技术点

技术点的逻辑主线就是

一、是什么？

1、定位/自己的理解/主要是用来干什么的

2、工作原理/执行流程/核心组件

3、项目中的应用（这点直接参考第三点，面试中可以加上）

二、 为什么用？

1、结合项目的具体应用场景去回答

2、和同类技术点的对比

三、 怎么用的？（这点是面试的重点，回答技术点的时候结合项目）

这个技术点的核心/机制/特性 在项目中的哪一块具体用到的

四、 用的过程中出现什么问题？

一、Dubbo

1、简单的介绍一下 Dubbo？(Dubbo 是什么)

【参考答案】

（1）Dubbo 是阿里开源的远程服务调用（RPC）的分布式框架，提供了 SOA 服务治理方案；（学生可以按照自己的理解来阐述）

（2）它的架构主要有五个角色/核心组件，分为是 Container（容器）、Provider（服务的提供方）、Registry（注册中心）、Consumer（服务的消费方）、Monitor（监控中心）。

容器主要负责启动、加载、运行服务提供者；

同时服务提供者在启动时，向注册中心注册自己提供的服务；

消费者向注册中心订阅自己的服务；

注册中心返回服务提供者列表给消费者，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；

对于服务消费者，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，再选另外一台调用；

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；

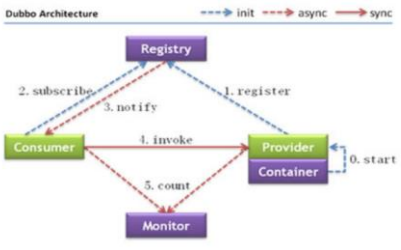


图 1-1

批注 [1]: RPC（Remote Procedure Call Protocol），通俗的说，就是两台服务器 A、B，应用 A 部署在 A 服务器上，应用 B 部署在 B 服务器上，A 应用想要调用 B 服务器上 B 应用提供的函数（方法），由于不在一个内存空间，不能直接调用，需要通过网络通讯、协议、寻址的方式来表达调用的语义和传达调用的数据。RPC 的亮点就是将远程调用的细节隐藏起来，使得调用远程服务像调用本地服务一样简单

批注 [2]: SOA（Service-Oriented Architecture）:面向服务的架构，也可以理解为资源调度和治理中心。所谓服务，拿项目二举个最简单的例子，把商品、订单、搜索等核心业务抽取出来，作为独立的服务。和面向对象来比较，面向对象的程序是由一个一个的类组成的，而面向服务的架构是由一个一个服务组成的。如果有多个系统、多个终端（PC、APP 等）都想调用我这个服务，那么就可以通过同一套协议和接口规范来调用。而服务越来越多，容量和资源利用率不相同的时候，我们可以增加一个调度中心基于访问的压力实时的管理集群容量，从而提高集群利用率，因此 Dubbo 就可以这中间起到的资源（服务）调度和治理中心。

批注 [3]: 这里的容器当 dubbo 和 spring 整合之后，可以理解为 spring 的容器

批注 [4]: 长连接：一个连接上可以连续的发送多个数据包，在连接保持期间，如果没有数据包发送，需要双方发检测包以维持此连接；比如：数据库的连接用长连接，聊天室，实时游戏

短连接：指通信的双方有数据交互时，建立一个连接，数据发送完成之后，则会断开连接。适用于网页浏览等数据刷新频度较低的场景。

批注 [5]: 【扩展】：Dubbo 提供的负载均衡策略有四种：

1、随机均衡算法：按权重设置随机概率，默认的

2、权重轮询均衡算法：按照公约后的权重设置轮询比率，能考虑到每台服务器的性能，所以，在实际应用中比较常见；

3、最少活跃调用数均衡算法

4、一致性 hash 均衡算法：相同参数的请求总是发到同一台提供者

这个可以自己了解，考查的概率不大

【扩展问题】

注册中心：

1、注册中心只负责地址的注册和查找，相当于我们的目录服务，只有在容器启动时，服务提供者和调用者与注册中心交互，整个过程，注册中心不参与数据传输，不转发请求，压力较小 （“两不一小”）；

2、注册中心宕机问题：

注册中心会部署集群，

如果集群中的任意一台宕机之后，将自动切换到另外一台，不会影响已运行的提供者和消费者；

如果集群中所有注册中心全部宕机之后，服务提供者和服务消费者仍能通过本地缓存通讯；

数据库宕机后，注册中心仍能通过缓存提供服务列表查询，但不能注册新服务；

监控中心：

1、监控中心负责统计各服务调用次数、调用时间等，统计先在内存汇总后每分钟一次发送到监控中心服务器，并以报表展示

2、监控中心宕机不影响使用，只是丢失部分的采样数据；

3、dubbo-admin 可以通过监控中心的可视化界面，进行禁止服务和截止消费者（大量恶意访问的 ip）；

服务提供者：

1、服务提供者出现宕机，如果任意一台宕机，由于服务提供者没有状态，不影响使用；如果所有的服务提供者全部宕机之后，服务消费者应用将无法使用，并无限次重连等待服务提供者恢复；

2、服务提供者出现变更，比如增加新的机器部署，注册中心基于长连接将推送服务提供者信息给消费者；

（3）可以参考 Dubbo 的第四个问题

2、说一下 Dubbo 的原理？（这个问题难度比较大，学员可以不会）

3、Dubbo 在你们项目中怎么用的？

【参考答案】 dubbo 在我们项目中主要用来实现不同系统之间的服务调用，由于我们项目是按照不同的功能分了不同的系统，按照三层架构又分了不同的服务，其中三层架构中的控制层作为服务的消费方，业务层和持久层共同作为服务的发布方，这样的架构实现了系统的服务化，提高了开发效率，实现了业务的解耦。

项目中通过 dubbo 和 spring 的整合，采用全 spring 配置方式，只需要用 spring 来加载 dubbo 的配置，完成了服务的发布和调用。我们主要在服务的暴露方通过<dubbo:service>标签来暴露服务，在服务的消费方通过<dubbo:reference>标签来引用服务，注册中心我们选用的是 zookeeper，对服务的 URL 进行了管理和配置。

