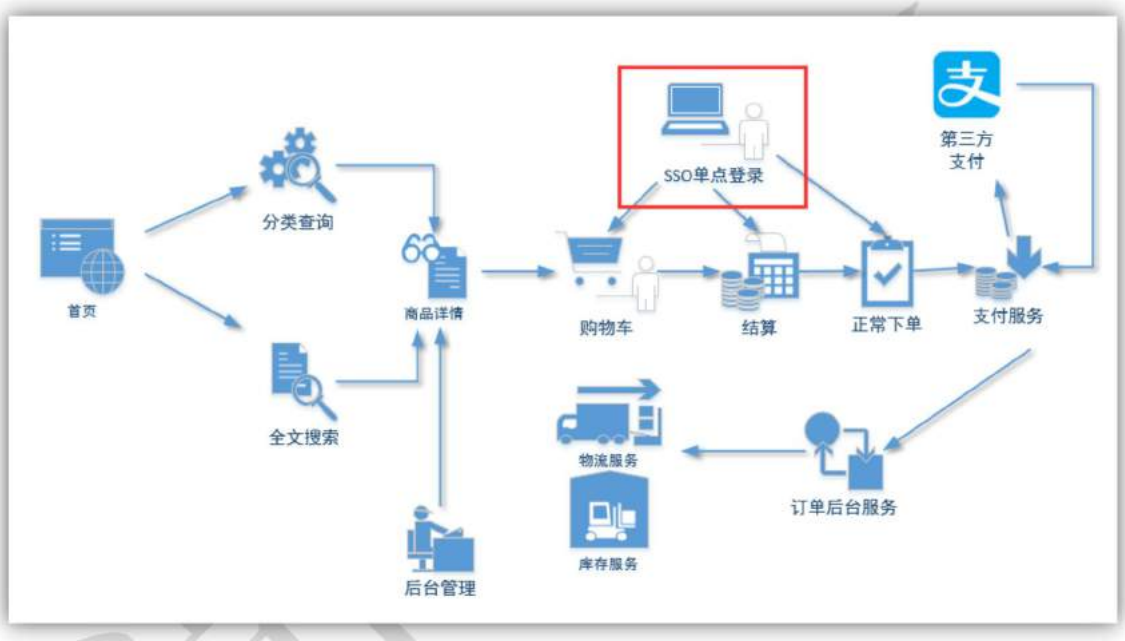
```
@Controller
public class ItemController {

    @Reference
    ListService listService;

    @Reference
    ManageService manageService;

    @RequestMapping("/{skuId}.html")
    public String getSkuInfo(@PathVariable("skuId") String skuId, Model model){
     ……
     listService.incrHotScore(skuId);  //最终应该由异步方式调用


}
```

单点登录

版本：V 1.0

www.atguigu.com

所在位置：



一、 单点登录业务介绍

早期单一服务器，用户认证

缺点：单点性能压力，无法扩展

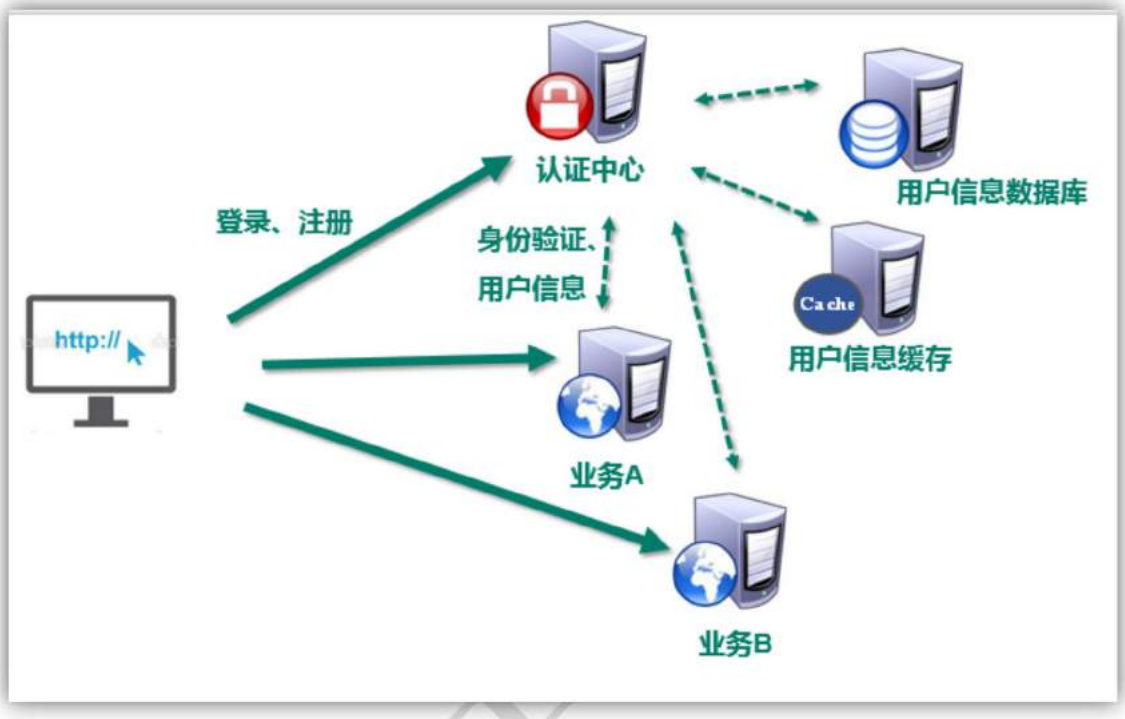
WEB 应用集群，session 共享模式

解决了单点性能瓶颈。

问题：

1、 多业务分布式数据独立管理，不适合统一维护一份 session 数据。
2、 分布式按业务功能切分，用户、认证解耦出来单独统一管理。
3、 cookie 中使用 jsessionId 容易被篡改、盗取。
4、 跨顶级域名无法访问。

分布式，SSO(single sign on)模式



解决 ：

用户身份信息独立管理，更好的分布式管理。

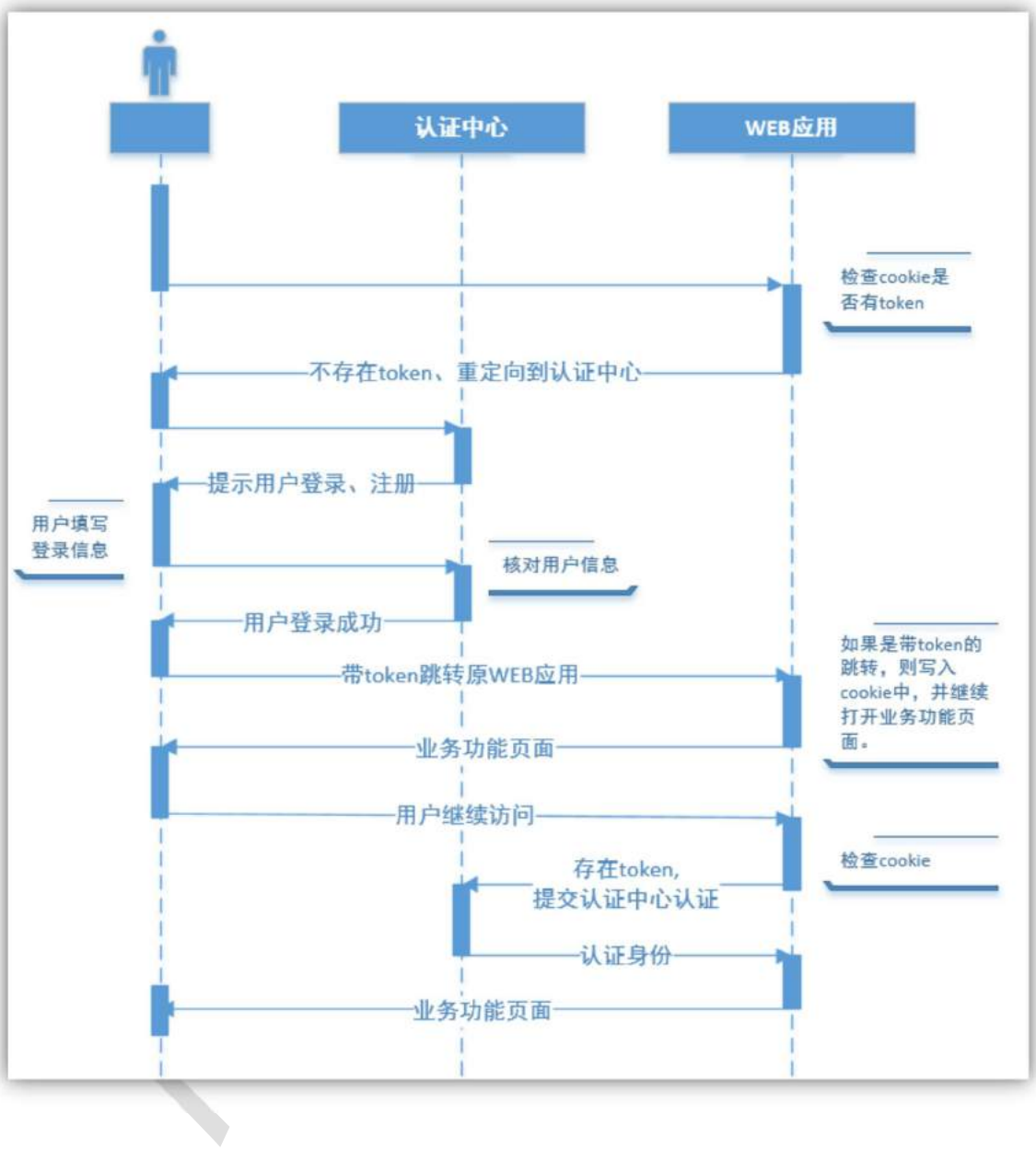
可以自己扩展安全策略

跨域不是问题

缺点：

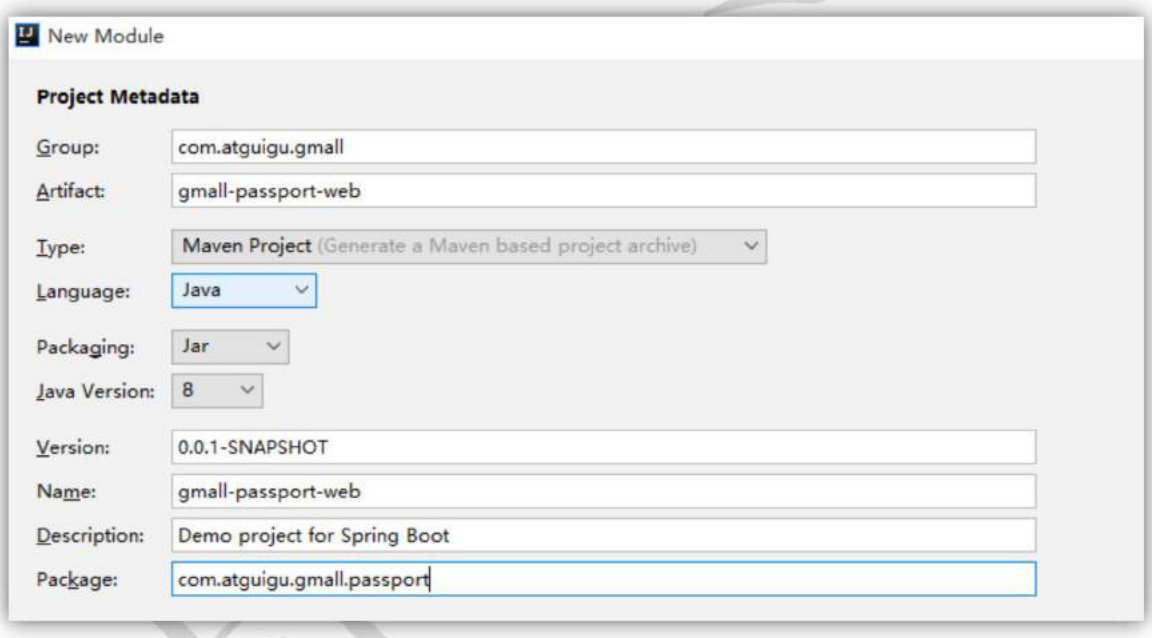
认证服务器访问压力较大。

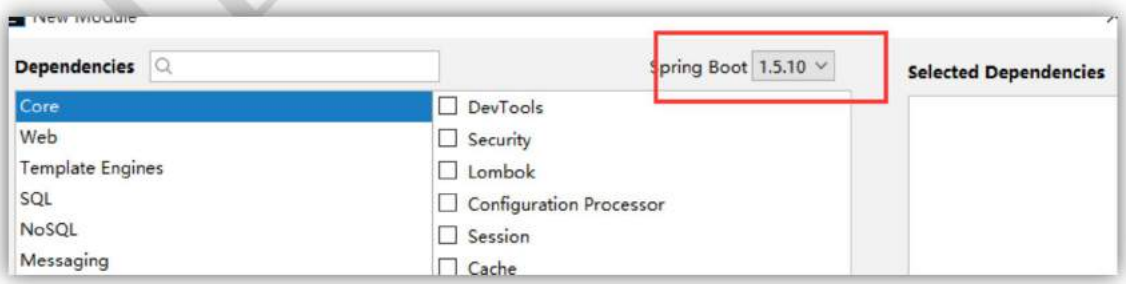
业务流程图

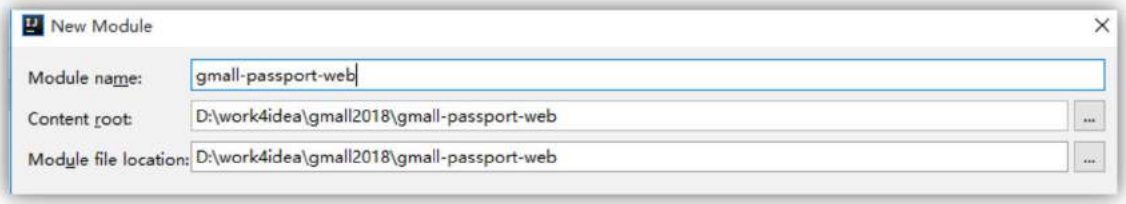

认证中心
WEB应用
检查cookie是否有token
不存在token、重定向到认证中心
提示用户登录、注册
用户填写登录信息
核对用户信息
用户登录成功
带token跳转原WEB应用
如果是带token的跳转，则写入cookie中，并继续打开业务功能页面。
业务功能页面
用户继续访问
检查cookie
存在token，提交认证中心认证
认证身份
业务功能页面

二、 认证中心模块

1 搭建认证中心模块







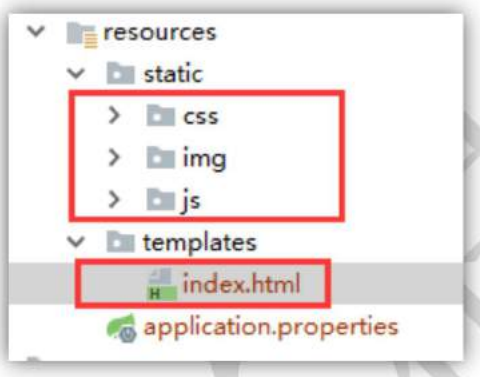
pom.xml

```
<?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.atguigu.gmall</groupId>
    <artifactId>gmall-passport-web</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>gmall-passport-web</name>
    <description>Demo project for Spring Boot</description>

    <parent>
        <groupId>com.atguigu.gmall</groupId>
        <artifactId>gmall-parent</artifactId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <dependencies>

        <dependency>
            <groupId>com.atguigu.gmall</groupId>
            <artifactId>gmall-interface</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>

        <dependency>
            <groupId>com.atguigu.gmall</groupId>
            <artifactId>gmall-web-util</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>


    </dependencies>

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


</project>
```

application.properties

```
server.port=8086

spring.thymeleaf.cache=false

spring.thymeleaf.mode=LEGACYHTML5

spring.dubbo.application.name=passport-web
spring.dubbo.registry.protocol=zookeeper
spring.dubbo.registry.address=192.168.67.163:2181
spring.dubbo.base-package=com.atguigu.gmall
spring.dubbo.protocol.name=dubbo
spring.dubbo.consumer.timeout=100000
spring.dubbo.consumer.check=false
```

导入静态资源和登录页面



把 index.html 中的路径从 ../static/ 换成 /

host 配置

```
192.168.67.163    passport.atguigu.com
```

nginx 配置

```
    upstream passport.atguigu.com{
        server 192.168.67.1:8086;
    }
   server {
      listen 80;
      server_name passport.atguigu.com;
      location / {
          proxy_pass http://passport.atguigu.com;
          proxy_set_header X-forwarded-for $proxy_add_x_forwarded_for;
      }
```

```
    }
```

2 登录页面 html

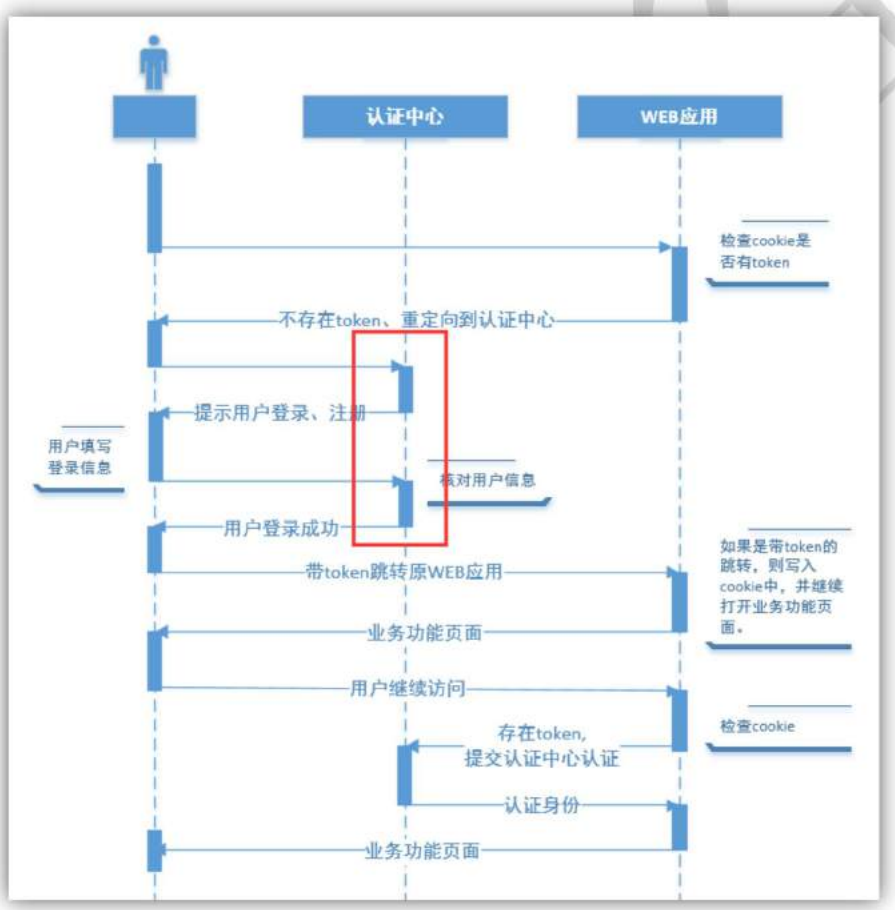
```
<div class="si  bom1 tab" style="display: block;">
    <div    class="error">
        请输入账户名和密码
    </div>

    <form id="loginForm" action="/login" method="post">
    <ul>
        <li class="top  1">
            <img src="/img/user_03.png" class="err_img1"/>

            <input type="text" name="loginName" placeholder="  邮箱/用户名/已验证手机"    class="user"/>
        </li>
        <li>
            <img src="/img/user  06.png" class="err  img2"/>
            <input type="password" name="passwd" placeholder="  密码" class="password"/>

        </li>
        <li class="bri">
            <a href="">忘记密码</a>
        </li>
        <li class="ent"><button id="btn2" class="btn2"><a class="a">登     录</a></button></li>

    </ul>
        <input type="hidden" id="originUrl" name="originUrl" th:value="${originUrl}"/>
    </form>
</div>
```

页面中的 js

```
btn2.onclick=function(){
    if(user.value == '' || pass.value == ''){
        err.style.display='block';
        user.style.border='1px solid red';
        pass.style.border='1px solid red';

        err_img1.src='img/img11.png';
        err_img2.src='img/img22.png';
    }else{
        $.post("/login",$("#loginForm").serialize(),function (data) {
            if(data&&data!='fail'){
                 var originUrl = $("#originUrl").val();
                originUrl = decodeURIComponent(originUrl);

                                var idx=originUrl.indexOf('?');

                                if(idx<0){
                                    originUrl+='?'
                }else{
                                    originUrl+='&'
                }
                 window.location.href=originUrl+"newToken="+data;
            }else{
                                $(".error").text("用户名密码错误！ ");
                $(".error").show();
            }
                    }  );
```

```
    }user.onfocus=function(){
    err_img1.src='img/grow1.png';
    user.style.border='1px solid #999';
}

}
```

3 登录功能



3.1 思路：

1、 用接受的用户名密码核对后台数据库

2、 将用户信息加载到写入 redis，redis 中有该用户视为登录状态。
3、 用 userId+当前用户登录 ip 地址+密钥生成 token
4、 重定向用户到之前的来源地址，同时把 token 作为参数附上。

3.2 核对后台登录信息+用户登录信息载入缓存

UserManageServiceImpl

```
public UserInfo login(UserInfo userInfo){
    String passwd = DigestUtils.md5Hex(userInfo.getPasswd());
    userInfo.setPasswd(passwd);

    UserInfo userInfoResult = userInfoMapper.selectOne(userInfo);
    if(userInfoResult!=null){

        String userInfoKey="user:"+userInfoResult.getId()+"info";

        Jedis jedis = redisUtil.getJedis();
        String userInfoJson = JSON.toJSONString(userInfoResult);
        jedis.setex(userInfoKey,UserConst.sessionExpire,userInfoJson);

        return userInfoResult ;
    }else{
        return null;
    }
}
```

3.3 生成 token

JWT 工具

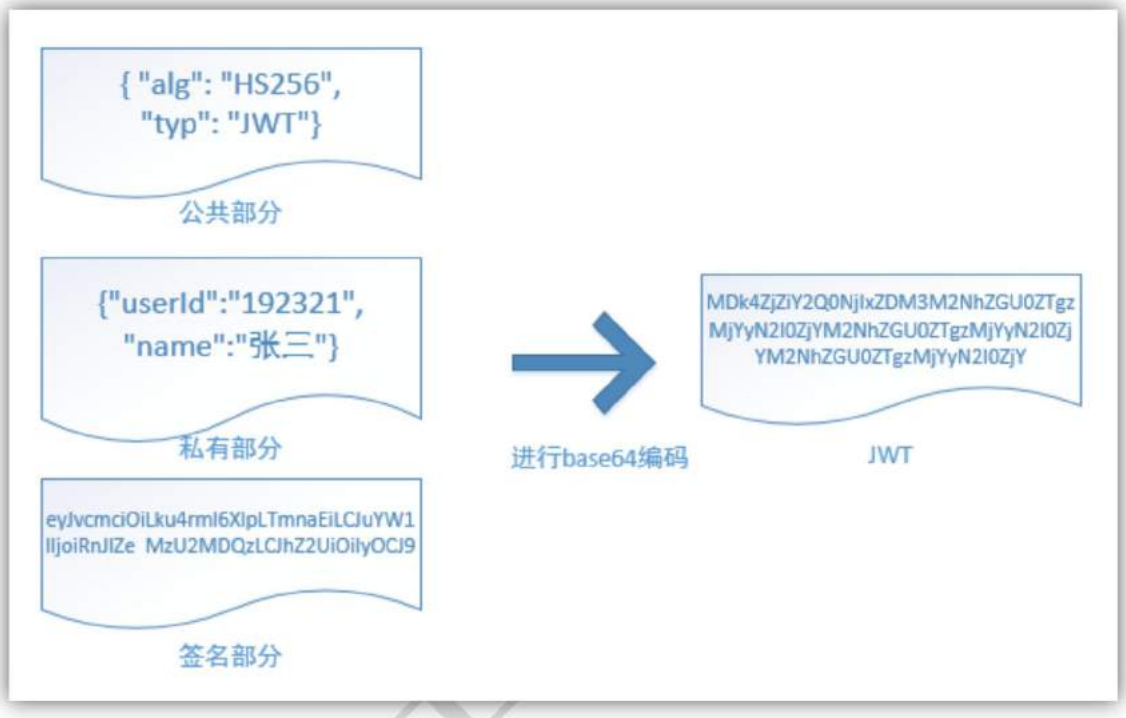
JWT（Json Web Token） 是为了在网络应用环境间传递声明而执行的一种基于 JSON 的开放标准。

JWT 的声明一般被用来在身份提供者和服务提供者间传递被认证的用户身份信息，以便于从资源服务器获取资源。比如用在用户登录上

JWT 最重要的作用就是对 token 信息的防伪作用。

JWT 的原理，

一个 JWT 由三个部分组成：公共部分、私有部分、签名部分。最后由这三者组合进行 base64 编码得到 JWT。



1、 公共部分

主要是该 JWT 的相关配置参数，比如签名的加密算法、格式类型、过期时间等等。

2、 私有部分

用户自定义的内容，根据实际需要真正要封装的信息。

3、 签名部分

根据用户信息+盐值+密钥生成的签名。如果想知道 JWT 是否是真实的只要把 JWT 的信息取出来，加上盐值和服务器中的密钥就可以验证真伪。所以不管由谁保存 JWT，只要没有密钥就无法伪造。

4、 base64 编码，并不是加密，只是把明文信息变成了不可见的字符串。但是其实只要用一些工具就可以吧 base64 编码解成明文，所以不要在 JWT 中放入涉及私密的信息，因为实际上 JWT 并不是加密信息。

pom 依赖　放到 gmall-web-util 中

```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.0</version>
</dependency>
```

使用：

制作 JWT 的工具类

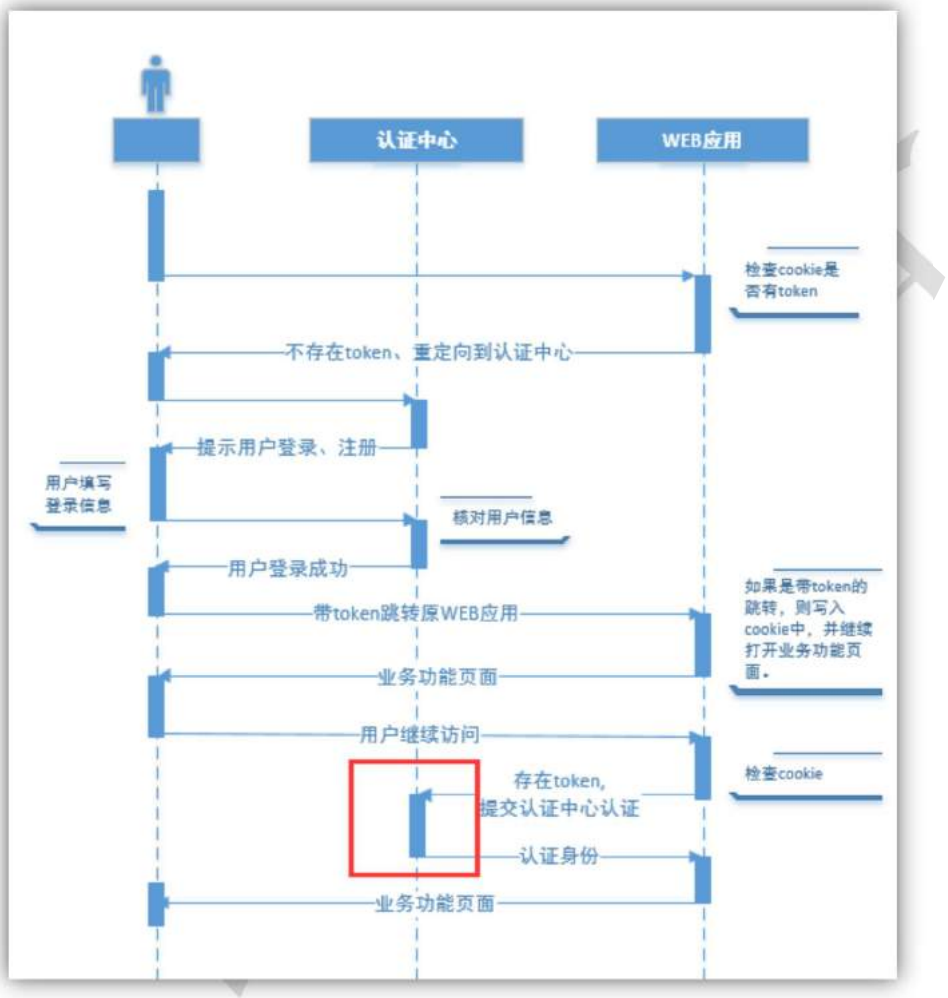
```
public class JwtUtil {

    public static String encode(String key,Map<String,Object> param,String salt){
        if(salt!=null){
            key+=salt;
        }
        JwtBuilder jwtBuilder = Jwts.builder().signWith(SignatureAlgorithm.HS256,key);

        jwtBuilder = jwtBuilder.setClaims(param);

        String token = jwtBuilder.compact();
        return token;

    }


    public   static Map<String,Object>   decode(String token ,String key,String salt){
        Claims claims=null;
        if (salt!=null){
            key+=salt;
        }
        try {
            claims= Jwts.parser().setSigningKey(key).parseClaimsJws(token).getBody();
        } catch ( JwtException e) {
            return null;
        }
        return   claims;
    }
}
```

3.4 Controller

PassportControler

```
@RequestMapping(value ="login",method = RequestMethod.POST)
@ResponseBody
public String   login(UserInfo userInfo,HttpServletRequest httpServletRequest){
    String remoteAddr = httpServletRequest.getHeader("x-forwarded-for");
    String userId = userManageService.login(userInfo);
    if(userId==null){
        return "fail";
    }else{
        Map map=new HashMap();
        map.put("userId",userId);
        map.put("nickName",userInfoResult.getNickName());

        String token = JwtUtil.encode(signKey, map, remoteAddr);
        return token;
    }

}
```

4 验证功能



功能：当业务模块某个页面要检查当前用户是否登录时，提交到认证中心，认证中心进行检查校验，返回登录状态、用户 Id 和用户名称。

4.1 思路：

1、 利用密钥和 IP 检验 token 是否正确，并获得里面的 userId
2、 用 userId 检查 Redis 中是否有用户信息,如果有延长它的过期时间。
3、 登录成功状态返回。

4.2 代码

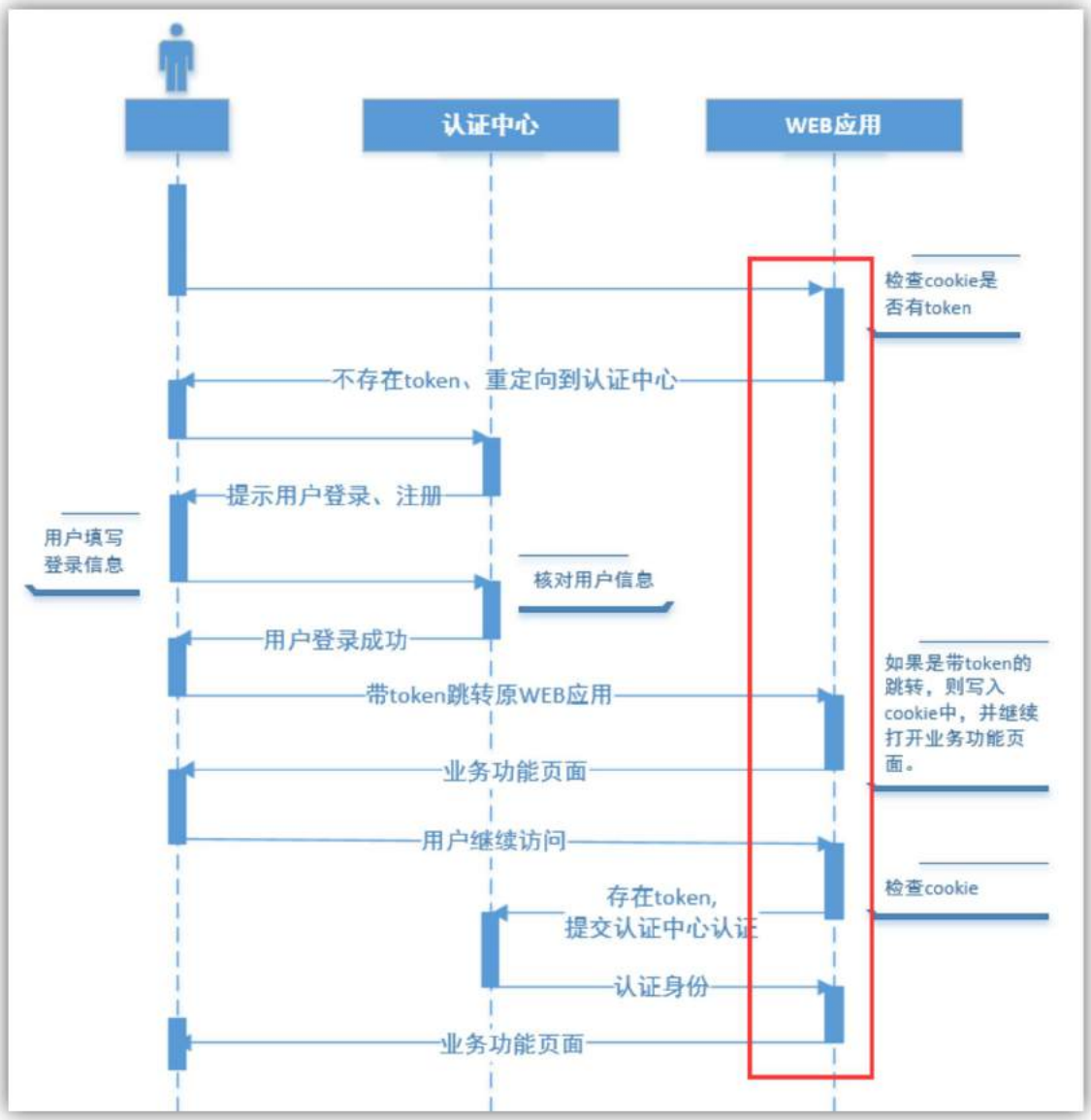
UserManageServiceImpl

```
public boolean verify(String userId){
        Jedis jedis = redisUtil.getJedis();
        String userInfoKey="user:"+userId+"info";
        Boolean exists = jedis.exists(userInfoKey);
        if(exists) {
              jedis.expire(userInfoKey, UserConst.sessionExpire);
        }
        return exists;
}
```

