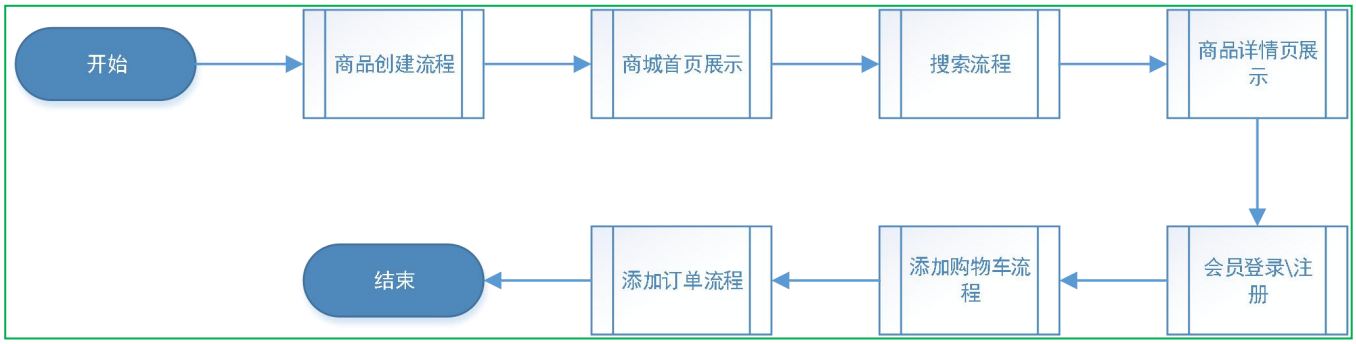
电商项目总结

1 电商核心业务主线

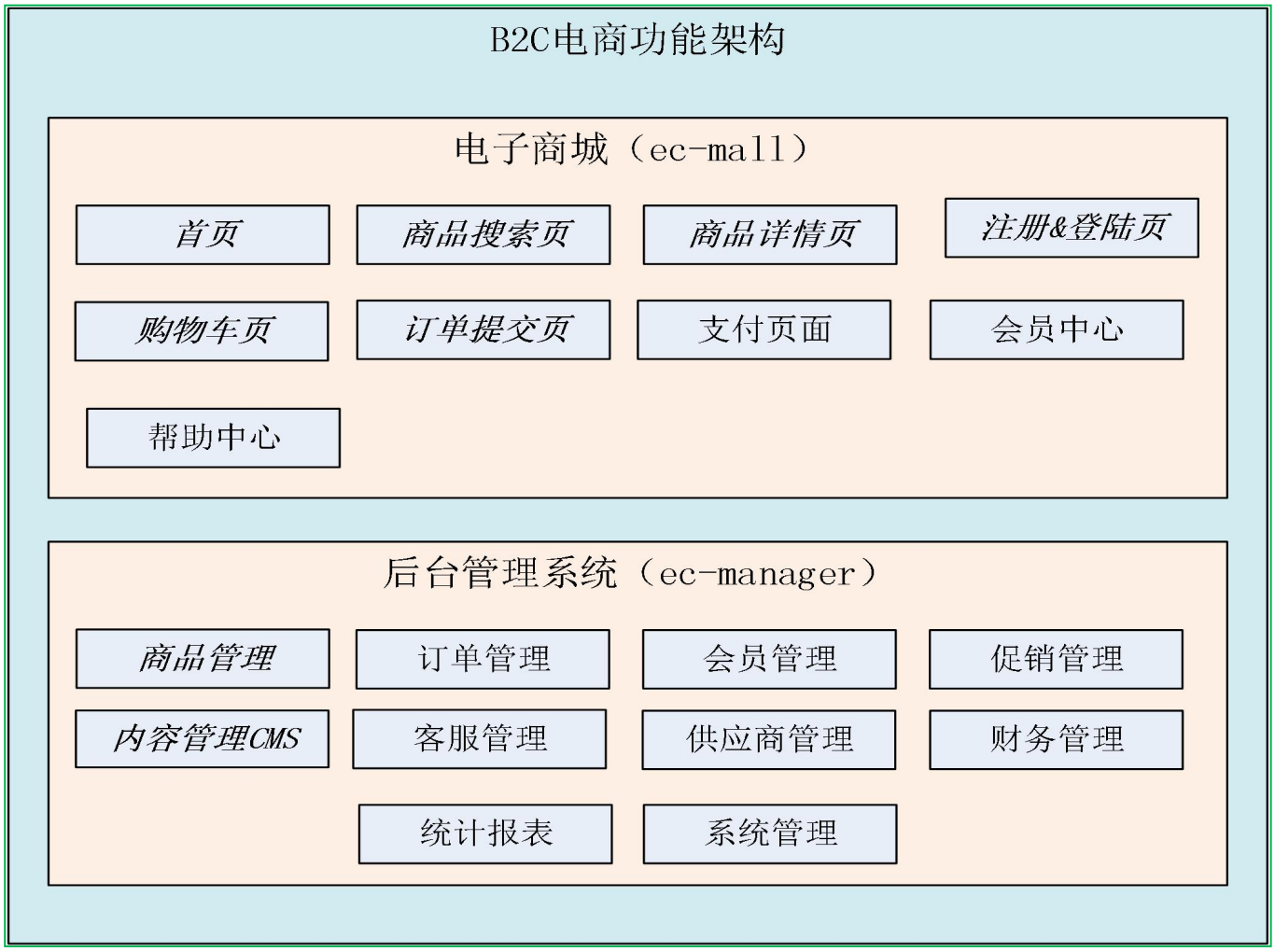


2.电商运营模式

按照电商行业的运营模型，一般分为以下几种模式：

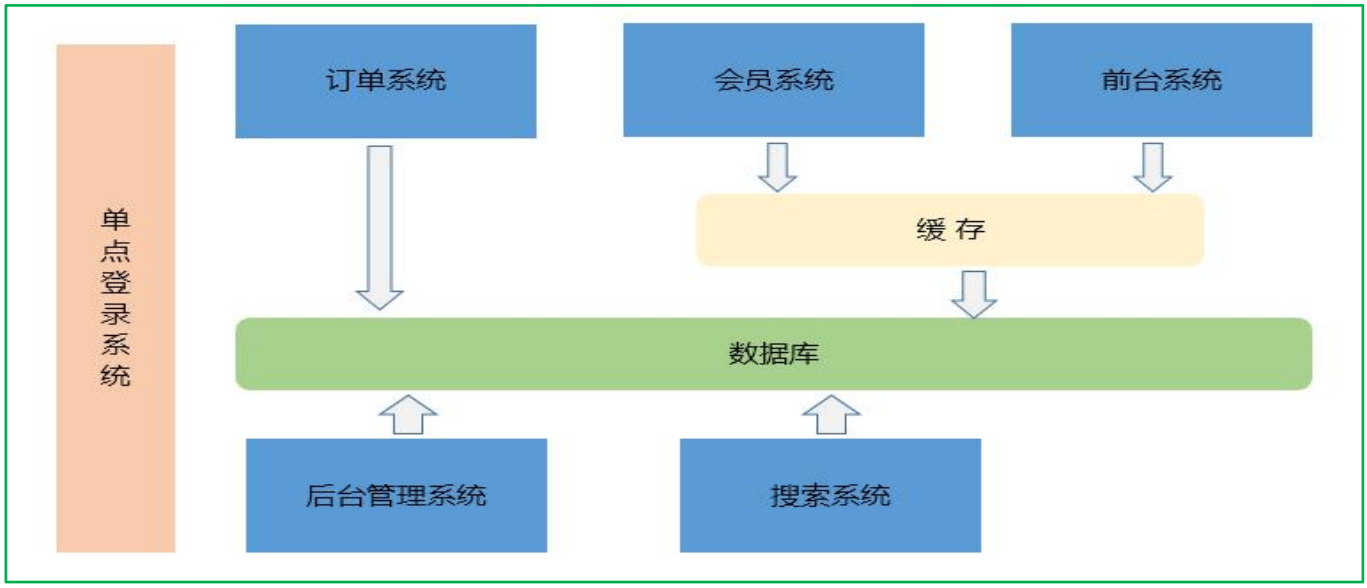
① B2B：商家到商家。例如：阿里巴巴（www.1688.com）、慧聪网。
② B2C：商家到用户。例如京东商城、国美网上商城。
③ B2B2C：商家到商家到用户，比如京东、天猫都有第三方商户。
④ C2C：用户到用户。比如淘宝。
⑤ O2O：线上到线下，比如百度外卖、美团、滴滴打车等。

3.功能架构



4.系统架构

(1)**分布式架构**：把系统按照系统模块拆分成多个独立的子系统。



优点：

①系统之间**弱耦合**，把模块拆分，使用接口通信，降低模块之间的耦合度。

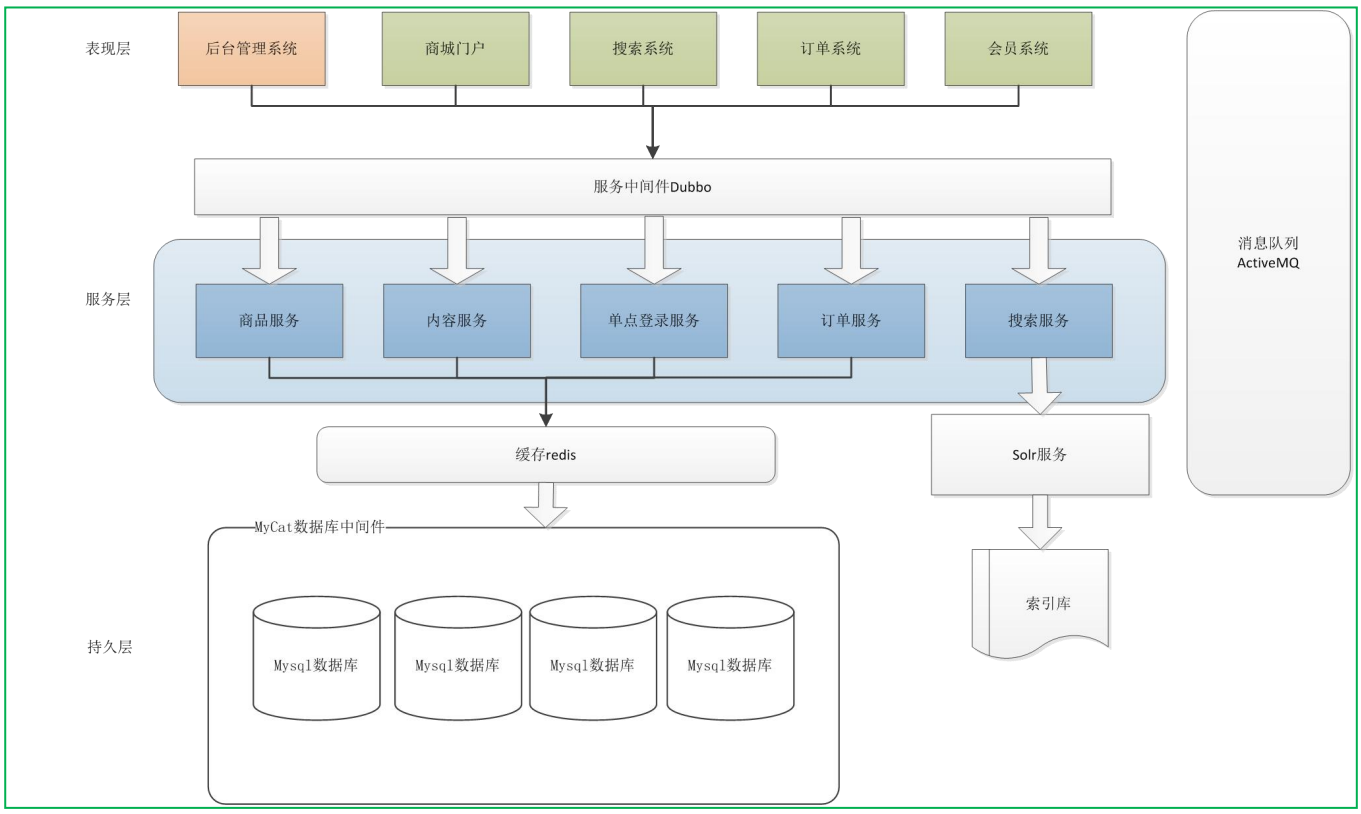
②系统之间**扩展性好**，有新功能时，添加子模块即可，不会对原有系统产生破坏，大大增强系统可扩展性。