批注 [6]: 注册中心在整个 dubbo 的过程中起了什么作用，有没有参与数据传输？ 这类问题的答案

批注 [7]: 面试中问到关于注册中心宕机问题的可以参考这个话术，比如：Dubbo 中 zookeeper 做注册中心，如果注册中心集群都挂掉，发布者和订阅者之间还能通信么？

批注 [8]: 监控中心的作用。监控中心在公司中主要由运维人员做监控的

4、Dubbo 都支持什么协议？这些协议有什么不同点？你们项目使用的是什么协议？

【参考答案】

Dubbo 支持 Dubbo 协议、RMI 协议、hessian 协议、Http 协议等

Dubbo 协议：缺省协议、采用了单一长连接和 NIO 异步通讯、使用线程池并发处理请求，能减少握手和加大并发效率、采用的是 Hession 二进制序列化、性能较好，推荐使用。

主要应用于传入传出参数数据包较小（建议小于 100K），消费者比提供者个数多，由于是单一连接，因为尽量不要传输大文件。

RMI 协议：采用 JDK 标准的 RMI 协议（基于 TCP 协议）、堵塞式短连接、JDK 标准序列化方式、同步通讯。适用于消费者和提供者个数差不多的，可传文件。测试发现偶尔会连接失败，需要重建 Stub。

Hessian 协议：采用 http 通讯，采用 Servlet 暴露服务，多连接短连接的同步传输方式，采用 hession 的二进制序列化，适合提供者比消费者多。

我们项目中使用的是默认 Dubbo 协议（其他两种简单的知道就行），因为我们项目的特点主要是并发大、消费者要比提供者多。

5、在使用 Duubo 的过程中你们遇到过什么问题？

【参考答案】

（1）序列化和反序列化异常：

调用服务，出现消息发送失败的时候，通常是接口方法的传入传出参数是新增加的扩展类，没有实现序列化接口；

B 服务远程调用 A 服务，返回值为 C 对象，结果断点 A 服务返回值对象 C 有值，但是 B 服务远程拿到的 C 对象为 null；

（2）服务调用问题

服务提供者是否正常运行？（这块可能涉及到调用的服务被禁止）

连接的注册中心是否正确？（有时会涉及到注册中心的分组问题）

相应的服务提供者是否注册到注册中心？

（3）调用版本问题（在开发过程中增加提供服务版本号和消费服务版本号）

当遇到多个环境（开发、测试、上线）、多个版本（app、PC 端）的时候，一个服务可能不满足我们的需求，因为对服务的版本号进行定义，可以为我们后期版本的迭代升级做好铺垫。

整个过程会遇到各种各样的问题，上面这些基本都是技术问题

批注 [9]: 缺省的意思是默认

批注 [10]: NIO(New IO):面向缓冲的非堵塞型 IO

批注 [11]: 序列化方式的一种，比 java 自带的序列化方式要高效

批注 [12]: 对于这两个为什么，大家不用计较，基本都是要不性能浪费，要不就是效率问题

批注 [13]: 存根，为了屏蔽客户调用远程主机上的对象，必须提供某种方式来模拟本地对象，这个本地对象称为存根。

批注 [14]: Dubbo 远程调用服务是通过序列化和反序列化实现的，Dubbo 默认是采用 Hession 的序列化方式，反序列话创建对象时，会取最少的构造函数创建对象，同时也为构造参数设置默认值，基本类型设置为基本类型的默认值，引用类型设置为 null，因为我们要在接口的传参类里面添加无参构造；

批注 [15]: 查看 dubbo 的源码发现：hession 序列化协议的时候，会将所有字段放在一个 ArrayList 中，然后依次写入二进制流中；反序列化的时候，所有字段放在一个 hashmap 中，hashmap 的 key 是不能重复的，如果对象 C 是继承父类 D 的，并且对象 C 和对象 D 有同名的字段，这时父类 D 同名字段的值就会把 C 的值覆盖，导致子类的字段值丢失。因为使用 hession 序列化的时候，一定要注意子类和父类不能有同名字段

PS：使用 hession 序列化时的约束：

（1）参数及返回值实现 Serializable 接口；

（2）参数及返回值不能自定义实现 List、Map、Number、Date 等接口，只能用 JDK 自带的实现，因为 hession 会做特殊处理，自定义实现类中的属性值都会丢失；

（3）hession 的序列化，只传成员变量和值的类型，不传方法或者静态变量；

（4）hession 序列化，默认都提供无参构造

批注 [16]: 这个问题也是一个面试问题，怎么禁止一个服务？

通过监控中心的可视化界面，我们可以对具体的服务设置禁止，也可以设置对应的权重。

如果设置了禁止按钮，消费者就调用不到具体的服务，出现超时问题，这时我们可以取消禁止按钮，但是实际过程中有本地缓存问题，需要注意。

批注 [17]: 注册中心可以配置不同的 group，这个涉及到测试环境和上线环境，为了防止冲突问题，可以设置不同的 group（了解即可）

二、Zookeeper

1、简单介绍一下 zookeeper?

【参考答案】

（1）Zookeeper 是 Apache Hadoop 的一个子项目，作为分布式协调作用（类似于我们的大脑），树形的目录结构，支持变更操作，同时也是 Dubbo 官方推荐的注册中心。

（2）Zookeeper 之所以能用来作为 dubbo 的注册中心来使用，主要是应用到 dubbo 的命名功能。在 SOA 架构、集群和分布式环境下，子项目之间的调用关系会变得越来越复杂，因为需要一个服务器专门给我们管理服务的信息和调节、管理这些服务，让我们的侧重点放在项目中的业务上，因为 zookeeper 的命名功能就可以充当这样一个服务器。

（3）我们项目中主要用 zookeeper 作为 Dubbo 的注册中心，集中管理所有服务的 URL；同时集中的管理集群的配置。

批注 [18]: 命名功能：利用 Zookeeper 的分层结构，可以把系统中的各种服务的名称、地址、以及目录信息存放在 Zookeeper 中，需要的时候去 Zookeeper 中读取