UserController

```
@RequestMapping(value ="verify",method = RequestMethod.POST)
@ResponseBody
public String verify(HttpServletRequest httpServletRequest){
        String token = httpServletRequest.getParameter("token");
        String curIp=httpServletRequest.getParameter("currentIp");
        Map<String, Object> map = JwtUtil.decode(token, signKey, curIp);
        JSONObject jsonObject=new JSONObject();
        String userId =(String) map.get("userId");
        boolean verify = userManageService.verify(userId);

        return    Boolean.toString(verify) ;

}
```

三、 业务模块页面登录情况检查



问题：

1 、由认证中心签发的 token 如何保存？保存到浏览器的 cookie 中

2 、难道每一个模块都要做一个 token 的保存功能？ 拦截器

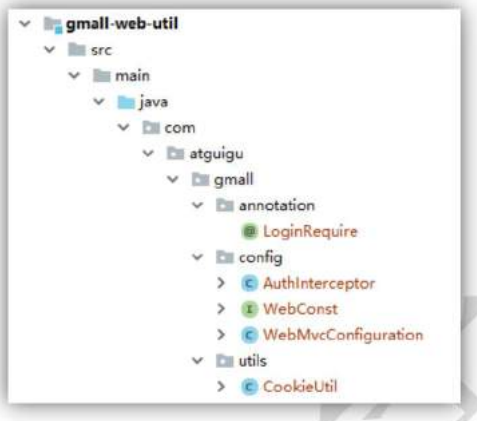
3 、如何区分请求是否一定要登录？自定义注解

1 加入拦截器

首先这个验证功能是每个模块都要有的，也就是所有 web 模块都需要的。在每个 controller 方法进入前都需要进行检查。可以利用在 springmvc 中的拦截器功能。

因为咱们是多个 web 模块分布式部署的，所以不能写在某一个 web 模块中，可以一个公共的 web 模块，就是 gmall-web-util 中。

位置：



首先自定义一个拦截器，继承成 springmvc 的 HandlerInterceptorAdapter，通过重新它的 preHandle 方法实现，业务代码前的校验工作

```
@Configuration
public class WebMvcConfiguration extends WebMvcConfigurerAdapter{
    @Autowired
    AuthInterceptor authInterceptor;

    @Override
    public void addInterceptors(InterceptorRegistry registry){
        registry.addInterceptor(authInterceptor).addPathPatterns("/**");
        super.addInterceptors(registry);
    }
}
```

2 登录成功后跳转回来的处理

登录成功后写入 cookie。

```
@Component
public class AuthInterceptor extends HandlerInterceptorAdapter


public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws
Exception {
    String newToken = request.getParameter("newToken");
    if(newToken!=null&&newToken.length()>0){
        CookieUtil.setCookie(request,response,"token",newToken,WebConst.cookieExpire,false);
    }


    return true;
}
}
```

其中用到了 CookieUtil 的工具。代码如下：

主要三个方法：从 cookie 中获得值，把值存入 cookie，设定 cookie 的作用域。

```
public class CookieUtil {


    public static String getCookieValue(HttpServletRequest request, String cookieName, boolean isDecoder)
{
        Cookie[] cookies = request.getCookies();
        if (cookies == null || cookieName == null){
            return null;
        }
        String retValue = null;
        try {
            for (int i = 0; i < cookies.length; i++) {
                if (cookies[i].getName().equals(cookieName)) {
                    if (isDecoder) {//如果涉及中文
                        retValue = URLDecoder.decode(cookies[i].getValue(), "UTF-8");
                    } else {
                        retValue = cookies[i].getValue();
                    }
                    break;
                }
            }
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
        return retValue;
    }


    public static void setCookie(HttpServletRequest request, HttpServletResponse response, String
cookieName, String cookieValue, int cookieMaxage, boolean isEncode) {
        try {
            if (cookieValue == null) {
                cookieValue = "";
            } else if (isEncode) {
                cookieValue = URLEncoder.encode(cookieValue, "utf-8");
            }
            Cookie cookie = new Cookie(cookieName, cookieValue);
```

```
                if (cookieMaxage >= 0)
                    cookie.setMaxAge(cookieMaxage);
                if (null != request)// 设置域名的 cookie
                    cookie.setDomain(getDomainName(request));
                cookie.setPath("/");
                response.addCookie(cookie);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }


        /**
         * 得到 cookie 的域名
         */
        private static final String getDomainName(HttpServletRequest request) {
            String domainName = null;

            String serverName = request.getRequestURL().toString();
            if (serverName == null || serverName.equals("")) {
                domainName = "";
            } else {
                serverName = serverName.toLowerCase();
                serverName = serverName.substring(7);
                final int end = serverName.indexOf("/");
                serverName = serverName.substring(0, end);
                final String[] domains = serverName.split("\\.");
                int len = domains.length;
                if (len > 3) {
                    // www.xxx.com.cn
                    domainName = domains[len - 3] + "." + domains[len - 2] + "." + domains[len - 1];
                } else if (len <= 3 && len > 1) {
                    // xxx.com or xxx.cn
                    domainName = domains[len - 2] + "." + domains[len - 1];
                } else {
                    domainName = serverName;
                }
            }

            if (domainName != null && domainName.indexOf(":") > 0) {
                String[] ary = domainName.split("\\:");
                domainName = ary[0];
            }
            System.out.println("domainName = " + domainName);
            return domainName;
        }
}
```

3 检查 cookie 中是否有 token

要做的工作：

4 检查 cookie 中是否有 token,如果有把 cookie 中的昵称取放入页面 request 属性中.

5 检查是否需要验证登录。

如果需要，调用认证模块接口

如果认证通过程序照常执行

如果认证不通过，跳转到登录页面。

6 检查是否是登陆页面跳转回来的，如果附带新的 token 则把 token 保存到 cookie 中。

思路：

。

AuthInterceptor

```
public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws
Exception {


        String newToken = request.getParameter("newToken");
        if(newToken!=null&&newToken.length()>0){
            CookieUtil.setCookie(request,response,"token",newToken,WebConst.cookieExpire,false);
        }

//1 进行如果能从 cookie 把 token 取出来，进行解析，显示页面上。
String token = CookieUtil.getCookieValue(request, "token", false);
String userId=null;
if(token!=null){
        Base64UrlCodec base64UrlCodec=new Base64UrlCodec();
        //  两个“.”之间的部分是实际内容
        String   tokenForDecode= StringUtils.substringBetween(token, ".");

        byte[] tokenByte = base64UrlCodec.decode(tokenForDecode);
        String tokenJson=new String(tokenByte,"UTF-8");
        System.out.println("tokenJson = " + tokenJson);

        JSONObject jsonObject = JSON.parseObject( tokenJson);

        userId = jsonObject.getString("userId");
        String nickName = jsonObject.getString("nickName");

        request.setAttribute("nickName",nickName);
}



            return true;
        }
}
```

4 检验方法是否需要验证用户登录状态

为了方便程序员在 controller 方法上标记，可以借助自定义注解的方式。

比如某个 controller 方法需要验证用户登录，在方法上加入自定义的@LoginRequie。像这样

```
@LoginRequire
@RequestMapping("/{skuId}.html")
public String getSkuInfo(@PathVariable("skuId") String skuId, Model model){

    SkuInfo skuInfo = manageService.getSkuInfo(skuId);
    model.addAttribute( s: "skuInfo",skuInfo);

    // 取出该spu的所有属性和属性值
/*      List<SpuSaleAttr> spuSaleAttrList = manageService.getSpuSaleAttrList(skuInfo.
```

如果方法被加了注解，在拦截器中就可以捕捉到。

添加自定义注解

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface LoginRequire {

    boolean autoRedirect() default true;
}
```

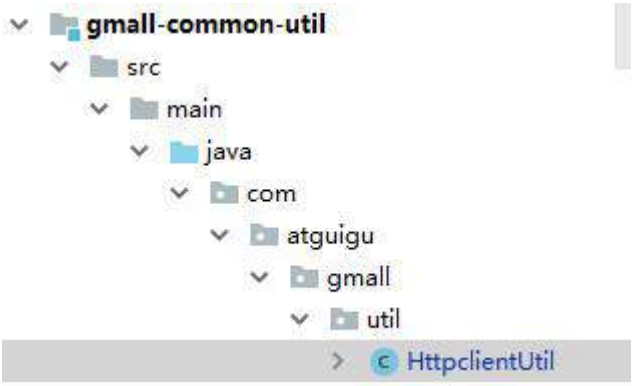
在拦截方法 preHandle 方法中继续添加，检验登录的代码。

```
//检查是否需要验证用户已经登录
HandlerMethod handlerMethod =(HandlerMethod) handler;
LoginRequire methodAnnotation = handlerMethod.getMethodAnnotation(LoginRequire.class);
if(methodAnnotation!=null){
    String currentIp = request.getHeader("x-forwarded-for");

    Map map=new HashMap();
    map.put("currentIp",currentIp);
    map.put("token",token);
    String result=null;
    if(token !=null) {
        result = HttpclientUtil.doPost(WebConst.VERIFY_URL, map);
    }
    if(result!=null&&result.equals("true")){
        request.setAttribute("userId",userId); //只有验证过才能取到userId
        return true;
    }else{
        if(methodAnnotation.autoRedirect()) {
            String url = URLEncoder.encode(request.getRequestURL().toString(), "utf-8");

            response.sendRedirect(WebConst.LOGIN_URL + "?originUrl=" + url);
            return false;
        }
    }
}
```

以上方法，检查业务方法是否需要用户登录，如果需要就把 cookie 中的 token 和当前登录人的 ip 地址发给远程服务器进行登录验证，返回的 result 是验证结果 true 或者 false。如果验证未登录，直接重定向到登录页面。

以上使用到了一个自定义的 HttpclientUtil 工具类，放在 gmall-common-util 模块中。专门负责通过 restful 风格调用接口。

位置：



代码如下。

```
public class HttpclientUtil {

    public static String doGet(String url)     {
        // 创建 Httpclient 对象
        CloseableHttpClient httpclient = HttpClients.createDefault();
        // 创建 http GET 请求
        HttpGet httpGet = new HttpGet(url);
        CloseableHttpResponse response = null;
        try {
            // 执行请求
            response = httpclient.execute(httpGet);
            // 判断返回状态是否为 200
            if (response.getStatusLine().getStatusCode() == HttpStatus.SC_OK) {
                HttpEntity entity = response.getEntity();
                String result = EntityUtils.toString(entity, "UTF-8");
                EntityUtils.consume(entity);
                httpclient.close();
                return result;
            }
            httpclient.close();
        }catch (IOException e){
            e.printStackTrace();
            return null;
        }
        return    null;
    }



    public static String doPost(String url, Map<String,String> paramMap)     {
        // 创建 Httpclient 对象
        CloseableHttpClient httpclient = HttpClients.createDefault();
        // 创建 http Post 请求
        HttpPost httpPost = new HttpPost(url);
        CloseableHttpResponse response = null;
        try {
            List<BasicNameValuePair> list=new ArrayList<>();
            for (Map.Entry<String, String> entry : paramMap.entrySet()) {
                list.add(new BasicNameValuePair(entry.getKey(),entry.getValue())) ;
            }
            HttpEntity httpEntity=new UrlEncodedFormEntity(list,"utf-8");
```

```
                httpPost.setEntity(httpEntity);
                // 执行请求
                response = httpclient.execute(httpPost);

                // 判断返回状态是否为 200
                if (response.getStatusLine().getStatusCode() == HttpStatus.SC_OK) {
                    HttpEntity entity = response.getEntity();
                    String result = EntityUtils.toString(entity, "UTF-8");
                    EntityUtils.consume(entity);
                    httpclient.close();
                    return result;
                }
                httpclient.close();
            }catch (IOException e){
                e.printStackTrace();
                return null;
            }

            return   null;
        }
}
```

WebConst 是常量类，

```
public interface WebConst {
    //登录页面
    public final static String LOGIN_URL="http://passport.atguigu.com/index";
    //认证接口
    public final static String VERIFY_URL="http://passport.atguigu.com/verify";
    //cookie 的有效时间：默认给 7 天
    public final static int cookieMaxAge=7*24*3600;
}
```

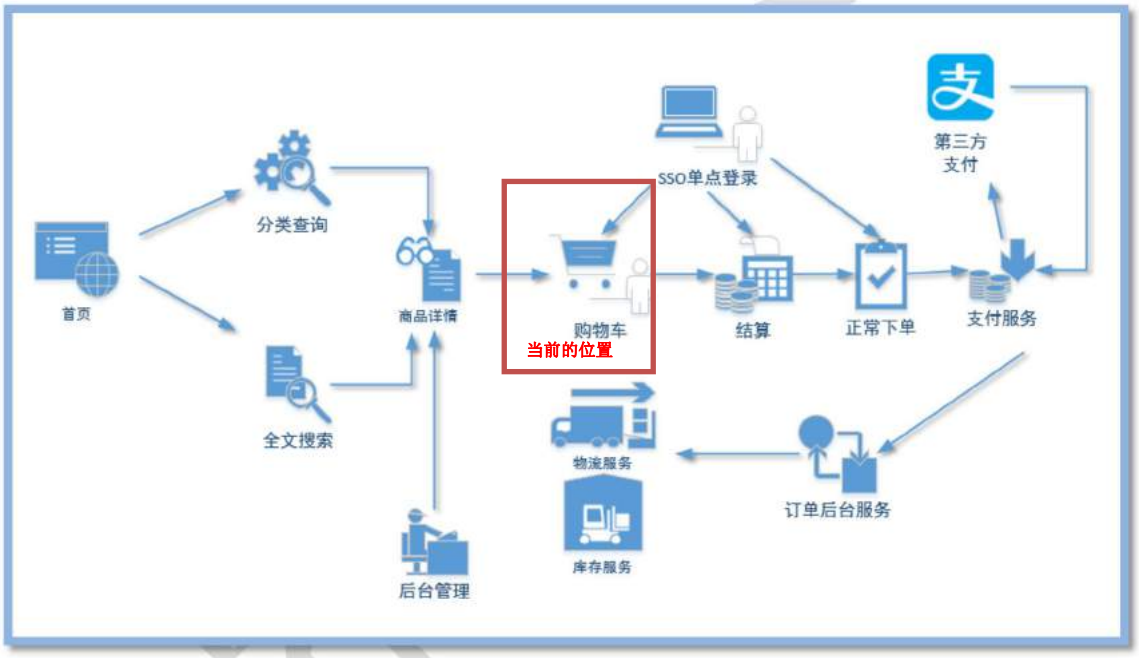
四、 测试效果

购物车

版本：V 1.0

www.atguigu.com

一、 购物车业务简介



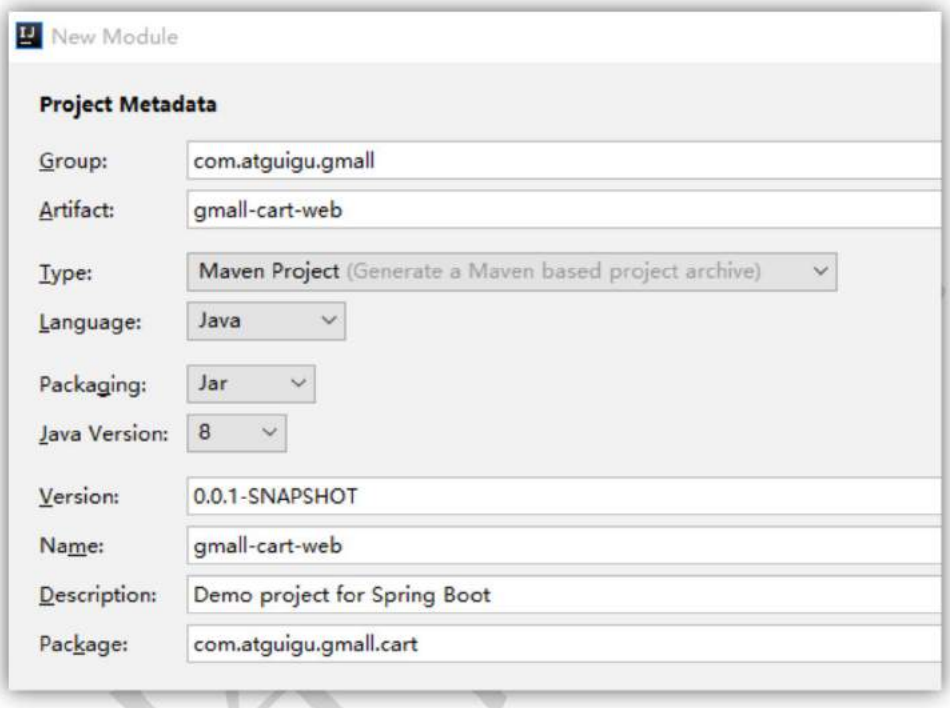
购物车模块要能过存储顾客所选的的商品，记录下所选商品，还要能随时更新，当用户决定购买时，用户可以选择决定购买的商品进入结算页面。

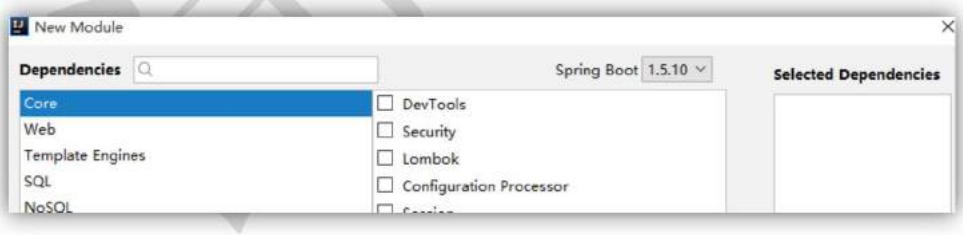
功能要求：

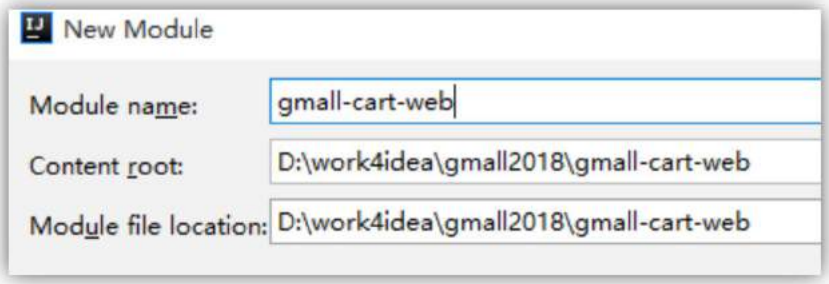
1) 要持久化，保存到数据库中。
2) 利用缓存提高性能。
3) 未登录状态也可以存入购物车，一旦用户登录要进行合并操作。

二 购物车模块搭建

1 gmall-cart-web







pom.xml

```
<?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.atguigu.gmall</groupId>
    <artifactId>gmall-cart-web</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>gmall-cart-web</name>
    <description>Demo project for Spring Boot</description>

    <parent>
        <groupId>com.atguigu.gmall</groupId>
        <artifactId>gmall-parent</artifactId>
        <version>1.0-SNAPSHOT</version>
    </parent>


    <dependencies>

    <dependency>
        <groupId>com.atguigu.gmall</groupId>
        <artifactId>gmall-interface</artifactId>
        <version>1.0-SNAPSHOT</version>
    </dependency>

    <dependency>
        <groupId>com.atguigu.gmall</groupId>
        <artifactId>gmall-web-util</artifactId>
        <version>1.0-SNAPSHOT</version>
    </dependency>
    </dependencies>

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


</project>
```

application.properties

```
server.port=8087
spring.thymeleaf.cache=false
spring.thymeleaf.mode=LEGACYHTML5

spring.dubbo.application.name=cart-web
spring.dubbo.registry.protocol=zookeeper
spring.dubbo.registry.address=192.168.67.163:2181
spring.dubbo.base-package=com.atguigu.gmall
spring.dubbo.protocol.name=dubbo
spring.dubbo.consumer.timeout=100000
spring.dubbo.consumer.check=false
```

host 文件

```
# gmall
192.168.67.163 cart.gmall.com passport.atguigu.com item.gmall.com list.gmall.com
manage.gmall.com www.gmall.com resource.gmall.com
```

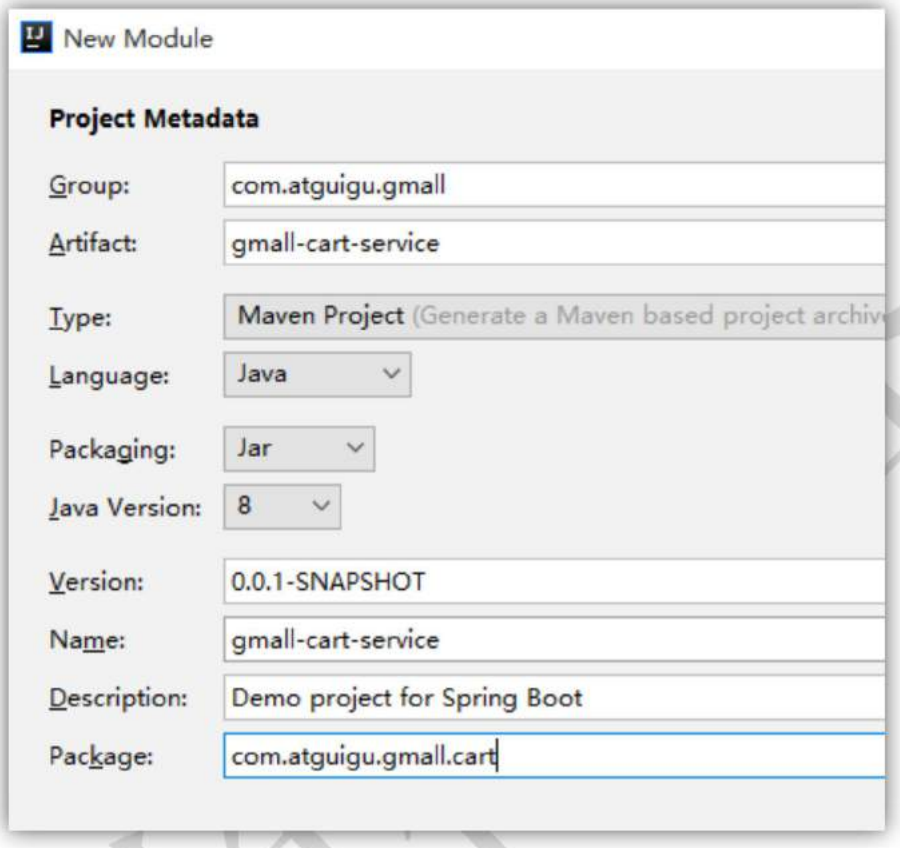
nginx.conf

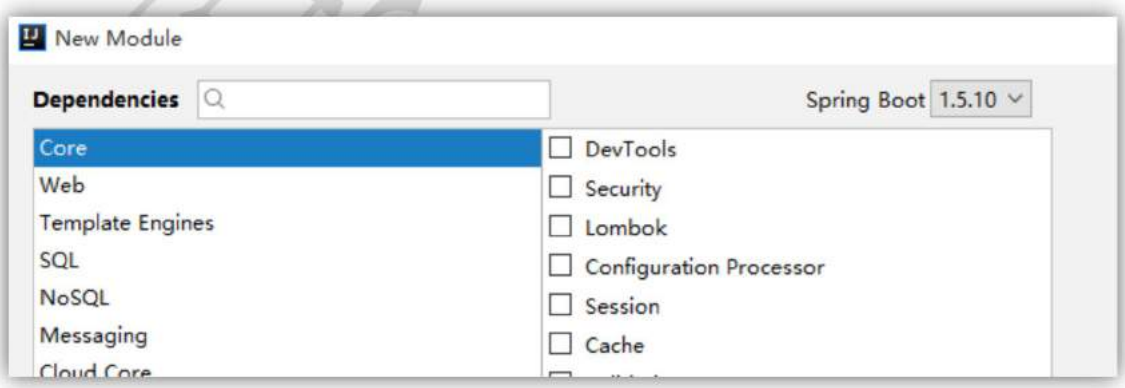
```
    upstream cart.gmall.com{
        server 192.168.67.1:8087;
    }
  server {
    listen 80;
    server_name cart.gmall.com;
    location / {
        proxy_pass http://cart.gmall.com;
        proxy_set_header X-forwarded-for $proxy_add_x_forwarded_for;
    }
  }
```

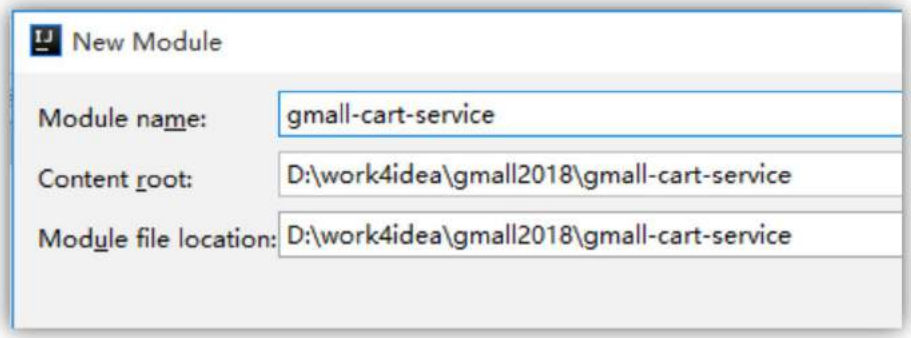
GmallCartWebApplication 提到跟 cart 目录同级

拷入静态文件和页面

2 gmall-cart-service







pom.xml

```
<?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.atguigu.gmall</groupId>
    <artifactId>gmall-cart-service</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>gmall-cart-service</name>
    <description>Demo project for Spring Boot</description>

    <parent>
        <groupId>com.atguigu.gmall</groupId>
        <artifactId>gmall-parent</artifactId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <dependencies>

        <dependency>
            <groupId>com.atguigu.gmall</groupId>
            <artifactId>gmall-interface</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>

        <dependency>
            <groupId>com.atguigu.gmall</groupId>
            <artifactId>gmall-service-util</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>

    </dependencies>
```

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


</project>
```

application.properties

```
spring.datasource.url=jdbc:mysql://59.110.141.236:3306/gmall?characterEncoding=UTF-8
spring.datasource.username=root
spring.datasource.password=123456

logging.level.root=debug
logging.level.com.atguigu.gmall.usermanage.mapper=debug

server.port=8077

spring.dubbo.application.name=usermanage
spring.dubbo.registry.protocol=zookeeper
spring.dubbo.registry.address=192.168.67.163:2181
spring.dubbo.base-package=com.atguigu.gmall
spring.dubbo.protocol.name=dubbo

spring.redis.host=192.168.67.163
spring.redis.port=6379
spring.redis.database=0
```

GmallCartServiceApplication 提到跟 cart 目录同级

三、功能—添加入购物车

1 功能解析：

1、 根据 skuId 查询出商品详情 skuInfo

2、 把 skuInfo 信息对应保存到购物车

3、 返回成功页面

2 设计购物车的数据结构

cart_info(购物车表)	
id	编号
user_id	用户id
sku_id	skuid
cart_price	放入购物车时价格
sku_num	数量
img_url	图片文件
sku_name	sku名称（冗余）

3 实体 bean—CartInfo

```
public class CartInfo implements Serializable{

    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    @Column
    String id;
    @Column
    String userId;
    @Column
    String skuId;
    @Column
    BigDecimal cartPrice;
    @Column
```

```
    Integer skuNum;
    @Column
    String imgUrl;
    @Column
    String skuName;

    @Transient
    BigDecimal skuPrice;

    @Transient
    String isChecked="0";
}
```

4 Redis 中的结构

利用 Hash 结构存储：

key:　　“user:[userId]:cart”

field:　　[skuId]

value:　　CartInfo　(Json)

5 gmall-cart-service 模块业务方法

1　思路：

1、 先检查该用户的购物车里是否已经有该商品

2、 如果有商品，只要把对应商品的数量增加上去就可以，同时更新缓存

3、 如果没有该商品，则把对应商品插入到购物车中，同时插入缓存。

2　CartServiceImpl

```
public void addToCart(String userId,SkuInfo skuInfo,Integer skuNum){
```

```
        String userCartKey="user:"+userId+":cart";
    //插入前先检查
        CartInfo cartInfoQuery=new CartInfo();
        cartInfoQuery.setSkuId(skuInfo.getId());
        cartInfoQuery.setUserId(userId);
        CartInfo cartInfoExist = cartInfoMapper.selectOne(cartInfoQuery);
        if(cartInfoExist!=null) {
            cartInfoExist.setSkuPrice(skuInfo.getPrice());
            cartInfoExist.setSkuNum(cartInfoExist.getSkuNum() + skuNum);
            cartInfoMapper.updateByPrimaryKeySelective(cartInfoExist);
            //更新缓存
            Jedis jedis = redisUtil.getJedis();
            jedis.hset(userCartKey,skuInfo.getId(),JSON.toJSONString(cartInfoExist) );
            jedis.close();
        }else{
            //插入数据库
            CartInfo cartInfo=new CartInfo();
            cartInfo.setSkuId(skuInfo.getId());
            cartInfo.setCartPrice(skuInfo.getPrice());
            cartInfo.setImgUrl(skuInfo.getSkuDefaultImg());
            cartInfo.setSkuName(skuInfo.getSkuName());
            cartInfo.setUserId(userId);
            cartInfo.setSkuNum(skuNum);
            cartInfo.setSkuPrice(skuInfo.getPrice());
            cartInfoMapper.insertSelective(cartInfo);

            //更新缓存
            Jedis jedis = redisUtil.getJedis();
            jedis.hset("user:"+userId+":cart",skuInfo.getId(),JSON.toJSONString(cartInfo));
            Long ttl = jedis.ttl("user:" + userId + ":info");
            jedis.expire(userCartKey, ttl.intValue());
            jedis.close();
        }

}
```

6 Web 模块业务方法（gmall-cart-web）

1 思路：

1、 获得参数：skuId 、num

2、 判断该用户是否登录，用 userId 判断

3、 如果登录则调用后台的 service 的业务方法

4、 如果未登录，要把购物车信息暂存到 cookie 中。

5、 实现利用 cookie 保存购物车的方法。

2 CartController

```
@Controller
public class CartController {

    @Reference
    ManageService manageService;

    @Reference
    CartService    cartService;

    @Autowired
    CartCookieHandler cartCookieHandler;

    @RequestMapping(value = "addToCart",method = RequestMethod.POST)
    @LoginRequire(autoRedirect = false)
    public String addCart(HttpServletRequest request, HttpServletResponse response){
        String skuId = request.getParameter("skuId");
        Integer num=Integer.parseInt( request.getParameter("num"));

        SkuInfo skuInfo = manageService.getSkuInfo(skuId);
        String userId=(String)request.getAttribute("userId");
        if(userId!=null){
            cartService.addToCart(userId,skuInfo,num);
        }else{
            cartCookieHandler.addToCart(skuInfo, num ,request,response);
        }
        request.setAttribute("skuInfo",skuInfo);
        request.setAttribute("num",num);
        return "success";
    }
```

7 实现利用 cookie 保存购物车的 CartCookieHandler

1 思路：

和 service 模块的方法类似

1、 先查询出来在 cookie 中的购物车，反序列化成列表。

2、 通过循环比较有没有该商品

3、 如果有，增加数量

4、 如果没有，增加商品

5、 然后把列表反序列化，利用之前最好的 CookieUtil 保存到 cookie 中。

2 CartCookieHandler

```
@Component
public class CartCookieHandler {

String cartCookieName="CART";

   public void addToCart(SkuInfo cartSkuInfo, Integer num, HttpServletRequest request,
                                        HttpServletResponse response) {
      try {
            List<CartInfo> cartList = getCartList(request);

            CartInfo cart = null;
            if (cartList != null && cartList.size() > 0) {
                  for (CartInfo c : cartList) {
                        // 判断购物车中是否存在该商品
                        if (c.getSkuId().equals(cartSkuInfo.getId())) {
                              cart = c;
                              break;
                        }
                  }
            }

            if (cart == null) {
                  // 当前的购物车没有该商品
                  cart = new CartInfo();
                  cart.setSkuId(cartSkuInfo.getId());
                  cart.setSkuName(cartSkuInfo.getSkuName());
                  // 设置商品主图
                  cart.setImgUrl(cartSkuInfo.getSkuDefaultImg());
                  cart.setSkuPrice(cartSkuInfo.getPrice());
                  cart.setCartPrice(cartSkuInfo.getPrice());
                  cart.setSkuNum(num);

                  cartList.add(cart);
            } else {
                  // 在购物车中存在该商品
                  cart.setSkuNum(cart.getSkuNum() + num);
            }
            // 设置购物车的商品，过期时间 7 天
            CookieUtil.setCookie(request, response, cartCookieName, JSON.toJSONString(cartList),
WebConst.cookieMaxAge,true);
```

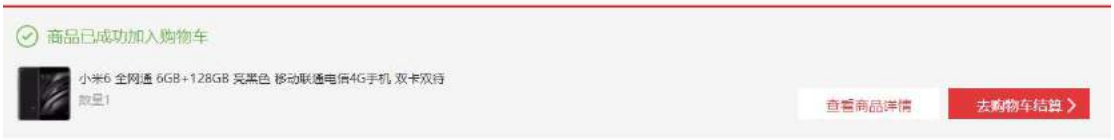
```
        } catch (Exception e) {
            e.printStackTrace();
        }

}

public List<CartInfo> getCartList(HttpServletRequest request) {

        try {
            String cartListJson = CookieUtil.getCookieValue(request,cartCookieName , true);
            if (cartListJson!= null) {
                List<CartInfo> cartList = JSON.parseArray(cartListJson, CartInfo.class);
                return cartList;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return new ArrayList<>();
}

}
```