③**部署灵活**方便，互联网应用需要进行分布式部署，使用这种架构，可以灵活的进行分布式部署。

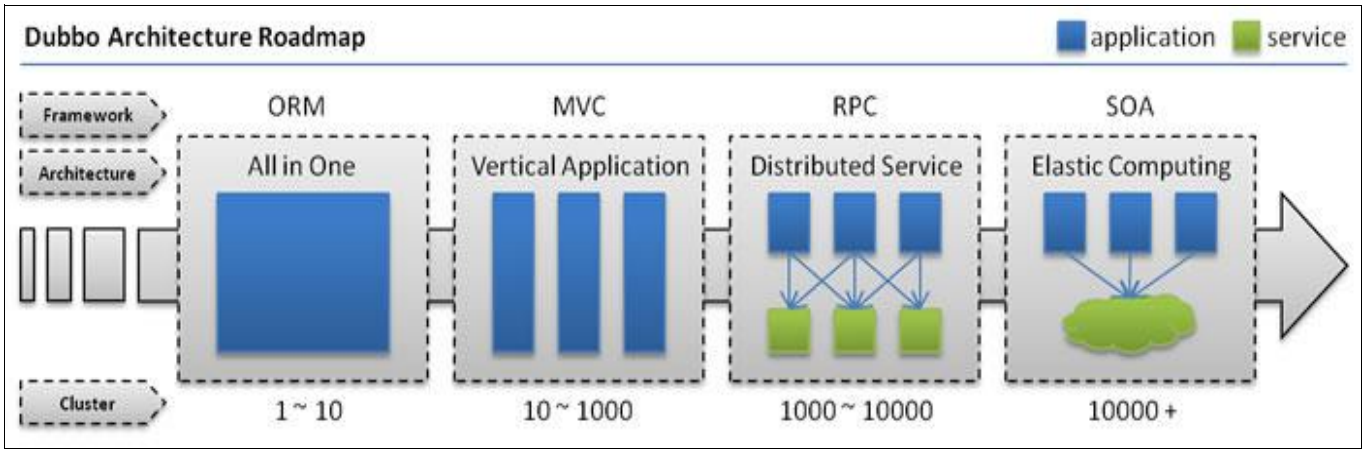
缺点：系统之间交互需要使用远程通信，接口开发增加工作量。

(2)SOA 架构：

电商的架构就是采用的分布式架构，只不过在此架构基础之上，为了实现面向服务的思想，所以目前大多数电商的架构，采用的都是**分布式服务架构**。



(3)架构演化过程：



- 单一应用架构
  - 当网站流量很小时，只需一个应用，将所有功能都部署在一起，以减少部署节点和成本。
- 垂直应用架构
  - 当访问量逐渐增大，单一应用增加机器带来的加速度越来越小，将应用拆成互不相干的几个应用，以提升效率。
- 分布式服务架构
  - 当垂直应用越来越多，应用之间交互不可避免，将核心业务抽取出来，作为独立的服务，逐渐形成稳定的服务中心，使前端应用能更快速的响应多变的市场需求。
- 流动计算架构
  - 当服务越来越多，容量的评估，小服务资源的浪费等问题逐渐显现，此时需增加一个调度中心基于访问压力实时管理集群容量，提高集群利用率。
  - 此时，用于提高机器利用率的 **SOA 服务治理方案**是关键。
  - **Dubbo** 就是 SOA 服务治理方案的核心框架。

5.常用技术选项

- Maven（管理依赖及工程构建）
- SVN（版本控制工具）
- SSM 框架（spring+springmvc+mybatis）
- Mysql（数据库）
- JSP、jQuery、jQuery EasyUI、KindEditor（富文本编辑器）
- Dubbo+zookeeper（调用&发布服务）
- Nginx（http&反向代理服务器）
- FastDFS（图片服务器）
- Redis（缓存服务器）
- Solr（搜索服务器）
- Freemarker（页面静态化）
- Activemq 消息中间件

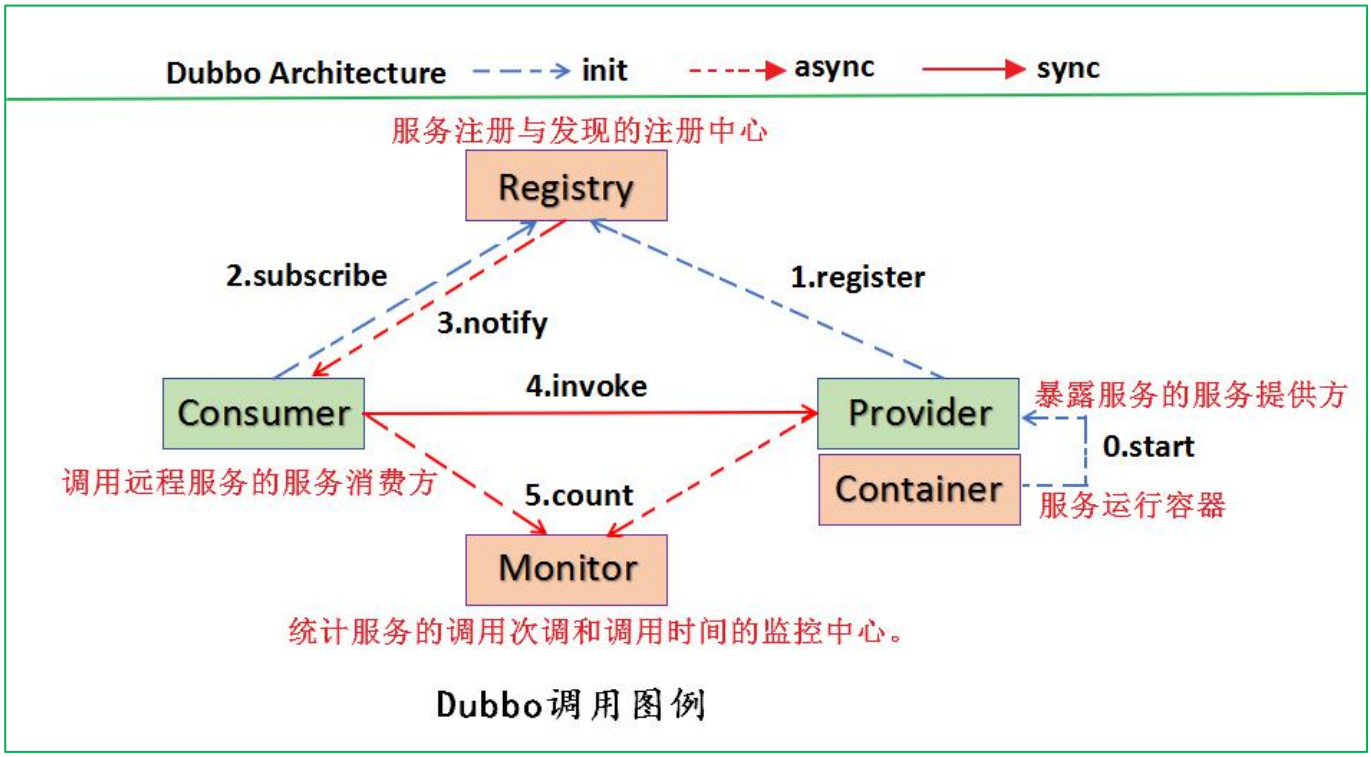
6.Duboo 服务中间件

Dubbo 是 Alibaba 开源的分布式服务框架，它最大的特点是按照分层的方式来架构，使用这种方式可以使各个层之间解耦合（或者最大限度地松耦合），比如表现层和业务层就需要解耦合。

从面向服务的角度来看，Dubbo 采用的是一种非常简单的模型，要么是提供方提供服务，要么是消费方消费服务，所以基于这一点可以抽象出服务提供方（Provider）和服务消费方（Consumer）两个角色。

除了以上两个角色，它还有注册中心和监控中心。它可以通过注册中心对服务进行注册和订阅；

可以通过监控中心对服务进行监控，这样的话，就可以知道哪些服务使用率高、哪些服务使用率低。对使用率高的服务增加机器，对使用率低的服务减少机器，达到合理分配资源的目的。



Dubbo调用图例

(1)调用关系说明

0. 服务容器负责启动，加载，运行服务提供者。
1. 服务提供者在启动时，向注册中心注册自己提供的服务。
2. 服务消费者在启动时，向注册中心订阅自己所需的服务。
3. 注册中心返回服务提供者地址列表给消费者，如果有变更，注册中心将基于长连接推送变更数据给消费者。
4. 服务消费者，从提供者地址列表中，基于软负载均衡算法，选一台提供者进行调用，如果调用失败，再选另一台调用。
5. 服务消费者和提供者，在内存中累计调用次数和调用时间，定时每分钟发送一次统计数据到监控中心。

(2)Dubbo 使用方法

Dubbo 采用全 Spring 配置方式，透明化接入应用，对应用没有任何 API 侵入，**只需用 Spring 加载 Dubbo 的配置即可。**

服务提供者.xml：

```
<!-- 和本地服务一样实现远程服务 -->
<bean id="xxxService" class="com.xxx.XxxServiceImpl" />
<!—配置注册中心-->
    <!-- 提供方应用信息，用于 dubbo monitor 计算依赖关系 -->
    <dubbo:application name="xxx-manager-service" />
    <!-- 使用 zookeeper 注册中心暴露服务地址 -->
    <dubbo:registry protocol="zookeeper" address="192.168.242.128:2181" />
    <!-- 使用 dubbo 协议在 20880 端口暴露服务 -->
    <dubbo:protocol name="dubbo" port="20880" />
<!-- 配置 Dubbo 发布 -->
<dubbo:service interface="com.xxx.XxxService" ref="xxxService" />
</bean>
```

服务消费者.xml：

```
<!—配置注册中心-->
    <!-- 消费方应用信息，用于计算依赖关系 -->
    <dubbo:application name="taotao-manager-web" />
    <!-- 使用 zookeeper 注册中心暴露服务地址 -->
    <dubbo:registry protocol="zookeeper" address="192.168.242.139:2181" />

<!-- 配置 Dubbo 引用 -->
<dubbo:reference interface="com.xxx.XxxService" id="xxxService" />

<!-- 和本地服务一样使用远程服务 -->
<bean id="xxxAction" class="com.xxx.XxxAction">
     <property name="xxxService" ref="xxxService" />
</bean>
```

7.PageHelper + EasyUI 实现分页

(1)EasyUI 页面分页

页面逻辑：页面初始化时，通过 jquery easyui 的 DataGrid（数据表格）的 url 属性异步加载，返回指定的 json 格式数据，在通过 pagination 属性，展示分页工具栏。

pagination	boolean	如果为true，则在DataGrid控件底部显示分页工具栏。

表现层分析：

请求 URL：/XXX/list

请求参数：Integer page、Integer rows（easyui 分页控件请求的参数），其中 page 默认为 1。

返回数据：json 格式的数据（easyui 分页控件请求的返回值 JSON 数据），格式如下：

```
{total:"2", rows:[{"id":"1","name":"张三"},{"id":"2","name":"李四"}]}
```

==> 将 Json 对应的数据格式封装成 POJO 类 DatagridResult。

业务逻辑分析：

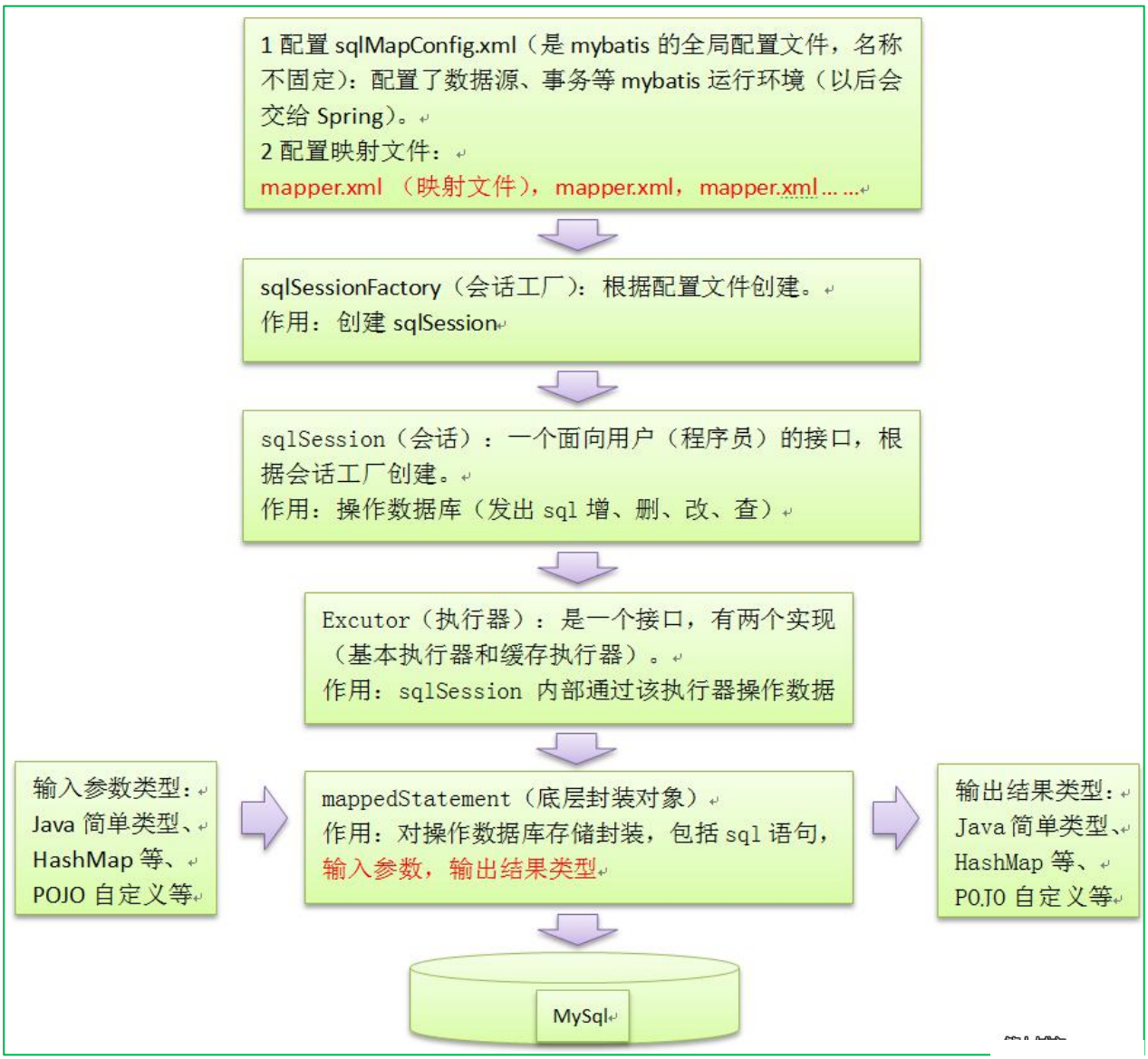
根据 page 和 rows 分页查询条件，使用分页插件 PageHelper 进行分页查询。