2、Zookeeper 的实现原理？（工作原理）

【参考答案】

Zookeeper 会维护一个类似于标准的文件系统的具有层次关系的数据结构。这个文件系统中每个子目录项都被称为 znode 节点，这个 znode 节点也可以有子节点，每个节点都可以存储数据，客户端也可以对这些 node 节点进行 getChildren，getData,exists 方法，同时也可以在 znode tree 路径上设置 watch（类似于监听），当 watch 路径上发生节点 create、delete、update 的时候，会通知到 client。client 可以得到通知后，再获取数据，执行业务逻辑操作。Zookeeper 的作用主要是用来维护和监控存储的 node 节点上这些数据的状态变化，通过监控这些数据状态的变化，从而达到基于数据的集群管理。

批注 [19]: 集群管理的原理

3、为什么要用 zookeeper 作为 dubbo 的注册中心？能选择其他的吗？

【参考答案】

Zookeeper 的数据模型是由一系列的 Znode 数据节点组成，和文件系统类似。但是与传统的磁盘文件系统不同的是，首先，zookeeper 的数据全部存储在内存中，性能高；其次，zookeeper 也支持集群，实现了高可用；同时基于 zookeeper 的特性，也支持事件监听（服务的暴露方发生变化，可以进行推送），因为 zookeeper 适合作为 dubbo 的注册中心区使用。

同时 redis、Simple 也可以作为 dubbo 的注册中心来使用。

Redis:用 key-value（Hash）来存储数据

主 key:服务器名和类型

Map 中的 key：url 地址

Map 中的 value：过期时间，判断脏数据，脏数据由监控中心删除（要求服务器时间必须相同）

利用 redis 中的 Publish/Subscribe 事件通知数据变更

批注 [20]: 标红的三点都是结合我们的项目特色或者 dubbo 注册中心的特点

总之，redis 作为注册中心来使用的话，支持集群，性能高，但是要求所有服务器的时间必须同步，要求较高。（redis 作为注册中心来使用，有的公司也在采用）

Simple：本身就是一个普通的 dubbo 服务，能减少第三方依赖，不支持集群，不适合生产环境。

批注 [21]: 了解即可

4、项目中主要用 zookeeper 做了什么？（作用）

【答案解析】

（1）作为注册中心用；（上面亮点已经介绍了为什么能用做注册中心）

主要是在服务器上搭建 zookeeper，其次在 spring 管理的 dubbo 的配置文件中配配

（PS：暴露方和消费方都需要配置）

```
<dubbo:registry address="zookeeper://192.168.37.161:2181"/>
```

（2）集中管理集群中的配置文件；（solr 集群 solrCloud 中集中管理 solr 的配置文件）

三、FastDFS

1、简单介绍一下 FastDFS？

【答案解析】

（1）开源的分布式文件系统，主要对文件进行存储、同步、上传、下载，有自己的容灾备份、负载均衡、线性扩容机制；

批注 [22]: 遭遇灾害的时候，系统仍能正常的运行

批注 [23]: 把同一个工作任务，分担到不同的人身上，共同完成

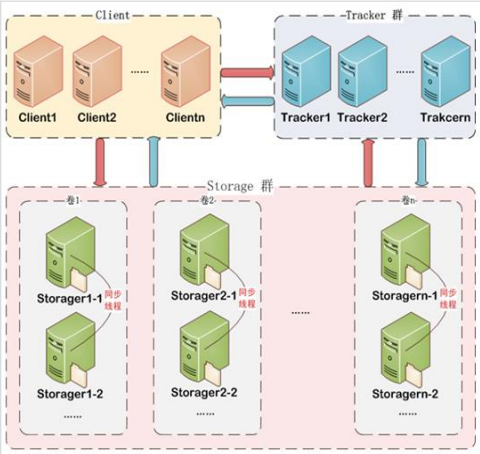
批注 [24]: 存储容量不够的时候，可以直接增加服务器的数量

批注 [25]: 跟踪器中的所有服务器都是对等的，可以根据服务器的压力随时增加或者减少

（2）FastDFS 架构主要包含 Tracker server 和 Storage server。客户端请求 Tracker server 进行文件上传、下载，通过 Tracker server 调度最终由 Storage server 完成文件上传和下载。

Tracker server：跟踪器或者调度器，主要起负载均衡和调度作用。通过 Tracker server 在文件上传时可以根据一些策略找到 Storage server 提供文件上传服务。

Storage server：存储服务器，作用主要是文件存储，完成文件管理的所有功能。客户端上传的文件主要保存在 Storage server 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去管理文件。



【扩展】：存储服务器采用了分组/分卷的组织方式。

整个系统由一个组或者多个组组成；

组与组之间的文件是相互独立的；

所有组的文件容量累加就是整个存储系统的文件容量；

一个组可以由多台存储服务器组成，一个组下的存储服务器中的文件都是相同的，组中的多台存储服务器起到了冗余备份和负载均衡的作用；

在组内增加服务器时，如果需要同步数据，则由系统本身完成，同步完成之后，系统自动将新增的服务器切换到线上提供使用；

当存储空间不足或者耗尽时，可以动态的添加组。只需要增加一台服务器，并为他们配置一个新的组，即扩大了存储系统的容量。

我们在项目中主要使用 fastdfs 来存储整个电商项目的图片。

2、为什么要使用 FastDFS 作为你们的图片服务器？

【答案解析】

（1）首先基于 fastDFS 的特点：存储空间可扩展、提供了统一的访问方式、访问效率高、容灾性好　等特点，再结合我们电商项目中图片的容量大、并发大等特点，因此我们选择了 FastDFS 作为我们的图片服务器；

【扩展】

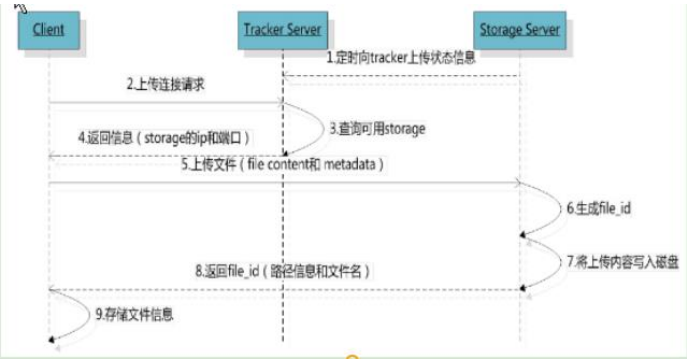
Nginx 也可以作为一台图片服务器来使用，因为 nginx 可以作为一台 http 服务器来使用，作为网页静态服务器，通过 location 标签配置；