3 添加成功后的展示页面

succuess.html

```
<div class="p-item">
        <div class="p-img">
        <a      href="/javascript:;"      target="_blank"><img      style="height:      60px;width:60px;"
th:src="@{${skuInfo.skuDefaultImg}}"></a>
        </div>
        <div class="p-info">
        <div class="p-name">
        <a    th:href="'http://item.gmall.com/'+${skuInfo.id}+'.html'"    th:title="${skuInfo.skuName}"
th:text="${skuInfo.skuName}">  商品名称</a>
</div>
<div class="p-extra"> <span class="txt" th:text="'数量:'+${num}">    数量：XX</span></div>
</div>
```

8 测试效果



四、功能—展示购物车列表

1 功能解析

1、 展示购物中的信息

2、 如果**用户已登录**从缓存中取值，如果缓存没有，加载数据库。

3、 如果**用户未登录**从 cookie 中取值。

2 CartServiceImpl

注意点：

1、redis 中取出来要进行反序列化

2、redis 的 hash 结构是无序的，要进行排序（可以用时间戳或者主键 id，倒序排序）

3、如果 redis 中没有要从数据库中查询，要连带把最新的价格也取出来，默认要显示最新价格而不是当时放入购物车的价格，如果考虑用户体验可以把两者的差价提示给用户。

4、加载入缓存时一定要设定失效时间，保证和用户信息的失效时间一致即可。

```
public List<CartInfo> getCartList(String userId){
    //优先从缓存中取值
    Jedis jedis = redisUtil.getJedis();
    List<String> skuJsonlist = jedis.hvals("user:" + userId + ":cart");
      List<CartInfo> cartInfoList=new ArrayList<>();

    if(skuJsonlist!=null &&skuJsonlist.size()!=0){
        //序列化
```

```
            for (String skuJson : skuJsonlist) {
                CartInfo cartInfo = JSON.parseObject(skuJson, CartInfo.class);
                cartInfoList.add( cartInfo);
            }
            //缓存中的值取出来是没有序的 用 id 进行排序
            cartInfoList.sort(new Comparator<CartInfo>() {
                @Override
                public int compare(CartInfo o1, CartInfo o2) {
                    return Long.compare(Long.parseLong(o2.getId()) ,Long.parseLong(o1.getId())) ;
                }
            });
            return cartInfoList;
        }else{
            //如果缓存没有就总数据库中加载
            cartInfoList = loadCartCache(    userId);
            return cartInfoList;
        }

}

//从数据中加载
public List<CartInfo> loadCartCache(String userId){
        //
        List<CartInfo> cartlist = cartInfoMapper.selectCartListWithCurPrice(Long.parseLong(userId));
        if(cartlist==null||cartlist.size()==0){
            return null;
        }
        Jedis jedis = redisUtil.getJedis();
        String userCartKey="user:"+userId+":cart";
        String userInfoKey="user:"+userId+":info";
        Map cartMap =new HashMap(cartlist.size());
        for (CartInfo cartInfo : cartlist) {
            cartMap.put(cartInfo.getSkuId(),JSON.toJSONString(cartInfo));
        }
        jedis.hmset(userCartKey,cartMap);
        Long ttl = jedis.ttl(userInfoKey);
        jedis.expire(userCartKey, ttl.intValue());
        return cartlist;
}
```

CartInfoMapper.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper SYSTEM "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.atguigu.gmall.cart.mapper.CartInfoMapper">
    <select    id ="selectCartListWithCurPrice" parameterType="long" resultMap="cartMap">
        SELECT c.*,s.price FROM cart_info c
        INNER JOIN sku_info s ON c.sku_id=s.id WHERE c.user_id=#{userId}
```

```
        order by c.id desc
    </select>
    <resultMap id="cartMap" type="com.atguigu.gmall.bean.CartInfo" autoMapping="true">
        <result property="id" column="id" ></result>
        <result property="skuPrice" column="price" ></result>
    </resultMap>
</mapper>
```

CartInfoMapper.java

```
public interface CartInfoMapper extends Mapper<CartInfo> {

    public  List<CartInfo> selectCartListWithCurPrice(long userId);
}
```

因为要取 mapper.xml 所以

application.properties 中要加入

```
mybatis.mapper-locations=classpath:mapper/*Mapper.xml
mybatis.configuration.mapUnderscoreToCamelCase=true
```

3 CartController

```
@RequestMapping("cartList")
@LoginRequire(autoRedirect = false)
public String cartList(HttpServletRequest request,HttpServletResponse response){
    String userId =(String) request.getAttribute("userId");
    List<CartInfo> cartCookieList = cartCookieHandler.getCartList(request);
    if(userId==null){
        request.setAttribute("cartList",cartCookieList );
    }else {
        if(cartCookieList==null||cartCookieList.size()==0) {
            List<CartInfo> cartList = cartService.getCartList(userId);
```

```
                request.setAttribute("cartList",cartList );
            }
        }

        return "cartList";
}
```

cartCookieHandler 中 getCartList 在前面的做添加购物车时已完成。

4 显示购物车列表页面

```
<div class="One_ShopCon">
    <ul>
        <li th:each="cartInfo:${cartList}">
            <div> </div>
            <div>
                <ol>
                    <li><input type="checkbox" class="check"
th:value="${cartInfo.skuId}" onchange="checkSku(this)"
th:checked="(${cartInfo.isChecked}=='1')?'true':'false'"/></li>

                    <li>
                        <dt><img th:src="${cartInfo.imgUrl}" src="img\shop1.jpg"
alt=""></dt>
                        <dd th:onclick="'toItem('+${cartInfo.skuId}+')'">
                            <p>
                                <span th:text="${cartInfo.skuName}"> 商品名称</span>
                            </p>
                        </dd>
                    </li>
                    <li>
                        <p class="dj" th:text="'￥'+${cartInfo.skuPrice}">￥钱</p>
                    </li>
                    <li>
                        <p>
                            <span>-</span>
                            <span th:text="${cartInfo.skuNum}">5</span>
                            <span>+</span>
                        </p>
                    </li>
                    <li style="font-weight:bold"><p class="zj" th:text="'
￥'+${cartInfo.totalAmount}">￥22995.00</p></li>
                    <li>
                        <p>删除</p>
                    </li>
                </ol>
```

```
            </div>
        </li>
    </ul>
</div>
```

5 测试效果

五、功能--合并购物车

由于加入购物车时，用户可能存在登录和未登录两种情况，登录前在 cookie 中保存了一部分购物车信息，如果用户登录了，那么对应的要把 cookie 中的购物车合并到数据库中，并且刷新缓存。

1 CartServiceImpl

思路：用数据库中的购物车列表与传递过来的 cookie 里的购物车列表循环匹配。

能匹配上的数量相加

匹配不上的插入到数据库中。

最后重新加载缓存

```
public List<CartInfo> mergeToCart(String userId ,List<CartInfo> cartInfoList){
    //查询用户名下的购物车清单
    CartInfo cartInfoQuery=new CartInfo();
    cartInfoQuery.setUserId(userId);
    List<CartInfo> cartInfoExistList = cartInfoMapper.select(cartInfoQuery);
    for (CartInfo cartInfo : cartInfoList) {
        for (CartInfo cartInfoExist : cartInfoExistList) {
            if( cartInfo.getSkuId().equals(cartInfoExist.getSkuId())){
                cartInfoExist.setSkuNum(cartInfoExist.getSkuNum()+cartInfo.getSkuNum());
                cartInfoMapper.updateByPrimaryKey(cartInfoExist);
            }
        }
        cartInfo.setUserId(userId);
        cartInfoMapper.insertSelective(cartInfo);
    }

    List<CartInfo> newCartInfoList = loadCartCache(userId);
    return newCartInfoList;
}
```

2 CartController

增加判断如果用户是登录状态的，但是 cookie 里却还有购物车，说明需要把 cookie 中的购物车合并进来，同时把 cookie 中的清空。

```
@RequestMapping("cartList")
@LoginRequire(autoRedirect = false)
public String cartList(HttpServletRequest request,HttpServletResponse response){
    String userId =(String) request.getAttribute("userId");
    List<CartInfo> cartCookieList = cartCookieHandler.getCartList(request);
    if(userId==null){
        request.setAttribute("cartList",cartCookieList );
    }else {
        if(cartCookieList==null||cartCookieList.size()==0) {
            List<CartInfo> cartList = cartService.getCartList(userId);
            request.setAttribute("cartList",cartList );
        }else {
            List<CartInfo> cartList = cartService.mergeToCart(userId, cartCookieList);
            cartCookieHandler.deleteCartCookie(request,response);
            request.setAttribute("cartList",cartList );
```

```
        }
    }

    return "cartList";
}
```

在 CartCookieHandler 中加入

```
public void deleteCartCookie(HttpServletRequest request,
                             HttpServletResponse response){
    CookieUtil.deleteCookie(request,response,cartCookieName);
}
```

在 web-util 的 CookieUtil 中增加

```
public static void deleteCookie(HttpServletRequest request,
HttpServletResponse response, String cookieName) {
    setCookie(request, response, cookieName, null, 0, false);
}
```

六、选中状态的变更

用户每次勾选购物车的多选框，都要把当前状态保存起来。由于可能会涉及更频繁的操作，所以这个勾选状态不必存储到数据库中。保留在缓存状态即可。

1 CartServiceImpl

把对应 skuId 的购物车的信息从 redis 中取出来，反序列化，修改 isChecked 标志。再保存回 redis 中。

同时保存另一个 redis 的 key 专门用来存储用户选中的商品，方便结算页面使用。

```
public void setCheckedCart(String userId,String skuId,String isChecked){
        Jedis jedis = redisUtil.getJedis();
        String userCartKey="user:"+userId+":cart";
        String cartInfoJson = jedis.hget(userCartKey, skuId);
        CartInfo cartInfo = JSON.parseObject(cartInfoJson, CartInfo.class);
        cartInfo.setIsChecked(isChecked);
        String newCartInfoJson=JSON.toJSONString(cartInfo);
        jedis.hset(userCartKey, skuId,newCartInfoJson);

        String cartCheckedKey="user:"+userId+":cartChecked";
        if(isChecked.equals("1")){
                jedis.hset(cartCheckedKey,skuId,newCartInfoJson);
                jedis.expire(cartCheckedKey, jedis.ttl(userCartKey).intValue());
        }else{
                jedis.hdel(cartCheckedKey,skuId );
        }

}
```

合并的时候要把未登录前 cookie 里的勾选附上。

```
public List<CartInfo> mergeToCart(String userId ,List<CartInfo> cartInfoCookieList){
        //查询用户名下的购物车清单
        CartInfo cartInfoQuery=new CartInfo();
        cartInfoQuery.setUserId(userId);
        List<CartInfo> cartInfoExistList = cartInfoMapper.select(cartInfoQuery);
        for (CartInfo cartInfo : cartInfoCookieList) {
                for (CartInfo cartInfoExist : cartInfoExistList) {
                        if( cartInfo.getSkuId().equals(cartInfoExist.getSkuId())){
                                cartInfoExist.setSkuNum(cartInfoExist.getSkuNum()+cartInfo.getSkuNum());
                                cartInfoMapper.updateByPrimaryKey(cartInfoExist);
                        }
                }
                cartInfo.setUserId(userId);
                cartInfoMapper.insertSelective(cartInfo);
        }

        List<CartInfo> newCartInfoList = loadCartCache(userId);

        for (CartInfo cartInfo : cartInfoCookieList) {
                if(cartInfo.getIsChecked().equals("1")){
                        setCheckedCart( userId,    cartInfo.getSkuId(), "1");
                }
        }

        return newCartInfoList;
}
```

2 CartController

同样这里要区分，用户登录和未登录状态。

如果登录，修改缓存中的数据，如果未登录，修改 cookie 中的数据。

```
@RequestMapping(value = "checkCart",method = RequestMethod.POST)
@LoginRequire(autoRedirect = false)
@ResponseBody
public void checkCart(HttpServletRequest request,HttpServletResponse response){
	String skuId = request.getParameter("skuId");
	String userId =(String) request.getAttribute("userId");
	String isChecked =(String) request.getParameter("isChecked");
	if(userId!=null){
		cartService.setCheckedCart(userId,skuId,isChecked);
	}else{
		cartCookieHandler.setCheckCartCookie(request,response,skuId,isChecked);
	}
	return ;
}
```

3 CartCookieHandler(修改 cookie 数据)

从 cookie 的购物车列表中依次匹配 skuId，匹配中则设入 isChecked 标志。

```
public void setCheckCartCookie(HttpServletRequest request,HttpServletResponse response,String skuId,String isChecked) {
	try {
		List<CartInfo> cartList = getCartList(request);
		CartInfo cart = null;
		if (cartList != null && cartList.size() > 0) {
			for (CartInfo c : cartList) {
				// 判断购物车中是否存在该商品
				if (c.getSkuId().equals(skuId)) {
					cart = c;
					break;
				}
			}
			cart.setIsChecked(isChecked);
		}

		CookieUtil.setCookie(request, response, cartCookieName, JSON.toJSONString(cartList), WebConst.cookieMaxAge, true);
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }
}
```

4 页面动作的 js

注意 jquery1.6 版本以后要用 prop("checked")取得多选框选中状态。

```
function checkSku( chkbox){
        console.log($(chkbox));
    var skuId= $(chkbox).attr("value");
    var checked=$(chkbox).prop("checked");
    var isCheckedFlag="0";
    if(checked){
            isCheckedFlag="1";
    }
     var param="isChecked="+isCheckedFlag+"&"+"skuId="+skuId;
     console.log(param);
     $.post("checkCart",param,function (data) {
            sumSumPrice();
     });
}
```

5 测试效果

七、点击结算要做的收尾工作

要解决用户在未登录且购物车中有商品的情况下，直接点击结算。

所以不能直接跳到结算页面，要让用户强制登录后，检查 cookie 并进行合并后再重定向到结算页面

1 CartController

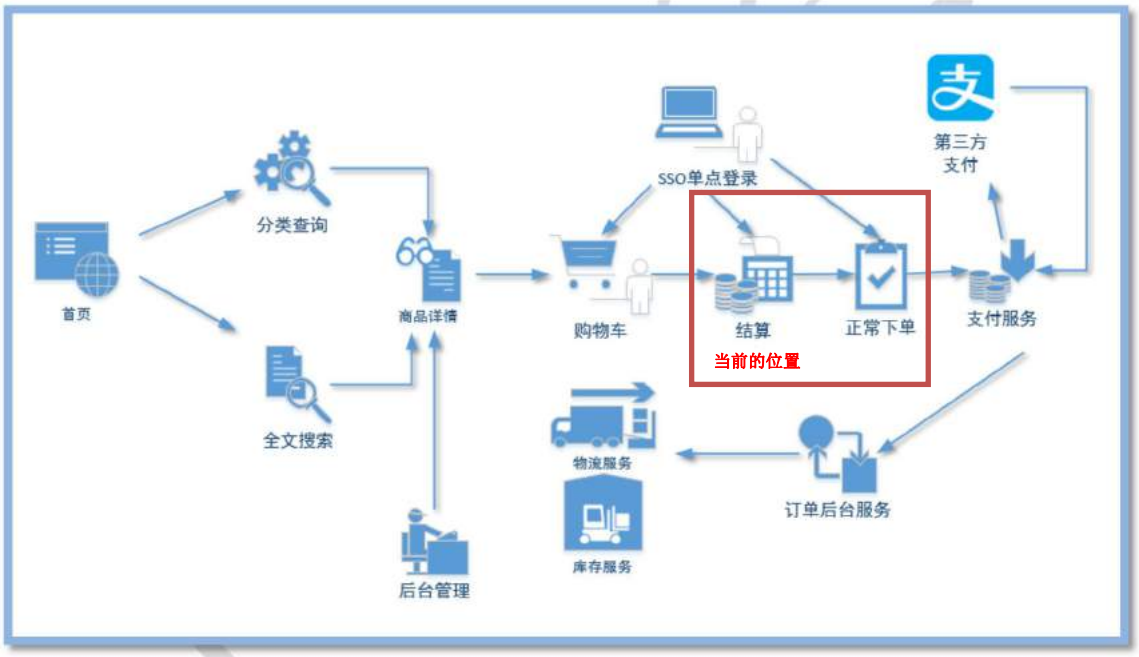
```
@RequestMapping("toTrade")
@LoginRequire(autoRedirect = true)
public String toTrade(HttpServletRequest request,HttpServletResponse response){
	String userId =(String) request.getAttribute("userId");
	List<CartInfo> cartCookieList = cartCookieHandler.getCartList(request);
	if(cartCookieList!=null&&cartCookieList.size()>0) {
		List<CartInfo> cartList = cartService.mergeToCart(userId, cartCookieList);
		cartCookieHandler.deleteCartCookie(request,response);
	}
	return "redirect://order.gmall.com/trade";
}
```

订　　单

版本：V 1.0

www.atguigu.com

一、业务介绍



订单业务在整个电商平台中处于核心位置，也是比较复杂的一块业务。是把“物”变为“钱”的一个中转站。

整个订单模块一共分四部分组成：

1. 结算
2. 下单
3. 对接支付服务
4. 对接库存管理系统

二、结算页

入口：购物车点击计算按钮

1 搭建模块

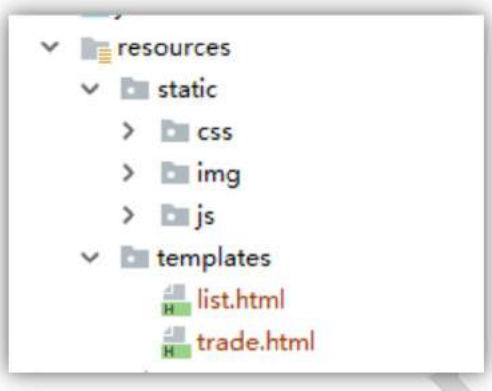
1.1 gmall-order-web（模块已添加）

application.properties

```
spring.dubbo.application.name=order-web
spring.dubbo.registry.protocol=zookeeper
spring.dubbo.registry.address=192.168.67.163:2181
spring.dubbo.base-package=com.atguigu.gmall
spring.dubbo.protocol.name=dubbo
spring.dubbo.consumer.timeout=10000
spring.dubbo.consumer.check=false

server.port=8088
```

```
spring.thymeleaf.cache=false
spring.thymeleaf.mode=LEGACYHTML5
```

拷贝静态资源文件和 html



host 文件

```
# gmall
192.168.67.163 cart.gmall.com passport.atguigu.com item.gmall.com list.gmall.com
manage.gmall.com www.gmall.com resource.gmall.com order.gmall.com
```

nginx.conf

```
    upstream order.gmall.com{
        server 192.168.67.1:8088;
    }
  server {
    listen 80;
    server_name order.gmall.com;
    location / {
        proxy_pass http://order.gmall.com;
        proxy_set_header X-forwarded-for $proxy_add_x_forwarded_for;
    }
  }
```

2 分析

分析页面需要的数据：

1、 用户地址列表（已完成）

2、 购物车中选择的商品列表

3 购物车中选择的商品列表

3.1 在 CartServiceImpl.java

```
public List<CartInfo> getCartCheckedList(String userId){
    //优先从缓存中取值
    Jedis jedis = redisUtil.getJedis();
    List<String> skuJsonlist = jedis.hvals("user:" + userId + ":cartChecked");
    List<CartInfo> cartInfoList=new ArrayList<>();
    if(skuJsonlist!=null &&skuJsonlist.size()!=0){
        //序列化
        for (String skuJson : skuJsonlist) {
            CartInfo cartInfo = JSON.parseObject(skuJson, CartInfo.class);
            cartInfoList.add( cartInfo);
        }
    }
    return cartInfoList;
}
```

要把查询出来的 cartInfoList 装配到 orderDetailList 中

3.2 订单明细的数据结构：

order_detail(订单明细表)	
id	编号
order_id	订单编号
sku_id	
sku_name	sku名称（冗余）
img_url	图片名称（冗余）
order_price	购买价格(下单时sku价格)
sku_num	购买个数

在 bean 下建立 order_detail

```
public class OrderDetail implements Serializable {
```

```
    @Id
    @Column
    private String id;
    @Column
    private String orderId;
    @Column
    private String skuId;
    @Column
    private String skuName;
    @Column
    private String imgUrl;
    @Column
    private BigDecimal orderPrice;
    @Column
    private Integer skuNum;

    @Transient
    private String hasStock;
}
```

其中 hasStock 是一个非持久化属性，用户传递【是否还有库存】的标志。

3.3 OrderController

加入 tradeInit 方法

```
@RequestMapping(value = "trade",method = RequestMethod.GET)
@LoginRequire
public String tradeInit(HttpServletRequest request, Model model){
    String userId=(String)request.getAttribute("userId");

    //查询用户地址信息
    List<UserAddress> addresssList = userManageService.getUserAddressList( userId );
    request.setAttribute("addressList",addresssList);

    //查询商品清单
    List<CartInfo> cartInfoList=cartService.getCartCheckedList(userId);

    List<OrderDetail> orderDetailList=new ArrayList();
    Integer totalNum=0;
    BigDecimal totalAmount=new BigDecimal(0);
    for (CartInfo cartInfo : cartInfoList) {
        OrderDetail orderDetail=new OrderDetail();

        orderDetail.setSkuId(cartInfo.getSkuId());
        orderDetail.setSkuName(cartInfo.getSkuName());
        orderDetail.setOrderPrice(cartInfo.getSkuPrice());
        orderDetail.setSkuNum(cartInfo.getSkuNum());
        orderDetail.setImgUrl(cartInfo.getImgUrl());
```

```
            totalAmount=                totalAmount.add(cartInfo.getSkuPrice().multiply(new
BigDecimal(cartInfo.getSkuNum())) );//总金额
            orderDetailList.add(orderDetail);
        }

        model.addAttribute("orderDetailList",orderDetailList);
        model.addAttribute("totalAmount",totalAmount);
        return "trade";
}
```

3.4 结算页面

```
<!--地址-->
<div class="top-3">
    <ul >
     <li class=".address default selected" th:each="address:${addressList}">
         <input        name="deliveryAddress"   type="radio"   th:value="${address.userAddress}"
th:checked="${address.isDefault=='1'}">
        <span                    th:text="${address.consignee}">              </span><span
th:text="${address.userAddress}"> </span>
    </li>
    </ul>
</div>
```

商品清单部分

```
<div class="to_right">
        <h5>商家：自营</h5>
        <!--图片-->
        <div class="yun1" th:each="detail:${orderDetailList}">
            <img th:src="${detail.imgUrl}" class="yun"/>
            <div class="mi">
                <div><p    style="width:    500px;"    th:text="${detail.skuName}">    </p>    <span
style="float: right"> <span align="center" style="color: red"    th:text="'￥'+${detail.orderPrice}">
</span> <span    th:text="'x'+${detail.skuNum}">    </span>    </div>

            </div>
        </div>
</div>
```

总金额

```
    <div class="yfze">
        <p  class="yfze_ a"><span  class="z">应 付 总 额 ：  </span><span  class="hq"  th:text="'
￥'+${totalAmount}"> </span></p>
```

```
    <button id="submitButton" class="tijiao">提交订单</button>
</div>
```

点击提交

增加一个表单，用于提交订单

```
<form action="/confirm_order" method="post" id="orderForm">
    <input name="consignee" id="consignee" type="hidden"/>
    <input name="deliveryAddress" id="deliveryAddress" type="hidden"/>
    <input name="orderDesc" id="orderDesc" type="hidden"/>
    <span     th:each="detail:${orderDetailList}">
        <input name="skuIds" type="hidden" th:value="${detail.skuId}" />
        <input name="skuNums" type="hidden"     th:value="${detail.skuNum}" />
    </span>

</form>
```

提交按钮的 js

```
$("#submitButton").click(function () {
    $("#consignee").val($("input[type='radio']:checked").next().text()) ;
    $("#deliveryAddress").val( $("input[type='radio']:checked").next().next().text());
    $("#orderDesc").val($("#order_desc_page").val());
    console.log($("#orderForm").html());
    $("#orderForm").submit();

});
```

三、下单

1 数据结构

id	主键。自动生成
consignee	收货人名称。页面获取
consignee_tel	收货人电话。页面获取
deliveryAddress	收货地址。页面获取
total_amount	总金额。计算
order_status	订单状态，用于显示给用户查看。设定初始值。
userId	用户 Id。从拦截器已放到请求属性中。
payment_way	支付方式（网上支付、货到付款）。页面获取
orderComment	订单状态。页面获取
out_trade_no	第三方支付编号。按规则生成
create_time	创建时间。设当前时间
expire_time	默认当前时间+1 天
process_status	订单进度状态，程序控制、 后台管理查看。设定初始值，
tracking_no	物流编号,初始为空，发货后补充
parent_order_id	拆单时产生，默认为空

id	主键，自动生成
order_id	订单编号，主表保存后给从表
sku_id	商品 id 页面传递
sku_name	商品名称，后台添加
img_url	图片路径，后台添加
order_price	商品单价，从页面中获取，并验价。
sku_num	商品个数，从页面中获取

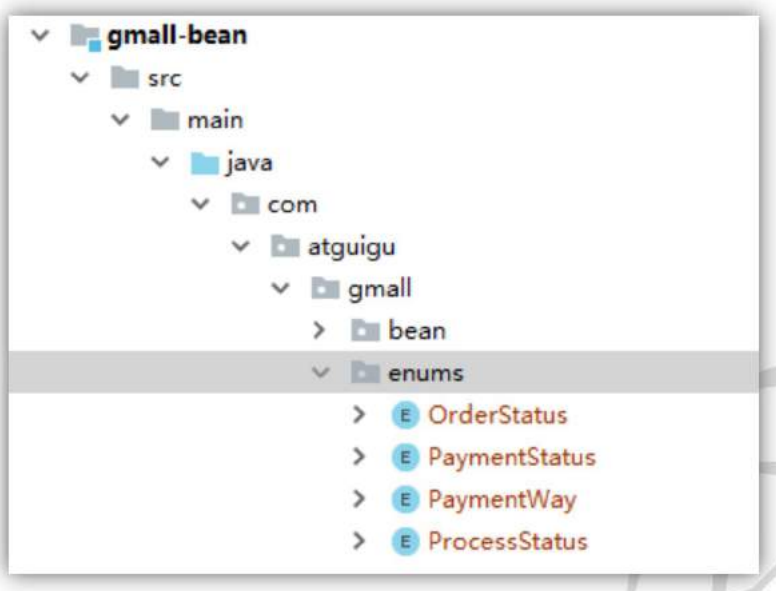
增加实体 Bean

```
public class OrderInfo implements Serializable {
    @Column
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private String id;

    @Column
    private String consignee;

    @Column
    private String consigneeTel;


    @Column
    private BigDecimal totalAmount;

    @Column
    private OrderStatus orderStatus;

    @Column
    private ProcessStatus processStatus;


    @Column
    private String userId;

    @Column
    private PaymentWay paymentWay;

    @Column
    private Date expectDeliveryTime;

    @Column
```

```
    private String deliveryAddress;

    @Column
    private String orderComment;

    @Column
    private Date createTime;

    @Column
    private String parentOrderId;

    @Column
    private String trackingNo;


    @Transient
    private List<OrderDetail> orderDetailList;

    @Transient
    private List<OrderInfo> orderSubList;

    @Transient
    private String wareId;

    @Column
    private String outTradeNo;

    public void sumTotalAmount(){
        BigDecimal totalAmount=new BigDecimal("0");
        for (OrderDetail orderDetail : orderDetailList) {
            totalAmount=        totalAmount.add(orderDetail.getOrderPrice().multiply(new
BigDecimal(orderDetail.getSkuNum())));
        }
         this.totalAmount=   totalAmount;

    }
}
```

增加枚举类

枚举类路径放到 gmall-bean 模块中和 bean 同级目录。




```
public enum OrderStatus {
        UNPAID("未支付"),
        PAID("已支付" ),
        WAITING_DELEVER("待发货"),
        DELEVERED("已发货"),
        CLOSED("已关闭"),
        FINISHED("已完结") ,
        SPLIT("订单已拆分");

    private String comment ;


    OrderStatus(String comment ){
        this.comment=comment;
    }

    public String getComment() {
        return comment;
    }

    public void setComment(String comment) {
        this.comment = comment;
    }


}
```

```
public enum ProcessStatus {
        UNPAID("未支付",OrderStatus.UNPAID),
        PAID("已支付",OrderStatus.PAID),
```

```
        NOTIFIED_WARE("已通知仓储",OrderStatus.PAID),
        WAITING_DELEVER("待发货",OrderStatus.WAITING_DELEVER),
        STOCK_EXCEPTION("库存异常",OrderStatus.PAID),
        DELEVERED("已发货",OrderStatus.DELEVERED),
        CLOSED("已关闭",OrderStatus.CLOSED),
        FINISHED("已完结",OrderStatus.FINISHED) ,
        PAY_FAIL("支付失败",OrderStatus.UNPAID),
        SPLIT("订单已拆分",OrderStatus.SPLIT);

    private String comment ;
    private OrderStatus orderStatus;

    ProcessStatus(String comment, OrderStatus orderStatus){
        this.comment=comment;
        this.orderStatus=orderStatus;
    }

    public String getComment() {
        return comment;
    }

    public void setComment(String comment) {
        this.comment = comment;
    }

    public OrderStatus getOrderStatus() {
        return orderStatus;
    }

    public void setOrderStatus(OrderStatus orderStatus) {
        this.orderStatus = orderStatus;
    }
```

```
public enum PaymentWay {
        ONLINE("在线支付"),
        OUTLINE("货到付款" );

    private String comment ;


    PaymentWay(String comment ){
        this.comment=comment;
    }

    public String getComment() {
        return comment;
    }

    public void setComment(String comment) {
        this.comment = comment;
    }
```

```
}
```

由于涉及枚举类所以　application.properties 中要加入

```
mapper.enum-as-simple-type=true
```

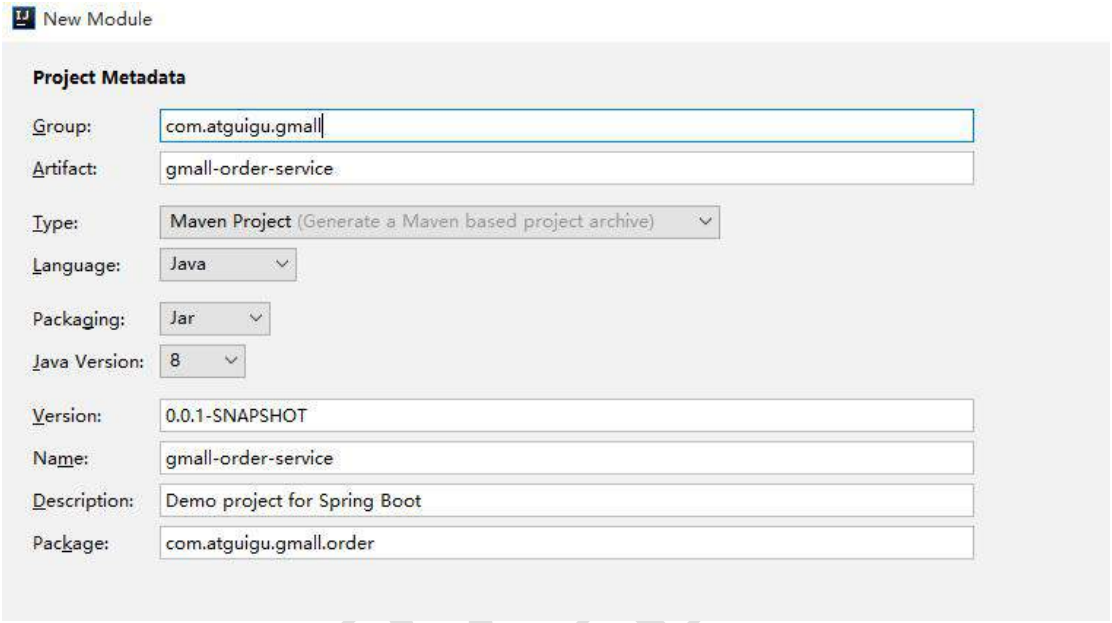
这个配置会把枚举类当成字符串处理。

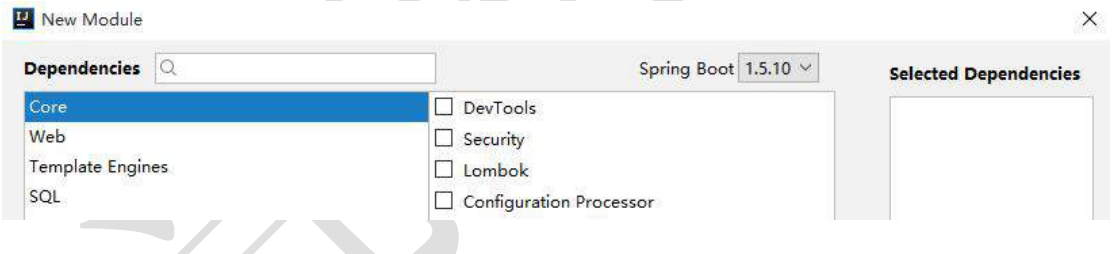
2 分析下单：

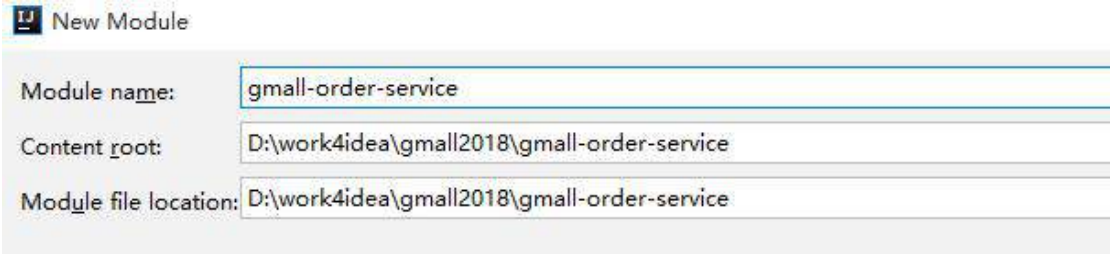
1. 保存单据前要做交易：验库存、验价
2. 保存单据。
3. 保存以后把购物车中的商品删除
4. 重定向到支付页面。

3 订单后台服务模块

3.1 搭建 gmall-order-service







pom.xml