将商品列表和记录总数封装到 PO 类对象中，并且将其转化为 Json 格式返回。

(2)PageHelper 数据库分页

mybatis 分页插件 PageHelper 目前支 Oracle,Mysql,MariaDB,SQLite,Hsqldb,PostgreSQL 六种数据库分页。

分页原理：



使用方法：

第一步：dao 层的 pom 文件中添加 pagehelper 依赖；

```
<dependency>
    <groupId>com.github.pagehelper</groupId>
    <artifactId>pagehelper</artifactId>
    <version>1.0</version>
</dependency>
```

第二步：在 Mybatis 配置 SqlMapConfig.xml 中配置拦截器插件；

```
<plugins>
    <!-- com.github.pagehelper 为 PageHelper 类所在包名 -->
    <plugin interceptor="com.github.pagehelper.PageHelper">
        <!-- 设置数据库方言 -->
        <property name="dialect" value="mysql" />
    </plugin>
</plugins>
```

第三步：将 Json 对应的数据格式封装成 POJO 类 DatagridResult 并实现序列化接口；

```
public class DatagridResult implements Serializable{
    private long total;// 记录总数
    private List rows; // 记录集合
}
```

第四步：开发 Dao 层，使用逆向工程生成代码；

第五步：开发 Service 层，主要逻辑：

```
@Override
    public DatagridResult queryItemList(Integer page, Integer rows) {
        if (page == null)
            page = 1;
        if (rows == null)
            rows = 30;
        // 1.设置分页信息
        PageHelper.startPage(page, rows);
        // 2.执行查询
        TbItemExample example = new TbItemExample();
        List<TbItem> list = mapper.selectByExample(example);
        // 3.获取分页查询后的数据
        PageInfo<TbItem> pageInfo = new PageInfo<>(list);
        // 4.封装结果对象，并返回
        DatagridResult result = new DatagridResult();
        result.setTotal(pageInfo.getTotal()); //设置总记录数
        result.setRows(list);  //设置每页展示数据集合
        return result;
    }
```

第六步：开发 Controller 层，主要逻辑：

```
@RequestMapping("/list")
@ResponseBody
 public DatagridResult list(@RequestParam(defaultValue= "1") Integer page, Integer
rows) {
        return service.queryItemList(page, rows);
    }
```

(3)超时及警告问题解决

原因：通过分页插件得到的结果，其实是 List 的子类 Page（该类由分页插件提供），而该类只在服务层，表现层没有该类，在反序列化的时候，抛出该警告，不影响使用。

```
警告: Hessian/Burlap: 'com.github.pagehelper.Page' is an unknown class in WebappClassLoader
  context:
  delegate: false
  repositories:
----------> Parent Classloader:
ClassRealm[plugin>org.apache.tomcat.maven:tomcat7-maven-plugin:2.2, parent: sun.misc.Launcher$AppClas
:
java.lang.ClassNotFoundException: com.github.pagehelper.Page
```

发布服务时，服务默认的响应时间为 1 秒，debug 时需要显示设置，单位是毫秒

```
<dubbo:service interface="com.Yxx.xxx.manager.service.ItemService"
    ref="itemServiceImpl" timeout="600000" />
```

8.JQuery EasyUI 的 tree 控件

注意：初始化时，只加载一级商品类目。展示子目录，需要重新加载。

（1）表现层分析：

异步加载商品类目的请求 URL：/itemCat/list

异步加载商品类目的请求参数：@RequestParam(“id”) Long parentId，默认为 0；

异步树控件

树控件内建异步加载模式的支持，用户先创建一个空的树，然后指定一个服务器端，执行检索后动态返回JSON数据来填充树并完成异步请求。例子如下：

```
1. <ul class="easyui-tree" data-options="url:'get_data.php'"></ul>
```

树控件读取URL。子节点的加载依赖于父节点的状态。当展开一个封闭的节点，如果节点没有加载子节点，它将会把节点id的值作为http请求参数并命名为'id'，通过URL发送到服务器上面检索子节点。

异步加载商品类目的请求返回值：json 格式的商品类目节点数据

树控件每个节点都具备以下属性：

树控件数据格式化

每个节点都具备以下属性：

- id：节点ID，对加载远程数据很重要。
- text：显示节点文本。
- state：节点状态，'open' 或 'closed'，默认：'open'。如果为'closed'的时候，将不自动展开该节点。
- checked：表示该节点是否被选中。
- attributes: 被添加到节点的自定义属性。
- children: 一个节点数组声明了若干节点。

对应的 json 数据格式为：

```
[
    {
        "id": 1,
        "text": "Node 1",
        "state": "closed"
    },
    {
        "id": 2,
        "text": "Node 2",
        "state": "closed"
    }
]
```

需要定义一个 POJO 类对 Json 数据进行封装，包含三个属性：id、text、state

（2）业务逻辑：

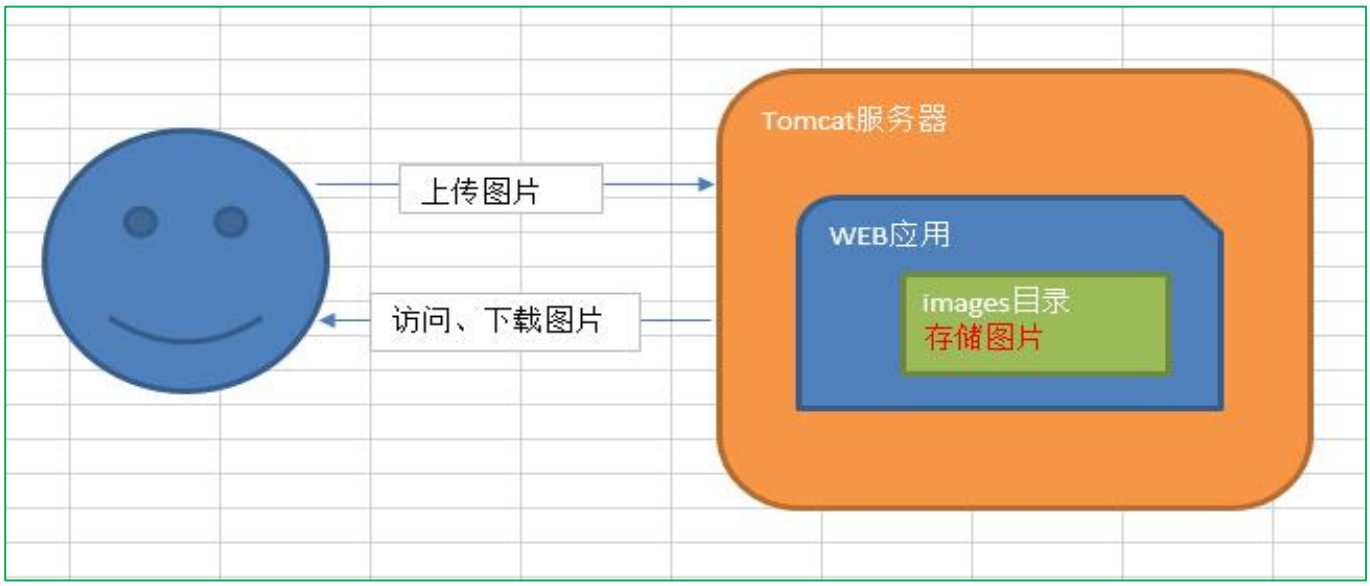
1）根据 parentId 查询商品类目列表

2）将查询结果封装到 POJO 中，如果是叶子节点，则 state 为 open。

3）将封装后的 POJO 对象，在 Controller 层转成 Json 对象返回。

9.商品图片上传及展示

传统方式



存在问题：

1）大并发量上传访问图片时，需要对 web 应用做负载均衡，但是会存在图片共享问题

2）web 应用服务器的存储空间有限，它是 web 应用服务器，而不是存储服务器。

3）web 应用服务器的本身的 io 读写性能不高，图片上传下载时，速度偏慢

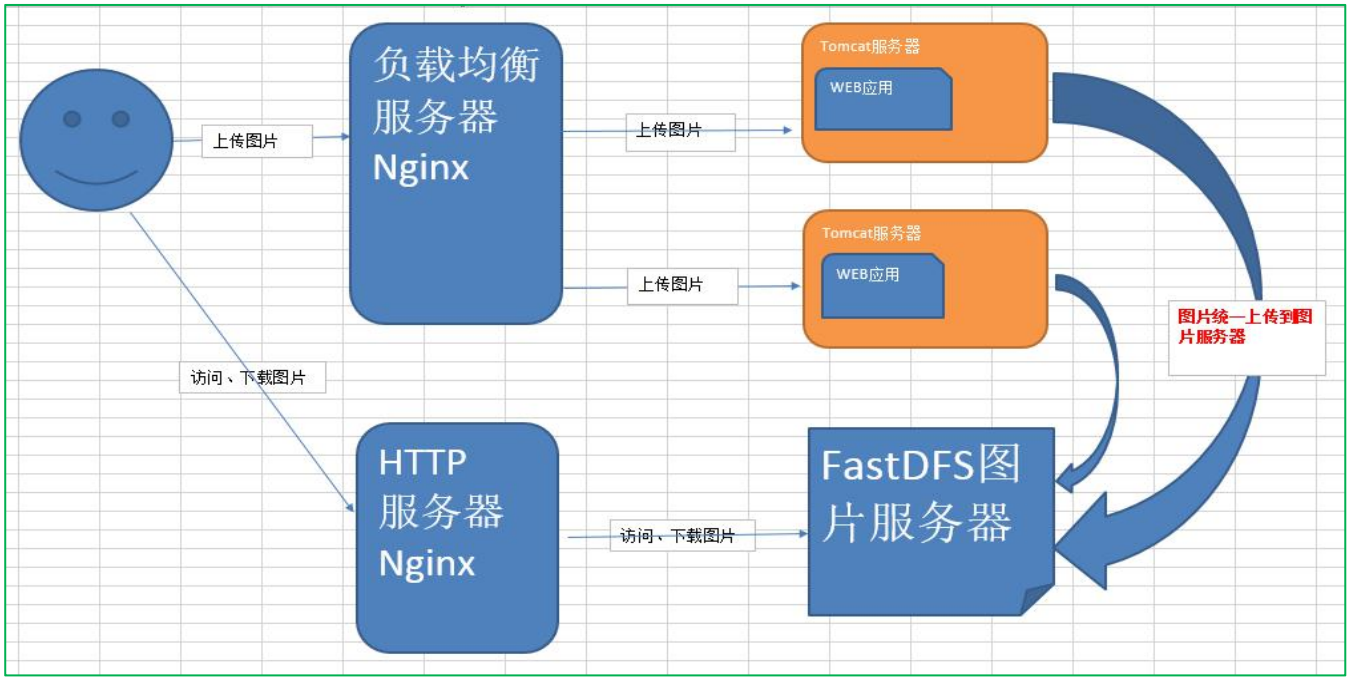
4）web 应用服务器访问图片时，由于图片内容较大，并发量大的时候，会占用 web 应用服务器的带宽，这样该 web 应用服务器的其他功能就会受到较大的影响。

解决方案：

1）将图片集中存储到 io 读写性能高的图片服务器中。

2）下载访问图片时，使用 http 服务器直接读取图片服务器中的图片。

集群模式



(1)Nginx

Nginx 是一个高性能的 HTTP 和反向代理服务器，也是一个 IMAP/POP3/SMTP 服务器。主要应用于：HTTP 服务器、反向代理服务器\负载均衡服务器、虚拟主机。