在公司中有的时候也用 ftp 作为图片服务器来使用。

3、FastDFS 中文件上传下载的具体流程？

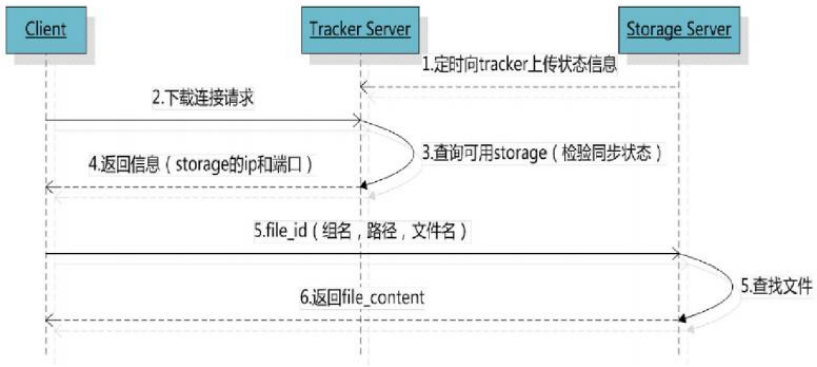
【答案解析】

FastDFS 文件上传的流程



客户端上传文件后生成一个 file_id，返回给客户端，客户端利用这个 file_id 结合 ip 地址，生成一个完成图片的 url，保存在数据库中。生成的那个 file_id 用于以后访问该文件的索引信息。

FastDFS 文件下载的流程



四、Redis

1、简单介绍一个 redis？

【答案解析】

（1）redis 是一个 key-value 类型的非关系型数据库，基于内存也可持久化的数据库，相对于关系型数据库（数据主要存在硬盘中），性能高，因此我们一般用 redis 来做缓存使用；并且 redis 支持丰富的数据类型，比较容易解决各种问题

（2）Redis 支持 5 种数据类型，string、hash、list、set、zset（sorted set）；

批注 [26]: 这块可以引出关系型数据库和非关系型数据库的区别？

String 类型是最简单的类型，一个 key 对应一个 value，项目中我们主要利用 string 类型的递增数字特性（incr）生成商品编号和订单编号；同时单点登录中的 token 也用 string 类型来存储；

Hash 类型中的 key 是 string 类型，value 又是一个 map（key-value），针对这种数据特性，比较适合存储对象，在我们项目中由于购物车是用 redis 来存储的，因为选择 redis 的散列（hash）来存储，同时首页大广告也是用 hash 类型存储的；

List 类型是按照插入顺序的字符串链表（双向链表），主要命令是 LPUSH 和 RPUSH，能够支持反向查找和遍历，在项目中我们主要存储商品评论列表，key 是该商品的 ID，value 是商品评论信息列表；

Set 类型是用哈希表类型的字符串序列，没有顺序，集合成员是唯一的，没有重复数据，支持集合间取差集、交集和并集，底层主要是由一个 value 永远为 null 的 hashmap 来实现的。我们的电商项目中没有用到这个数据类型。这个应用场景一般存储一个列表数据，但列表里面又不希望出现重复数据，比如微博应用中，可以将一个用户所有关注的对象放在一个集合中，将其所有粉丝存在一个集合，这样我们就可以实现两个人的共同好友、共同关注等需求；

zset（sorted set）类型和 set 类型基本是一致的，不同的是 zset 这种类型会给每个元素关联一个 dubbo 类型的分数（score），这样就可以为成员排序，并且插入是有序的。底层通过 hashmap 和跳跃表（SkipList）来保证数据的存储和有序。这种数据类型在我们项目中主要用来统计商品的销售排行榜，比如：items:sellsort 10 1001 20 1002 这个代表编号是 1001 的商品销售数量为 10,编号为 1002 的商品销售数量为 20。

（3）我们项目中主要用 redis 的 java 客户端 Jedis 来操作 redis 数据库，用来缓存各种操作频繁，不经常修改的数据，这样就减轻了数据库的访问压力，提高了查询效率

2、你还用过其他的缓存吗？这些缓存有什么区别？都在什么场景下去用？

【答案解析】

对于缓存了解过 redis 和 memcache，redis 我们在项目中用的比较多，memcache 没用过，但是了解过一点；

Memcache 和 redis 的区别：

数据支持的类型：

存储方式：redis 不仅仅支持简单的 k/v 类型的数据，同时还支持 list、set、zset、hash 等数据结构的存储；memcache 只支持简单的 k/v 类型的数据，key 和 value 都是 string 类型

可靠性：memcache 不支持数据持久化，断电或重启后数据消失，但其稳定性是有保证的；redis 支持数据持久化和数据恢复，允许单点故障，但是同时也会付出性能的代价

性能上：对于存储大数据，memcache 的性能要高于 redis

应用场景：

Memcache：适合多读少写，大数据量的情况（一些官网的文章信息等）

Redis：适用于对读写效率要求高、数据处理业务复杂、安全性要求较高的系统

批注 [27]: 这种散列对应的 value 内部实际是一个 hashmap，底层有两种实现方式：
（1）hash 的成员比较少时，Redis 为了节省内存会采用类似一维数组的方式来紧凑存储，而不会采用真正的 hashmap 结构，value 是 redisObject 的 zipmap;
（2）当成员数量比较多的时候，会自动转化成真正的 hashmap

批注 [28]: 比如我们要存储一个用户信息对象数据，包含以下信息：

用户 ID，为查找的 key，存储的 value 用户对象包含姓名 name，年龄 age，生日 birthday 等信息，如果用普通的 key/value 结构来存储，主要有以下 2 种存储方式：

第一种方式将用户 ID 作为查找 key,把其他信息封装成一个对象以序列化的方式存储，

如：set u001 "李三,18,20010101"

这种方式的缺点是，增加了序列化/反序列化的开销，并且在需要修改其中一项信息时，需要把整个对象取回，并且修改操作需要对并发进行保护，引入 CAS 等复杂问题。

第二种方法是这个用户信息对象有多少成员就存成多少个 key-value 对儿，用用户 ID+对应属性的名称作为唯一标识来取得对应属性的值，

如：mset user:001:name "李三 "user:001:age18 user:001:birthday "20010101"