```
<?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.atguigu.gmall</groupId>
    <artifactId>gmall-order-service</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>gmall-order-service</name>
    <description>Demo project for Spring Boot</description>

    <parent>
        <groupId>com.atguigu.gmall</groupId>
        <artifactId>gmall-parent</artifactId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <dependencies>

        <dependency>
            <groupId>com.atguigu.gmall</groupId>
            <artifactId>gmall-interface</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>

        <dependency>
            <groupId>com.atguigu.gmall</groupId>
            <artifactId>gmall-service-util</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
    </dependencies>

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


</project>
```

application.properties

```
server.port=8078

logging.level.root=debug

spring.dubbo.application.name=order-service
```

```
spring.dubbo.registry.protocol=zookeeper
spring.dubbo.registry.address=192.168.67.163:2181
spring.dubbo.base-package=com.atguigu.gmall
spring.dubbo.protocol.name=dubbo
spring.dubbo.consumer.timeout=100000
spring.dubbo.consumer.check=false

spring.datasource.url=jdbc:mysql://59.110.141.236:3306/gmall?characterEncoding=UTF-8
spring.datasource.username=root
spring.datasource.password=123456


mybatis.mapper-locations=classpath:mapper/*Mapper.xml
mybatis.configuration.mapUnderscoreToCamelCase=true

spring.redis.host=192.168.67.163
spring.redis.port=6379
spring.redis.database=0
```

GmallOrderServiceApplication

```
@SpringBootApplication
@MapperScan(basePackages = "com.atguigu.gmall.order.mapper")
@ComponentScan(basePackages = "com.atguigu.gmall")
public class GmallOrderServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(GmallOrderServiceApplication.class, args);
    }
}
```

3.2 增加 OrderServiceImpl

```
@Service
public class OrderServiceImpl implements OrderService{

    @Autowired
    OrderInfoMapper orderInfoMapper;

    @Autowired
    OrderDetailMapper orderDetailMapper;

    @Override
    @Transactional
    public void saveOrder(OrderInfo orderInfo){
        //增加时间、状态、生成外部流水号
```

```
            orderInfo.setCreateTime(new Date());
            if(orderInfo.getOrderStatus()==null){
                orderInfo.setOrderStatus(ProcessStatus.UNPAID.getOrderStatus());
            }
            if(orderInfo.getProcessStatus()==null){
                orderInfo.setProcessStatus(ProcessStatus.UNPAID);
            }

orderInfo.setOutTradeNo(OrderConst.out_trade_no_prefix+System.currentTimeMillis()+"000"+n
ew Random().nextInt(1000));

            orderInfoMapper.insertSelective(orderInfo);
            System.out.println(orderInfo);

            List<OrderDetail> orderDetailList=orderInfo.getOrderDetailList();
            for (OrderDetail orderDetail : orderDetailList) {
                orderDetail.setOrderId(orderInfo.getId());
                orderDetailMapper.insertSelective(orderDetail);

            }
        }
}
```

3.3 CartServiceImpl 中增加删除购物车方法

```
public void delCartCheckedList(String userId,List<String> skuIds){

        Example example=new Example(CartInfo.class);
        example.createCriteria().andIn("skuId",skuIds);
        cartInfoMapper.deleteByExample(example);

        Jedis jedis=redisUtil.getJedis();
        jedis.pipelined();
        for (String skuId : skuIds) {
            jedis.hdel("user:" + userId + ":cartChecked",skuId);
            jedis.hdel("user:" + userId + ":cart",skuId);
        }
        jedis.sync();
        jedis.close();

}
```

3.4 OrderController

```
@RequestMapping(value = "submitOrder" ,method = RequestMethod.POST)
@LoginRequire(debugUserId = "8")
public String submitOrder(OrderInfo orderInfo    ,    HttpServletRequest request){
      String userId = (String) request.getAttribute("userId");

      List<OrderDetail> orderDetailList = orderInfo.getOrderDetailList();
      for (OrderDetail orderDetail : orderDetailList) {
            SkuInfo skuInfo = manageService.getSkuInfo(orderDetail.getSkuId());
            //验价
            if(!orderDetail.getOrderPrice().equals(skuInfo.getPrice())){
                  cartService.loadCartCache(userId);
                  request.setAttribute("errMsg","商品["+skuInfo.getSkuName()+"]价格已发生变化，请在购物车页面重新进行结算");
                  return "tradeFail";
            }

            orderDetail.setImgUrl(skuInfo.getSkuDefaultImg());
            orderDetail.setSkuName(skuInfo.getSkuName());
      }

      orderInfo.setUserId(userId);
      orderInfo.sumTotalAmount();

      orderService.saveOrder(orderInfo);

      List<String> skuIdList =new ArrayList<>();
      for (OrderDetail orderDetail : orderDetailList) {
            skuIdList.add(orderDetail.getSkuId());
      }
      cartService.delCartCheckedList(userId,skuIdList);
      return "redirect://payment.gmall.com/index";
}
```

3.5 页面 trade.html

```
<form action="/submitOrder" method="post" id="orderForm">
   <input name="consignee" id="consignee" type="hidden"/>
   <input name="deliveryAddress" id="deliveryAddress" type="hidden"/>
   <input name="paymentWay" id="paymentWay" type="hidden"/>
   <input name="orderComment" id="orderComment" type="hidden"/>
   <span   th:each="detail,stat:${orderDetailList}">
      <input th:name="'orderDetailList['+${stat.index}+'].skuId'"
```

```
type="hidden"    th:value="${detail.skuId}" />
      <input th:name="'orderDetailList['+${stat.index}+'].skuNum'"
type="hidden"   th:value="${detail.skuNum}" />
      <input th:name="'orderDetailList['+${stat.index}+'].orderPrice'"
type="hidden"   th:value="${detail.orderPrice}" />
   </span>

</form>
```

```
$("#submitButton").click(function () {
   $("#consignee").val($("input[type='radio']:checked").next().text()) ;

$("#deliveryAddress").val( $("input[type='radio']:checked").next().next().te
xt());
   $("#paymentWay").val("ONLINE");
   $("#orderComment").val($("#orderCommentPage").val());
   console.log($("#orderForm").html());
   $("#orderForm").submit();

});
```

4 如何解决用户利用浏览器刷新和回退重复提交订单？

在进入结算页面时，生成一个结算流水号，然后保存到结算页面的隐藏元素中，每次用户提交都检查该流水号与页面提交的是否相符，订单保存以后把后台的流水号删除掉。那么第二次用户用同一个页面提交的话流水号就会匹配失败，无法重复保存订单。

4.1 修改结算页增加流水号的生成。

4.1.1 OrderServiceImpl

```
//生成流水号
@Override
public String genTradeNo(String userId){
	Jedis jedis = redisUtil.getJedis();
	String userTradeNoKey="user:"+userId+":tradeNo";
	UUID uuid = UUID.randomUUID();
	jedis.setex(userTradeNoKey,OrderConst.tradeExpire,uuid.toString());
	return uuid.toString();
}

//验证流水号
@Override
public	boolean checkTradeNo(String userId,String trackNo){
	Jedis jedis = redisUtil.getJedis();
	String userTradeNoKey="user:"+userId+":tradeNo";
	String	uuid = jedis.get(userTradeNoKey);
	if(trackNo!=null&&trackNo.equals(uuid)){
		return	true;
	}
	return	false;
}

//删除流水号
@Override
public	void delTradeNo(String userId ){
	Jedis jedis = redisUtil.getJedis();
	String userTradeNoKey="user:"+userId+":tradeNo";
	jedis.del(userTradeNoKey);
	return	;
}
```

4.1.2 常量类

```
public interface OrderConst {

	public static final String out_trade_no_prefix="ATGUIGU";

	public static final int tradeExpire=10*60;
}
```

4.2 OrderController 中的 trade 方法中

```
String tradeNo = orderService.genTradeNo(userId);
request.setAttribute("tradeNo",tradeNo);
```

4.3 OrderController 中的 submitOrder 方法中

```
@RequestMapping(value = "submitOrder" ,method = RequestMethod.POST)
@LoginRequire(debugUserId = "8")
public String submitOrder(OrderInfo orderInfo    ,    HttpServletRequest request){
      String userId = (String) request.getAttribute("userId");

      String tradeNo= request.getParameter("tradeNo");
      Boolean hasTradeNo=orderService.checkTradeNo(userId,tradeNo);
      if(!hasTradeNo){
            request.setAttribute("errMsg","结算页面过期或已失效，请重新结算。");
            return "tradeFail";
      }

      List<OrderDetail> orderDetailList = orderInfo.getOrderDetailList();
      for (OrderDetail orderDetail : orderDetailList) {
            SkuInfo skuInfo = manageService.getSkuInfo(orderDetail.getSkuId());
            //验价
            if(!orderDetail.getOrderPrice().equals(skuInfo.getPrice())){
                  cartService.loadCartCache(userId);
                  request.setAttribute("errMsg","商品["+skuInfo.getSkuName()+"]价格已发生变
化，请在购物车页面重新进行结算");
                  return "tradeFail";
            }

            orderDetail.setImgUrl(skuInfo.getSkuDefaultImg());
            orderDetail.setSkuName(skuInfo.getSkuName());
      }

      orderInfo.setUserId(userId);
      orderInfo.sumTotalAmount();

      orderService.saveOrder(orderInfo);
```

```
        List<String> skuIdList =new ArrayList<>();
        for (OrderDetail orderDetail : orderDetailList) {
            skuIdList.add(orderDetail.getSkuId());
        }
        cartService.delCartCheckedList(userId,skuIdList);
        orderService.delTradeNo(userId);
        return "redirect://payment.gmall.com/index";
}
```

5 验库存

通过 restful 接口查询商品是否有库存

一般电商系统的商品库存，都不由电商系统本身来管理，由另外一套仓库管理系统，或者进销存系统来管理，电商系统通过第三方接口调用该系统。

由于库管系统可能是异构的系统，所以不在 dubbo 的分布式体系之内。只支持 restful 风格的 webservice 调用和消息队列的调用。

详见《库存管理系统手册》

根据手册中的接口文档，编写调用代码。

1、 查询库存

接口	/hasStock
请求参数	skuId： 商品 skuId num： 商品数量
请求方式	get
例	/hasStock?skuId=10221&num=2

5.1 OrderController 中注入 application.properties 的值

```
@Value("${ware_sys_url}")
public String ware_sys_url;
```

增加方法

```
private String hasStorkBySkuidAndNum(String skuid,String num) {
    String url=ware_sys_url+"/hasStock";
    url=url+"?skuId="+skuid+"&num="+num;
    String result = HttpclientUtil.doGet(url);
    return result;
}
```

5.2 submitOrder 中增加 方法

```
@RequestMapping(value = "submitOrder" ,method = RequestMethod.POST)
@LoginRequire(debugUserId = "8")
public String submitOrder(OrderInfo orderInfo    ,    HttpServletRequest request){
    String userId = (String) request.getAttribute("userId");

    String tradeNo= request.getParameter("tradeNo");
    Boolean hasTradeNo=orderService.checkTradeNo(userId,tradeNo);
    if(!hasTradeNo){
        request.setAttribute("errMsg","结算页面过期或已失效，请重新结算。");
        return "tradeFail";
    }

    List<OrderDetail> orderDetailList = orderInfo.getOrderDetailList();
    for (OrderDetail orderDetail : orderDetailList) {
        SkuInfo skuInfo = manageService.getSkuInfo(orderDetail.getSkuId());
        //验价
        if(!orderDetail.getOrderPrice().equals(skuInfo.getPrice())){
            cartService.loadCartCache(userId);
            request.setAttribute("errMsg","商品["+skuInfo.getSkuName()+"]价格已发生变化，请在购物车页面重新进行结算");
            return "tradeFail";
        }
        String hasStock = hasStorkBySkuidAndNum(orderDetail.getSkuId(), String.valueOf(orderDetail.getSkuNum()));
        if (hasStock==null||hasStock.equals("0")){
            request.setAttribute("errMsg","商品["+skuInfo.getSkuName()+"]暂时缺货。");
            return "tradeFail";
        }

        orderDetail.setImgUrl(skuInfo.getSkuDefaultImg());
        orderDetail.setSkuName(skuInfo.getSkuName());
```

```
        }

        orderInfo.setUserId(userId);
        orderInfo.sumTotalAmount();

        orderService.saveOrder(orderInfo);

        List<String> skuIdList =new ArrayList<>();
        for (OrderDetail orderDetail : orderDetailList) {
            skuIdList.add(orderDetail.getSkuId());
        }
        cartService.delCartCheckedList(userId,skuIdList);
        orderService.delTradeNo(userId);
        return "redirect://payment.gmall.com/index";
}
```

我的订单(不做讲解)

OrderInfoMapper

```
public interface OrderInfoMapper extends Mapper<OrderInfo>{
    public List<OrderInfo> selectOrderListByUser(Long userId);
}
```

OrderInfoMapper.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper SYSTEM "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.atguigu.gmall.order.mapper.OrderInfoMapper">
    <select id="selectOrderListByUser" parameterType="long"
resultMap="orderInfoListMap">
        SELECT o.id,
        o.consignee,
        o.consignee_tel,
        o.total_amount,
        o.order_status,
        o.user_id ,
        payment_way,
        o.delivery_address,
        o.order_comment,
        o.out_trade_no,
        o.create_time ,
```

```
            o.expire_time ,
            o.process_status,
            o.tracking_no,
            o.parent_order_id,
            od.order_id,
            od.order_price,
            od.sku_num,
            od.id order_detail_id,
            od.img_url,
            od.sku_id,
            od.sku_name
            FROM    order_info o INNER JOIN order_detail od ON o.id=od.order_id
            WHERE user_id=#{userId} order by create_time desc

    </select>
    <resultMap          id="orderInfoListMap"          type="com.atguigu.gmall.bean.OrderInfo"
autoMapping="true">
        <result property="id" column="id" ></result>

        <collection   property="orderDetailList"   ofType="com.atguigu.gmall.bean.OrderDetail"
autoMapping="true">
            <result property="id" column="order_detail_id" ></result>
        </collection>

    </resultMap>

</mapper>
```

OrderServiceImpl

```
@Override
public List<OrderInfo> getOrderListByUser(String userId) {
    // 优先去查缓存
    //缓存未命中 去查库

    List<OrderInfo> orderInfoList =
orderInfoMapper.selectOrderListByUser(Long.parseLong(userId));
    return orderInfoList;
}
```

OrderController

```
@RequestMapping(value = "list",method = RequestMethod.GET)
@LoginRequire
public String getOrderList(HttpServletRequest httpServletRequest,Model model){
        String userId =(String) httpServletRequest.getAttribute("userId");
        List<OrderInfo> orderList    = orderService.getOrderListByUser(userId);
        model.addAttribute("orderList", orderList );
        return "list";
}
```

list.html (涉及渲染的片段)

```
<table class="table" th:each="orderInfo:${orderList}">
   <tr>
      <td colspan="7" style="background:#F7F7F7"    class="order-header" >
         <span                                                    style="color:#AAAAAA"
th:text="${#dates.format(orderInfo.createTime,'yyyy-mm-dd HH:MM:SS')}"> </span>
         <span><ruby          style="color:#AAAAAA">        订        单        号        :</ruby><span
th:text="${orderInfo.id}"></span>    </span>
         <span th:text="${orderInfo.orderStatus.getComment()}">已拆分  </span>
      </td>
   </tr>

   <!--不拆单情况---------------->
   <!--不拆单情况---------------->
   <!--不拆单情况---------------->
   <tr                                              th:if="${orderInfo.orderStatus.toString()!='SPLIT'}"
th:each="orderDetail,state:${orderInfo.orderDetailList}">
      <td colspan="3" class="item">
         <img src="img/ding.jpg" alt="" class="img">
         <div>
            <p th:text="${orderDetail.skuName}"></p>
         </div>
         <div style="margin-left:15px;" th:text="'x'+${orderDetail.skuNum}"></div>
      </td>

      <td                th:if="${state.index==0}"                th:rowspan="${state.size}"        ><span
th:text="${orderInfo.consignee}"></span> </td>
      <td th:if="${state.index==0}"    th:rowspan="${state.size}">
         <div style="margin-left:15px;"    th:text="'总额  ￥'+${orderInfo.totalAmount}"> </div>
         <hr style="width:90%;">
         <p th:text="${orderInfo.paymentWay.getComment()}" >在线支付</p>
      </td>
   <td th:if="${state.index==0}"    th:rowspan="${state.size}">
      <ul>
         <li  style="color:#71B247;"  th:text="${orderInfo.orderStatus.getComment()}">等待收货
```

```
</li>
        <li    style="margin:4px    0;"    class="hide"><i    class="table_i2"></i>    跟    踪    <i
class="table_i3"></i>
          <div class="hi">
            <div class="p-tit">
              普通快递    运单号:390085324974
            </div>
            <div class="hideList">
              <ul>
                <li>
                  [北京市] 在北京昌平区南口公司进行签收扫描,快件已被拍照(您
                  的快件已签收,感谢您使用韵达快递)签收
                </li>
                <li>
                  [北京市] 在北京昌平区南口公司进行签收扫描,快件已被拍照(您
                  的快件已签收,感谢您使用韵达快递)签收
                </li>
                <li>
                  [北京昌平区南口公司] 在北京昌平区南口公司进行派件扫描
                </li>
                <li>
                  [北京市] 在北京昌平区南口公司进行派件扫描;派送业务员:业务员;联系
电话:17319268636
                </li>
              </ul>
            </div>
          </div>
        </li>
        <li class="tdLi">订单详情</li>
      </ul>
    </td>
    <td th:if="${state.index==0}"    th:rowspan="${state.size}">
      <button>确认收货</button>
      <p style="margin:4px 0; ">取消订单</p>
      <p>催单</p>
    </td>
    </tr>
</table>
```

支付宝接口

版本：V 1.0

www.atguigu.com

一、支付业务

1　支付宝业务简介

买家在商户网站选择需购买的商品，填写订单信息后，点击立即购买。

网页跳转到支付宝收银台页面。

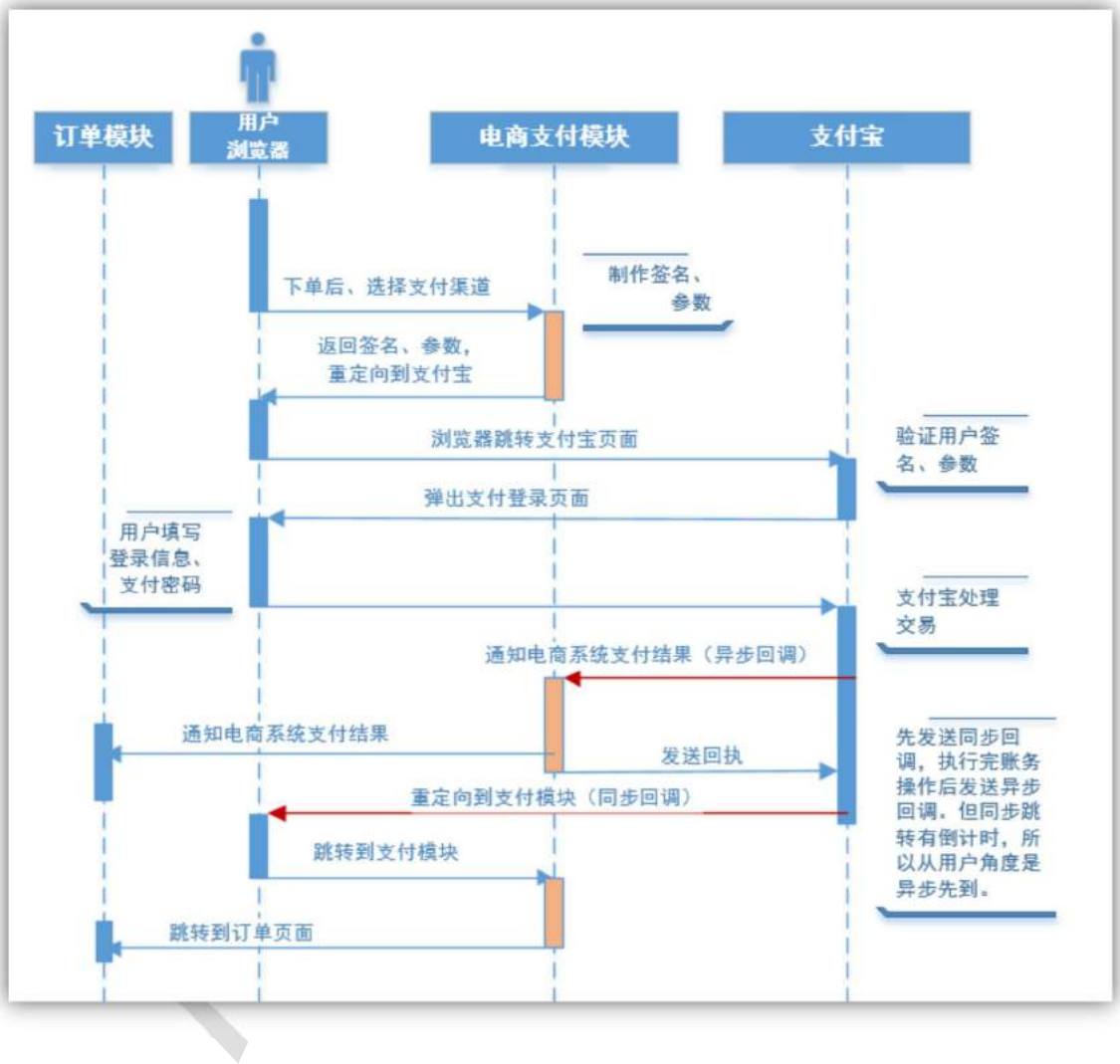
用户可以使用支付宝 App 扫一扫屏幕二维码，待手机提示付款后选择支付工具输入密码即可完成支付；

如果不使用手机支付，也可以点击上图右侧的“登录账户付款”，输入支付宝账号和支付密码登录 PC 收银台。

用户选择付款方式，输入支付密码后点击“确认付款”。

付款成功。

2 过程分析



3 对接支付宝的准备工作

1、申请条件

1. 企业或个体工商户可申请；
2. 提供真实有效的营业执照，且支付宝账户名称需与营业执照主体一致；
3. 网站能正常访问且页面信息有完整商品内容；
4. 网站必须通过 ICP 备案，个体户备案需与账户主体一致。

（团购类网站不支持个体工商户签约）

支付手续费

电脑网站支付

服务名称	费率	服务期限
单笔费率	0.6%	1年

费率说明：
助力中小商户，从签约日至2018.12.31日优惠费率为0.55%（不包括特殊行业）
特殊行业：休闲游戏；网络游戏点卡、游戏渠道代理；游戏系统商；网游周边服务、交易平台；网游运营商（含网页游戏）

4 申请步骤：

1、 支付宝商家中心中申请　https://www.alipay.com/

一个工作日后登录到蚂蚁金服开发者中心中：

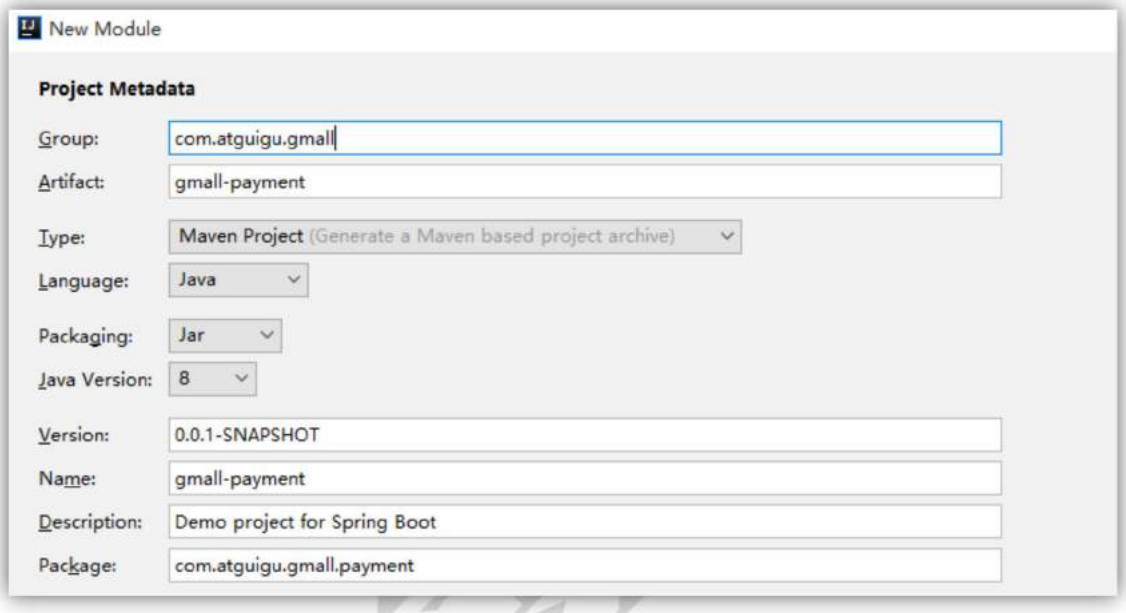
可以查看到一个已经签约上线的应用。 其中非常重要的是这个 APPID，需要记录下来之后的程序中要用到这个参数。

点击查看

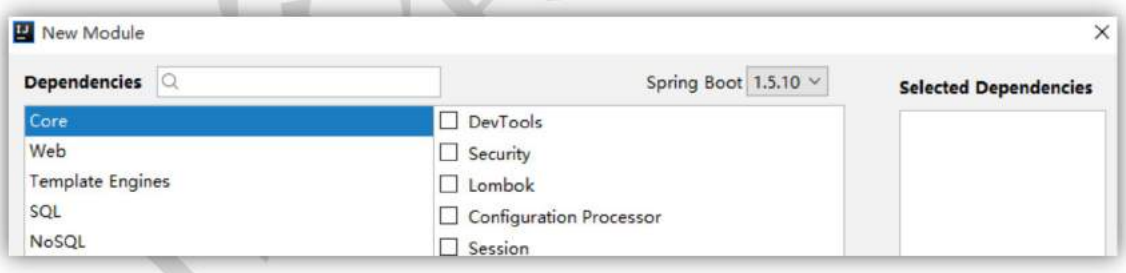
到此为止，电商网站可以访问支付宝的最基本的准备已经完成。

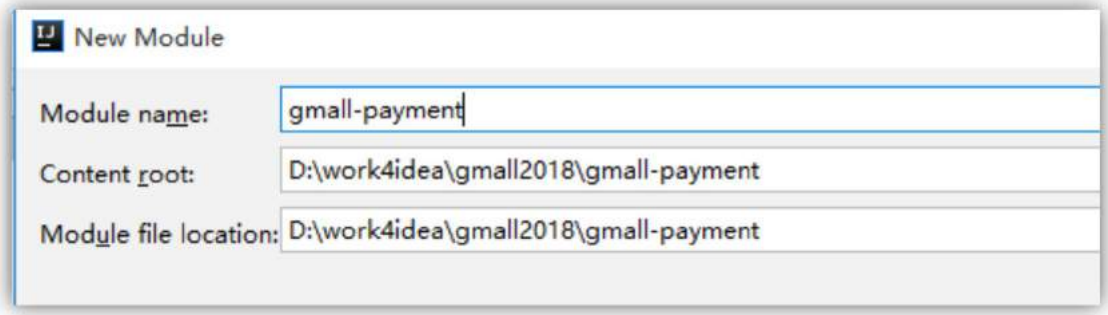
接下来搭建支付模块

二、支付模块搭建



这里由于支付的页面部分非常简单，便于讲解所以将支付模块的 service 和 web 工程使用一个模块





pom.xml

```
<?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.atguigu.gmall</groupId>
    <artifactId>gmall-payment</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>gmall-payment</name>
    <description>Demo project for Spring Boot</description>

    <parent>
        <groupId>com.atguigu.gmall</groupId>
        <artifactId>gmall-parent</artifactId>
        <version>1.0-SNAPSHOT</version>
    </parent>


    <dependencies>

        <dependency>
            <groupId>com.atguigu.gmall</groupId>
            <artifactId>gmall-interface</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>

        <dependency>
            <groupId>com.atguigu.gmall</groupId>
            <artifactId>gmall-web-util</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
```

```
        <dependency>
            <groupId>com.atguigu.gmall</groupId>
            <artifactId>gmall-service-util</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
    </dependencies>

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


</project>
```

application.properties

```
server.port=8090

logging.level.root=debug

spring.thymeleaf.cache=false
spring.thymeleaf.mode=LEGACYHTML5

spring.dubbo.application.name=payment
spring.dubbo.registry.protocol=zookeeper
spring.dubbo.registry.address=192.168.67.163:2181
spring.dubbo.base-package=com.atguigu.gmall
spring.dubbo.protocol.name=dubbo
spring.dubbo.consumer.timeout=1000000
spring.dubbo.consumer.check=false

spring.datasource.url=jdbc:mysql://59.110.141.236:3306/gmall?characterEncoding=UTF-8
spring.datasource.username=root
spring.datasource.password=123456

mapper.enum-as-simple-type=true
mybatis.mapper-locations=classpath:mapper/*Mapper.xml
mybatis.configuration.mapUnderscoreToCamelCase=true
```

GmallPaymentApplication

```
@SpringBootApplication
```

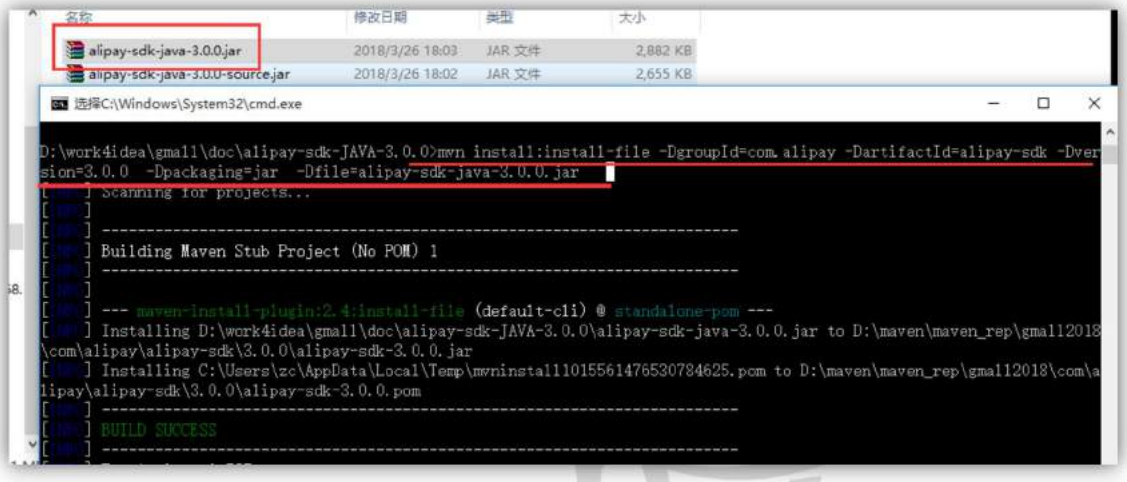
```
@ComponentScan(basePackages = "com.atguigu.gmall")
@MapperScan(basePackages = "com.atguigu.gmall.payment.mapper")
public class GmallPaymentApplication {

   public static void main(String[] args) {
      SpringApplication.run(GmallPaymentApplication.class, args);
   }
}
```

增加导入支付宝的 sdk 包到 maven 仓库。

调用支付宝的服务必须使用它的 jar 包，但是目前该 jar 包没有上传到网上的中心仓库上，所以只能手工导入到本地仓库中。

首先从蚂蚁金服的文档中心下载该 jar 包。

在 jar 包所在目录下，打开命令行工具



执行如下命令

```
mvn install:install-file -DgroupId=com.alipay -DartifactId=alipay-sdk -Dversion=3.0.0 -Dpackaging=jar -Dfile=alipay-sdk-java-3.0.0.jar
```

这样在 pom.xml 中就可以引入依赖

```
<dependency>
    <groupId>com.alipay</groupId>
    <artifactId>alipay-sdk</artifactId>
    <version>3.0.0</version>
</dependency>
```

配置 host

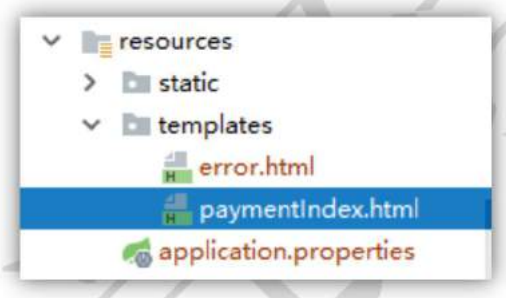
```
# common
0.0.0.0 account.jetbrains.com

# ------------------------------------------
# gmall
192.168.67.163 payment.gmall.com cart.gmall.com pas
```

配置 nginx

```
upstream payment.gmall.com{
          server 192.168.67.1:8090;
     }
    server {
       listen 80;
       server_name payment.gmall.com;
       location / {
            proxy_pass http://payment.gmall.com;
            proxy_set_header X-forwarded-for $proxy_add_x_forwarded_for;
       }
     }
```

三、支付渠道的选择页面

引入静态文件及页面



PaymentController

```
@Controller
public class PaymentController {

    @Reference
    OrderService orderService;

    @RequestMapping(value = "index",method = RequestMethod.GET)
    public String paymentIndex(@RequestParam("orderId")String  orderId, Model
model){

        OrderInfo orderInfo=orderService.getOrderInfo( orderId );
```

```
        model.addAttribute("orderId",orderInfo.getId());
        model.addAttribute("totalAmount",orderInfo.getTotalAmount());

        return "paymentIndex";
    }
}
```

在 OrderServiceImpl 中增加 getOrderInfo 方法

```
@Override
public OrderInfo getOrderInfo(String id) {
    OrderInfo orderInfo = orderInfoMapper.selectByPrimaryKey(id);

    OrderDetail orderDetailQuery=new OrderDetail();
    orderDetailQuery.setOrderId(orderInfo.getId());
    List<OrderDetail> orderDetailList = orderDetailMapper.select(orderDetailQuery);
    orderInfo.setOrderDetailList(orderDetailList);

    return orderInfo;
}
```