Nginx 的安装（另见《电商常用软件安装篇》）

Nginx 的使用（通过修改 nginx.conf 配置文件实现）

1）虚拟主机配置

将网络中的每一台计算机分成多个虚拟主机，每个虚拟主机可以单独对外提供 www 服务。

a)基于端口配置（ip 地址一致，端口不一致）

```
#基于 80 端口的虚拟主机
server {
        listen          80;
        server_name 192.168.242.128;
        location / {
            root html;
            index index.html index.htm;
        }
    }
```

```
#基于 81 端口的虚拟主机
server {
    listen          81;
    server_name 192.168.242.128;
    location / {
        root html81;
        index index.html index.htm;
    }
}
```

b)基于域名配置

①修改 hosts 文件，实现 ip 地址和域名的映射配置：

Hosts 文件的位置：C:\Windows\System32\drivers\etc\hosts

建议使用 SwitchHosts 工具修改本地 DNS 配置。

②配置 nginx.conf

```
#基于 www.aaa.com 域名的虚拟主机
server {
    listen          80;
    server_name www.aaa.com;
    location / {
        root htmlaaa;
        index index.html index.htm;
    }
}
```

```
#基于 www.bbb.com 域名的虚拟主机
server {
    listen          80;
    server_name www.bbb.com;
    location / {
        root htmlbbb;
        index index.html index.htm;
    }
}
```

2）反向代理配置

通常的代理服务器，只用于代理内部网络对 Internet 的连接请求；而反向代理（Reverse Proxy）方式是指以代理服务器来接受 internet 上的连接请求。

简单理解：以 tomcat 为中心，以内外网为前提，tomcat 主动出击是正向代理，而 tomcat 被动

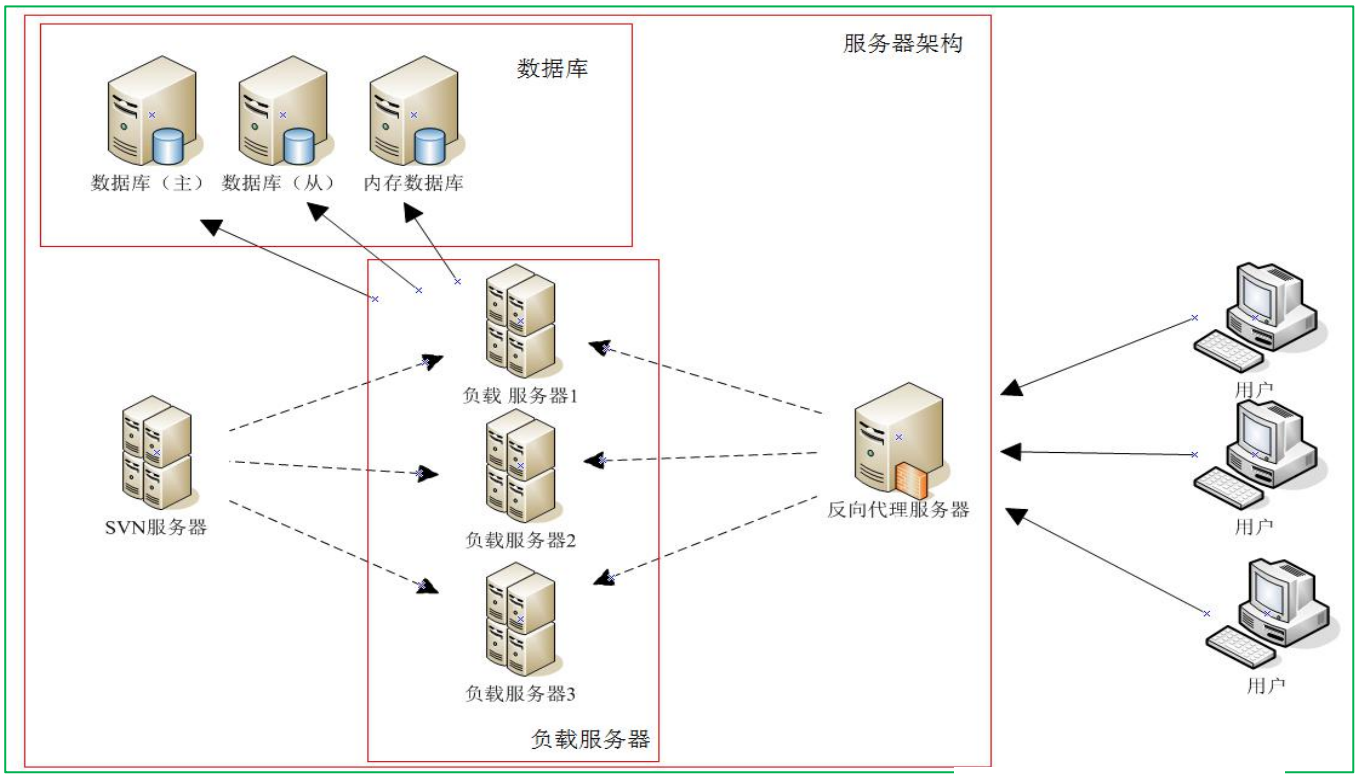
接受请求就是反向代理。

3）负载均衡配置

简单理解：反向代理负载均衡技术是把将来自 internet 上的连接请求以反向代理的方式动态地转发给内部网络上的多台服务器进行处理，从而达到负载均衡的目的。

① 负载均衡的方式：硬负载（F5 服务器）、软负载（Nginx）

② 负载均衡的策略：轮询、ip_hash 等



③ 具体配置

tomcat+nginx 演示：

反向代理案例中，通过域名 www.tomcat1.com 访问时，后台只有 8080 端口服务进行响应。

负载均衡配置时，通过域名 www.tomcat1.com 访问时，后台除了 8080 端口响应，再添加一台 8282 服务器响应服务。

修改 nginx 配置文件，修改如下：

```
upstream tomcat1{
    #weight 权重，默认是 1，权重越高，被分配的几率越大
    server 192.168.242.128:8080   weight=2;
    server 192.168.242.128:8282;
}
#配置一个虚拟主机
server {
    listen 80;
    server_name www.tomcat1.com;
    location / {
        proxy_pass http://tomcat1;
    }
}
```

(2)FastDFS

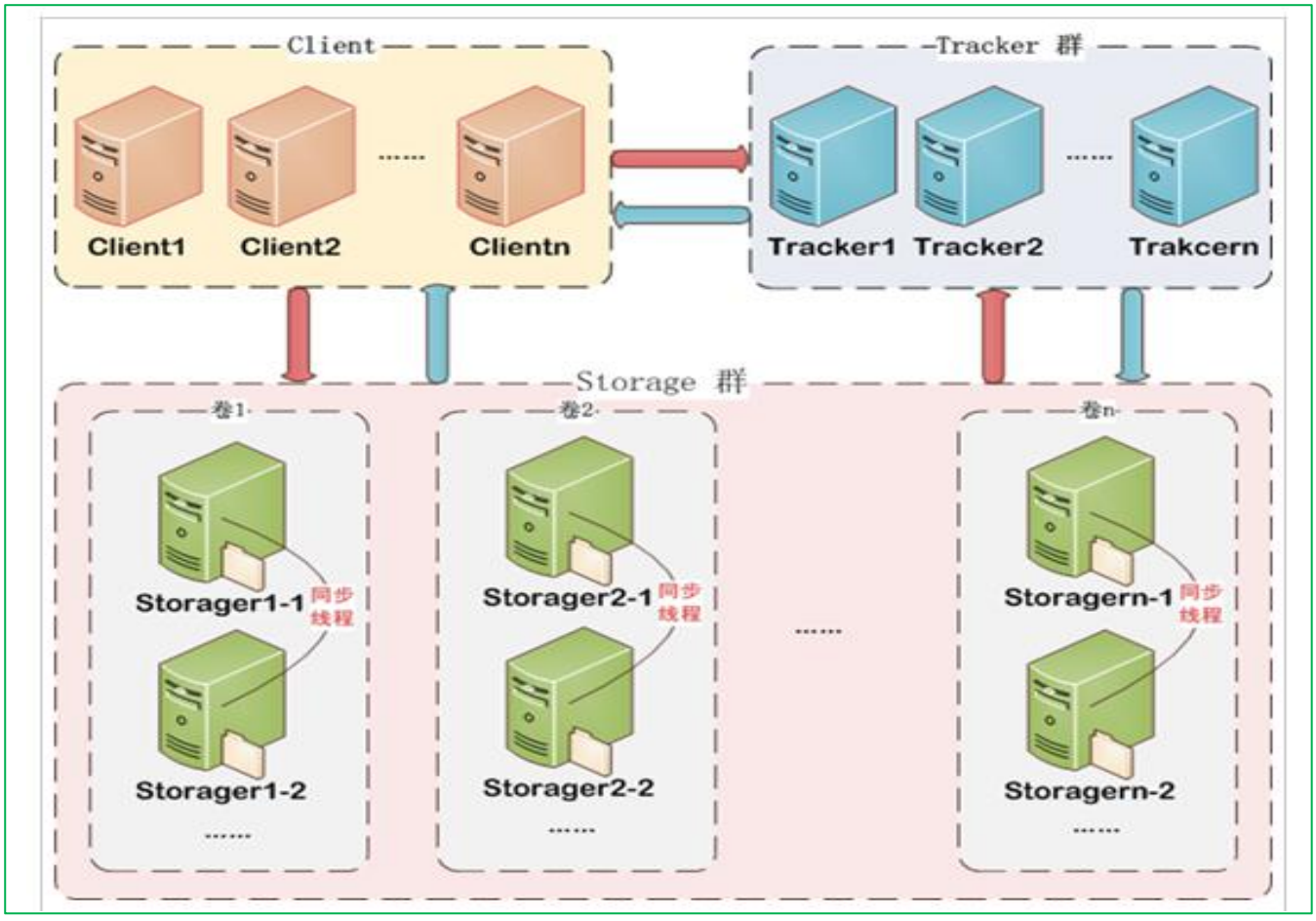
FastDFS 是用 C 语言编写的一款开源的分布式文件系统。FastDFS 为互联网量身定制，充分考虑了冗余备份、负载均衡、线性扩容等机制，并注重高可用、高性能等指标，使用 FastDFS 很容易搭建一套高性能的文件服务器集群提供文件上传、下载等服务。

1）FastDFS 架构（Tracker server+Storage server）

① **Tracker server** 作用是**对 Storage server 进行负载均衡和调度**，在文件上传时会直接请求 Tracker server，然后 Tracker server 可以根据一些策略找到 Storage server 来提供文件上传服务。所以**可以将 tracker 称为追踪服务器或调度服务器**。

② **Storage server** 作用是**文件存储**，客户端上传的文件最终存储在 Storage 服务器上，Storage server 没有实现自己的文件系统而是利用操作系统的文件系统来管理文件。**可以将 storage 称为存储服务器**。

具体架构如下图：



2）Tracker 集群

FastDFS 集群中的 Tracker server 可以有多台，Tracker server 之间是相互平等关系同时提供服务，Tracker server 不存在单点故障。客户端请求 Tracker server 采用**轮询方式**，如果请求的 tracker 无法提供服务则换另一个 tracker。