批注 [29]: 商品评论列表：
一个商品会被不同的用户评论，并且商品的评论按照时间顺序排序
Key ： items:comment:1001
Value:'{"id":1,"name":"商品很好！","date":1432079}'

批注 [30]: Set 之所以不能存储重复的数据，就是通过计算 hash 的方式来快速排重的。
PS：个人觉得和 java 中的 hashset 唯一性就是因为 hashset 的底层是用 hashmap 实现的，这两点类似

批注 [31]: 者块可以利用 redis 中求两个集合的交集、并集、差集等

批注 [32]: 底层实现原理：hashmap 中存放的是成员到 score 的映射，而所有的成员放在跳跃表里面。排序主要应用的是 score，使用跳跃表的话能提高查找效率。

批注 [33]: 跳跃表：在一个节点维持多个指向其他节点的指针，从而能够快速访问其节点。跳跃表这个数据结构在 redis 中集群的时候也用到过，用作集群节点中内部数据结构。（了解即可）

3、Redis 在你们项目中是怎么用的？

【答案解析】

（1）门户系统中的首页内容信息的展示。（商品类目、广告、热门商品等信息）门户系统的首页是用户访问量最大的，而且这些数据一般不会经常修改，因此为了提高用户的体验，我们选择将这些内容放在缓存中；

（2）单点登录系统中也用到了 redis。因为我们是分布式系统，存在 session 之间的共享问题，因此在做单点登录的时候，我们利用 redis 来模拟了 session 的共享，来存储用户的信息，实现不同系统的 session 共享；

（3）我们项目中同时也将购物车的信息设计存储在 redis 中，购物车在数据库中没有对应的表，用户登录之后将商品添加到购物车后存储到 redis 中，key 是用户 id，value 是购物车对象；

（4）因为针对评论这块，我们需要一个商品对应多个用户评论，并且按照时间顺序显示评论，为了提高查询效率，因此我们选择了 redis 的 list 类型将商品评论放在缓存中；

（5）在统计模块中，我们有个功能是做商品销售的排行榜，因此选择 redis 的 zset 结构来实现；

还有一些其他的应用场景，主要就是用来作为缓存使用。

4、对 redis 的持久化了解不？

【答案解析】

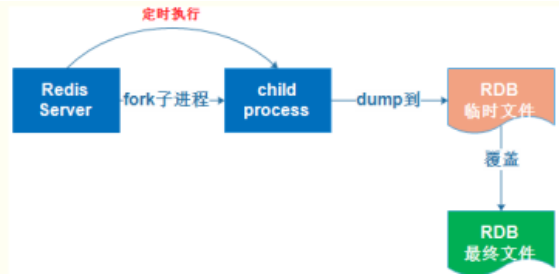
Redis 是内存型数据库，同时它也可以持久化到硬盘中，redis 的持久化方式有两种：

（1）RDB（半持久化方式）：

按照配置不定期的通过异步的方式、快照的形式直接把内存中的数据持久化到磁盘的一个 dump.rdb 文件（二进制文件）中；

这种方式是 redis 默认的持久化方式，它在配置文件（redis.conf）中的格式是：save N M，表示的是在 N 秒之内发生 M 次修改，则 redis 抓快照到磁盘中；

原理：当 redis 需要持久化的时候，redis 会 fork 一个子进程，这个子进程会将数据写到一个临时文件中；当子进程完成写临时文件后，会将原来的.rdb 文件替换掉，这样的好处是写时拷贝技术（copy-on-write），可以参考下面的流程图；



优点：只包含一个文件，对于文件备份、灾难恢复而言，比较实用。因为我们可以轻松的将一个单独的文件转移到其他存储媒介上；性能最大化，因为对于这种半持久化方式，使用的是写时拷贝技术，可以极大的避免服务进程执行 IO 操作；想对于 AOF 来说，如果数据集很大，RDB 的启动效率就会很高

批注 [34]: 每个应用场景可以参考业务中具体的业务逻辑

批注 [35]: 这种单点登录的实现方式不建议说，因为只要是市场上经常招人的公司基本都知道这是传智实现单点登录的方式

批注 [36]: 这块可以给大家提供了一个数据：Redis 启动后读取 RDB 快照文件，将数据从硬盘载入到内存中，根据数据量大小、结构、服务器的性能来说：通常记录一千万个字符串类型键、大小为 1GB 的快照文件载入到内存中需要花费 20-30 秒

批注 [37]: 这个配置也叫 redis 持久化的执行策略。对应的这种半持久化方式我们也可以手动的执行 save 或者 bgsave（异步）做快照。

批注 [38]: 扩展内容

批注 [39]: Fork：分支的意思，在这里可以理解为创建了一个和原来线程几乎完全相同的分支

批注 [40]: 这个和 GC 算法中的复制算法类似，可以比较着来理解

缺点：如果想保证数据的高可用（最大限度的包装数据丢失），那么 RDB 这种半持久化方式不是一个很好的选择，因为系统一旦在持久化策略之前出现宕机现象，此前没有来得及持久化的数据将会产生丢失；rdb 是通过 fork 进程来协助完成持久化的，因此当数据集较大的时候，我们就需要等待服务器停止几百毫秒甚至一秒；

（2）AOF（全持久化的方式）

把每一次数据变化都通过 write() 函数将你所执行的命令追加到一个 appendonly.aof 文件里面；

Redis 默认是不支持这种全持久化方式的，需要将 no 改成 yes

实现文件刷新的三种方式：

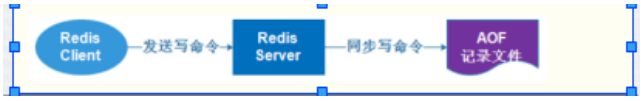
```
#
# Redis supports three different modes:
#
# no: don't fsync, just let the OS flush the data when it wants. Faster.
# always: fsync after every write to the append only log . Slow, Safest.
# everysec: fsync only if one second passed since the last fsync. Compromise.
#
```

no:不会自动同步到磁盘上，需要依靠 OS（操作系统）进行刷新，效率快，但是安全性就比较差；

always:每提交一个命令都调用 fsync 刷新到 aof 文件，非常慢，但是安全；

everysec:每秒钟都调用 fsync 刷新到 aof 文件中，很快，但是可能丢失一秒内的数据，推荐使用，兼顾了速度和安全；

原理：redis 需要持久化的时候，fork 出一个子进程，子进程根据内存中的数据库快照，往临时文件中写入重建数据库状态的命令；父进程会继续处理客户端的请求，除了把写命令写到原来的 aof 中，同时把收到的写命令缓存起来，这样包装如果子进程重写失败的话不会出问题；当子进程把快照内容以命令方式写入临时文件中后，子进程会发送信号给父进程，父进程会把缓存的写命令写入到临时文件中；接下来父进程可以使用临时的 aof 文件替换原来的 aof 文件，并重命名，后面收到的写命令也开始往新的 aof 文件中追加。下面的图为最简单的方式，其实也是利用写时复制原则。