页面 html

```
<dd>
  <span th:text="'订单提交成功，请尽快付款！订单号：'+${orderId}"> </span>
  <span th:text="'应付金额'+${totalAmount}+'元'"> </span>
</dd>
```

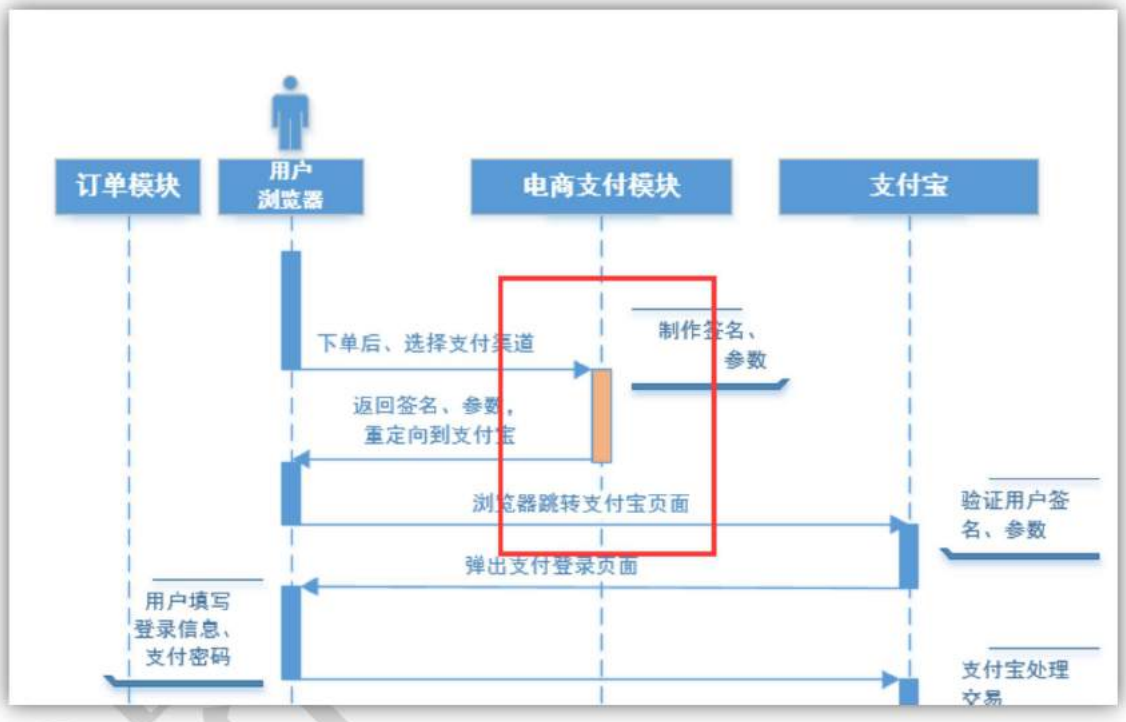
```
<form    method="post" id="paymentForm">

    <input type="hidden" name="orderId" th:value="${orderId}" >
</form>
```

js

```
<script type="text/javascript">
    $(function() {
        $("#paymentButton").click(function () {

$("#paymentForm").attr("action","/"+$("input[type='radio']:checked").val()+
"/submit")  ;
            $("#paymentForm").submit();
            console.log($("#paymentForm").html()) ;
            console.log($("#paymentForm").attr("action")) ;
        })


    })
```

测试启动，支付渠道选择页面

四、跳转支付宝



1 、分析

功能要求：

1、 制作支付宝需要的各种参数

2、 保存支付信息，作用：追踪交易状态、去重、对账

3、 帮助用户跳转到支付宝的页面

分析支付宝需要什么参数？

查看蚂蚁金服的文档中心中的电脑网站支付说明

关键参数说明:

配置参数	示例值解释	获取方式/示例值
URL	支付宝网关（固定）	https://openapi.alipay.com/gateway.do
APPID	APPID 即创建应用后生成	获取见上面创建应用
APP_PRIVATE_KEY	开发者私钥，由开发者自己生成	获取详见上面配置密钥
FORMAT	参数返回格式，只支持json	json（固定）
CHARSET	编码集，支持GBK/UTF-8	开发者根据实际工程编码配置
ALIPAY_PUBLIC_KEY	支付宝公钥，由支付宝生成	获取详见上面配置密钥
SIGN_TYPE	商户生成签名字符串所使用的签名算法类型，目前支持RSA2和RSA，推荐使用RSA2	RSA2

这些参数可以一次性注入到阿里提供 alipayClient 中，以后就不用再赋值了。

业务参数

请求参数

参数	类型	是否必填	最大长度	描述	示例值
out_trade_no	String	是	64	商户订单号，64个字符以内，可包含字母、数字、下划线；需保证在商户端不重复	20150320010101001
product_code	String	是	64	销售产品码，与支付宝签约的产品码名称。 注：目前仅支持FAST_INSTANT_TRADE_PAY	FAST_INSTANT_TRADE_PAY
total_amount	Price	是	11	订单总金额，单位为元，精确到小数点后两位，取值范围[0.01,100000000]	88.88
subject	String	是	256	订单标题	Iphone6 16G

2 、 支付信息的保存

表结构 payment_info

id	主键自动生成
out_trade_no	订单中已生成的对外交易编号。订单中获取
alipay_trade_no	订单编号　初始为空，支付宝回调时生成
total_amount	订单金额。订单中获取

subject	交易内容。利用商品名称拼接。
payment_status	支付状态，默认值未支付。
create_time	创建时间，当前时间
callback_time	回调时间，初始为空，支付宝异步回调时记录
callback_content	回调信息，初始为空，支付宝异步回调时记录

实体 Bean

```
public class PaymentInfo {

    @Column
    @Id
    private String    id;

    @Column
    private String outTradeNo;

    @Column
    private String orderId;

    @Column
    private String alipayTradeNo;

    @Column
    private BigDecimal totalAmount;

    @Column
    private String Subject;

    @Column
    private PaymentStatus paymentStatus;

    @Column
    private Date createTime;

    @Column
    private Date confirmTime;

    @Column
    private String callbackContent;
}
```

PaymentStatus

```
public enum PaymentStatus {

    UNPAID("支付中"),
    PAID("已支付"),
```

```
    PAY_FAIL("支付失败"),
    ClOSED("已关闭");

    private String name ;

    PaymentStatus(String name) {
        this.name=name;
    }
}
```

3 PaymentServiceImpl

```
@Override
public void savePaymentInfo(PaymentInfo paymentInfo) {
    //必须保证每个订单只有唯一的支付信息，所以如果之前已经有了该笔订单的支付信息，
那么只更新时间
    PaymentInfo paymentInfoQuery=new PaymentInfo();
    paymentInfoQuery.setOrderId(paymentInfo.getOrderId());

    PaymentInfo paymentInfoExists =
paymentInfoMapper.selectOne(paymentInfoQuery);
    if(paymentInfoExists!=null){
        paymentInfoExists.setCreateTime(new Date());
        paymentInfoMapper.updateByPrimaryKey(paymentInfoExists);
        return;
    }

    paymentInfo.setCreateTime(new Date());
    paymentInfoMapper.insertSelective(paymentInfo);
}
```

PaymentInfoMapper

```
public interface PaymentInfoMapper extends Mapper<PaymentInfo>{
}
```

OrderInfo

```
//生成摘要
public String getOrderSubject(){
	String body="";
	if(orderDetailList!=null&&orderDetailList.size()>0){
		body=	orderDetailList.get(0).getSkuName();
	}
	body+="等"+getTotalSkuNum()+"件商品";
	return body;

}

public Integer getTotalSkuNum(){
	Integer totalNum=0;
	for (OrderDetail orderDetail : orderDetailList) {
		totalNum+=	orderDetail.getSkuNum();
	}
	return totalNum;
}
```

初始化 AlipayClient

```
@Configuration
@PropertySource("classpath:alipay.properties")
public class AlipayConfig {

	@Value("${alipay_url}")
	private String alipay_url;

	@Value("${app_private_key}")
	private String app_private_key;

	@Value("${app_id}")
	private String app_id;


	public final static String format="json";
	public final static String charset="utf-8";
	public final static String sign_type="RSA2";


	public static String return_payment_url;

	public static	String notify_payment_url;

	public static	String return_order_url;
```

```
    public static    String alipay_public_key;

    @Value("${alipay_public_key}")
    public     void setAlipay_public_key(String alipay_public_key) {
        AlipayConfig.alipay_public_key = alipay_public_key;
    }

    @Value("${return_payment_url}")
    public     void setReturn_url(String return_payment_url) {
        AlipayConfig.return_payment_url = return_payment_url;
    }

    @Value("${notify_payment_url}")
    public     void setNotify_url(String notify_payment_url) {
        AlipayConfig.notify_payment_url = notify_payment_url;
    }

    @Value("${return_order_url}")
    public     void setReturn_order_url(String return_order_url) {
        AlipayConfig.return_order_url = return_order_url;
    }

    @Bean
    public AlipayClient alipayClient(){
        AlipayClient                                                    alipayClient=new
DefaultAlipayClient(alipay_url,app_id,app_private_key,format,charset,
alipay_public_key,sign_type );

        return alipayClient;
    }
}
```

增加 alipay.properties

```
alipay_url=https://openapi.alipay.com/gateway.do

app_id=2018020102122556

app_private_key=MIIEvQI……N1b/88D5yMuaZhZNFeUdWb+SmtP9DAzAW
YEfJzetw/3cIimWu4NJzVQOaojaGA58oo2+fub43Xn25Jq4rvSVe3oLdb5xWkw5Q=

alipay_public_key=MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAhkZi6W0wn/prX
+NIIF9ATb5Z8ReKK4hFYtBrwe……..G+qbjTcZAzgNPfuiD0zXgt/YYjMQMzck75BOmwnYO
am2ajODUSQn8Xybsa7wQIDAQAB

return_payment_url=http://payment.gmall.com/alipay/callback/return
```

```
notify_payment_url=http://60.205.215.91/alipay/callback/notify
‘
return_order_url=http://order.gmall.com/list
```

问题：1 密钥如何得来，为什么有两个

利用工具生成　保存本地私钥 和支付宝公钥　把本地公钥上传给支付宝

2、url 为什么有的用域名，有的用 ip 地址。

4 PaymentController

分析 ：

1、通过 orderId 取得订单信息

2、组合对应的支付信息保存到数据库。

3、组合需要传给支付宝的参数。

4、根据返回的表单 html，传给浏览器。

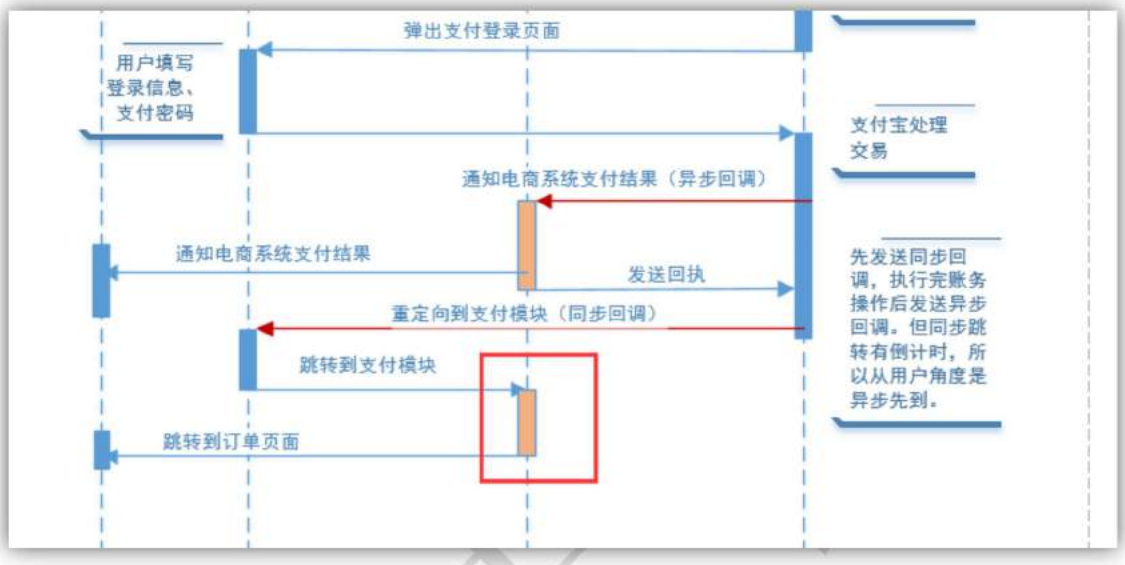
支付宝开发手册：https://docs.open.alipay.com/270/105900/

```
@RequestMapping(value = "/alipay/submit",method = RequestMethod.POST)
@ResponseBody
public ResponseEntity<String> paymentAlipay(HttpServletRequest request, HttpServletResponse
httpServletResponse, Model model) throws IOException {


    String orderId = request.getParameter("orderId");
    if(orderId==null||orderId.length()==0){
        return  ResponseEntity.status(HttpStatus.BAD_REQUEST).build();
    }
    //获取订单信息
    OrderInfo orderInfo=    orderService.getOrderInfo( orderId );
    if(orderInfo==null ){
        //没有对应的订单
        return  ResponseEntity.status(HttpStatus.BAD_REQUEST).build();
    }
    //保存支付信息
    PaymentInfo paymentInfo =new PaymentInfo();
```

```
        paymentInfo.setOrderId(orderId);
        paymentInfo.setOutTradeNo(orderInfo.getOutTradeNo());
        paymentInfo.setSubject(orderInfo.getOrderSubject());
        paymentInfo.setPaymentStatus(PaymentStatus.UNPAID);
        paymentInfo.setTotalAmount(orderInfo.getTotalAmount());
        paymentService.savePaymentInfo(paymentInfo);


        //利用支付宝客户端生成表单页面
        AlipayTradePagePayRequest alipayRequest=new AlipayTradePagePayRequest();

        alipayRequest.setReturnUrl(AlipayConfig.return_payment_url);
        alipayRequest.setNotifyUrl(AlipayConfig.notify_payment_url);

        Map<String,String> paramMap=new HashMap<>();
        paramMap.put("out_trade_no",paymentInfo.getOutTradeNo());
        paramMap.put("product_code","FAST_INSTANT_TRADE_PAY");
        paramMap.put("total_amount",paymentInfo.getTotalAmount().toString());
        paramMap.put("subject",paymentInfo.getSubject());
        String paramJson = JSON.toJSONString(paramMap);
        alipayRequest.setBizContent(paramJson);
        String form="";
        try {
            form = alipayClient.pageExecute(alipayRequest).getBody();
        } catch (AlipayApiException e) {
            e.printStackTrace();
        }

        httpServletResponse.setContentType("text/html;charset=utf-8" );

        //把表单 html 打印到客户端浏览器
        return    ResponseEntity.ok().body(form) ;
}
```

测试页面

五　支付后回调—同步回调

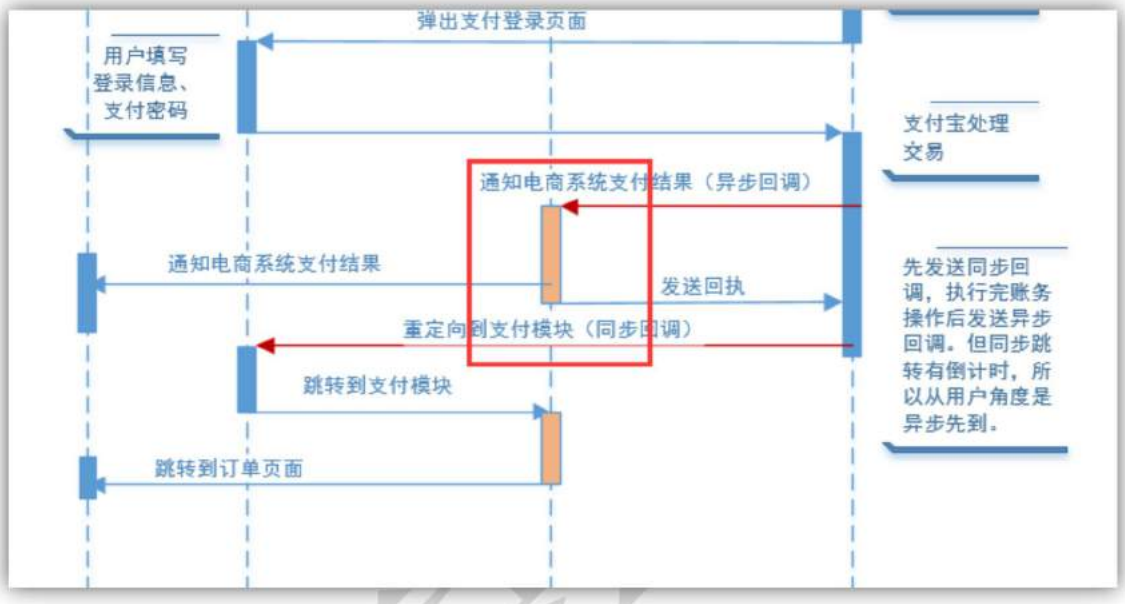


PaymentController

```
@RequestMapping(value="/alipay/callback/return",method = RequestMethod.GET)
public String callbackReturn(HttpServletRequest request, Model model) throws
UnsupportedEncodingException {
    System.out.println(" callback return to "+AlipayConfig.return_order_url);

    return  "redirect:"+AlipayConfig.return_order_url;
}
```

这里 requestMapping 对应的路径必须与之前传给支付宝的 alipayRequest.setReturnUrl 保持一致。

六　支付宝回调—异步回调



异步回调有两个重要的职责：

确认并记录用户已付款，通知电商模块。新版本的支付接口已经取消了同步回调的支付结果传递。所以用户付款成功与否全看异步回调。

接收到回调要做的事情：

1、 验证回调信息的真伪

2、 验证用户付款的成功与否

3、 把新的支付状态写入支付信息表中。

4、 通知电商

5、 给支付宝返回回执。

PaymentController

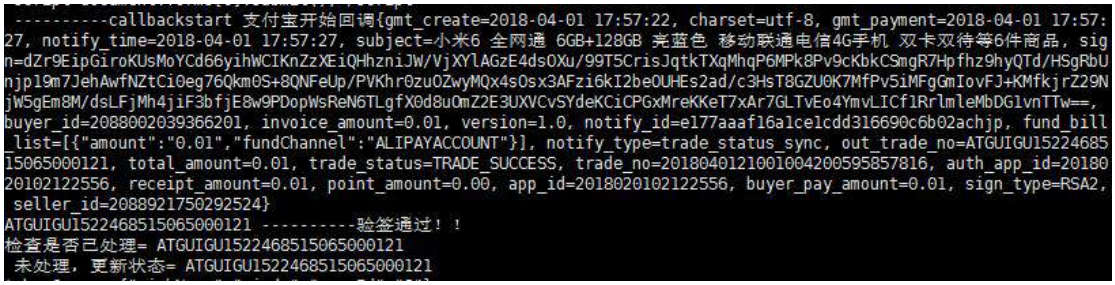
```
@RequestMapping(value="/alipay/callback/notify",method = RequestMethod.POST)
@ResponseBody
public String callbackNotify(@RequestParam Map<String,String> paramMap) {
      System.out.println(" ----------callbackstart  支付宝开始回调"+paramMap.toString() );
```

```
        //验证签名
        boolean isCheckPass=false;
        try {
            isCheckPass = AlipaySignature.rsaCheckV1(paramMap, AlipayConfig.alipay_public_key,
AlipayConfig.charset, AlipayConfig.sign_type);
        } catch (AlipayApiException e) {
            e.printStackTrace();
        }
        if(!isCheckPass){
            System.out.println(" ----------验签不通过!! " );
            return "验签不通过!! ";
        }
        System.out.println(" ----------验签通过!! " );
        //验证成功标志
        String trade_status = paramMap.get("trade_status");
        if("TRADE_SUCCESS".equals(trade_status)){
            //检查当前支付状态
            String outTradeNo = paramMap.get("out_trade_no");
            PaymentInfo paymentInfoQuery=new PaymentInfo();
            paymentInfoQuery.setOutTradeNo(outTradeNo);
            PaymentInfo paymentInfo = paymentService.getPaymentInfo(paymentInfoQuery);
            if (paymentInfo==null) {
                return "error: not exists out_trade_no:"+outTradeNo;
            }
            System.out.println("检查是否已处理= " +outTradeNo );
            if(paymentInfo.getPaymentStatus()==PaymentStatus.PAID){
                //如果已经处理过了 就直接返回成功标志
                System.out.println(" 已处理= " +outTradeNo );
                return "success";
            }else {
                //先更新支付状态
                System.out.println(" 未处理，更新状态= " +outTradeNo );
                PaymentInfo paymentInfo4Upt=new PaymentInfo();
                paymentInfo4Upt.setPaymentStatus(PaymentStatus.PAID);
                paymentInfo4Upt.setConfirmTime(new Date());
                paymentInfo.setCallbackContent(paramMap.toString());

paymentService.updatePaymentInfoByOutTradeNo(outTradeNo,paymentInfo4Upt);

                //发送通知给订单
                paymentService.sendPaymentResult2MQ(paymentInfo.getOrderId());
                return  "success";

            }

        }
        return "";

}
```

PaymentServiceImpl

```
/***
 * 根据 outTradeNo 更新支付信息
 * @param outTradeNo
 * @param paymentInfo
 */
public void updatePaymentInfoByOutTradeNo(String outTradeNo , PaymentInfo
paymentInfo){
    Example example=new Example(PaymentInfo.class);
    example.createCriteria().andEqualTo("outTradeNo",outTradeNo);

    paymentInfoMapper.updateByExampleSelective(paymentInfo,example);

}


/**
 * 根据 outTrade 查询支付信息
 * @param outTradeNo
 * @return
 */
public PaymentInfo getPaymentInfoByOutTradeNo(String outTradeNo) {
    Example example=new Example(PaymentInfo.class);
    example.createCriteria().andEqualTo("outTradeNo",outTradeNo);

    PaymentInfo paymentInfo = paymentInfoMapper.selectOneByExample(example);
    return paymentInfo;
}
```

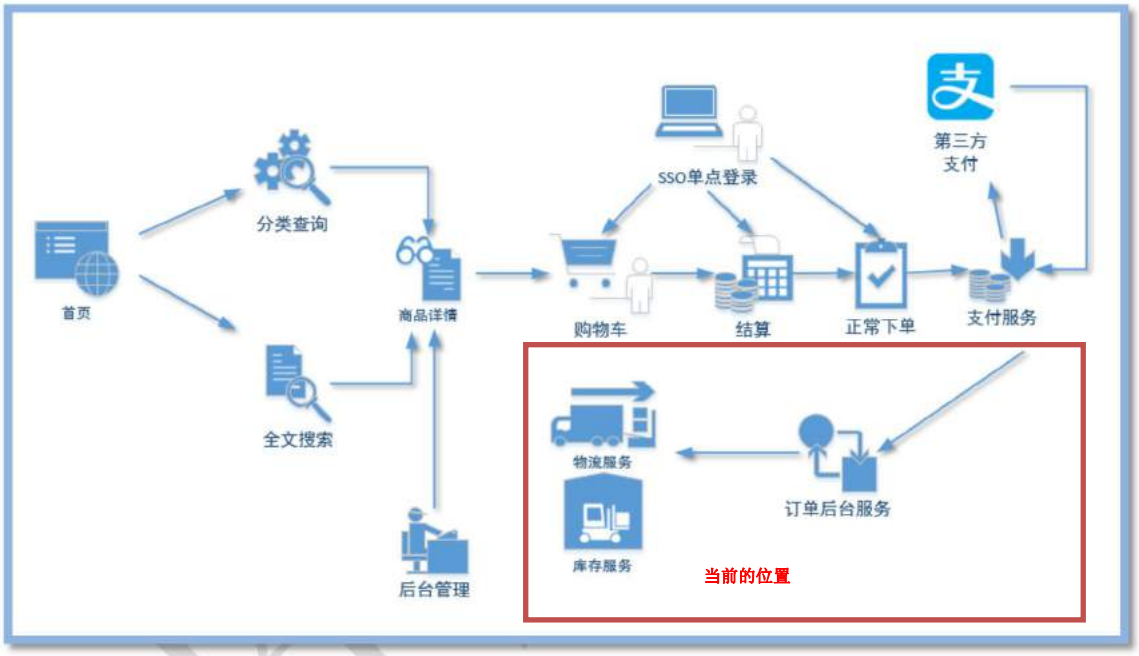
测试将应用程序发布到 aliyun 服务器上，支付并观察支付宝的回调。



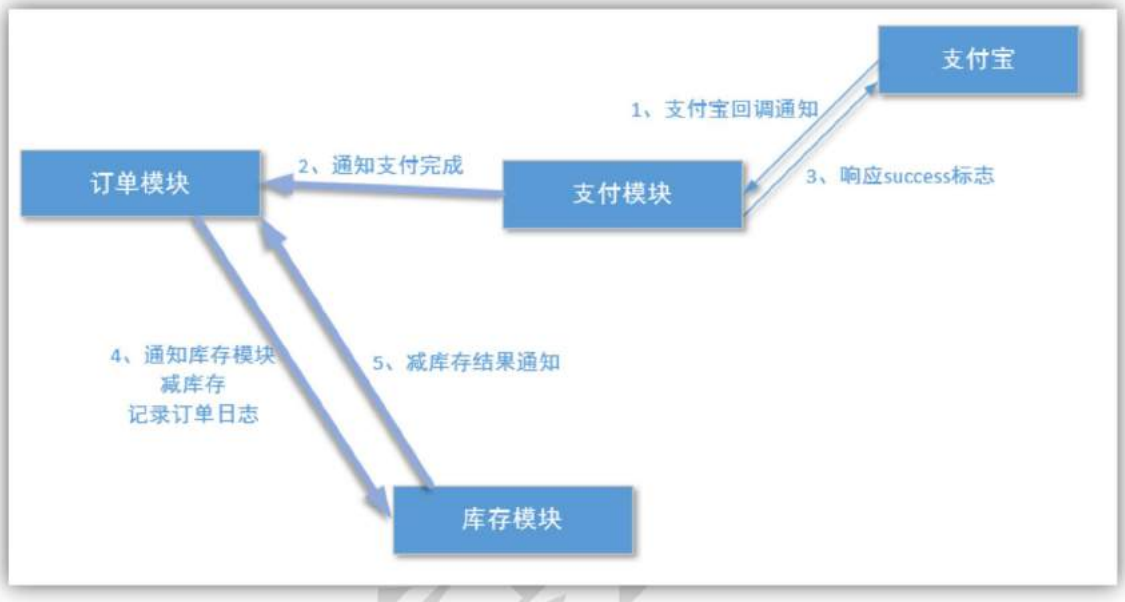
异步通信

版本：V 1.0

www.atguigu.com



一、分布式的业务场景



1 、如何高效完成各个分布式系统的协作

通过消息队列来达到异步解耦的效果，减少了程序之间的阻塞等待时间，降低了因为服务之间调用的依赖风险。

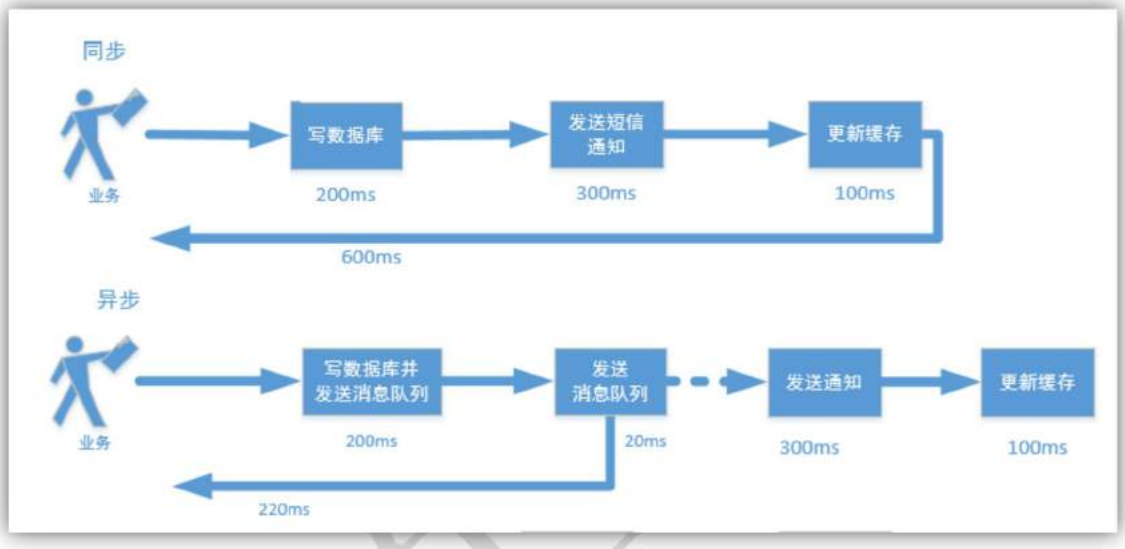
2、消息的弊端？如何解决？

消息队列的问题在于**不确定性**，不能绝对保证消息的准确到达，所以要引入延迟、周期性的主动轮询，来发现未到达的消息，从而进行补偿。

二、消息队列简介

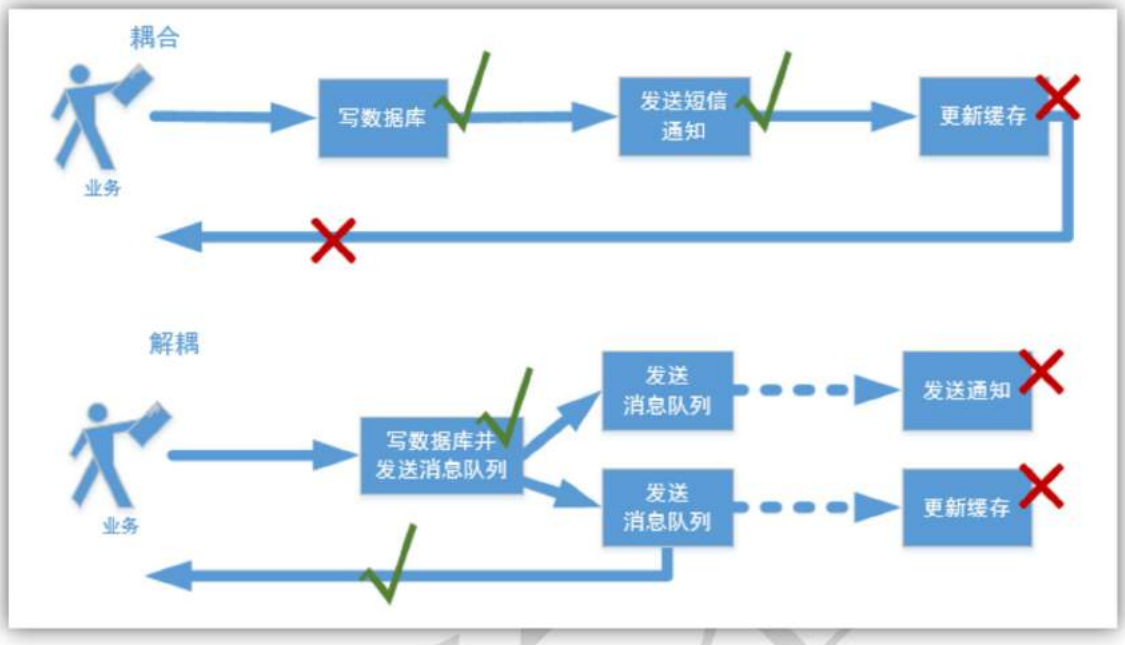
消息队列，也叫消息中间件。消息的传输过程中保存消息的容器。
消息队列都解决了什么问题？