3）Storage 集群

Storage 集群采用了**分组存储方式**。storage 集群由一个或多个组构成，一个组由一台或多台存

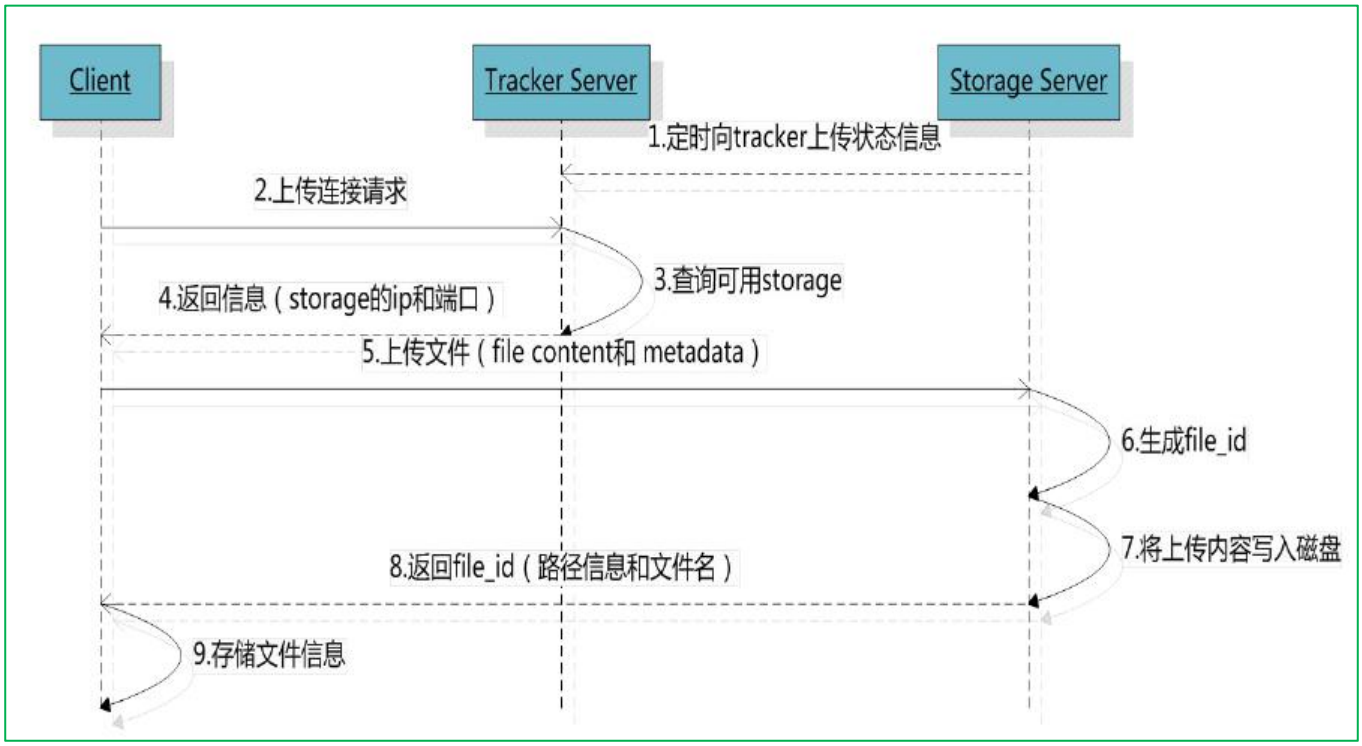
储服务器组成，组内的 Storage server 之间是平等关系，**不同组的 Storage server 之间不会相互通信，同组内的 Storage server 之间会相互连接进行文件同步，从而保证同组内每个 storage 上的文件完全一致的。一个组的存储容量为该组内存储服务器容量最小的那个，集群存储总容量为集群中所有组的存储容量之和**，由此可见组内存储服务器的软硬件配置最好是一致的。

采用分组存储方式的好处是灵活、可控性较强。比如上传文件时，可以由客户端直接指定上传到的组也可以由 tracker 进行调度选择。一个分组的存储服务器访问压力较大时，可以在该组增加存储服务器来扩充服务能力（**纵向扩容**）。当系统容量不足时，可以增加组来扩充存储容量（**横向扩容**）。

4）Storage 状态收集

Storage server 会连接集群中所有的 Tracker server，定时向他们报告自己的状态，包括磁盘剩余空间、文件同步状况、文件上传下载次数等统计信息。

5）文件上传流程

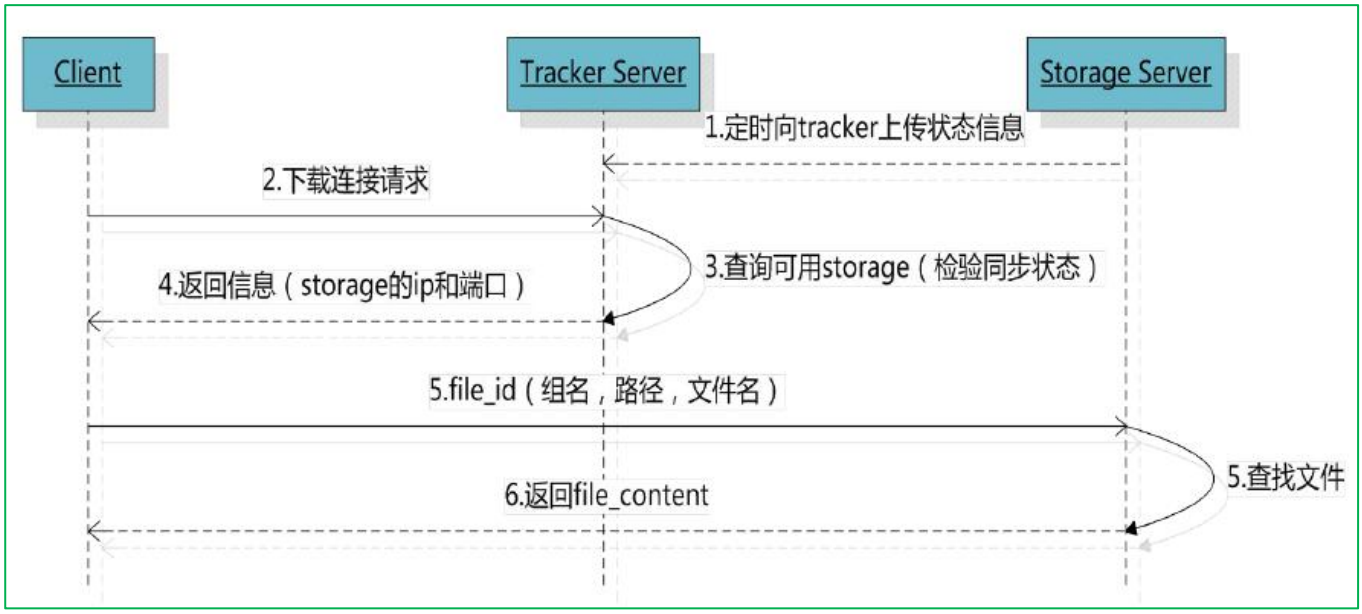


客户端上传文件后存储服务器将文件 ID 返回给客户端，此文件 ID 用于以后访问该文件的索引信息。文件索引信息包括：组名，虚拟磁盘路径，数据两级目录，文件名。

group1 /M00 /02/44/ wKgDrE34E8wAAAAAAAAGkEIYJK42378.sh

- **组名**：文件上传后所在的 storage 组名称，在文件上传成功后有 storage 服务器返回，需要客户端自行保存。
- **虚拟磁盘路径**：storage 配置的虚拟路径，与磁盘选项 store_path*对应。如果配置了 store_path0 则是 M00，如果配置了 store_path1 则是 M01，以此类推。
- **数据两级目录**：storage 服务器在每个虚拟磁盘路径下创建的两级目录，用于存储数据文件。
- **文件名**：与文件上传时不同。是由存储服务器根据特定信息生成，文件名包含：源存储服务器 IP 地址、文件创建时间戳、文件大小、随机数和文件拓展名等信息。

6）文件下载流程



tracker 根据请求的文件路径即文件 ID 来快速定义文件。
比如请求下边的文件：

group1 /M00 /02/44/ wKgDrE34E8wAAAAAAAAGkEIYJK42378.sh

1.通过组名 tracker 能够很快的定位到客户端需要访问的存储服务器组是 group1，并选择合适的存储服务器提供客户端访问。
2.存储服务器根据“文件存储虚拟磁盘路径”和“数据文件两级目录”可以很快定位到文件所在目录，并根据文件名找到客户端需要访问的文件。

10.KindEditor 富文本编辑器的使用

第一步：将 KindEditor 下载下来并放到项目中
第二步：在页面中添加 KindEditor 的 js 和 css 的引用

item-add.jsp

```
1 <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
2 <link href="/js/kindeditor-4.1.10/themes/default/default.css" type="text/css" rel="stylesheet">
3 <script type="text/javascript" charset="utf-8" src="/js/kindeditor-4.1.10/kindeditor-all-min.js"></sc
4 <script type="text/javascript" charset="utf-8" src="/js/kindeditor-4.1.10/lang/zh_CN.js"></script>
5 <div style="padding:10px 10px 10px 10px">
6     <form id="itemAddForm" class="itemForm" method="post">
7         <table cellpadding="5">
8             <tr>
```

第三步：定义 textarea 控件，富文本编辑器都是基于 textarea 控件实现的
注意：textarea 要设置不可见、给定 name 值

```
<td>商品描述:</td>
<td>
    <textarea style="width:800px;height:300px;visibility:hidden;" name="desc"></textarea>
</td>
```

第四步：使用 KindEditor 创建富文本编辑器

11.Redis

(1)Redis 安装步骤

第一步：在 Linux 下安装 gcc 环境
第二步：将下载的 Redis 源码包 redis-3.0.0.tar.gz 上传到 Linux 服务器中，并解压缩
第三步：执行 make 命令，编译 redis 源码。
第四步：安装 redis

(2)缓存使用分析

①确定要存储的数据是什么？
②确定要存储到 redis 中的数据类型是什么？

a) **Redis 的数据类型有 string、hash**、list、set、zset，项目中一般使用 string 和 hash 类型进行存储。
b) **注意：redis 中的数据主要用来查询时，选用 string 或者 hash 都可以。但是如果 redis 中的数据主要用来增删改时，选用 hash 类型。**

③在使用缓存的业务中添加通用缓存逻辑

c) 查询数据时，先调用 JedisCluster 查询缓存有没有该数据？
d) 缓存中没有相应数据，再去查询数据库，然后将结果存入到缓存并返回。
e) 缓存中有相应数据，从缓存中取出数据并返回。

(3)Redis 和 Memcached 的区别

Redis和Memcached的区别

Redis的作者Salvatore Sanfilippo曾经对这两种基于内存的数据存储系统进行过比较：

1. Redis支持服务器端的数据操作：Redis相比Memcached来说，拥有更多的数据结构和并支持更丰富的数据操作，通常在Memcached里，你需要将数据拿到客户端来进行类似的修改再set回去。这大大增加了网络IO的次数和数据体积。在Redis中，这些复杂的操作通常和一般的GET/SET一样高效。所以，如果需要缓存能够支持更复杂的结构和操作，那么Redis会是不错的选择。
2. 内存使用效率对比：使用简单的key-value存储的话，Memcached的内存利用率更高，而如果Redis采用hash结构来做key-value存储，由于其组合式的压缩，其内存利用率会高于Memcached。
3. 性能对比：由于Redis只使用单核，而Memcached可以使用多核，所以平均每一个核上Redis在存储小数据时比Memcached性能更高。而在100k以上的数据中，Memcached性能要高于Redis，虽然Redis最近也在存储大数据的性能上进行优化，但是比起Memcached，还是稍有逊色。

(4)Redis 的认证

①修改配置文件

Redis 的配置文件默认在/etc/redis.conf，找到如下行：

#requirepass foobared 去掉前面的注释，并修改为所需要的密码：

②设置 Redis 认证密码后，客户端登录时需要**使用-a 参数输入认证密码**，不添加该参数虽然也可以登录成功，但是没有任何操作权限。

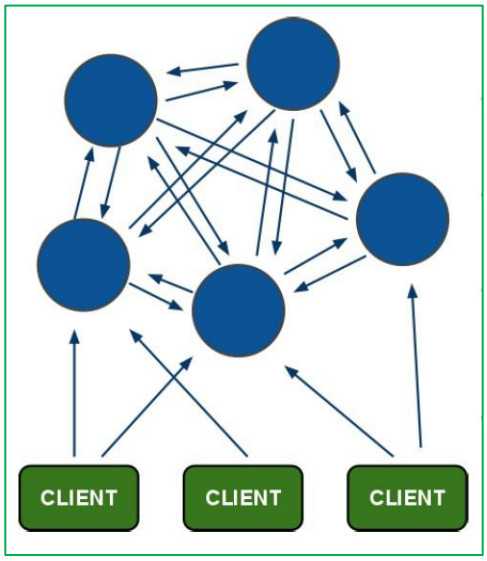
③在 Redis 集群中使用认证密码

如果 Redis 服务器，使用了集群。除了在 master 中配置密码外，也需要在 slave 中进行相应配置。在 slave 的配置文件中找到如下行，去掉注释并修改与 master 相同的密码即可：

```
# masterauth master-password
```

12.Redis-Cluster（Redis 集群）

(1)redis-cluster 架构图



架构细节:

(1)所有的 redis 节点彼此互联(**PING-PONG 机制**),内部使用二进制协议优化传输速度和带宽.

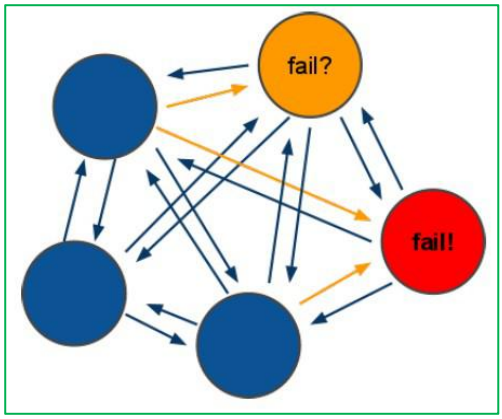
(2)节点的 fail 是通过**集群中超过半数的节点检测失效**时才生效.

(3)客户端与 redis 节点直连,不需要中间 proxy 层.客户端不需要连接集群所有节点,**连接集群中任何一个可用节点即可**

(4)redis-cluster 把所有的物理节点映射到[0-16383]slot 上,cluster 负责维护 node<->slot<->value

Redis 集群中内置了 **16384** 个哈希槽，当需要在 Redis 集群中放置一个 key-value 时，redis 先对 key 使用 crc16 算法算出一个结果，然后把结果对 16384 求余数，这样每个 key 都会对应一个编号在 0-16383 之间的哈希槽，redis 会根据节点数量大致均等的将哈希槽映射到不同的节点

(2)redis-cluster 投票:容错



(1)集群中所有 master 参与投票,如果**半数以上 master 节点与其中一个 master 节点通信超过(cluster-node-timeout),认为该 master 节点挂掉**.