优点：

数据安全性高

该机制对日志文件的写入操作采用的是 append 模式，因此在写入过程中即使出现宕机问题，也不会破坏日志文件中已经存在的内容；

如果日志文件过大，可以采用 rewriter 机制

缺点：

对于数量相同的数据集来说，aof 文件通常要比 rdb 文件大，因此 rdb 在恢复大数据集时的速度大于 AOF；

根据同步策略的不同，AOF 在运行效率上往往慢于 RDB，每秒同步策略的效率是比较高的，同步禁用策略的效率和 RDB 一样高效；

批注 [41]: 事实上，不会立即将命令写入硬盘文件中，而是写入硬盘缓存，可以配置策略，配置多久从硬盘缓存写入到硬盘文件中。
Appendfsync always
Appendfsync everysec 默认的
Appendfsync no 不主动，默认 30 秒一次

批注 [42]: 原理可以不会，属于扩展内容

批注 [43]: 如果本次操作只是写入了一半数据就出现系统崩溃问题，可以在 redis 下一次启动之前，我们可以通过 redis-check-aof 工具来帮助我们解决数据一致性的问题
【扩展问题】
如何修复损坏的 aof 文件？
1）将现有已经损坏的 aof 文件额外拷贝一份出来；
2）执行“redis-check-aof --fix <filename>”命令来修复损坏的 aof 文件；
3）用修复后的 aof 文件重新启动 redis 服务器

批注 [44]: Rewriter 机制：aof 文件中存放了所有的 redis 操作指令，当 aof 文件达到一定条件或者手动 bgrewriteaof 命令都可以触发 rewriter
Rewriter 函数执行完成之后，aof 文件中会保存 keys 的最后状态，清楚之前冗余的状态，从而达到缩小这个文件的作用
可以在配置文件中设置
Auto-aof-rewrite-percentage 100：当前写入日志文件的大小超过上一次 rewriter 之后的文件大小的百分之 100 时（也就是 2 倍的时候）触发 rewriter 这个函数

针对以上两种不同的持久化方式，如果缓存数据安全性要求比较高的话，用 aof 这种持久化方式（比如项目二中的购物车）；如果对于大数据集要求效率高的话，就可以使用默认的。而且这两种持久化方式可以同时使用。

5、做过 redis 的集群吗？你们做集群的时候搭建了几台，都是怎么搭建的？

针对这类问题，我们首先考虑的是为什么要搭建集群？（这个需要针对我们的项目来说）

Redis 的数据是存放在内存中的，这就意味着 redis 不适合存储大数据，大数据存储一般公司常用 hadoop 中的 Hbase 或者 MogoDB。因此 redis 主要用来处理高并发的，用我们的项目二来说，电商项目如果并发大的话，一台单独的 redis 是不能足够支持我们的并发，这就需要我们扩展多台设备协同合作，即用到集群。

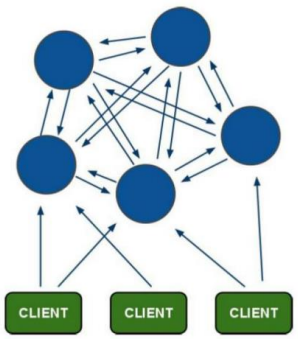
Redis 搭建集群的方式有多种，例如：客户端分片、Twemproxy、Codis 等，但是 redis3.0 之后就支持 redis-cluster 集群，这种方式采用的是无中心结构，每个节点保存数据和整个集群的状态，每个节点都和其他所有节点连接。我们项目中也使用的是 redis-cluster 集群。

集群这块直接说是公司运维搭建的，小公司的话也有可能由我们自己搭建，开发环境我们也可以直接用单机版的。但是学员可以了解一下 redis 的集群版。搭建 redis 集群的时候，对于用到多少台服务器，每家公司都不一样，大家针对自己项目的大小去衡量。举个简单的例子：

我们项目中 redis 集群主要搭建了 6 台，3 主（为了保证 redis 的投票机制）3 从（高可用），每个主服务器都有一个从服务器，作为备份机。

【扩展】

1、架构图如下：



（1）所有的节点都通过 PING-PONG 机制彼此互相连接；

（2）每个节点的 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 集群连接，只需要连接集群中的任何一个节点即可；

（4）Redis-cluster 把所有的物理节点映射到【0-16383】slot 上，负责维护

2、容错机制（投票机制）

（1）选举过程是集群中的所有 master 都参与，如果半数以上 master 节点与故障节点连接超过时间，则认为该节点故障，自动会触发故障转移操作；

（2）集群不可用？

a:如果集群任意 master 挂掉，并且当前的 master 没有 slave，集群就会 fail；

批注 [45]: 这条会涉及到 redis-cluster 的原理问题：Redis-cluster 中内置了 16384 个哈希槽，当需要 set 一个值的时候，redis 先会对 key 进行 crc16 算法算出一个结果，然后把这个结果对 16384 取余，这样的话每个 key 就会对应一个 0-16283 之间的哈希槽，redis 会根据节点的数量进行均等的分配哈希槽映射到不同的节点上。比如：有三个节点 A1、A2、A3

A1：【0-5000】

A2：【5001-10000】

A3：【10001-16383】

批注 [46]: 这块就会涉及到集群挂的面试题，这个要学员灵活理解，面试中面试官经常会出各种场景，问学生集群挂不挂的问题

b:如果集群超过半数以上 master 挂掉，无论是否有 slave，整个集群都会 fail;