1、 异步

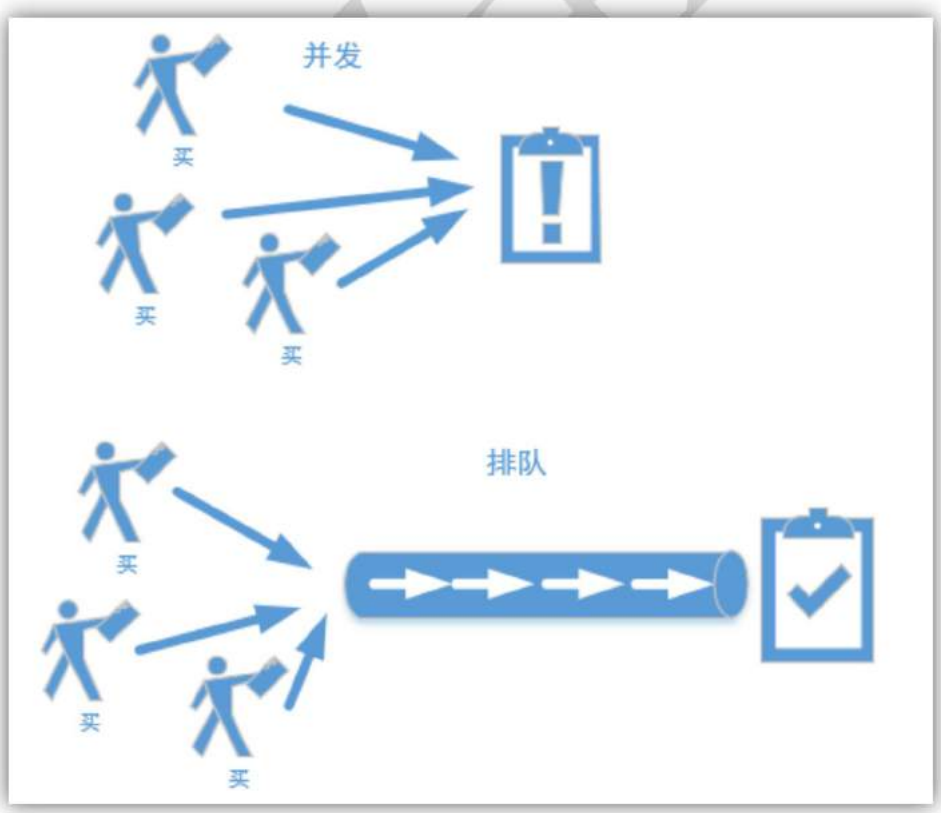


2、并行

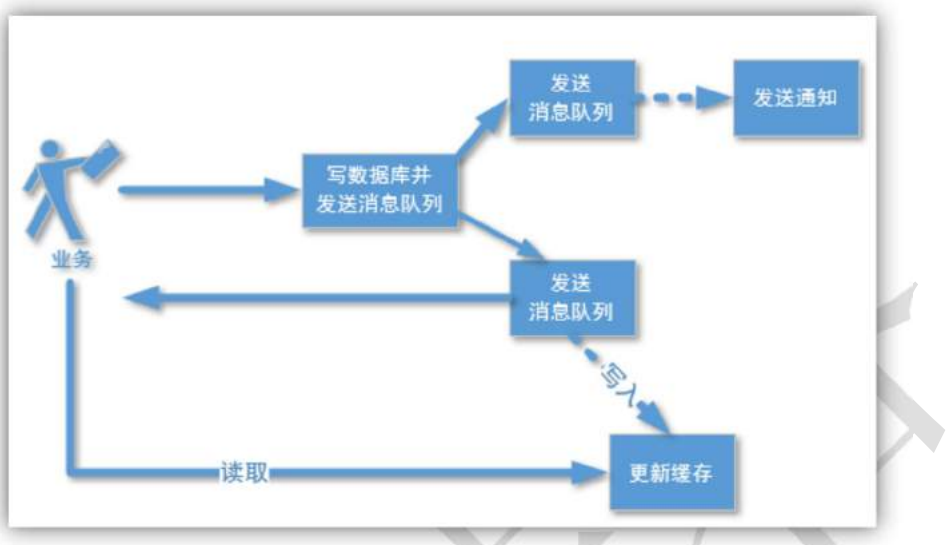
3、解耦



4、排队（削峰填谷）



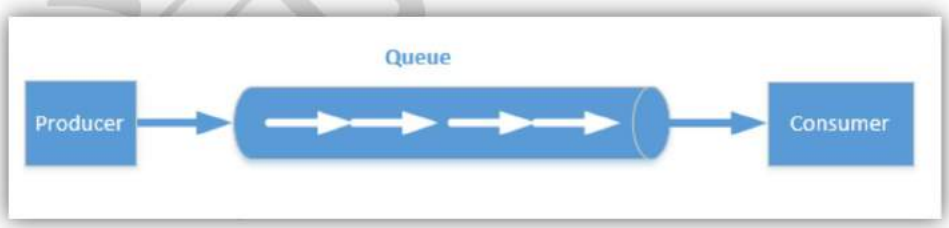
5 弊端：不确定性和延迟



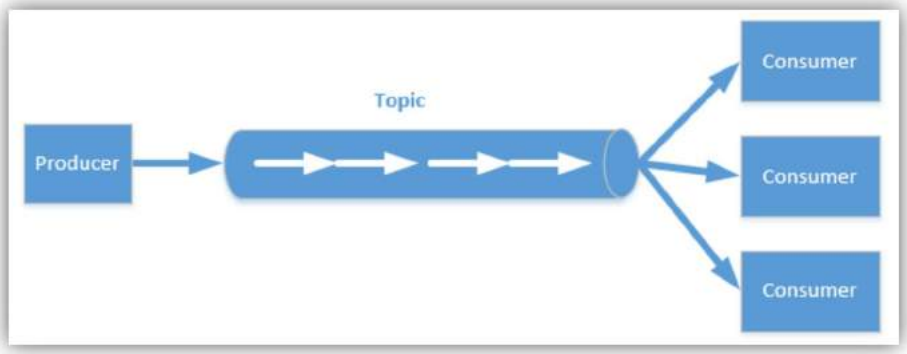
解决方案：最终一致

消息模式

点对点



订阅



三 消息队列工具 ActiveMQ

1 、简介

同类产品： RabbitMQ 、 Kafka、Redis（List）

1.1 对比 RabbitMQ

最接近的同类型产品，经常拿来比较，性能伯仲之间，基本上可以互相替代。最主要区别是二者的协议不同 RabbitMQ 的协议是 AMQP(Advanced Message Queueing Protoco)，而 ActiveMQ 使用的是 JMS(Java Messaging Service)协议。顾名思义 JMS 是针对 Java 体系的传输协议，队列两端必须有 JVM，所以如果开发环境都是 java 的话推荐使用 ActiveMQ，可以用 Java 的一些对象进行传递比如 Map、BLob、Stream 等。而 AMQP 通用行较强，非 java 环境经常使用，传输内容就是标准字符串。

另外一点就是 RabbitMQ 用 Erlang 开发，安装前要装 Erlang 环境，比较麻烦。ActiveMQ 解压即可用不用任何安装。

1.2 对比 KafKa

Kafka 性能超过 ActiveMQ 等传统 MQ 工具，集群扩展性好。

弊端是：

在传输过程中可能会出现消息重复的情况，

不保证发送顺序

一些传统 MQ 的功能没有，比如消息的事务功能。

所以通常用 Kafka 处理大数据日志。

1.3 对比 Redis

其实 Redis 本身利用 List 可以实现消息队列的功能，但是功能很少，而且队列体积较大时性能会急剧下降。对于数据量不大、业务简单的场景可以使用。

2 安装 ActiveMQ

拷贝 apache-activemq-5.14.4-bin.tar.gz 到 Linux 服务器的/opt 下

解压缩 tar -zxvf apache-activemq-5.14.4-bin.tar.gz

重命名 mv apache-activemq-5.14.4 activemq

vim /opt/activemq/bin/activemq

```
46
47 # -----------------------------------------------------------------------------
48 # IMPROVED DEBUGGING (execute with bash -x)
49 # export PS4=' ${BASH_SOURCE}:${LINENO}(${FUNCNAME[0]}) '
50 #
51 # Backup invocation parameters
52 COMMANDLINE_ARGS="$@"
53 EXEC_OPTION=""
54 JAVA_HOME="/opt/jdk1.8.0_152"
55 JAVA_CMD="/opt/jdk1.8.0_152/bin"
56 # -----------------------------------------------------------------------------
57 # HELPERS
```

增加两行

```
JAVA_HOME="/opt/jdk1.8.0_152"

JAVA_CMD="/opt/jdk1.8.0_152/bin
```

注册服务

```
ln -s    /opt/activemq/bin/activemq    /etc/init.d/activemq

chkconfig --add activemq
```

启动服务

service activemq start

```
[root@jack init.d]# service activemq start
INFO: Loading '/opt/activemq//bin/env'
INFO: Using java '/opt/jdk1.8.0_152/bin/java'
INFO: Starting - inspect logfiles specified in logging.properties and log4j.properties to get details
INFO: pidfile created : '/opt/activemq//data/activemq.pid' (pid '4846')
[root@jack init.d]# ps -ef|grep java
```

关闭服务

service activemq stop

通过 netstat 查看端口

```
tcp6       0      0 :::61614                :::*                    LISTEN      4846/java
tcp6       0      0 :::34222                :::*                    LISTEN      4846/java
tcp6       0      0 :::61616                :::*                    LISTEN      4846/java
tcp6       0      0 :::1883                 :::*                    LISTEN      4846/java
tcp6       0      0 127.0.0.1:7005          :::*                    LISTEN      882/java
tcp6       0      0 :::8161                 :::*                    LISTEN      4846/java
tcp6       0      0 :::7009                 :::*                    LISTEN      882/java
tcp6       0      0 :::39811                :::*                    LISTEN      878/java
tcp6       0      0 :::2181                 :::*                    LISTEN      878/java
```

activemq 两个重要的端口，一个是提供消息队列的默认端口：61616

另一个是控制台端口 8161

通过控制台测试

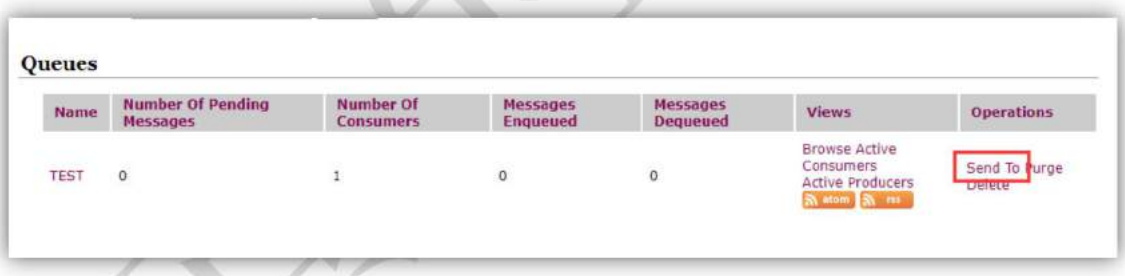
启动消费端

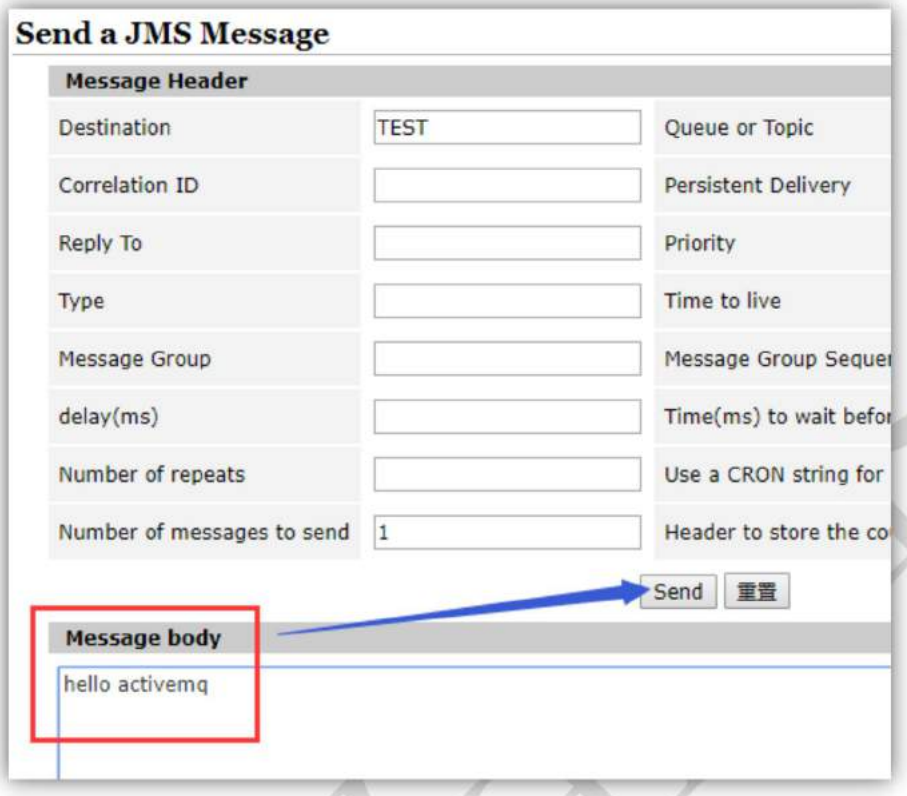
```
[root@jack init.d]# service activemq consumer
INFO: Loading '/opt/activemq//bin/env'
INFO: Using java '/opt/jdk1.8.0_152/bin/java'
Java Runtime: Oracle Corporation 1.8.0_152 /opt/jdk1.8.0_152/jre
  Heap sizes: current=62976k  free=61992k  max=932352k
    JVM args: -Xms64M -Xmx1G -Djava.util.logging.config.file=logging.pro
pt/activemq//conf/login.config -Dactivemq.classpath=/opt/activemq//conf:
/activemq/ -Dactivemq.base=/opt/activemq/ -Dactivemq.conf=/opt/activemq/
Extensions classpath:
  [/opt/activemq/lib,/opt/activemq/lib/camel,/opt/activemq/lib/optional,
a]
ACTIVEMQ_HOME: /opt/activemq
ACTIVEMQ_BASE: /opt/activemq
ACTIVEMQ_CONF: /opt/activemq/conf
ACTIVEMQ_DATA: /opt/activemq/data
 INFO | Connecting to URL: failover://tcp://localhost:61616 (null:null)
 INFO | Consuming queue://TEST
 INFO | Sleeping between receives 0 ms
 INFO | Running 1 parallel threads
 INFO | Successfully connected to tcp://localhost:61616
 INFO | consumer-1 wait until 1000 messages are consumed
```

进入网页控制台

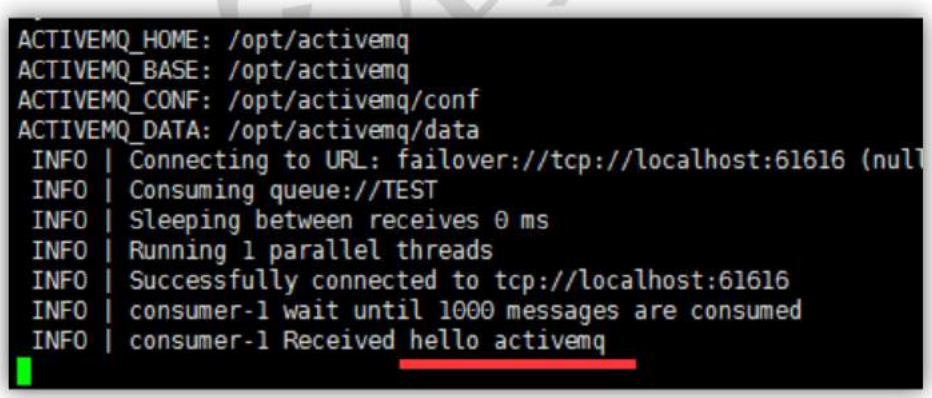
账号/密码默认： admin/admin

点击 **Queues**





观察客户端



3 在 Java 中使用消息队列

3.1 在 gmall-service-util 中导入依赖坐标

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-activemq</artifactId>
   <exclusions>
      <exclusion>
         <groupId>org.slf4j</groupId>
         <artifactId>slf4j-log4j12</artifactId>
      </exclusion>
   </exclusions>
</dependency>

<dependency>
   <groupId>org.apache.activemq</groupId>
   <artifactId>activemq-pool</artifactId>
   <version>5.15.2</version>
   <exclusions>
      <exclusion>
         <groupId>org.slf4j</groupId>
         <artifactId>slf4j-log4j12</artifactId>
      </exclusion>
   </exclusions>
</dependency>
```

3.2 producer 端

```
public static void main(String[] args) {

    ConnectionFactory connect = new
ActiveMQConnectionFactory("tcp://192.168.67.163:61616");
    try {
        Connection connection = connect.createConnection();
        connection.start();
        //第一个值表示是否使用事务，如果选择 true，第二个值相当于选择 0
        Session session = connection.createSession(true, Session.SESSION_TRANSACTED);
        Queue testqueue = session.createQueue("TEST1");

        MessageProducer producer = session.createProducer(testqueue);
        TextMessage textMessage=new ActiveMQTextMessage();
        textMessage.setText("今天天气真好！");
        producer.setDeliveryMode(DeliveryMode.PERSISTENT);
        producer.send(textMessage);
        session.commit();
        connection.close();

    } catch (JMSException e) {
        e.printStackTrace();
    }

}
```

3.3 consumer

```
public static void main(String[] args) {
    ConnectionFactory connect = new
ActiveMQConnectionFactory(ActiveMQConnection.DEFAULT_USER,ActiveMQConnec
tion.DEFAULT_PASSWORD,"tcp://192.168.67.163:61616");
    try {
        Connection connection = connect.createConnection();
        connection.start();
        //第一个值表示是否使用事务，如果选择 true，第二个值相当于选择 0
        Session session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);
        Destination testqueue = session.createQueue("TEST1");

        MessageConsumer consumer = session.createConsumer(testqueue);
```

```
        consumer.setMessageListener(new MessageListener() {
            @Override
            public void onMessage(Message message) {
                if(message instanceof TextMessage){
                    try {
                        String text = ((TextMessage) message).getText();
                        System.out.println(text);

                        //session.rollback();
                    } catch (JMSException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    }
                }
            }
        });


    }catch (Exception e){
        e.printStackTrace();
    }
}
```

3.4 关于事务控制

producer 提交时的事务	事务开启	只执行 send 并不会提交到队列中，只有当执行 session.commit()时，消息才被真正的提交到队列中。
	事务不开启	只要执行 send，就进入到队列中。
consumer 接收时的事	事务开启，签收必须写	收到消息后，消息并没有

务	Session.*SESSION_TRANSACTED*	**真正的被消费。消息只是被锁住。一旦出现该线程死掉、抛异常，或者程序执行了 session.rollback()那么消息会释放，重新回到队列中被别的消费端再次消费。**
	事务不开启，签收方式选择 Session.AUTO_ACKNOWLEDGE	只要调用 comsumer.receive 方法，自动确认。
	事务不开启，签收方式选择 Session.CLIENT_ACKNOWLEDGE	需要客户端执行 message.acknowledge(), 否则视为未提交状态，线程结束后，其他线程还可以接收到。 这种方式跟事务模式很像，区别是不能手动回滚,而且可以单独确认某个消息。
	事务不开启，签收方式选择 Session.DUPS_OK_ACKNOWLEDGE	在Topic模式下做批量签收时用的，可以提高性能。但是某些情况消息可能会被重复提交，使用这种模式的 consumer要可以处理重复提交的问题。

3.5 持久化与非持久化

通过 producer.setDeliveryMode(DeliveryMode.*PERSISTENT*) 进行设置

持久化的好处就是当 activemq 宕机的话，消息队列中的消息不会丢失。非持久化会丢

失。但是会消耗一定的性能。

四 与 springboot 整合

1 配置类 ActiveMQConfig

```
@Configuration
public class ActiveMQConfig {

    @Value("${spring.activemq.broker-url:disabled}")
    String brokerURL ;

    @Value("${activemq.listener.enable:disabled}")
    String listenerEnable;

    @Bean
    public ActiveMQUtil getActiveMQUtil() throws JMSException {
        if(brokerURL.equals("disabled")){
            return null;
        }
        ActiveMQUtil activeMQUtil=new ActiveMQUtil();
        activeMQUtil.init(brokerURL);
        return activeMQUtil;
    }

    //定义一个消息监听器连接工厂，这里定义的是点对点模式的监听器连接工厂
    @Bean(name = "jmsQueueListener")
    public DefaultJmsListenerContainerFactory
jmsQueueListenerContainerFactory(ActiveMQConnectionFactory activeMQConnectionFactory ) {
        DefaultJmsListenerContainerFactory factory = new
DefaultJmsListenerContainerFactory();
        if(!listenerEnable.equals("true")){
            return null;
        }

        factory.setConnectionFactory(activeMQConnectionFactory);
        //设置并发数
        factory.setConcurrency("5");

        //重连间隔时间
```

```
        factory.setRecoveryInterval(5000L);
        factory.setSessionTransacted(false);
        factory.setSessionAcknowledgeMode(Session.CLIENT_ACKNOWLEDGE);

        return factory;
    }


    @Bean
    public ActiveMQConnectionFactory activeMQConnectionFactory ( ){

        ActiveMQConnectionFactory activeMQConnectionFactory =
                new ActiveMQConnectionFactory(    brokerURL);
        return activeMQConnectionFactory;
    }

}
```

2 工具类 ActiveMQUtil

```
public class ActiveMQUtil {
    PooledConnectionFactory pooledConnectionFactory=null;

    public ConnectionFactory init(String brokerUrl) {

        ActiveMQConnectionFactory factory = new ActiveMQConnectionFactory(brokerUrl);
         //加入连接池
        pooledConnectionFactory=new PooledConnectionFactory(factory);
        //出现异常时重新连接
        pooledConnectionFactory.setReconnectOnException(true);
        //
        pooledConnectionFactory.setMaxConnections(5);
        pooledConnectionFactory.setExpiryTimeout(10000);
        return pooledConnectionFactory;
    }

    public ConnectionFactory getConnectionFactory(){
        return pooledConnectionFactory;
    }
}
```

五 在支付业务模块中应用

1 支付成功通知

支付模块利用消息队列通知订单系统，支付成功

在支付模块中配置 application.properties

```
spring.activemq.broker-url=tcp://mq.server.com:61616
```

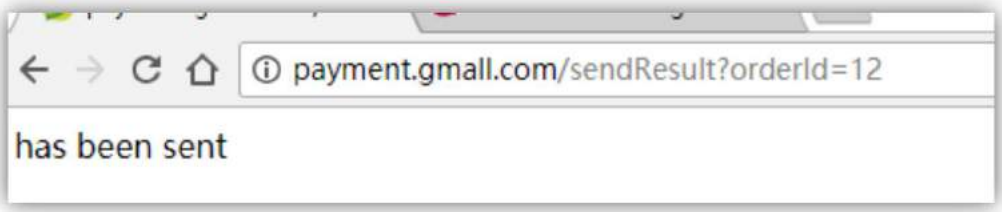
在 PaymentServiceImpl 中增加发送方法：

```
public void sendPaymentResult(String orderId,String result){
    ConnectionFactory connectionFactory = activeMQUtil.getConnectionFactory();
    Connection connection=null;
    try {
        connection = connectionFactory.createConnection();
        connection.start();
        Session session = connection.createSession(true, Session.SESSION_TRANSACTED);
        Queue paymentResultQueue = session.createQueue("PAYMENT_RESULT_QUEUE");
        MapMessage mapMessage=new ActiveMQMapMessage();
        mapMessage.setString("orderId",orderId);
        mapMessage.setString("result",result);
        MessageProducer producer = session.createProducer(paymentResultQueue);
        producer.send(mapMessage);
        session.commit();

        producer.close();
        session.close();
        connection.close();
    } catch (JMSException e) {
        e.printStackTrace();
    }
}
```

在 PaymentController 中增加一个方法用来测试

```
@RequestMapping("sendResult")
@ResponseBody
public String sendPaymentResult(@RequestParam("orderId") String orderId){
    paymentService.sendPaymentResult(orderId,"success" );
    return "has been sent";
}
```

在浏览器中访问：



查看队列内容：有一个在队列中没有被消费的消息。

2 订单模块消费消息

application.properties

```
spring.activemq.broker-url=tcp://mq.server.com:61616
activemq.listener.enable=true
```

订单消息消息后要更新订单状态，先准备好订单状态更新的方法

```
public void updateProcessStatus(String orderId , ProcessStatus processStatus, Map<String,String>... paramMaps) {
    OrderInfo orderInfo = new OrderInfo();
    orderInfo.setId(orderId);
    orderInfo.setOrderStatus(processStatus.getOrderStatus());
    orderInfo.setProcessStatus(processStatus);

    //动态增加需要补充更新的属性
    if (paramMaps != null && paramMaps.length > 0) {
        Map<String, String> paramMap = paramMaps[0];
        for (Map.Entry<String, String> entry : paramMap.entrySet()) {
            String properties = entry.getKey();
            String value = entry.getValue();
            try {
                BeanUtils.setProperty(orderInfo, properties, value);
            } catch (IllegalAccessException e) {
                e.printStackTrace();
            } catch (InvocationTargetException e) {
                e.printStackTrace();
            }
        }
    }
    orderInfoMapper.updateByPrimaryKeySelective(orderInfo);
}
```

消息队列的消费端

```
@JmsListener(destination = "PAYMENT_RESULT_QUEUE",containerFactory = "jmsQueueListener")
public void consumePaymentResult(MapMessage mapMessage) throws JMSException {
    String orderId = mapMessage.getString("orderId");
    String  result = mapMessage.getString("result");
    if(!"success".equals(result)){
        orderService.updateProcessStatus(  orderId , ProcessStatus.PAY_FAIL);
    }else{
        orderService.updateProcessStatus(  orderId , ProcessStatus.PAID);
```

```
    }
    orderService.sendOrderResult(orderId);
}
```

3 订单模块发送减库存通知

订单模块除了接收到请求改变单据状态，还要发送库存系统

查看看《库存管理系统接口手册》中**【减库存的消息队列消费端接口】**中的描述，组织相应的消息数据进行传递。

```
@Transactional
public void sendOrderResult(String orderId){
    OrderInfo orderInfo = getOrderInfo(orderId);
    Map<String, Object> messageMap = initWareOrderMessage(orderInfo);
    String wareOrderJson= JSON.toJSONString(messageMap);
    Session session = null;
    try {
        Connection conn = activeMQUtil.getConnection();

        session = conn.createSession(true, Session.SESSION_TRANSACTED);
        Queue queue = session.createQueue("ORDER_RESULT_QUEUE");
        MessageProducer producer = session.createProducer(queue);

        TextMessage message =new ActiveMQTextMessage();
        message.setText(wareOrderJson);
        producer.send(message);

        updateProcessStatus(orderInfo.getId(), ProcessStatus.NOTIFIED_WARE);

        session.commit();
        producer.close();
        conn.close();
    } catch (JMSException e) {
        e.printStackTrace();
    }
}
```

针对接口手册中需要的消息进行组织

```
public    Map<String,Object>    initWareOrderMessage( OrderInfo    orderInfo ) {

    //准备发送到仓库系统的订单
    String wareId = orderInfo.getWareId();
```

```
        HashMap<String, Object> hashMap = new HashMap<>();
        hashMap.put("orderId", orderInfo.getId());
        hashMap.put("consignee", orderInfo.getConsignee());
        hashMap.put("consigneeTel", orderInfo.getConsigneeTel());
        hashMap.put("orderComment", orderInfo.getOrderComment());
        hashMap.put("orderBody", orderInfo.getOrderSubject());

        hashMap.put("deliveryAddress", orderInfo.getDeliveryAddress());
        hashMap.put("paymentWay", "2");//1 货到付款 2 在线支付

        hashMap.put("wareId",wareId);

        List<HashMap<String, String>> details = new ArrayList<>();
        List<OrderDetail> orderDetailList = orderInfo.getOrderDetailList();
        for (OrderDetail orderDetail : orderDetailList) {
            HashMap<String, String> detailMap = new HashMap<>();
            detailMap.put("skuId", orderDetail.getSkuId());
            detailMap.put("skuNum", "" + orderDetail.getSkuNum());
            detailMap.put("skuName", orderDetail.getSkuName());
            details.add(detailMap);
        }

        hashMap.put("details", details);

        return hashMap;
    }
```

4 消费减库存结果

给仓库系统发送减库存消息后，还要接受减库存成功或者失败的消息。

同样根据《库存管理系统接口手册》中**【商品减库结果消息】**的说明完成。消费该消息的消息队列监听程序。

接受到消息后主要做的工作就是更新订单状态。

```
@JmsListener(destination = "SKU_DEDUCT_QUEUE",containerFactory = "jmsQueueListener")
public void consumeSkuDeduct(MapMessage mapMessage) throws JMSException {
    String orderId = mapMessage.getString("orderId");
    String    status = mapMessage.getString("status");
    if("DEDUCTED".equals(status)){
        orderService.updateProcessStatus(    orderId , ProcessStatus.WAITING_DELEVER);
        return ;
    }else{
        orderService.updateProcessStatus(    orderId , ProcessStatus.STOCK_EXCEPTION);
        return ;
    }
```

```
}
```

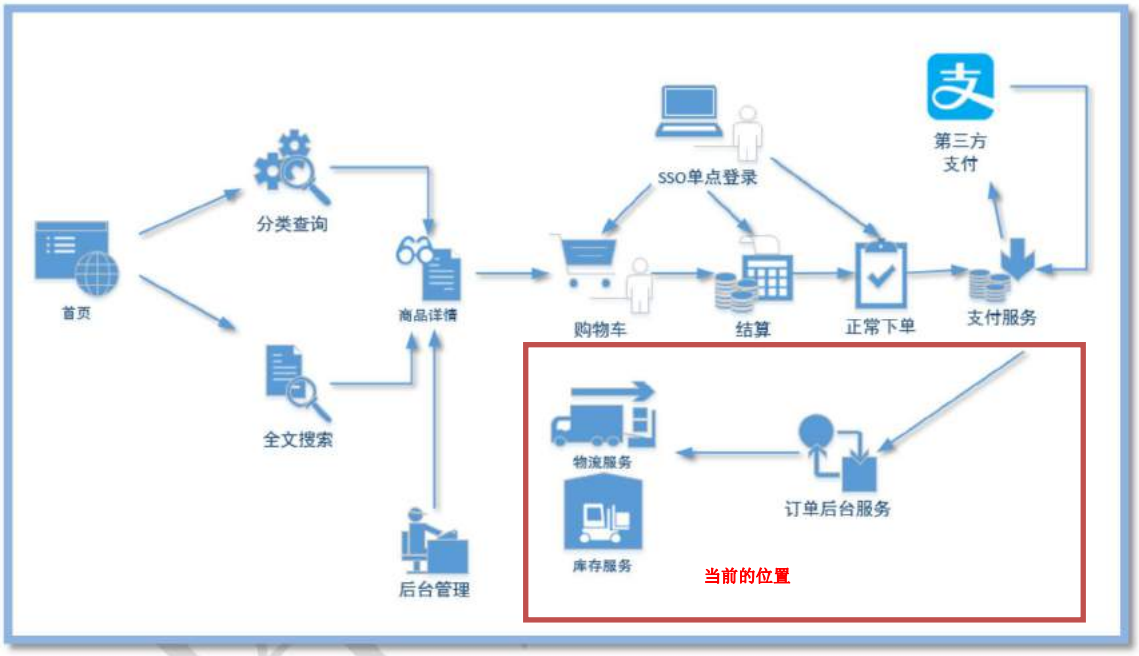
最后一次支付完成后，所有业务全部走通应该可以在订单列表中，查看到对应的订单是待发货状态。

5 验证结果

延迟队列与轮询

版本：V 1.0

www.atguigu.com



一、分布式事务的异步通信问题

使用分布式事务异步通信的结构，一个很大的问题就是**不确定性**。一个消息发送过去了，不管结果如何发送端都不会原地等待接收端。直到接收端再推送回来回执消息，发送端才直到结果。但是也有可能发送端消息发送后，石沉大海，杳无音信。这时候就需要一种机制能够**对这种不确定性进行补充**。

比如你给有很多笔友，平时写信一去一回，但是有时候会遇到迟迟没有回信的情况。那么针对这种偶尔出现的情况，你可以选择两种策略。一种方案是你发信的时候用定个闹钟，设定 1 天以后去问一下对方收没收到信。另一种方案就是每天夜里定个时间查看一下所有发

过信但是已经一天没收到回复的信。然后挨个打个电话问一下。

第一种策略就是实现起来就是延迟队列，第二种策略就是定时轮询扫描。

二者的区别是**延迟队列**更加精准，但是如果周期太长，任务留在延迟队列中时间的就会非常长，会把队列变得冗长。比如用户几天后待办提醒，生日提醒。

那么如果遇到这种长周期的事件，而且并不需要精确到分秒级的事件，可以利用**定时扫描**来实现，尤其是比较消耗性能的大范围扫描，可以安排到夜间执行。

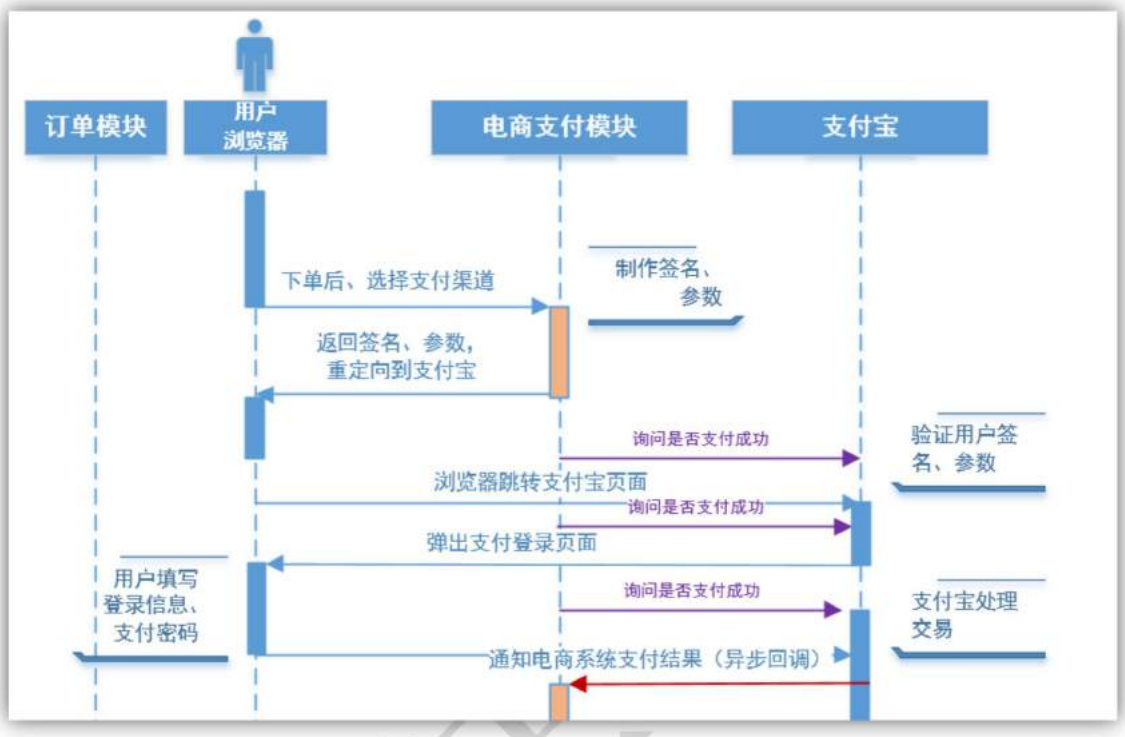
二、延迟队列

1 应用场景

当用户选择支付后，通常来说用户都会在支付宝正常支付，支付宝转账成功后，通过后台异步发送成功的请求到电商支付模块。

但是如果用户点击支付后，支付模块可能会长时间没有收到支付宝的支付成功通知。这种情况会‘有两种可能性，一种是用户在弹出支付宝付款界面时没有继续支付，另一种就是用户支付成功了，但是因为网络等各种问题，支付模块没有收到通知。

如果是上述第二种可能性，对于用户来说体验是非常糟糕的，甚至会怀疑平台的诚信。所以为了尽可能避免第二种情况，在用户点击支付后一段时间后，不管用户是否付款，都要去主动询问支付宝，该笔单据是否付款。



图中紫线部分，就是支付模块一旦帮助用户重定向到支付宝后，就要每隔一段时间询问支付宝用户是否支付成功，直到收到支付宝的回复，或者超过了询问次数。

2 实现思路

首先，需要知道如何主动查询支付宝中某笔交易的状态。

支付宝查询接口文档：https://docs.open.alipay.com/api_1/alipay.trade.query

其次，利用延迟队列反复调用。

3、实现支付宝订单状态查询

支付宝文档中的样例

1、首先通过基本参数初始化 AlipayClient,此处和支付模块部分相同，不再详述。

2、业务参数

业务参数就两个，选哪个都可以，其中 out_trade_no 是电商系统生成的，trade_no 是支付宝回调后产生的。因为有可能一直就没收到支付宝的回调，也就没有 trade_no，所以咱们这里使用 out_trade_no。