(2):什么时候整个集群不可用(cluster_state:fail)?

- 如果集群**任意 master 挂掉,且当前 master 没有 slave**，则集群进入 fail 状态。也可以理解成集群的[0-16383]slot 映射不完全时进入 fail 状态。
- 如果集群**超过半数以上 master 挂掉**，无论是否有 slave，集群进入 fail 状态。

(3)安装步骤

第一步：搭建 Ruby 环境；
第二步：创建一个 redis 实例，需要修改 redis.conf 文件
①修改 port 为 7001
②修改 daemonize 为 yes
③修改 cluster-enable 为 yes
第三步：复制 6 分，然后修改 port 端口即可。
第四步：编辑一个启动脚本，启动所有的 redis 实例。

13.导入索引库

(1)业务逻辑分析

页面分析：
显示功能页面请求 URL：/system/index-import
显示功能页面请求返回值：String （导入页面的逻辑视图名称 index-import）
①**确定要导入索引库的数据**：全部商品数据
②**需要索引的字段**：商品 ID、商品名称、商品卖点、商品价格、商品分类名称、商品描述、商品图片。**需**

要定义 solr 的业务域。

③根据要索引的商品字段，定义出查询商品的 sql 语句。在索引字段中，商品分类名称不属于商品表，所以查询商品数据时，需要关联商品分类表。逆向工程代码不能实现，需要单独开发持久层代码，并在 TbItemExt 类中添加 catName 属性。

表现层分析：

①导入索引库的请求 URL：/index/importAll

②导入索引库的请求返回值：TaotaoResult，导入成功后，返回成功失败状态。

业务逻辑分析：

①查询所有商品信息。

②将查询出的商品数据，通过 solrj 导入到索引库中。

(2)Solr

1）Solr 简介

Solr 的目标是打造一款企业级的搜索引擎系统，它是基于 Lucene 一个搜索引擎服务，可以独立运行；而 Lucene 是一个开放源代码的全文检索引擎工具包，它不是一个完整的全文检索应用。

2）Solr 的使用

```
// 1、 根据sql查询商品数据      List<TbItemExt> list = mapper.queryItemList();
// 2、 创建SolrServer    @Autowired      private SolrServer solrServer;
// 3、 将查询结果数据，转成SolrInputDocument对象
        List<SolrInputDocument> docs = new ArrayList<>();
        SolrInputDocument doc;
        for (TbItemExt item : list) {
            doc = new SolrInputDocument();
            doc.addField("id", item.getId());
            .....
            docs.add(doc);
        }
// 4、 调用solrServer将SolrInputDocument对象添加到索引库     solrServer.add(docs);
// 5、 提交    solrServer.commit();
```

3）Solr 安装配置（单机版安装）

①创建 solrhome（里面有一个 collection1 的 solrcore）

②修改 solr.war

a)添加 example/lib/ext 目录下的 jar 包到 solr/WEB-INF/lib 目录下

b)添加 example/resource 目录下的文件到 solr/WEB-INF/classes 目录下

c)在 web.xml 中配置 solrhome 和 solr.war 的关联关系

③配置中文分词器 Ikanalyzer

a)添加 Ikanalyzer 的 jar 包到 solr/WEB-INF/lib 目录下

b)添加 Ikanalyzer 的配置文件和扩展文件到 solr/WEB-INF/classes 目录下

c)在 shema.xml 文件中，配置 FieldType，指定分词器为 Ikanalyzer

④商品数据导入索引库（全量）

a)索引库中存储的数据是全部商品数据

b)索引库中是通过 Field 来存储数据的，但是 solr 中的 Field 必须先定义后使用。

<field name="item_title" type="text_ik" indexed="true" stored="true"/>

⑤Solr 如何实现搜索结果排名靠前？

Lucene 对查询关键字和索引文档的相关度进行打分，得分高的就排在前边。而相关度主要依赖权重，Term（索引的最小单位）对文档的重要性称为权重；也可手动设置加权：textField.setBoost(100f);

影响 Term 权重有两个因素：一个是 **TF**，一个是 **DF**：

TF 指此 Term 在此文档中出现了多少次，tf 越大说明越重要;

DF 指有多少文档包含此 Term，df 越大说明越不重要。

14.索引同步

(1)业务逻辑

1）索引分页展示逻辑

①创建 SolrQuery 对象，设置查询条件、分页、默认域、高亮等属性。

②通过 SolrServer 执行 SolrQuery 对象，进行搜索。

③对搜索结果进行高亮处理

④计算总页数 totalPages

⑤将 itemList、totalPages、totalRecords 封装到 Map<String,Object>中并返回。

经分析，服务层只需要调用 solrj 实现搜索，不需要调用持久层代码。

2）索引同步逻辑

①需要保证更新索引操作时，不能影响原有业务，所以使用 activemq 消息中间件进行异步通信。

②消息发送，选用发布\订阅模式，这种消息可以有多个消费者接收消息。因为在商品信息变更时，还需要进行缓存同步等其他操作。

具体操作：

Ⅰ）在 taotao-manager-service 工程中的添加商品方法里面添加**发送商品 ID 消息**逻辑。

a) 消息目标：Topic。

b) 消息内容：商品 ID，使用 TextMessage

Ⅱ）在 taotao-search-service 工程中**接收消息**

a)定义一个监听器接收信息

◆接收商品 ID

◆根据商品 ID 查询要索引的商品数据，包含商品分类名称，需要开发 mapper

◆将查询出的数据转成 SolrInputDocument 对象，然后调用 SolrServer 添加到索引库。

b) 在 activemq 的配置文件中配置接收消息的 Destination、MessageListener、MessageListenerContainer。

3）添加商品信息缓存逻辑

1、 分析要缓存的数据：商品数据

2、 分析存储到 Redis 中的类型

使用 string 和 hash 存储该数据都可以。

String

key	value
item:id	{id:1,price:999,title:'大电视'}

Hash

Key	field	value
item	id	{id:1,price:999,title:'大电视'}

a) 考虑到商品数据量太大，全部长时间放到缓存中不合适，所以需要对缓存的商品数据设置有效期，操作方法就是针对 Redis 中的 key 设置 expire。在这种情况下，每个商品都需要一个单独的 key。

```
cluster.expire(REDIS_KEY_ITEM_PRE + itemId, REDIS_ITEM_EXPIRE_SECOND);
```

cluster.expire(REDIS_KEY_ITEM_PRE + itemId, REDIS_ITEM_EXPIRE_SECOND);

b) **建议使用 String 类型**存储商品数据。

c) key 就是商品 ID。不过用商品 ID 作为 key，可能会存在冲突，解决该问题的办法是 key 中定一个前缀，比如[item：商品 ID]

d) value 就是 json 格式的商品数据。

3、 在 taotao-manager-service 工程的添加商品服务中，添加缓存逻辑

a) 查询商品时，先根据商品 ID 查询 redis 缓存

b) 如果缓存中有数据，则将数据取出，转换成 TbItem 对象返回。

c) 如果缓存中没有数据，则根据商品 ID 查询数据库。然后将查询结果转成 json 格式的数据，存入 redis 缓存。

(2)SolrCloud

1)SolrCloud 简介

①SolrCloud 概述

SolrCloud(solr 云)是 Solr 提供的基于 Solr 和 Zookeeper 的分布式搜索方案，查询时自动负载均衡。

SolrCloud 的主要思想是使用 Zookeeper 作为 SolrCloud 集群的配置信息中心，统一管理 solrcloud 的配置，比 如 solrconfig.xml 和 schema.xml 等配置。

②应用场景：

a) 大规模，容错，分布式索引和检索能力时使用 SolrCloud。

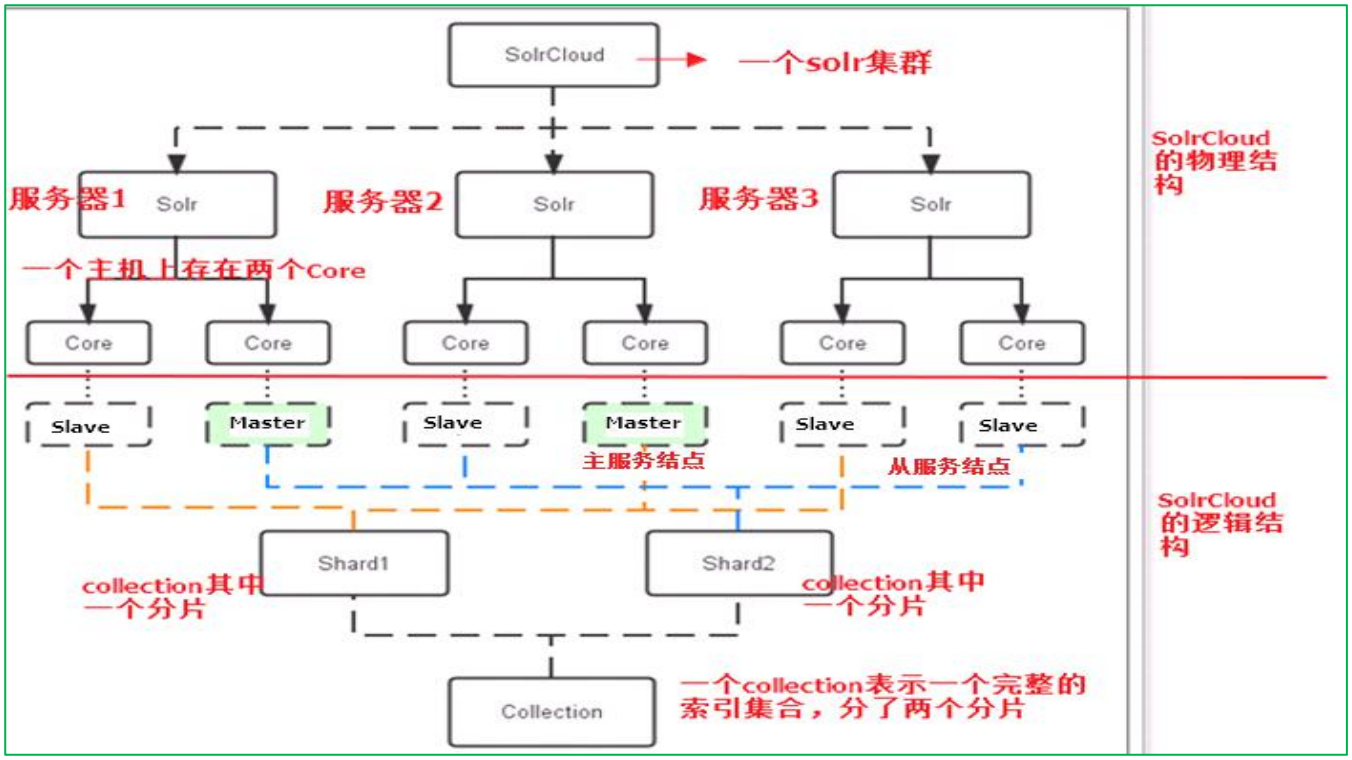
b) 索引量很大，搜索请求并发很高时，同样需要使用 SolrCloud 来满足这些需求。

2)Solrcloud 结构

solrcloud 为了降低单机的处理压力，需要由多台服务器共同来完成索引和搜索任务。实现的思路是**将索引数据进行 Shard 分片**，每个分片由多台服务器共同完成，当一个索引或搜索请求过来时会分别从不同的 Shard 的服务器中操作索引。

solrcloud 是基于 solr 和 zookeeper 部署，zookeeper 是一个集群管理软件，solrcloud 需要由多台 solr 服务器组成，然后由 zookeeper 来进行协调管理。

下图是一个 SolrCloud 应用的例子：



从逻辑结构来说:

a) 整个 solrcloud 就看成一个大的 solrcore，也就是一个 collection。而一个 collection 被分成两个 shard 分片（shard1 和 shard2）。

b) shard1 和 shard2 又分别由三个 solrcore 组成，其中一个 Leader 两个 Replication。Leader 是由 zookeeper 选举产生，zookeeper 控制每个 shard 上三个 Core 的索引数据一致，解决高可用问题。

c) 用户发起索引请求分别从 shard1 和 shard2 上获取，解决高并发问题。

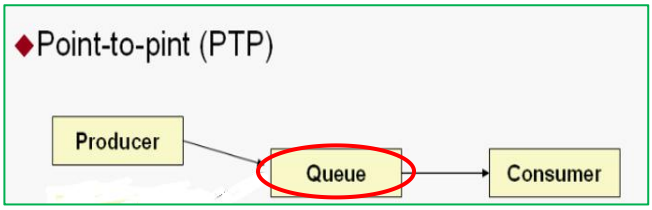
3)Solrcloud 搭建

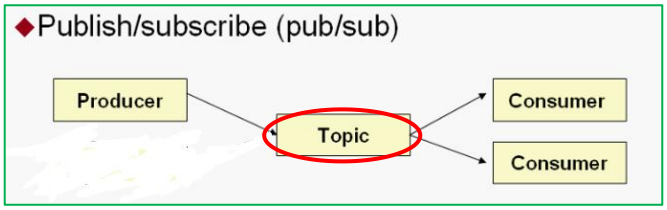
solrcloud 是通过 zookeeper 统一管理配置文件(solrconfig.xml、schema.xml)，所以搭建 solrcloud 之前，需要先搭建 zookeeper。由于 solrcloud 一般都是解决大数据量、大并发的搜索服务，所以搭建 solrcloud，对 zookeeper 也需要搭建集群。其核心是在 web.xml 配置 tomcat 和 solrhome 的关系。