6、redis 有事务吗？

Redis 是有事务的，redis 中的事务是一组命令的集合，这组命令要么都执行，要不都不执行，redis 事务的实现，需要用到 MULTI（事务的开始）和 EXEC（事务的结束）命令 ；

```
redis 127.0.0.1:6379> MULTI
OK
redis 127.0.0.1:6379> set url http://baidu.com
QUEUED
redis 127.0.0.1:6379> set title 百度
QUEUED
redis 127.0.0.1:6379> EXEC
1) OK
2) OK
redis 127.0.0.1:6379>
```

当输入 MULTI 命令后，服务器返回 OK 表示事务开始成功，然后依次输入需要在本次事务中执行的所有命令，每次输入一个命令服务器并不会马上执行，而是返回”QUEUED”，这表示命令已经被服务器接受并且暂时保存起来，最后输入 EXEC 命令后，本次事务中的所有命令才会被依次执行，可以看到最后服务器一次性返回了两个 OK，这里返回的结果与发送的命令是按顺序一一对应的，这说明这次事务中的命令全都执行成功了。

Redis 的事务除了保证所有命令要不全部执行，要不全部不执行外，还能保证一个事务中的命令依次执行而不被其他命令插入。同时，redis 的事务是不支持回滚操作的。

【扩展】

Redis 的事务中存在一个问题，如果一个事务中的 B 命令依赖上一个命令 A 怎么办？

这会涉及到 redis 中的 WATCH 命令：可以监控一个或多个键，一旦其中有一个键被修改（或删除），之后的事务就不会执行，监控一直持续到 EXEC 命令（事务中的命令是在 EXEC 之后才执行的，EXEC 命令执行完之后被监控的键会自动被 UNWATCH）。

应用场景：待定

批注 [47]: 这部分了解即可。不用深入追究，这个 watch 命令相当于 mysql 中的锁

【扩展】

1、redis 的安全机制（你们公司 redis 的安全这方面怎么考虑的？）

漏洞介绍：redis 默认情况下，会绑定在 0.0.0.0:6379，这样就会将 redis 的服务暴露到公网上，如果在没有开启认证的情况下，可以导致任意用户在访问目标服务器的情况下未授权访问 redis 以及读取 redis 的数据，攻击者就可以在未授权访问 redis 的情况下可以利用 redis 的相关方法，成功在 redis 服务器上写入公钥，进而可以直接使用私钥进行直接登录目标主机；

比如：可以使用 FLUSHALL 方法，整个 redis 数据库将被清空

解决方案：

（1）禁止一些高危命令。修改 redis.conf 文件，用来禁止远程修改 DB 文件地址，比如 rename-command FLUSHALL "" 、rename-command CONFIG"" 、rename-command EVAL “”等；

批注 [48]: 这个一般开发的时候不会考虑，也不会配置那么多。在公司项目上线之后，结果出现过被别人删除数据的问题，这个也相应的在线上环境做了预防。

（2）以低权限运行 redis 服务。为 redis 服务创建单独的用户和根目录，并且配置禁止登录；

（3）为 redis 添加密码验证。修改 redis.conf 文件，添加 requirepass mypassword；

（4）禁止外网访问 redis。修改 redis.conf 文件，添加或修改 bind 127.0.0.1，使得 redis 服务只在当前主机使用；

（5）做 log 监控，及时发现攻击；

2、redis 的哨兵机制（redis2.6 以后出现的）

哨兵机制：

监控：监控主数据库和从数据库是否正常运行；

提醒：当被监控的某个 redis 出现问题的时候，哨兵可以通过 API 向管理员或者其他应用程序发送通知；

自动故障迁移：主数据库出现故障时，可以自动将从数据库转化为主数据库，实现自动切换；

具体的配置步骤面试中可以说参考的网上的文档。要注意的是，如果 master 主服务器设置了密码，记得在哨兵的配置文件（sentinel.conf）里面配置访问密码

3、缓存穿透

缓存查询一般都是通过 key 去查找 value，如果不存在对应的 value，就要去数据库中查找。如果这个 key 对应的 value 是一定不存在的，并且对该 key 并发请求很大，就会对数据库产生很大的压力，这就叫缓存穿透

解决方案：如果一个查询返回的结果为空，仍然对空结果进行缓存；

对一定不存在的 key 进行过滤，将这些 key 放到一个 map 中，查询时将这个 map 过滤；

4、缓存雪崩

当缓存服务器重启或者大量缓存集中在某个时间失效，这样在失效的瞬间对数据库的访问压力就比较大

解决方案：不同的 key 设置不同的过期时间，让缓存失效的时间点尽量均匀一些；

在缓存失效之前，预先更新缓存；

5、redis 中对于生存时间的应用

Redis 中可以使用 expire 命令设置一个键的生存时间，到时间后 redis 会自动删除；

应用场景：

（1）设置限制的优惠活动的信息；

（2）一些及时需要更新的数据，积分排行榜；

（3）手机验证码的时间；

（4）限制网站访客访问频率；

五、Solr

1、简单介绍一下 solr？

【答案解析】

（1）是一个基于 lucene 的全文搜素服务器；

（2）流程：主要分为两个部分，一个部分是创建索引，客户端主要通过 POST 方法向 solr 服务器发送一个描述 Field 及其内容的 XML 文档，Solr 服务器根据 xml 文档添加、删除、更新索引；第二个部分是搜索索引，客户端用 GET 方法向 Solr 服务器发送请求，然后对 Solr 服务器返回 Xml、json 等格式的查询结果进行解析，组织页面布局。Solr 不提供构建页面 UI 的功能，但是 Solr 提供了一个管理界面，通过管理界面可以查询 Solr 的配置和运行情况（solr 的客户端）；

（3）我们在项目中主要用 solr 服务器实现电商网站中的用户对于商品的搜索功能，可以通过商品的分类、品牌、规格、商品名称等进行搜索，同时对于新增加的商品，我们会将其添加到 solr 的索引库中，对于一些敏感词汇进行配置停用，对于一些网络新词配置扩展词。

2、Solr 如何实现的搜索？（solr 的底层原理）

【答案解析】

3、数据库可以直接建索引实现搜索，你们为什么要用 solr？

【答案解析】

（1）数据库中如果我们要去做搜索的话，主要应用的是 like 关键字，而如果用 like 关键字去匹配的话，即使在字段上建立了索引，在某些情况下，数据库的索引是失效的，因为在数据库中建不建索引，效果不大；