```
@Autowired
AlipayClient alipayClient;

public PaymentStatus checkAlipayPayment(PaymentInfo paymentInfo){
```

```
        System.out.println(" 开 始 主 动 检 查 支 付 状 态 ， paymentInfo.toString() = " +
paymentInfo.toString());
        //先检查当前数据库是否已经变为“已支付状态”
        if(paymentInfo.getId()==null){
            System.out.println("outTradeNo:"+paymentInfo.getOutTradeNo()    );
            paymentInfo = getPaymentInfo(paymentInfo);
        }
        if (paymentInfo.getPaymentStatus()== PaymentStatus.PAID){
            System.out.println("该单据已支付:"+paymentInfo.getOutTradeNo());
            return PaymentStatus.PAID;
        }

        //如果不是已支付，继续去查询 alipay 的接口
        System.out.println("%% % %  查询 alipay 的接口"    );
        AlipayTradeQueryRequest alipayTradeQueryRequest=new AlipayTradeQueryRequest();

alipayTradeQueryRequest.setBizContent("{\"out_trade_no\":\""+paymentInfo.getOutTradeNo()
+"\"}");
        AlipayTradeQueryResponse response=null;
        try {
            response = alipayClient.execute(alipayTradeQueryRequest);
        } catch (AlipayApiException e) {
            e.printStackTrace();
        }


        if(response.isSuccess()){
            String tradeStatus = response.getTradeStatus();

            if ("TRADE_SUCCESS".equals(tradeStatus)){
                System.out.println("支付完成  ======================  "      );
                //如果结果是支付成功 ,则更新支付状态
                PaymentInfo paymentInfo4Upt=new PaymentInfo();
                paymentInfo4Upt.setPaymentStatus(PaymentStatus.PAID);
                paymentInfo4Upt.setCallbackTime(new Date());
                paymentInfo4Upt.setCallbackContent(response.getBody());
                paymentInfo4Upt.setId(paymentInfo.getId());
                paymentInfoMapper.updateByPrimaryKeySelective(paymentInfo4Upt);

                // 然后发送通知给订单
                sendPaymentResult(paymentInfo,"success");
                return PaymentStatus.PAID;
            }else{
                System.out.println("支付尚未完成 ？？？？？？？？？？  "      );
                return PaymentStatus.UNPAID;
            }
        }else{
            System.out.println("支付尚未完成 ？？？？？？？？？？  "      );
            return      PaymentStatus.UNPAID;
        }
```

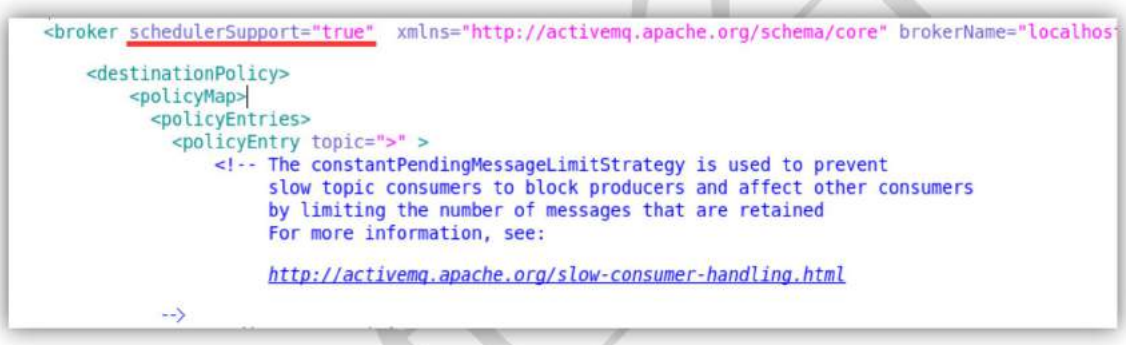
```
}
```

4、 利用延迟队列反复调用查询接口。

执行策略：

选择支付渠道后，点击支付后提交到延迟队列，每隔一分钟执行一次查询操作，查询三次。

首先在消息队列中打开延迟队列配置：在 activemq 的 conf 目录下 activemq.xml 中



开启 schedulerSupport="true”

发送延迟队列

```
public void sendDelayPaymentResult(String outTradeNo,int delaySec,int checkCount){
    //发送支付结果
    Connection connection = activeMQUtil.getConnection();
    try {
        connection.start();
        Session session = connection.createSession(true, Session.SESSION_TRANSACTED);
        Queue paymentResultQueue = session.createQueue("PAYMENT_RESULT_CHECK_QUEUE");
        MessageProducer producer = session.createProducer(paymentResultQueue);
        producer.setDeliveryMode(DeliveryMode.PERSISTENT);
        MapMessage mapMessage= new ActiveMQMapMessage();
        mapMessage.setString("outTradeNo",outTradeNo);
        mapMessage.setInt("delaySec",delaySec);
        mapMessage.setInt("checkCount",checkCount);

        mapMessage.setLongProperty(ScheduledMessage.AMQ_SCHEDULED_DELAY,delaySec*1000);
        producer.send(mapMessage);
```

```
            session.commit();
            producer.close();
            session.close();
            connection.close();

        } catch (JMSException e) {
            e.printStackTrace();
        }

}
```

接收延迟队列的消费端

```
@Component
public class PaymentConsumer {

    @Autowired
    PaymentService paymentService;


    @JmsListener(destination = "PAYMENT_RESULT_CHECK_QUEUE",containerFactory = "jmsQueueListener")
    public void consumeCheckResult(MapMessage mapMessage) throws JMSException {
        int delaySec = mapMessage.getInt("delaySec");
        String outTradeNo = mapMessage.getString("outTradeNo");
        int checkCount = mapMessage.getInt("checkCount");

        PaymentInfo paymentInfo=new PaymentInfo();
        paymentInfo.setOutTradeNo(outTradeNo);
        PaymentStatus paymentStatus = paymentService.checkAlipayPayment(paymentInfo);
        if(paymentStatus==PaymentStatus.UNPAID&&checkCount>0){
            System.out.println("checkCount = " + checkCount);
            paymentService.sendDelayPaymentResult(outTradeNo,delaySec,checkCount-1);
        }

    }


}
```

三 轮询扫描

1 应用场景

长期没有付款的订单，要定期关闭掉。

如果时限比较小，比如 30 分钟未付款的订单就关闭（一般是锁了库存的订单），也可以用延时队列解决。

如果时限比较长比如 1-2 天，可以选择用轮询扫描。

2 、实现方式 spring task

轮询扫描有很多工具，比较经典的就是 quartz。

但是 springboot 整合了自家的 spring task ，功能上基本和 quartz 差不多，但是配置更简单，全程只用注解就可以，不用额外的 xml。

测试 Demo

```
@Component
@EnableScheduling
public class OrderTask {

    @Autowired
    OrderService orderService;

    @Scheduled(cron = "0/5 * * * * ?")
    public void work() throws InterruptedException {
        System.out.println("thread = ==============" + Thread.currentThread());
    }
}
```

默认扫描是单线程的即一次任务执行完，第二次的任务才能执行。如果第一次的任务被一些其他情况阻塞住了，那么第二次的扫描就没法开始了。

```
@Bean
public TaskScheduler taskScheduler() {
    ThreadPoolTaskScheduler taskScheduler = new ThreadPoolTaskScheduler();
    taskScheduler.setPoolSize(5);
    return taskScheduler;
}
```

关于@Scheduled

秒	0-59
分	0-59
小时	0-23
日期	1-31
月份	1-12
星期	1-7
年（可选）	1970-2099

3 业务扫描

```
@Scheduled(cron = "0/30 * * * * ?")
public void checkUnpaidOrder() {
	System.out.println("开始检查未付款单据 = ");
	Long beginTime=System.currentTimeMillis();
	List<OrderInfo> unpaidOrderList = orderService.getUnpaidOrderList();
	for (OrderInfo orderInfo : unpaidOrderList) {
		orderService.checkExpireOrder(orderInfo);
	}
	Long costtime=System.currentTimeMillis()-beginTime;
	System.out.println("开始检查完毕未付款单据 = 共消耗"+costtime);
}
```

```

```

```
public	void	checkExpireOrder(OrderInfo orderInfo	)	{

		updateProcessStatus(orderInfo.getId(), ProcessStatus.CLOSED);
		paymentService.closePayment(orderInfo.getId());


	return ;
```

```
}
```

4 利用多线程实现异步并发操作

```
@Configuration
@EnableAsync
public class AsyncTaskConfig implements AsyncConfigurer {
    @Override
    @Bean
    public Executor getAsyncExecutor() {
        ThreadPoolTaskExecutor threadPoolTaskExecutor=new
ThreadPoolTaskExecutor();
        threadPoolTaskExecutor.setCorePoolSize(10);    //线程数
        threadPoolTaskExecutor.setQueueCapacity(100);    //等待队列容量 ， 线程
数不够任务会等待
        threadPoolTaskExecutor.setMaxPoolSize(50);     // 最大线程数，等待数不
够会增加线程数，直到达此上线  超过这个范围会抛异常
        threadPoolTaskExecutor.initialize();
        return threadPoolTaskExecutor;

    }

    @Override
    @Bean
    public AsyncUncaughtExceptionHandler getAsyncUncaughtExceptionHandler() {
        return null;
    }
}
```

在代码中的方法上可以标记@Async

```
@Async
public   void    checkExpireOrder(OrderInfo orderInfo    )    {
      Date expireDate= DateUtil.addDays(orderInfo.getCreateTime(),1) ;
          if (new Date().after(expireDate)){
```

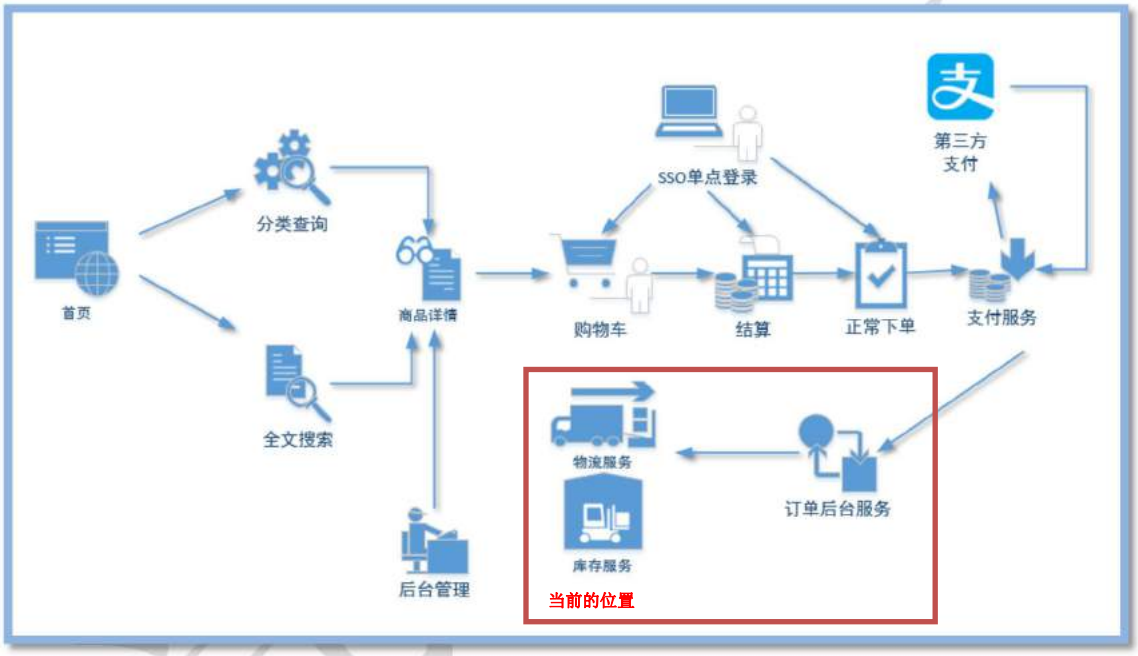
```
                updateProcessStatus(orderInfo.getId(), ProcessStatus.CLOSED);
                paymentService.closePayment(orderInfo.getId());
            }
        return ;
}
```

```
public void closePayment(String orderId){
    Example example=new Example(PaymentInfo.class);
    example.createCriteria().andEqualTo("orderId",orderId);
    PaymentInfo paymentInfo=new PaymentInfo();
    paymentInfo.setPaymentStatus(PaymentStatus.CLOSED);
    paymentInfoMapper.updateByExampleSelective(paymentInfo,example);

}
```

拆单

版本：V 1.0

www.atguigu.com



一、拆单简介

1 为什么要拆单

因为同一个订单中不同的商品可能会因为商家不同、仓库不同、物流渠道不同。分成多个子订单分别进行跟踪。

2 何时拆单

可以选择下单时拆分，也可以选择支付时拆单。谷粒商城的项目以京东为参考选择支付时拆单。

3 拆单对用户支付是否有影响？

用户支付时不影响拆单，如果是因为多商家进行的拆单，用户付款的金额会进行分账到不同的商家。

二 开发

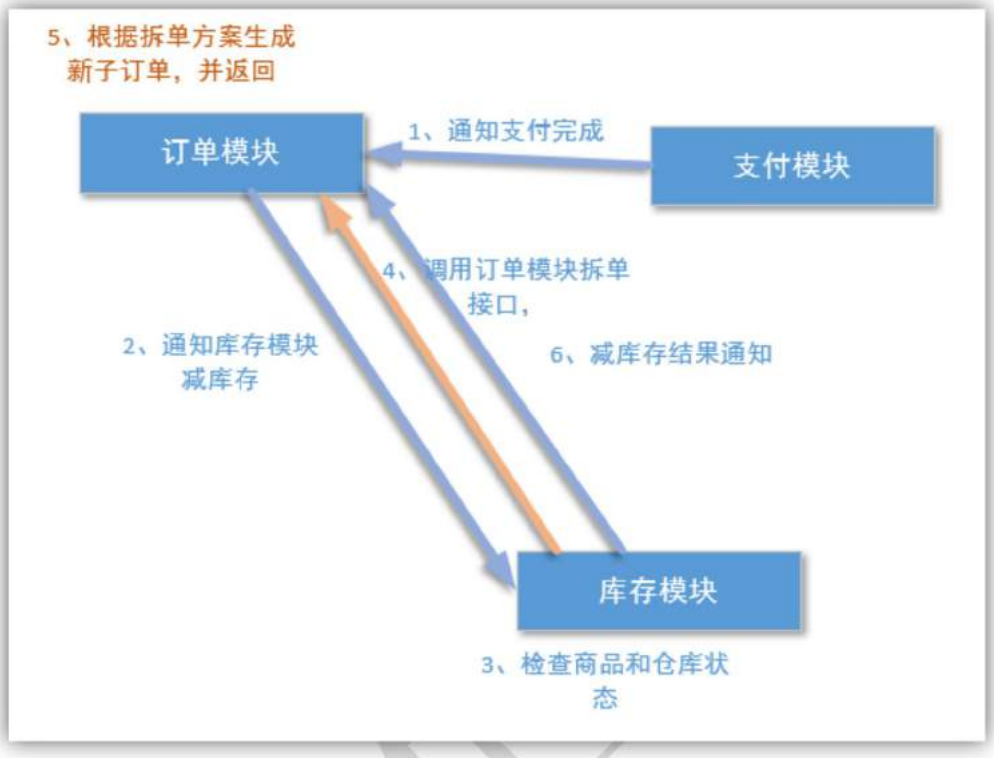
1 分析

本章案例讲解有库存系统检查发现单笔订单中的商品存在于不同仓库之中，而触发拆单动作。由库存模块调用订单模块的拆单接口。

通信方式：

由于拆单动作是不同于通知性任务，必须获得拆单后新的订单编号和结构才能继续往下进行，因此不选择利用消息队列采用异步通信。而采用 **restful** 这种同步通信的方式。

2 示意图



3 库存系统调用拆单的接口

拆单接口说明

由库存系统发起申请

调用接口	http://order.gmall.com/orderSplit	
请求参	orderId	订单系统的订单 ID

数		
	wareSkuMap	仓库编号与商品的对照关系 例如 [{"wareId":"1","skuIds":["2","10"]},{"wareId":"2","skuIds":["3"]}] 表示：sku 为 2 号，10 号的商品在 1 号仓库 sku 为 3 号，10 号的商品在 2 号仓库
请求方式	post	
返回值格式	json 集合	子订单的集合 json
	orderId	订单系统的订单 ID
	consignee	收货人
	consigneeTel	收货电话
	orderComment	订单备注
	orderBody	订单概要
	deliveryAddress	发货地址
	paymentWay	支付方式： ‘1’ 为货到付款，‘2’为在线支付。
	details: skuId skuNum skuName	购买商品明细 例如： details:[{skuId:101,skuNum:1,skuName: ’小米手 64G’}, {skuId:201,skuNum:1,skuName:’索尼耳机’}]
	wareId	传入时的仓库编号
返回值例：	[{ "orderBody": "小米 红米 5 移动联通电信 4G 手机 双卡双待等 1 件商品", "consignee": "晨哥", "orderComment": "123123",	

```
    "wareId": "1",
    "orderId": "16",
    "deliveryAddress": "宏福科技综合楼",
    "details": [
      {
        "skuName": "小米 红米 5 动联通电信 4G 手机 双卡双待",
        "skuId": "2",
        "skuNum": 7
      }
    ],
    "paymentWay": "2"
  },
  {
    "orderBody": "小米 红米 5 Plus 全面屏手 版 4GB+64GB 黑色 等 1 件商品",
    "consignee": "晨哥",
    "orderComment": "123123",
    "wareId": "2",
    "orderId": "17",
    "deliveryAddress": "宏福科技综合楼",
    "details": [
      {
        "skuName": "小米 红米 5 Plus 全 +64GB 黑色 ",
        "skuId": "3",
        "skuNum": 3
      }
    ],
    "paymentWay": "2"
  }
]
```


三 代码实现

OrderController

```
@RequestMapping(value = "orderSplit",method = RequestMethod.POST)
@ResponseBody
public String orderSplit(HttpServletRequest request){
    String orderId = request.getParameter("orderId");
    String wareSkuMapJson = request.getParameter("wareSkuMap");
     List<OrderInfo> subOrderInfoList = orderService.splitOrder(orderId,wareSkuMapJson);
```

```
        List wareSkuMapList=new ArrayList();
        for (OrderInfo orderInfo : subOrderInfoList) {
            Map map = orderService.initWareOrder(orderInfo);
            wareSkuMapList.add(map);
        }
        String newWareSkuMapJson = JSON.toJSONString(wareSkuMapList);
        return newWareSkuMapJson;
    }
```

OrderServiceImpl

```
public    List<OrderInfo>    splitOrder(String orderId,String wareSkuMapJson){
        List<OrderInfo> subOrderList=new ArrayList<>();
        //转 List    循环 list 获得  仓库与 sku 的对应关系
        List<Map> wareSkuMapList = JSON.parseArray(wareSkuMapJson, Map.class);
        OrderInfo orderInfo = getOrderInfo(orderId);
        for (Map map : wareSkuMapList) {
            String wareId = (String)map.get("wareId");
            List<String> skuidList = (List)map.get("skuIds");

            //每个仓库拆分成一个子订单
            //构造主表
            OrderInfo subOrderInfo=new OrderInfo();
            try {
                BeanUtils.copyProperties(subOrderInfo,orderInfo);
            } catch (IllegalAccessException e) {
                e.printStackTrace();
            } catch (InvocationTargetException e) {
                e.printStackTrace();
            }
            subOrderInfo.setParentOrderId(orderInfo.getId());
            subOrderInfo.setId(null);
            subOrderInfo.setWareId(wareId);
            List<OrderDetail> subOrderDetailList=new ArrayList<>();
            for (String skuId : skuidList) {
                for (OrderDetail orderDetail :orderInfo.getOrderDetailList()) {
                    if(skuId.equals(orderDetail.getSkuId())){
                        orderDetail.setOrderId(null);
                        orderDetail.setId(null);
                        subOrderDetailList.add(orderDetail);
                    }
                }
            }
            subOrderInfo.setOrderDetailList(subOrderDetailList);
```

```
            subOrderInfo.sumTotalAmount();

            saveOrder(subOrderInfo);

            updateOrderStatus(orderInfo.getId(),ProcessStatus.SPLIT);

            subOrderList.add(subOrderInfo);
        }

    return subOrderList;
}
```

在不影响原程序下改造 initWareOrder 方法

```
public String initWareOrder(String orderId) {
     OrderInfo orderInfo = getOrderInfo(orderId);
     Map map = initWareOrder(orderInfo);
     return JSON.toJSONString(map);
}


public Map  initWareOrder(OrderInfo orderInfo){

    Map map=new HashMap();
    map.put("orderId",orderInfo.getId());
    map.put("consignee", orderInfo.getConsignee());
    map.put("consigneeTel",orderInfo.getConsigneeTel());
    map.put("orderComment",orderInfo.getOrderComment());
    map.put("orderBody",orderInfo.getTradeBody());
    map.put("deliveryAddress",orderInfo.getDeliveryAddress());
    map.put("paymentWay","2");
    map.put("wareId",orderInfo.getWareId());

    List detailList=new ArrayList();
    List<OrderDetail> orderDetailList = orderInfo.getOrderDetailList();
    for (OrderDetail orderDetail : orderDetailList) {
        Map  detailMap =new HashMap();
        detailMap.put("skuId",orderDetail.getSkuId());
        detailMap.put("skuName",orderDetail.getSkuName());
```

```
        detailMap.put("skuNum",orderDetail.getSkuNum());

        detailList.add(detailMap);
    }

    map.put("details",detailList);

    return  map;

}
```

安装 elasticsearch

版本：v1.0

www.atguigu.com

1 安装包下载

Elasticsearch 官网： https://www.elastic.co/products/elasticsearch

https://www.elastic.co/downloads/past-releases/elasticsearch-5-6-4

本课程选择的版本是 elasticsearch-5.6.4

下载好后放到/opt/目录下

2 安装 elasticsearch

拷贝 elasticsearch-5.6.4.rpm 到/opt 目录下

```
[root@jack opt]# rpm -ivh elasticsearch-5.6.4.rpm
warning: elasticsearch-5.6.4.rpm: Header V4 RSA/SHA512 Signature, key ID d88e42b4: NOKEY
Preparing...                ########################################### [100%]
Creating elasticsearch group... OK
Creating elasticsearch user... OK
   1:elasticsearch          ########################################### [100%]
### NOT starting on installation, please execute the following statements to configure elasti
 sudo chkconfig --add elasticsearch
### You can start elasticsearch service by executing
 sudo service elasticsearch start
```

2.1 注册并启动服务

```
[root@jack opt]# cd /etc/init.d
[root@jack init.d]# ll
总用量 408
-rwxr-xr-x. 1 root root  1288 5月  12 2016 abrt-ccpp
-rwxr-xr-x. 1 root root  1628 5月  12 2016 abrtd
-rwxr-xr-x. 1 root root  1642 5月  12 2016 abrt-oops
-rwxr-xr-x. 1 root root  1818 2月  17 2016 acpid
-rwxr-xr-x. 1 root root  2062 2月  20 2015 atd
-rwxr-xr-x. 1 root root  3580 5月  11 2016 auditd
-r-xr-xr-x. 1 root root  1343 5月  11 2016 blk-availability
-rwxr-xr-x. 1 root root   710 11月 11 2010 bluetooth
-rwxr-xr-x. 1 root root 11864 7月  24 2015 cpuspeed
-rwxr-xr-x. 1 root root  2826 11月 10 2015 crond
-rwxr-xr-x. 1 root root  3034 5月  11 2016 cups
-rwxr-xr-x. 1 root root  1734 5月  11 2016 dnsmasq
-rwxr-x---. 1 root root  5113 11月  1 02:57 elasticsearch
-rwxr-xr-x. 1 root root  1186 11月  5 00:19 fdfs_storaged
-rwxr-xr-x. 1 root root  1186 11月  5 00:19 fdfs_trackerd
-rwxr-xr-x. 1 root root  3245 7月   9 2013 firstboot
-rw-r--r--. 1 root root 25419 4月  12 2016 functions
-rwxr-xr-x. 1 root root  1801 10月 15 2014 haldaemon
-rwxr-xr-x. 1 root root  5985 4月  12 2016 halt
-rwxr-xr-x. 1 root root  2001 5月  12 2016 htcacheclean
```

CentOS6.8 通过 chkconfig --list 可以查看

```
[root@jack init.d]# chkconfig --add elasticsearch
[root@jack init.d]# chkconfig --list
NetworkManager  0:关闭  1:关闭  2:启用  3:启用  4:启用  5:启用  6:关闭
abrt-ccpp       0:关闭  1:关闭  2:关闭  3:启用  4:关闭  5:启用  6:关闭
abrtd           0:关闭  1:关闭  2:关闭  3:启用  4:关闭  5:启用  6:关闭
acpid           0:关闭  1:关闭  2:启用  3:启用  4:启用  5:启用  6:关闭
atd             0:关闭  1:关闭  2:关闭  3:启用  4:启用  5:启用  6:关闭
auditd          0:关闭  1:关闭  2:启用  3:启用  4:启用  5:启用  6:关闭
blk-availability        0:关闭  1:启用  2:启用  3:启用  4:启用  5:启用  6:关闭
bluetooth       0:关闭  1:关闭  2:关闭  3:启用  4:启用  5:启用  6:关闭
cpuspeed        0:关闭  1:启用  2:启用  3:启用  4:启用  5:启用  6:关闭
crond           0:关闭  1:关闭  2:启用  3:启用  4:启用  5:启用  6:关闭
cups            0:关闭  1:关闭  2:启用  3:启用  4:启用  5:启用  6:关闭
dnsmasq         0:关闭  1:关闭  2:关闭  3:关闭  4:关闭  5:关闭  6:关闭
elasticsearch   0:关闭  1:关闭  2:启用  3:启用  4:启用  5:启用  6:关闭
firstboot       0:关闭  1:关闭  2:关闭  3:关闭  4:关闭  5:关闭  6:关闭
haldaemon       0:关闭  1:关闭  2:关闭  3:启用  4:启用  5:启用  6:关闭
htcacheclean    0:关闭  1:关闭  2:关闭  3:关闭  4:关闭  5:关闭  6:关闭
httpd           0:关闭  1:关闭  2:关闭  3:关闭  4:关闭  5:关闭  6:关闭
ip6tables       0:关闭  1:关闭  2:启用  3:启用  4:启用  5:启用  6:关闭
iptables        0:关闭  1:关闭  2:启用  3:启用  4:启用  5:关闭  6:关闭
```

CentOS7.x 可以通过 systemctl list-unit-files|grep elasticsearch

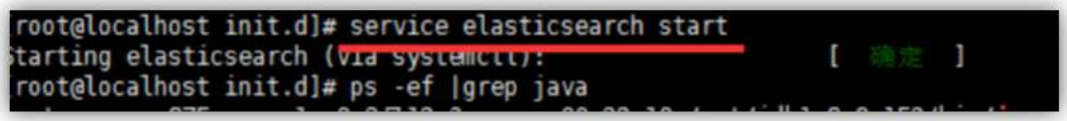
```
[root@localhost init.d]# systemctl list-unit-files |grep elasticsearch
elasticsearch.service                         disabled
[root@localhost init.d]#
```

启动之前为 elasticsearch 配置 jdk

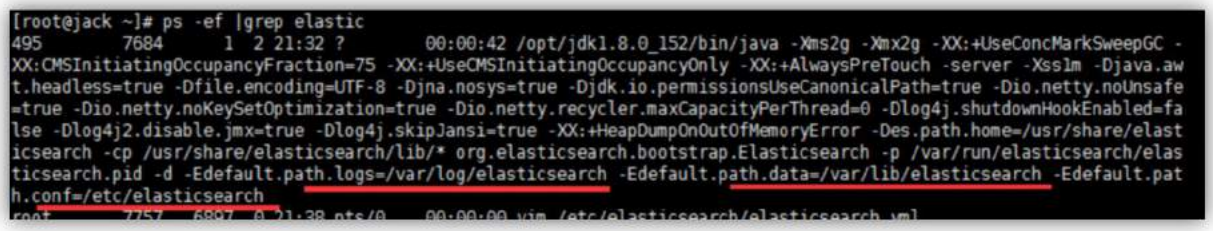
vim /etc/sysconfig/elasticsearch 中修改 JAVA_HOME 路径的路径

```
# Elasticsearch home directory
#ES_HOME=/usr/share/elasticsearch

# Elasticsearch Java path
JAVA_HOME=/opt/jdk1.8.0_152

# Elasticsearch configuration directory
#CONF_DIR=/etc/elasticsearch

# Elasticsearch data directory
#DATA_DIR=/var/lib/elasticsearch
```

启动 elasticsearch



查看进程



核心文件

/etc/elasticsearch/elasticsearch.yml

数据文件路径

/var/lib/elasticsearch/

日志文件路径e

/var/log/elasticsearch/elasticsearch.log

2.2 修改配置文件

```
vim /etc/elasticsearch/elasticsearch.yml
```

修改 yml 配置的注意事项:
每行必须顶格，不能有空格
“：”后面必须有一个空格

集群名称，同一集群名称必须相同

```
#
# Use a descriptive name for your cluster:
#
cluster.name: my-es
#
# ------------------------------------ Node ------------------------------------
```

单个节点名称

```
# ------------------------------------ Node ------------------------------------
#
# Use a descriptive name for the node:
#
node.name: node-1
#
```

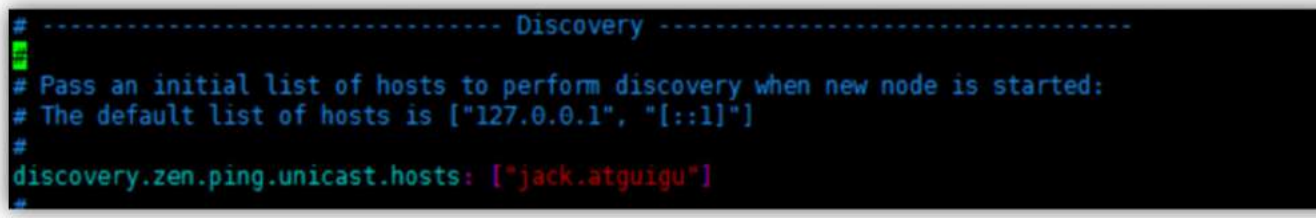
网络部分　改为当前的 ip 地址　，端口号保持默认 9200 就行

```
#
# Set the bind address to a specific IP (IPv4 or IPv6):
#
network.host: 192.168.67.163
#http.host: 192.168.67.163
#transport.host: 192.168.67.147
# Set a custom port for HTTP:
#
```

把 bootstrap 自检程序关掉

```
# ---------------------------------- Memory ----------------------------------
#
# Lock the memory on startup:
#
bootstrap.memory_lock: false
bootstrap.system_call_filter: false
#
# Make sure that the heap size is set to about half the memory available
# on the system and that the owner of the process is allowed to use this
# limit.
```

自发现配置：新节点向集群报到的主机名



2.3 修改 linux 配置

为什么要修改 linux 配置？

默认 elasticsearch 是单机访问模式，就是只能自己访问自己。

但是我们之后一定会设置成允许应用服务器通过网络方式访问。这时，elasticsearch 就会因为嫌弃单机版的低端默认配置而报错，甚至无法启动。

所以我们在这里就要把服务器的一些限制打开，能支持更多并发。

问题1：max file descriptors [4096] for elasticsearch process likely too low, increase to at least [65536] elasticsearch

原因：系统允许 Elasticsearch 打开的最大文件数需要修改成 65536
解决：vi /etc/security/limits.conf
添加内容：
* soft nofile 65536
* hard nofile 131072
* soft nproc 2048
* hard nproc 65536

注意：“*” 不要省略掉

问题2：max number of threads [1024] for user [judy2] likely too low, increase to at least [2048] （CentOS7.x 不用改）

原因：允许最大进程数修该成 2048
解决：vi /etc/security/limits.d/90-nproc.conf
修改如下内容：
* soft nproc 1024
#修改为
  * soft nproc 2048

问题3：max virtual memory areas vm.max_map_count [65530] likely too low, increase to at least [262144] （CentOS7.x　不用改）

原因：一个进程可以拥有的虚拟内存区域的数量。
解决：可零时提高 vm.max_map_count 的大小
命令：sysctl -w vm.max_map_count=262144

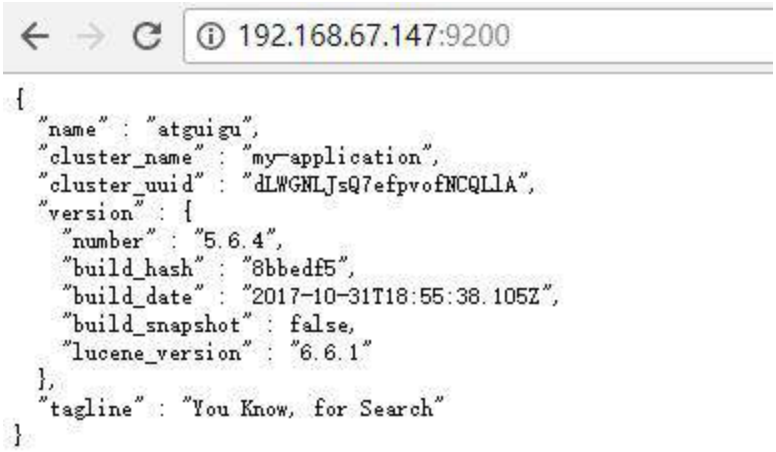
2.4 重启 linux

2.5 测试