(3)ActiveMQ

1)ActiveMQ 简介

ActiveMQ 是 Apache 出品的，按照 **JMS**（Java Message Service）规范实现的**消息中间件**。它主要是让**生产者生产消息之后，先发送给消息中间件**，然后**消息消费者去消息中间件进行获取**，发送和接收的方式有两种：**点对点**、**发布\订阅**模式。消息格式一般使用 **TextMessage**。





2)ActiveMQ 使用

a) Producer

1、 创建 ConnectionFactory。
2、 通过 ConnectionFactory 创建 Connection
3、 启动 Connection
4、 通过 Connection 创建 Session
5、 通过 Session 创建 Destination（目的地），目的地有两种：**Topic**、**Queue**
6、 通过 Session 创建 Producer
7、 通过 Session 创建 Message
8、 通过 Producer 发送 Message
9、 释放资源（Producer、Session、Connection）。

b) Consumer

1、 创建 ConnectionFactory
2、 通过 ConnectionFactory 创建 Connection
3、 启动 Connection
4、 通过 Connection 创建 Session
5、 通过 Session 创建 Destination：**Topic**、**Queue**
6、 通过 Session 创建 Consumer
7、 调用 Consumer 的接收消息（receive 方法或者 MessageListener 接口）
8、 处理消息
9、 关闭资源（Consumer、Session、Connection）。

c) 与 Spring 整合

生产者：

```
        <!-- Spring 提供的 JMS 工具类 JmsTemplate，它可以进行消息发送、接收等 -->
        <!--这个是队列目的地，点对点的 ActiveMQQueue -->
        <!--这个是主题目的地，一对多的 ActiveMQTopic -->
@Autowired
private JmsTemplate JmsTemplate;
@Resource(name = "itemTopic")
private Destination destination;
......
// 通过 ActiveMQ 发送消息
        JmsTemplate.send(destination, new MessageCreator() {

            @Override
            public Message createMessage(Session session) throws JMSException {
                // 将商品 ID 发送出去
                TextMessage message = session.createTextMessage(itemId + "");
                return message;
            }
        });
```

消费者：

```
    <!-- 配置 ActiveMQQueue 这个是队列目的地，点对点的 -->
    <!-- 配置 MessageListener -->
    <!-- 配置 DefaultMessageListenerContainer -->
```

```
public class ItemMessageListener implements MessageListener {
    @Autowired
    private SolrServer solrServer;
    @Autowired
    private TbItemExtMapper itemExtMapper;
    @Override
    public void onMessage(Message message) {
        try {
            if (message instanceof TextMessage) {
                TextMessage mqTextMessage = (TextMessage) message;
                // 获取商品 ID
                Long itemId = Long.parseLong(mqTextMessage.getText());
                // 根据商品 ID 查询数据库中的商品信息
                TbItemExt item = itemExtMapper.queryItemById(itemId);
                // 将商品数据封装到 SolrInputDocument 对象
                SolrInputDocument doc = new SolrInputDocument();
                doc.addField("id", itemId);
                .....
                // 添加到索引库
                solrServer.add(doc);
                // 提交
                solrServer.commit();
```

```
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

15.页面静态化

(1)业务逻辑

1、在 service 工程的添加商品服务中发送[商品 ID]消息。

2、在 web 工程中创建监听器接收[商品 ID]消息

a)接收消息中的商品 ID。

b)从 spring 上下文中获取 Freemarker 的 **configuration** 对象。

c)通过 configuration **创建 template** 对象，模板的名称在 java 配置文件中配置。

d)创建模板数据：根据商品 ID 查询出商品信息，如果没有数据，抛出异常。

e)指定输出文本。文本的路径在 java 配置文件中配置，文本的名称是商品 ID，文本的后缀在 java 配置文件中配置。

f)调用 **template.process(map, writer)方法，生成输出文件**。

(2)代码实现

```
<!-- 配置 Configuration -->
<bean class="org.springframework.web.servlet.view.freemarker.FreeMarkerConfigurer">
    <!-- 配置模板文件路径 -->
    <property name="templateLoaderPath" value="classpath:ftl/"></property>
    <!-- 指定模板的默认编码 -->
    <property name="defaultEncoding" value="UTF-8"></property>
</bean>
```

```
@Autowired
private FreeMarkerConfigurer configurer;
// a) 获取 Freemarker 的配置文件
Configuration configuration = configurer.getConfiguration();
// c) 通过 configuration 创建 template 对象，模板的名称在 java 配置文件中配置。
Template template = configuration.getTemplate(ITEM_TEMPLATE_NAME);
// d) 创建模板数据：根据商品 ID 查询出商品信息
TbItem item = service.queryItemById(itemId);
TbItemExt itemExt = new TbItemExt();
BeanUtils.copyProperties(item, itemExt);
Map<String, Object> map = new HashMap<>();
map.put("item", itemExt);
// e) 指定输出文本
String htmlPath = ITEM_HTML_PATH_PRE + itemId + ITEM_HTML_PATH_EXT;
Writer writer = new FileWriter(new File(htmlPath));
// f) 调用 template 的 process 方法，生成输出文件。
template.process(map, writer);
// 关闭 writer
writer.close();
```

(3)为什么要实现伪静态化？

①提高用户对主页的信任度；
②最主要的是为了搜索引擎方便搜索引擎蜘蛛（Spider）来抓取主页上的相关内容；
③由于静态地址比动态地址权重高，可以影响网站在搜索引擎中的排名；

(4)Freemarker

1)FreeMarker 概述

FreeMarker 是一个用 Java 语言编写的模板引擎，它基于模板来生成文本并输出。

FreeMarker 不仅可以用作表现层的实现技术，类似于 JSP 的功能一样，而且还可以用于生成 XML，JSP 或 Java 源代码等。

2）使用方法

① 创建 ftl 模板
② 编写代码
   a) 创建 Configuration 对象
      i. 设置模板路径
      ii. 设置模板默认编码
   b) 通过 Configuration 对象获取模板
   c) 创建模型数据
   d) 创建 Writer，指定输出文件路径
   e) 调用模板的 process 方法，生成文件
   f) 释放资源

3）模板介绍

① 模板包括：文本、注释、**插值、ftl 标签**。
② 插值语法（类似于 jsp 中的 el 表达式）
   g) 读取简单类型，命令${key}
   h) 读取 pojo 类型，命令${pojo.field}
   i) 读取日期类型，命令${变量名?内置函数}，其中内置函数包含 date、time、datetime、string(“yyyy-MM-dd HH24:mm:ss”)
   j) Null 值，命令${变量名!默认值}
③ Ftl 标签（类似于 jsp 中的 c 标签）
   k) 循环，命令<#list 集合变量名 as 声明集合内元素的变量名></#list>
   l) 循环下标，命令${声明集合内元素的变量名_index}
   m) 判断，命令<#if 判断表达式></#if>
   n) 包含文件，命令<#include “被包含的文件”></#include>

16.登录功能

(1)登录功能分析

http 协议是无状态的，那么怎么记录浏览器的登录状态呢？
实际上是通过 Session 和 cookie 技术实现登录状态的：

1、 通过浏览器，访问服务器。服务器会针对这个浏览器请求，产生一个 session 对象。该 session 对象会产生一个 jsessionid，该 jsessionid 会随着请求响应会浏览器
2、 浏览器会将请求响应中的 jsessionid，记录到浏览器中的 cookie 里面
3、 下一次，通过该浏览器，访问服务器，会带着该 cookie 中的 jsessionid 去访问，服务器根据该 jsessionid，查找对应的 session 对象。
4、 登录操作步骤，将用户名和密码，存储到一个 session 中（该浏览器产生的 session。）

- 存在问题：**session 分离**。

集群环境下登录，用户信息只会保存在第一次登录的服务器中，其他服务器的 session 中没有该用户的登录信息。

- 解决方法：**session 共享**。一般来说 session 共享有两种方案。
  - 第一种：**使用 Tomcat 自身的同步机制复制 session 到集群中的其他服务器**。Tomcat 的同步机制是广播形式，也就是说集群中所有 Tomcat 都是相同的 session 信息。但是这种情况下，Tomcat 越多，效率越低。而官方给出的最佳集群数量是 5 台 tomcat。

  - 第二种：**使用缓存服务器集中存储 session 信息达到 session 共享需求**。不管在哪台服务器登录成功之后，都会将 session 信息保存在统一的 Redis 缓存服务器中。第二次登录时，查询 Redis 缓存服务器，获取用户登录信息。

- 存在问题：**切换不同系统需要再次登录。**

系统进行分布式部署时，对于商品系统、用户中心、订单系统会员只需登录一次，但是 session 只会存储到商城系统，其他系统访问时需要再次登录。

- 解决办法：**单点登录**

  单点登录有两种方式：

  - 第一种方式：**使用 token 登录令牌**

系统 1 登录成功后，产生一个 token 令牌，该令牌会被保存到 cookie 中和数据库中，从系统 1 的页面直接访问系统 2 的功能时，需要先从 cookie 中取出 token 令牌，然后带着该令牌访问系统 2 的登录认证功能，如果令牌和第一次登录存储到数据库的令牌可以匹配成功，则理解为该用户在系统 2 也登录成功。

- 存在问题：**该模式，需要每套系统中都需要开发验证 token 的逻辑，麻烦。**
  - 第二种方式：**开发 SSO 单点登录系统**

  所有的系统访问一个共同的登录系统，这样就能解决分布式系统登录问题。

(2)业务逻辑

1、 根据用户名查询用户信息
2、 如果用户信息为空，则登录不成功
3、 如果用户信息不为空，则将页面输入的密码进行加密之后与查询出的密码进行匹配
4、 如果为 false，则登录不成功
5、 如果为 true，则表示登录成功。需要将该用户信息写入模拟的 session 中。如何将该模拟 session 放入 redis 缓存中。Redis 中的类型使用 hash 类型。Key 为 session 的唯一标示，value 为 map 结构数据，其中一条记录是该用户信息记录。
   a) 模拟 session 的 sessionId，声明 token 值，使用 UUID 生成
   b) 模拟 session 的有效期，对 redis 中的 key 设置 expire 即可。
6、 返回 TaotaoResult，需要封装 token 值

(3)Redis 存储类型分析

考虑使用 string 还是 hash 存储 session 信息?
要考虑是查询多还是增删改多？

登录分析：Session 和 cookie 的作用
原先
1、 Session 中的数据结构就是一种 hashmap 结构，所以使用 hash 类型存储会更方便。
2、 Session 会产生一个唯一标示 jsessionid。
3、 Session 可以设置过期时间。
session.setAttribute(“user1”, “111”)

session.setAttribute(“user2”, “222”)

结果：

Jsessionid----------session（类似于 map），map 有多对 key-value 值，map 在 redis 中的存储结构，应该是对应 hash 类型

Key----------Field----------value

jsessionid	登录用户信息	{id:1,name:zhangsan}
jsessionid	权限信息	{id:2,name:zhangsan2}

(3)商城系统显示用户名称

1、 根据 token 去 redis 中查询该用户登录信息
2、 如果没有数据，则登录不成功，返回登录不成功。
3、 如果有数据，则登录成功，重新设置 session 的期限
4、将数据转成 TbUser 对象，封装到 TaotaoResult 中返回。

(4)出现的问题—JS 跨域

1）什么是 JS 跨域

跨域问题是由于 javascript 语言安全限制中造成的。

简单来说,JavaScript 语言不允许 ajax 跨域请求数据，这个跨域指的是**主机名、协议和端口号**的组合。

比如：

1、 域名相同、端口不同，比如：www.liyan.com:8081 和 www.liyan.com:8082
2、 域名不同、端口相同，比如：www.liyan.com 和 www.search.com

JS 不支持跨域请求 XMLHttpRequest 数据，即通过 JS 访问一个跨域的服务，是可以访问到该服务的，但是返回结果无法传递给跨域访问的调用者。

但是，JS 支持跨域访问并加载 JS 脚本，即使用<script>标签加载其他域的另一段 JS 文件到本 JS 文件中。

2）使用 jsonp 解决跨域

Jquery 提供了 jsonp 访问格式来解决 js 跨域问题。

Jsonp 解决跨域请求的思路：

①jquery 设置 dataType 为 jsonp，会发送一个 key 为 callback 的请求参数到后台。callback 的值是由 jquery 生成的随机 js 方法名称。

②jquery 设置 dataType 为 jsonp，则返回的数据会被解析成一段 js 脚本。比如：

<script type="text/javascript">callback(“{id:1,name:"zhangsan"}”) </script>

③页面定义一个回调函数，比如叫 function callback(data){}方法。

④return callback+”(“+**{status:200}**+”)”;页面得到返回的 js 脚本之后，会立即执行该 js 方法。而返回的数据，就可以在该 js 方法中获取并进行业务处理。