```
[root@centos147 ~]# curl http://192.168.67.147:9200
{
  "name" : "atguigu",
  "cluster_name" : "my-application",
  "cluster_uuid" : "dLWGNLJsQ7efpvofNCQLlA",
  "version" : {
    "number" : "5.6.4",
    "build_hash" : "8bbedf5",
    "build_date" : "2017-10-31T18:55:38.105Z",
    "build_snapshot" : false,
    "lucene_version" : "6.6.1"
  },
```

```
  "tagline" : "You Know, for Search"
}
```

或者直接浏览器访问 http://192.168.67.147:9200



2.6 如果启动未成功

如果启动未成功，请去查看相关日志

vim /var/log/elasticsearch/my-es.log

3 安装 kibana

拷贝 kibana-5.6.4-linux-x86_64.tar 到/opt 下
解压缩
进入 kibana 主目录的 config 目录下

```
drwxr-xr-x.   2 atguigu atguigu     76 3月   16 23:08 bin
drwxrwxr-x.   2 atguigu atguigu     24 3月   16 23:07 config
drwxrwxr-x.   2 atguigu atguigu     18 3月   16 22:57 data
-rw-rw-r--.   1 atguigu atguigu    562 11月   1 03:04 LICENSE.txt
drwxrwxr-x.   6 atguigu atguigu    108 11月   1 03:04 node
drwxrwxr-x. 618 atguigu atguigu  20480 11月   1 03:04 node_modules
-rw-rw-r--.   1 atguigu atguigu 798420 11月   1 03:04 NOTICE.txt
drwxrwxr-x.   3 atguigu atguigu     45 11月   1 03:04 optimize
-rw-rw-r--.   1 atguigu atguigu    721 11月   1 03:04 package.json
drwxrwxr-x.   2 atguigu atguigu      6 11月   1 03:04 plugins
-rw-rw-r--.   1 atguigu atguigu   4899 11月   1 03:04 README.txt
drwxr-xr-x.  13 atguigu atguigu    165 11月   1 03:04 src
drwxrwxr-x.   5 atguigu atguigu     52 11月   1 03:04 ui_framework
drwxr-xr-x.   2 atguigu atguigu   4096 11月   1 03:04 webpackShims
[root@jack kibana-5.6.4-linux-x86_64]# cd config
[root@jack config]#
```

vim kibana.yml

```
# Kibana is served by a back end server. This setting specifies th
#server.port: 5601

# Specifies the address to which the Kibana server will bind. IP a
# The default is 'localhost', which usually means remote machines
# To allow connections from remote users, set this parameter to a
server.host: "0.0.0.0"

# Enables you to specify a path to mount Kibana at if you are runn
# the URLs generated by Kibana, your proxy is expected to remove t
# to Kibana. This setting cannot end in a slash.
#server.basePath: ""

# The maximum payload size in bytes for incoming server requests.
#server.maxPayloadBytes: 1048576

# The Kibana server's name.  This is used for display purposes.
#server.name: "your-hostname"

# The URL of the Elasticsearch instance to use for all your querie
elasticsearch.url: "http://192.168.67.163:9200"

# When this setting's value is true Kibana uses the hostname speci
```

启动

在 kibana 主目录 bin 目录下执行

nohup ./kibana &

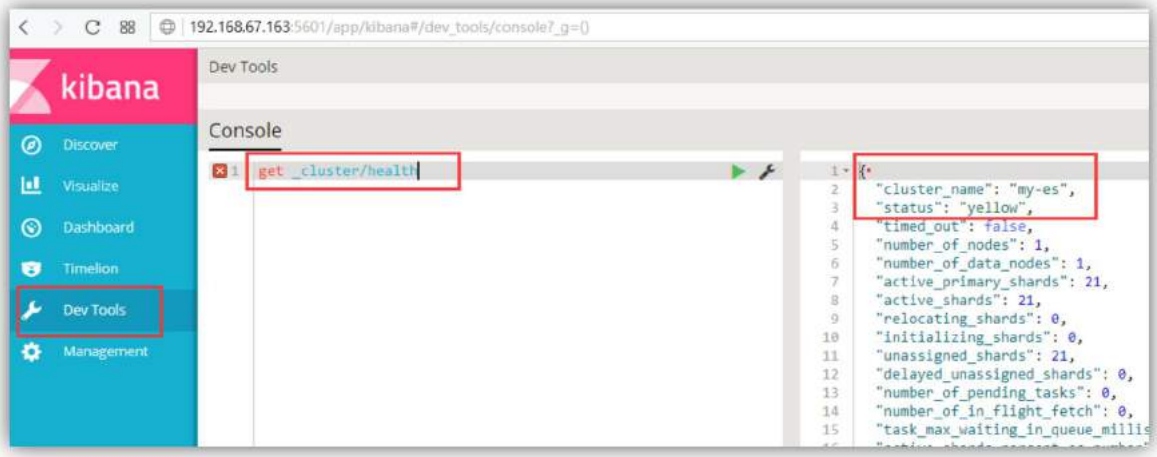
然后 ctrl+c 退出

执行 ps -ef

```
root       1757   1285  4 23:38 pts/0    00:00:10 ./../node/bin/node --no-warnings ./../src/cli
```

如上图,1757 号进程就是 kibana 的进程

用浏览器打开
http://192.168.xx.xx:5601/



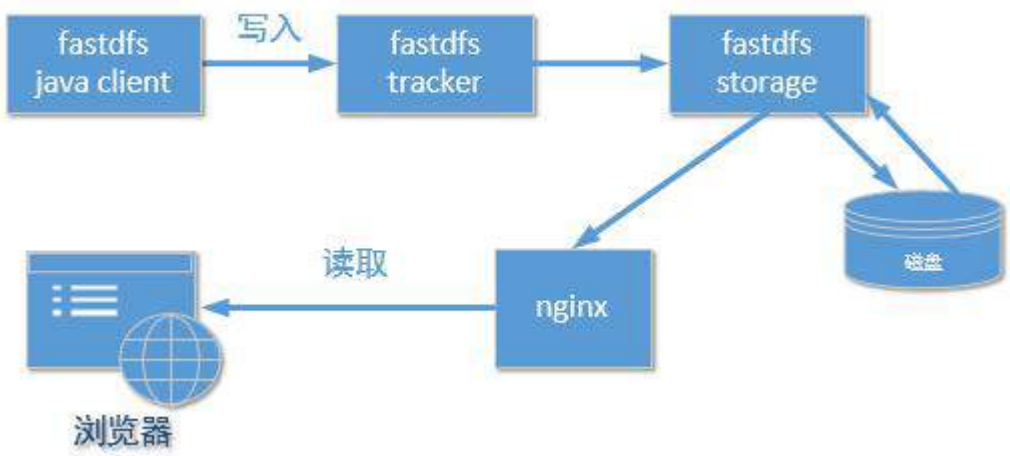
点击左边菜单 DevTools
在 Console 中
执行 get _cluster/health
右边的结果中，status 为 yellow 或者 green。
表示 es 启动正常，并且与 kibana 连接正常。

FastDfs 安装文档

版本：v1.0

www.atguigu.com

fastdfs 结构说明



一 、FastDFS--tracker 安装

1 FastDFS 安装环境

FastDFS 是 C 语言开发，建议在 linux 上运行，本教程使用 Centos7.4 作为安装环境。

安装 gcc 依赖环境 `yum install gcc-c++ -y`

2 安装 libevent

2.1 yum -y install libevent

3 安装 libfastcommon

1、上传压缩包文件 libfastcommonV1.0.7.tar.gz 到 /usr/local 目录下，并解压。

2、tar -zxvf libfastcommonV1.0.7.tar.gz

3、进入到解压后的文件夹中

```
[root@localhost local]# cd libfastcommon-1.0.7/
[root@localhost libfastcommon-1.0.7]# pwd
/usr/local/libfastcommon-1.0.7
```

4、进行编译 ./make.sh

5、如果出现编译 perl 不识别 运行下面这段命令

yum -y install zlib zlib-devel pcre pcre-devel gcc gcc-c++ openssl openssl-devel libevent libevent-devel perl unzip net-tools wget

```
-rw-rw-r--. 1 root root 2170 9月  16 2014 HISTORY
-rw-rw-r--. 1 root root  582 9月  16 2014 INSTALL
-rw-rw-r--. 1 root root 1341 9月  16 2014 libfastcommon.spec
-rwxrwxr-x. 1 root root 2151 9月  16 2014 make.sh
-rw-rw-r--. 1 root root  617 9月  16 2014 README
drwxrwxr-x. 2 root root 4096 9月  16 2014 src
[root@localhost libfastcommon-1.0.7]# ./make.sh
```

安装 ./make.sh install

```
cc -Wall -D_FILE_OFFSET_BITS=64 -g -DDEBUG_FLAG -DOS_LINUX -DIOEVENT_USE_EPOLL -c
cc -Wall -D_FILE_OFFSET_BITS=64 -g -DDEBUG_FLAG -DOS_LINUX -DIOEVENT_USE_EPOLL -c
cc -Wall -D_FILE_OFFSET_BITS=64 -g -DDEBUG_FLAG -DOS_LINUX -DIOEVENT_USE_EPOLL -c

cc -Wall -D_FILE_OFFSET_BITS=64 -g -DDEBUG_FLAG -DOS_LINUX -DIOEVENT_USE_EPOLL -c
ueue.c
cc -Wall -D_FILE_OFFSET_BITS=64 -g -DDEBUG_FLAG -DOS_LINUX -DIOEVENT_USE_EPOLL -c
cc -Wall -D_FILE_OFFSET_BITS=64 -g -DDEBUG_FLAG -DOS_LINUX -DIOEVENT_USE_EPOLL -c

cc -Wall -D_FILE_OFFSET_BITS=64 -g -DDEBUG_FLAG -DOS_LINUX -DIOEVENT_USE_EPOLL -c
cc -Wall -D_FILE_OFFSET_BITS=64 -g -DDEBUG_FLAG -DOS_LINUX -DIOEVENT_USE_EPOLL -c
pool.c
ar rcs libfastcommon.a hash.o
[root@localhost libfastcommon-1.0.7]# ./make.sh install
```

注意：libfastcommon 安装好后会自动将库文件拷贝至/usr/lib64 下，由于 FastDFS 程序引用 usr/lib 目录所以需要将/usr/lib64 下的库文件拷贝至 /usr/lib 下。

cp /usr/lib64/libfastcommon.so /usr/lib/

4 tracker 编译安装

1、 上传资料 FastDFS_v5.05.tar.gz 到 /usr/local 目录下

2、解压编译安装

tar -zxvf FastDFS_v5.05.tar.gz

cd FastDFS

./make.sh

./make.sh install

3、安装成功之后，将安装目录下的 conf 下的文件拷贝到/etc/fdfs/下。

cd conf

cp * /etc/fdfs/

4、修改配置文件

vim /etc/fdfs/tracker.conf

```
# false for enabled
# true for disabled
disabled=false

# bind an address of this host
# empty for bind all addresses of this host
bind_addr=

# the tracker server port
port=22122

# connect timeout in seconds
# default value is 30s
connect_timeout=30

# network timeout in seconds
# default value is 30s
network_timeout=60

# the base path to store data and log files
base_path=/opt/fastdfs
```

5、 创建 fastdfs 文件夹

6、 mkdir /opt/fastdfs

5 设置启动项

mkdir /usr/local/fdfs

拷贝安装目录下 stop.sh 和 restart.sh 到/usr/local/fdfs/

cp restart.sh /usr/local/fdfs/

cp stop.sh /usr/local/fdfs/

```
-rw-r--r--. 1 8980 users  2802 12月   2 2014 fastdfs.spec
-rw-r--r--. 1 8980 users 31386 12月   2 2014 HISTORY
drwxr-xr-x. 2 8980 users    48 3月    9 21:43 init.d
-rw-r--r--. 1 8980 users  7755 12月   2 2014 INSTALL
-rwxr-xr-x. 1 8980 users  5813 12月   2 2014 make.sh
drwxr-xr-x. 2 8980 users  4096 12月   2 2014 php_client
-rw-r--r--. 1 8980 users  2380 12月   2 2014 README.md
-rwxr-xr-x. 1 8980 users  1768 12月   2 2014 restart.sh
-rwxr-xr-x. 1 8980 users  1680 12月   2 2014 stop.sh
drwxr-xr-x. 4 8980 users  4096 3月    9 21:04 storage
drwxr-xr-x. 2 8980 users  4096 12月   2 2014 test
drwxr-xr-x. 2 8980 users  4096 3月    9 21:04 tracker
[root@localhost FastDFS_soft]# cp restart.sh /usr/local/fdfs/
```

修改启动脚本

vim /etc/init.d/fdfs_trackerd

```
. /etc/init.d/functions

PRG=/usr/bin/fdfs_trackerd
CONF=/etc/fdfs/tracker.conf

if [ ! -f $PRG ]; then
  echo "file $PRG does not exist!"
  exit 2
fi

if [ ! -f /usr/local/fdfs/stop.sh ]; then
  echo "file /usr/local/bin/stop.sh does not exist!"
  exit 2
fi

if [ ! -f /usr/local/fdfs/restart.sh ]; then
  echo "file /usr/local/bin/restart.sh does not exist!"
  exit 2
fi

if [ ! -f $CONF ]; then
  echo "file $CONF does not exist!"
  exit 2
fi

CMD="$PRG $CONF"
RETVAL=0
```

```
stop() {
        /usr/local/fdfs/stop.sh $CMD
        RETVAL=$?
        return $RETVAL
}
rhstatus() {
        status fdfs_storaged
}
restart() {
        /usr/local/fdfs/restart.sh $CMD &
}
```

把启动脚本中的路径按照上图修改
修改完毕后
注册服务
chkconfig --add fdfs_trackerd

然后可以用 service fdfs_trackerd start 启动测试 如下图

```
[root@localhost opt]# ps -ef |grep tracker
root       9736   9135  0 22:00 pts/2    00:00:00 grep --color=auto tracker
[root@localhost opt]# service fdfs_trackerd start
Starting fdfs_trackerd (via systemctl):                    [  确定  ]
[root@localhost opt]# ps -ef |grep tracker
root       9799      1  0 22:00 ?        00:00:00 /usr/bin/fdfs_trackerd /etc/fdfs/tracker.conf
root       9807   9135  0 22:00 pts/2    00:00:00 grep --color=auto tracker
```

二、FastDFS--storage 安装

1 修改配置文件

vim /etc/fdfs/storage.conf

```
# network timeout in seconds
# default value is 30s
network_timeout=60

# heart beat interval in seconds
heart_beat_interval=30

# disk usage report interval in seconds
stat_report_interval=60

# the base path to store data and log files
base_path=/opt/fastdfs

# max concurrent connections the server supported
# default value is 256
# more max_connections means more memory will be used
max_connections=256
```

```
# store_path# , based 0, if store_path0 not exists, it's value is base_path
# the paths must be exist
store_path0=/opt/fastdfs/fdfs_storage
#store_path1 /home/yuqing/fastdfs2

# subdir_count  * subdir_count directories will be auto created under each
# store_path (disk), value can be 1 to 256, default value is 256
subdir_count_per_path=256

# tracker_server can ocur more than once, and tracker_server format is
#   "host:port", host can be hostname or ip address
tracker_server=192.168.67.163:22122
```

2 创建 fdfs_storage 文件夹

mkdir /opt/fastdfs/fdfs_storage

3 设置启动服务

vim /etc/init.d/fdfs_storaged

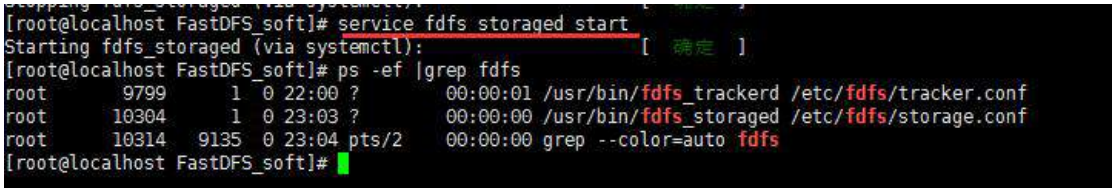
```
PRG=/usr/bin/fdfs_storaged
CONF=/etc/fdfs/storage.conf

if [ ! -f $PRG ]; then
  echo "file $PRG does not exist!"
  exit 2
fi

if [ ! -f /usr/local/fdfs/stop.sh ]; then
  echo "file /usr/local/fdfs/stop.sh does not exist!"
  exit 2
fi

if [ ! -f /usr/local/fdfs/restart.sh ]; then
  echo "file /usr/local/fdfs/restart.sh does not exist!"
  exit 2
fi
```

```
stop() {
        /usr/local/fdfs/stop.sh $CMD
        RETVAL=$?
        return $RETVAL
}
rhstatus() {
        status fdfs_storaged
}
restart() {
        /usr/local/fdfs/restart.sh $CMD &
}
```

chkconfig --add fdfs_storaged

启动服务

service fdfs_storaged start



4 功能文件目录总结说明

/opt/fastdfs/	数据文件及日志
/usr/bin/fdfs_trackerd 、 fdfs_storaged	启动执行程序
/usr/local/fdfs/ stop.sh 、 restart.sh	关闭、重启脚本
/etc/init.d/fdfs_tracked 、fdfs_storaged	服务启动脚本
/etc/fdfs/	配置文件

5 上传图片测试

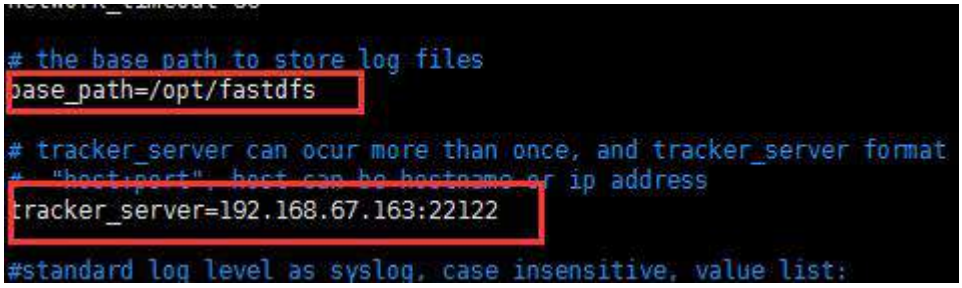
FastDFS 安装成功可通过/usr/bin/fdfs_test 测试上传、下载等操作。

修改/etc/fdfs/client.conf

[root@localhost ~]# vim /etc/fdfs/client.conf

base_path=/opt/fastdfs

tracker_server=192.168.67.163:22122



比如将/root 下的日志上传到 FastDFS 中：

/usr/bin/fdfs_test /etc/fdfs/client.conf upload

/root/winteriscoming.jpg

```
[root@localhost FastDFS_soft]# /usr/bin/fdfs_test /etc/fdfs/client.conf upload /root/winteriscoming.jpg
This is FastDFS client test program v5.05

Copyright (C) 2008, Happy Fish / YuQing

FastDFS may be copied only under the terms of the GNU General
Public License V3, which may be found in the FastDFS source kit.
Please visit the FastDFS Home Page http://www.csource.org/
for more detail.

[2018-03-09 23:18:16] DEBUG - base_path=/opt/fastdfs, connect_timeout=30, network_timeout=60, tracker_server_count=1, a
token=0, anti_steal_secret_key length=0, use_connection_pool=0, g_connection_pool_max_idle_time=3600s, use_storage_id=0
server id count: 0

tracker_query_storage_store_list_without_group:
        server 1. group_name=, ip_addr=192.168.67.163, port=23000

group_name=group1, ip_addr=192.168.67.163, port=23000
storage_upload_by_filename
group_name=group1, remote_filename=M00/00/00/wKhDo1qipbiAJC6iAAB1tayPlqs094.jpg
source ip address: 192.168.67.163
file timestamp=2018-03-09 23:18:16
file size=30133
file crc32=2895091371
example file url: http://192.168.67.163/group1/M00/00/00/wKhDo1qipbiAJC6iAAB1tayPlqs094.jpg
storage_upload_slave_by_filename
group_name=group1, remote_filename=M00/00/00/wKhDo1qipbiAJC6iAAB1tayPlqs094_big.jpg
source ip address: 192.168.67.163
file timestamp=2018-03-09 23:18:16
file size=30133
file crc32=2895091371
example file url: http://192.168.67.163/group1/M00/00/00/wKhDo1qipbiAJC6iAAB1tayPlqs094_big.jpg
[root@localhost FastDFS_soft]#
```

对应的上传路径：

/opt/fastdfs/fdfs_storage/data

/00/00/wKhDo1qipbiAJC6iAAB1tayPlqs094_big.jpg

```
-rw-r--r--. 1 root root 30133 3月   9 23:18 wKhDo1qipbiAJC6iAAB1tayPlqs094_big.jpg
-rw-r--r--. 1 root root    49 3月   9 23:18 wKhDo1qipbiAJC6iAAB1tayPlqs094_big.jpg-m
-rw-r--r--. 1 root root 30133 3月   9 23:18 wKhDo1qipbiAJC6iAAB1tayPlqs094.jpg
-rw-r--r--. 1 root root    49 3月   9 23:18 wKhDo1qipbiAJC6iAAB1tayPlqs094.jpg-m
[root@localhost 00]# pwd
/opt/fastdfs/fdfs_storage/data/00/00
```

三、FastDFS 整合 nginx

1 安装前配置 fastdfs-nginx-module

上传 fastdfs-nginx-module_v1.16.tar.gz 上传到 /usr/local，并解压

tar -zxvf fastdfs-nginx-module_v1.16.tar.gz

编辑配置文件：修改 config 文件将/usr/local/路径改为/usr/

vim

fastdfs-nginx-module/src/config

```
ngx_addon_name=ngx_http_fastdfs_module
HTTP_MODULES="$HTTP_MODULES ngx_http_fastdfs_module"
NGX_ADDON_SRCS="$NGX_ADDON_SRCS $ngx_addon_dir/ngx_http_fastdfs_module.c"
CORE_INCS="$CORE_INCS /usr/include/fastdfs /usr/include/fastcommon/"
CORE_LIBS="$CORE_LIBS -L/usr/lib -lfastcommon -lfdfsclient"
CFLAGS="$CFLAGS -D_FILE_OFFSET_BITS=64 -DFDFS_OUTPUT_CHUNK_SIZE='256*1024' -DFDFS_MOD_CONF_FILENAME='\"/etc/fdfs/mod_fastdfs.conf\
"'"
~
~
~
~
~
~
~
~
~
~
-- 插入 --                                                    4,48          全部
```

将 FastDFS-nginx-module/src 下的 mod_fastdfs.conf 拷贝至/etc/fdfs/下

[root@localhost src]# cp mod_fastdfs.conf /etc/fdfs/

并修改 mod_fastdfs.conf 的内容：

vim /etc/fdfs/mod_fastdfs.conf

```
# the base path to store log files
base_path=/opt/fastdfs

# if load FastDFS parameters from tracker server
# since V1.12
# default value is false
load_fdfs_parameters_from_tracker=true

# storage sync file max delay seconds
```

继续修改

```
# FastDFS tracker_server can ocur more than once, and tracker_server format is
#  "host:port", host can be hostname or ip address
# valid only when load_fdfs_parameters_from_tracker is true
tracker_server=192.168.67.163:22122

# the port of the local storage server
# the default value is 23000
storage_server_port=23000
```

继续修改 url 中包含 group 名称

```
tracker_server=192.168.67.130:22122

# the port of the local storage server
# the default value is 23000
storage_server_port=23000

# the group name of the local storage server
group_name=group1

# if the url / uri including the group name
# set to false when uri like /M00/00/00/xxx
# set to true when uri like ${group_name}/M00/00/00/xxx, such as group1/M00/xxx
# default value is false
url_have_group_name = true

# path(disk or mount point) count, default value is 1
# must same as storage.conf
-- 插入 --
```

继续修改　#指定文件存储路径

```
# store_path#, based 0, if store_path0 not exists, it's value is base_path
# the paths must be exist
# must same as storage.conf
store_path0=/opt/fastdfs/fdfs_storage
#store_path1=/home/yuqing/fastdfs1
```

将 libfdfsclient.so 拷贝至/usr/lib 下

[root@localhost src]# cp /usr/lib64/libfdfsclient.so /usr/lib/

2 安装 fastdfs-nginx-module

创建 nginx/client 目录

[root@localhost src]# mkdir -p /var/temp/nginx/client

cd nginx 的原始程序目录

```
./configure \
--prefix=/usr/local/nginx \
--pid-path=/var/run/nginx/nginx.pid \
--lock-path=/var/lock/nginx.lock \
--error-log-path=/var/log/nginx/error.log \
--http-log-path=/var/log/nginx/access.log \
--with-http_gzip_static_module \
--http-client-body-temp-path=/var/temp/nginx/client \
--http-proxy-temp-path=/var/temp/nginx/proxy \
--http-fastcgi-temp-path=/var/temp/nginx/fastcgi \
--http-uwsgi-temp-path=/var/temp/nginx/uwsgi \
```

```
--http-scgi-temp-path=/var/temp/nginx/scgi \
--add-module=/opt/fastdfs-nginx-module/src
```

```
./configure --add-module=/opt/fastdfs-nginx-module/src
```

配置成功

```
Configuration summary
  + using system PCRE library
  + OpenSSL library is not used
  + using builtin md5 code
  + sha1 library is not found
  + using system zlib library

  nginx path prefix: "/usr/local/nginx"
  nginx binary file: "/usr/local/nginx/sbin/nginx"
  nginx configuration prefix: "/usr/local/nginx/conf"
  nginx configuration file: "/usr/local/nginx/conf/nginx.conf"
  nginx pid file: "/var/run/nginx/nginx.pid"
  nginx error log file: "/var/log/nginx/error.log"
  nginx http access log file: "/var/log/nginx/access.log"
  nginx http client request body temporary files: "/var/temp/nginx/client"
  nginx http proxy temporary files: "/var/temp/nginx/proxy"
  nginx http fastcgi temporary files: "/var/temp/nginx/fastcgi"
  nginx http uwsgi temporary files: "/var/temp/nginx/uwsgi"
  nginx http scgi temporary files: "/var/temp/nginx/scgi"

[root@localhost nginx-1.8.0]#
```

编译

[root@localhost nginx-1.12.2]# make

安装

[root@localhost nginx-1.12.2]# make install

3 编辑 nginx.conf

vim /usr/local/nginx/conf/nginx.conf

```
server {
        listen          80;
        server_name     file.gmall.com;
        #charset koi8-r;
```

```
        #access_log    logs/host.access.log    main;
        location / {
            root    html;
            index    index.html index.htm;
        }
        location /group1/M00/ {
            ngx_fastdfs_module;
        }
```

启动 nginx

/usr/local/nginx/sbin/nginx

设置开机启动

[root@iZ2zednyjjxxq7k3i2dwsfZ nginx-1.12.2]# vim /etc/rc.d/rc.local

```
touch /var/lock/subsys/loca
/usr/bin/fdfs_trackerd /etc/fdfs/tracker.conf restart
/usr/bin/fdfs_storaged /etc/fdfs/storage.conf restart
/usr/local/nginx/sbin/nginx
~
```

需要关闭防火墙

service iptables stop

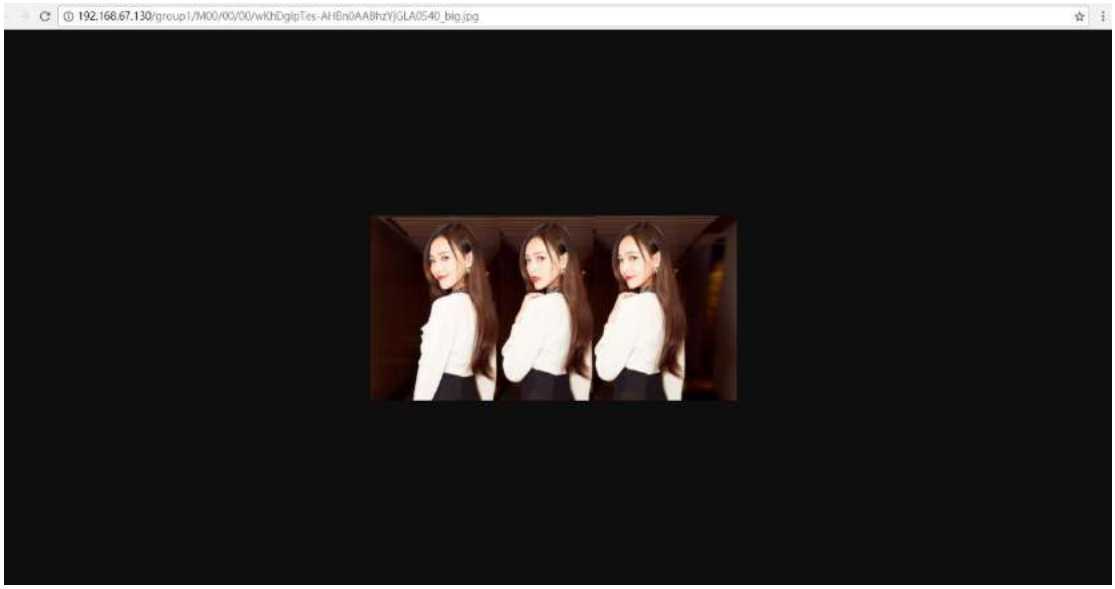
永久关闭 chkconfig iptables off

测试

/usr/bin/fdfs_test /etc/fdfs/client.conf upload /root/ty.jpg

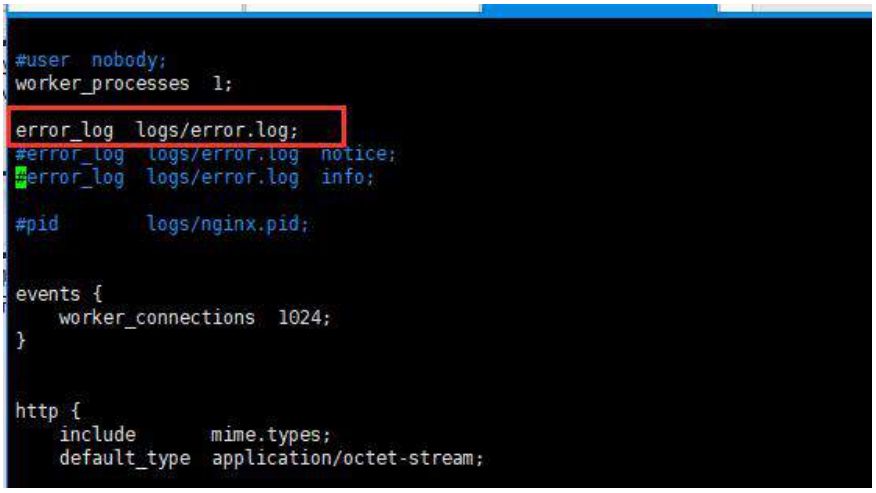
```
[root@localhost nginx-1.8.0]# /usr/bin/fdfs_test /etc/fdfs/client.conf upload /root/ty.jpg
This is FastDFS client test program v5.05

Copyright (C) 2008, Happy Fish / YuQing

FastDFS may be copied only under the terms of the GNU General
Public License V3, which may be found in the FastDFS source kit.
Please visit the FastDFS Home Page http://www.csource.org/
for more detail.

[2018-01-08 22:06:07] DEBUG - base_path=/home/fastdfs, connect_timeout=30, network_timeout=60, tracker_server_count=1, anti_stea
token=0, anti_steal_secret_key length=0, use_connection_pool=0, g_connection_pool_max_idle_time=3600s, use_storage_id=0, storage
erver id count: 0

tracker_query_storage_store_list_without_group:
        server 1. group_name=, ip_addr=192.168.67.130, port=23000

group_name=group1, ip_addr=192.168.67.130, port=23000
storage_upload_by_filename
group_name=group1, remote_filename=M00/00/00/wKhDglpTes-AHBn0AABhzYjGLA0540.jpg
source ip address: 192.168.67.130
file timestamp=2018-01-08 22:06:07
file size=25037
file crc32=2294688781
example file url: http://192.168.67.130/group1/M00/00/00/wKhDglpTes-AHBn0AABhzYjGLA0540.jpg
storage_upload_slave_by_filename
group_name=group1, remote_filename=M00/00/00/wKhDglpTes-AHBn0AABhzYjGLA0540_big.jpg
source ip address: 192.168.67.130
file timestamp=2018-01-08 22:06:07
file size=25037
file crc32=2294688781
example file url: http://192.168.67.130/group1/M00/00/00/wKhDglpTes-AHBn0AABhzYjGLA0540_big.jpg
[root@localhost nginx-1.8.0]#
```

显示结果：



4 问题排查

打开 vim /usr/local/nginx/conf/nginx.conf



然后去 logs/error.log 查看报错。

四、 附：nginx 注册服务脚本

```
#!/bin/bash
#chkconfig:2345 21 91
#decription: nginx-server
```

```
nginx=/usr/local/nginx/sbin/nginx
case "$1" in
        start)
                netstat -anlpt | grep nginx
            if
                [ $? -eq 0 ]
             then
                echo " the nginx-server is already running"
            else
                echo " ther nginx-server is starting to run"
                $nginx
            fi
          ;;
        stop)
              netstat -anlpt | grep nginx
                if
                [ $? -eq 0 ]
              then
                    $nginx -s stop
                    if [ $? -eq 0 ]
                        then
                            echo " the nginx-server is stopped "
                    else
                            echo " failed to stop the nginx-server"
                     fi
            else
                 echo " the nginx-server has stopped you needn't to stop it "
            fi
          ;;
      restart)
                  $nginx -s reload
             if
                  [ $? -eq 0 ]
                then
                    echo "the nginx-server is restarting "
              else
                    echo " the nginx-server failed to restart"
             fi
          ;;

        status)
                     netstat -anlpt | grep nginx
```

```
                if
                    [ $? -eq 0 ]
                  then
                      echo " the nginx-server is running "
               else
                      echo " the nginx-server is not running ,please try again"
                fi
        ;;

         status)
                      netstat -anlpt | grep nginx
                if
                    [ $? -eq 0 ]
                  then
                      echo " the nginx-server is running "
               else
                      echo " the nginx-server is not running ,please try again"
                fi
          ;;
         *)
                  echo "please enter { start|stop|status|restart}"
         ;;
esac
```