```
@RequestMapping("/token/{token}")
@ResponseBody
public String doCheck(@PathVariable String token, String callback) {
    // return " <script >callback({id:1,name:"zhangsan"}) </script>";
    TaotaoResult result = service. doCheck (token);
    if (StringUtils.isEmpty(callback)) {
```

```
            return JsonUtils.objectToJson(result);
        }
        // 需要跨域处理  callback("{id:1,name:'zhangsan'}")
        String func = callback + "(" + JsonUtils.objectToJson(result) + ")";
        return func;
    }
```

17.购物车功能

购买商品前，需要将商品先添加到购物车中。然后进入购物车页面，查看购物车中商品是否是自己选购的，可以对商品的数量进行修改，也可以删除购物车商品。

(1)购物车实现方案

1、 Cookie
   a) 优点：简单
   b) 缺点：依赖于浏览器本身的 cookie 禁用问题、信息会在浏览器中看到。
2、 Session
   a) 优点：数据安全
   b) 缺点：应用服务器会存在大量的 session，每个 session 中会存在大量的购物车信息。
3、 数据库
   a) 优点：数据可以持久化，并且安全
   b) 缺点：频繁访问数据库，响应速度相对来说，会比较慢
4、 **Redis**
   a) 优点：数据安全、访问响应速度快
   b) 缺点：存储空间有限
5、**登录前使用 Cookie，登录之后使用 Redis；（推荐）**

淘淘商城现阶段使用的仅仅是把购物车的商品写入 cookie 中，这样服务端基本上么有存储的压力。但是弊端就是用户更换电脑后购物车不能同步。打算下一步这么实现：当用户没有登录时向购物车添加商品是添加到 cookie 中，当用户登录后购物车的信息是存储在 redis 中的并且是跟用户 id 向关联的，此时你更换电脑后使用同一账号登录购物车的信息就会展示出来。

(2)添加购物车逻辑

表现层逻辑：

1、 使用 CookieUtils 工具类从 cookie 中取出购物车中商品列表信息(json 格式字符串)。
2、 调用 service 将页面传入的商品添加到从 cookie 中取出的商品列表中，并返回添加后的商品集合。
3、 使用 CookieUtils 工具类将添加后的商品集合存入 cookie 中。

业务逻辑：

1、 判断 cookie 中的购物车商品列表是否为空
2、 如果为空，则创建新的购物车商品集合
3、 如果不为空，则判断购物车商品列表中是否有新添加的商品
4、 如果有，则只需将 num 值追加。
5、 如果没有，需要根据商品 ID 查询数据库中的商品信息。
6、 将查询出的商品信息放到购物车商品集合中，并设置 num 值。
7、 返回购物车商品列表。

(3)修改购物车逻辑

表现层逻辑：

1、 使用 CookieUtils 工具类从 cookie 中取出购物车中商品列表信息(json 格式字符串)。
2、 调用 service 修改购物车中商品的数量，并返回修改后的商品集合。

3、 使用 CookieUtils 工具类将添加后的商品集合存入 cookie 中。

业务逻辑：

1、 遍历购物车商品集合

2、 将指定商品 ID 的商品 num 值改为页面传入的值

3、 返回修改后的购物车商品集合。

(4)删除购物车逻辑

表现层逻辑：

1、 使用 CookieUtils 工具类从 cookie 中取出购物车中商品列表信息(json 格式字符串)。

2、 调用 service 删除购物车中指定的商品，并返回修改后的商品集合。

3、 使用 CookieUtils 工具类将添加后的商品集合存入 cookie 中。

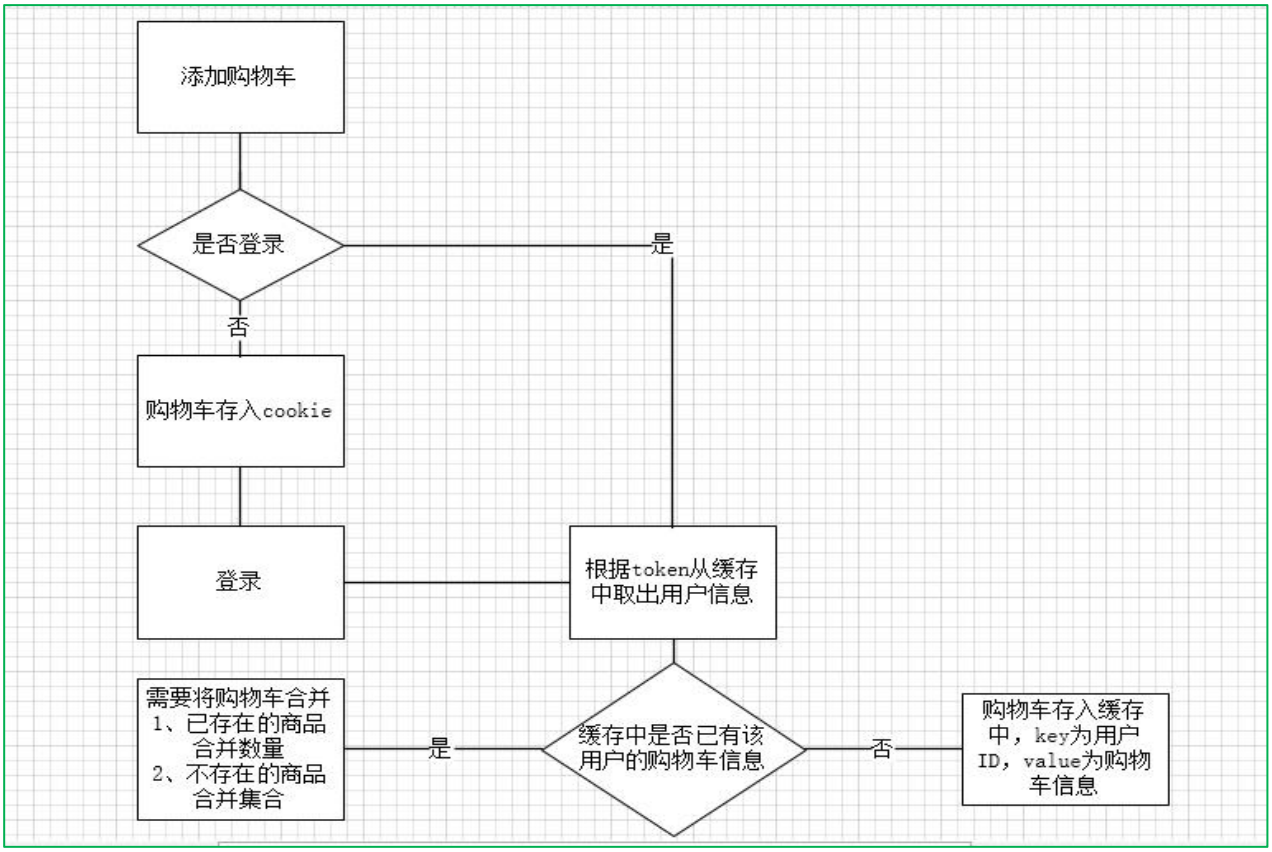
业务逻辑：

1、 遍历购物车商品集合

2、 删除指定商品 ID 对应的商品

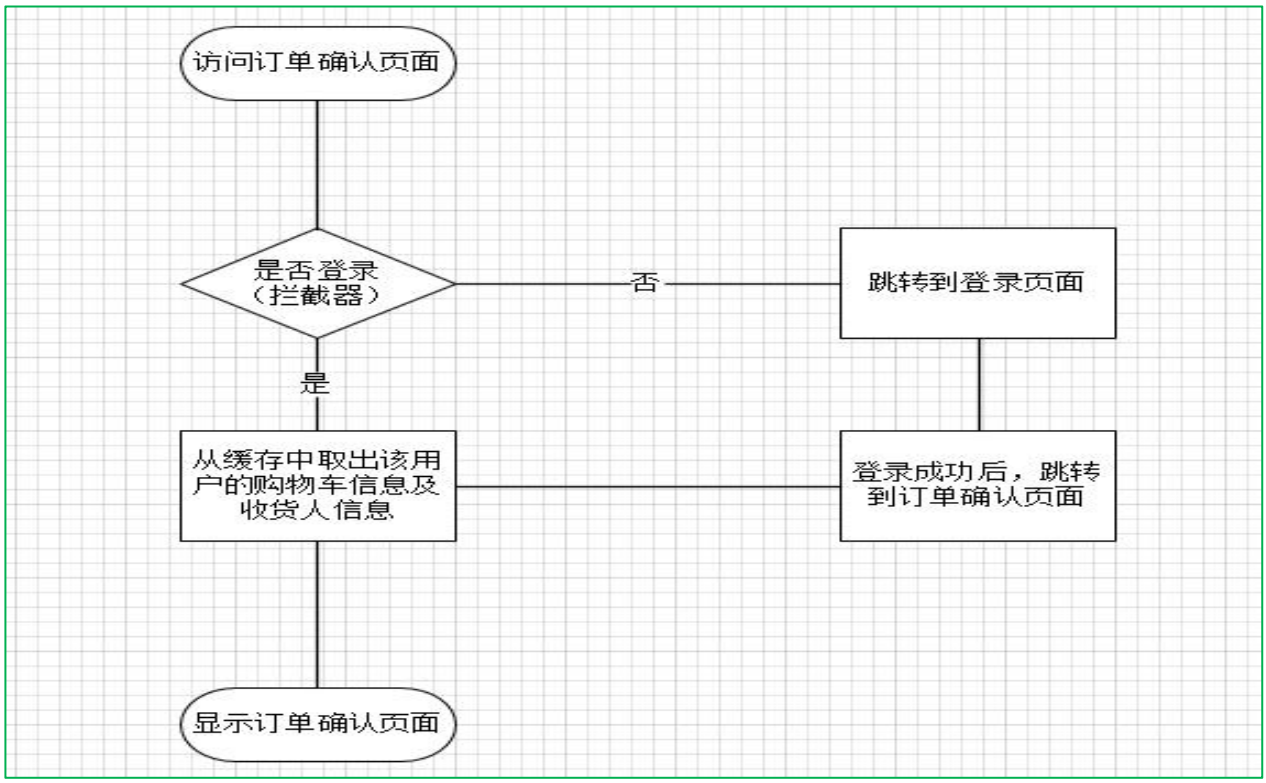
3、 返回修改后的购物车商品集合

(5)购物车同步逻辑



18.订单功能

(1)下单功能实现原理



(2)下单功能业务逻辑

表现层逻辑：

1、 使用拦截器进行登录拦截。

2、 从 cookie 中获取购物车信息，然后放到 request 域中。

3、 从 session 中获取用户信息，根据用户 ID 获取用户收货地址信息（未实现，模拟即可）

拦截器逻辑：

1. 从 cookie 中取出登录用户的 token。
2. 如果为空，则没有登录，跳转到 SSO 登录页面，同时带有订单确认页面的重定向 URL。
3. 如果不为空，根据 token 调用 sso 的服务取出用户信息。
4. 如果用户信息为空，则登录超时，跳转到 SSO 登录页面，同时带有订单确认页面的重定向 URL。
5. 如果用户信息不为空，将用户信息放到 request 域中，并且放行，显示订单确认页面

(3)提交订单功能业务逻辑

1、 补全页面中传入的订单信息、订单明细信息、订单配送信息。

2、 插入 tb_order 订单表，需要返回订单 ID

3、 插入 tb_order_item **订单商品明细表**

4、 插入 tb_order_shipping **订单商品送货信息表**

5、 返回成功页面中，需要传入生成的订单 ID

19.其他功能

(1)电商活动倒计时方案

1、确定一个基准时间。可以使用一个 sql 语句从数据库中取出一个当前时间。SELECT NOW()；

2、活动开始的时间是固定的。

3、使用活动开始时间-基准时间可以计算出一个秒为单位的数值。

4、在 redis 中设置一个 key（活动开始标识）。设置 key 的过期时间为第三步计算出来的时间。

5、展示页面的时候取出 key 的有效时间。Ttl 命令。使用 js 倒计时。

6、一旦活动开始的 key 失效，说明活动开始。

7、需要在活动的逻辑中，先判断活动是否开始。

(2)秒杀方案：

1、把商品的数量放到 redis 中。

2、秒杀时使用 decr 命令对商品数量减一。如果不是负数说明抢到。

3、一旦返回数值是负数说明商品已售完。