（2）相对于数据库的索引来说，solr 建立索引去搜索的话效率要高，针对于电商项目中存在的高并发问题，我们使用 solr 服务器做搜索，减少了与数据库的访问压力；

（3）用 solr 服务做搜索的话，可以配置一些搜索的停用词、热词、还有对搜索结果的高亮处理等。

4、Solr 在项目中怎么用的？

【答案解析】

我们项目中针对于前台商品搜索这块单独做了一个搜索系统。在搜索系统中我们使用 solr 全文搜索技术+IK 分词器等技术，

首先根据数据库的字段对 solr 的业务域进行配置，项目中我们主要对 itemid、item_title、item_price、item_image、item_category、item_seller、item_brand、item_keywords 这些字段配置了业务域，测试中存在一些性能问题，因此在配置这些业务域的时候我们后期也做了修改：

（1）将只用于搜索、而不需要作为结果的 filed 的 stored 设置为 false；（比如 item_keywords）

（2）将不需要搜索、而只需要作为结果的 filed 的 indexed 设置为 false；（比如 item_image）

（3）删除所有不必要的 copyfiled；

（4）将所有 textfileds 的 index 设置为 false，并且使用 copyfield 域将他们复制到一个总的 textfile 上进行搜索（比如 item_keywords）；

其次我们将所有的商品同步到索引库中，这样我们在前台系统进行搜索的时候直接调用 solr 的服务接口，solr 的服务接口我们采用的是 solr 的 java 客户端 solrj 完成的，主要是针对我们前台的搜索条件调用 solrJ 封装的工具类去索引库中去搜索，然后对搜索结果进行处理（高亮、按热度排名等处理）返回给前台系统做展示。

因为我们项目使用的中文分词器是 IK 分词器，针对于一些明感词、停用词、新词的扩展，我们会将其定时的修改 IK 分词器对应的配置文件 stopword.dic 和 ext.dic。

我们项目中对于新增加审核通过的一些商品要同步到索引库中去，方便用户能够搜索到新的商品，这块我们采用的是消息队列中的 ActiveMq。在商品审核通过之后对索引库发送消息，让索引库这边去数据库查询，然后添加到数据库中，这样做的好处就是每次添加索引库的时候只需要添加新添加的商品进去就行,不用把数据库里的数据全部重新添加一次。

当然我们为了防止有异常的发生也在后台添加了全部添加的功能,但只对部分人员进行权限得开放。由于考虑到后期访问人数增加,项目扩展,以及可能出现的一些其他异常,我们还对 solr 进行了集群的搭建。

5、Solr 和 lucene 有什么区别和联系？

【答案解析】

（1）lucene 是一个全文搜索引擎工具包，是一个做搜索用的类库，但不包含搜索引擎系统，它包含了索引结构、读写索引工具、排序等功能，因此在使用 lucene 时需要我们来关注搜索引擎系统，即数据的获取、解析、分词等；

（2）solr 是基于 lucene 可直接运行的应用程序，作为一个单独的搜索引擎系统来使用，提供了更加丰富的基于 REST 的搜索服务接口，我们就不需要将搜索逻辑耦合在应用中，并且我们可以通过配置文件定义数据解析的方式，比较像一个搜索框架，支持集群、热切换、facet 等工作，

因此，针对 solr 和 lucene 的特性，lucene 的使用更加灵活，但需要自己处理搜索引擎系统，开发者需要自己维护索引文件；而 solr 更相当于一个框架，封装了 lucene 各种处理复杂搜索业务的服务接口。因此我们使用 solr 作为我们搜索系统的应用。

批注 [49]: 这个 file 域是将经常查询的 textfiled 域进行 copyfiled 的一些域对象

批注 [50]:

六、FreeMarker

1、FreeMarker 是什么？

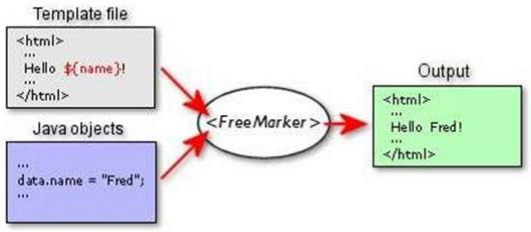
【答案解析】

（1）FreeMarker 是一款用 java 编写的模板引擎，即基于模板、用于生成输出文本的通用工具；

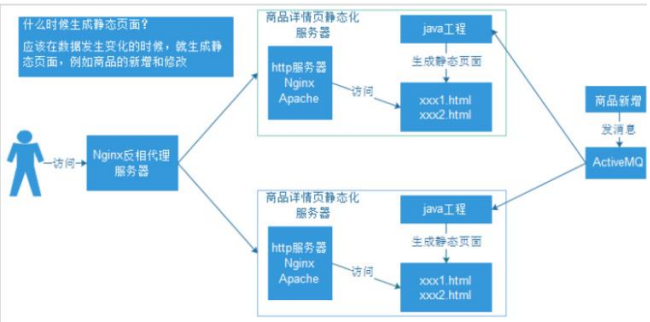
（2）原理：将页面中所需要的样式放入 FreeMarker 文件中，然后将页面中所需要的数据动态绑定，并放入 Map 中，通过调用 FreeMarker 模板文件解析类 process()方法完成静态页面的生成；模板+数据模型=输出

批注 [51]: 模板：一份已经写好了基本内容，有着固定格式的文档，其中空出或者用占位符标识的内容，由使用者来填充，不同的使用者给出的数据是不同的。

批注 [52]: 数据模型：这些信息来自于数据库、文件，甚至于在程序中直接生成，在前台页面通过表达式的方式嵌入。在模板运行时，由模板引擎来解析模板，并采用动态数据替换占位符部分的内容。



（3）freeMarker 可以作为 MVC 框架中的 view 层组件、html 页面静态化、代码生成工具、CMS 模板引擎。类似于 jsp 的功能，也可以用于生成 xml、pdf、java 等文件。考虑到效率问题，我们最近的项目主要用 freemarker 来生成商品详情页，在后台系统对于商品审核之后，利用消息队列 ActiveMq 发送消息，将商品 id 发送给页面生成监听工程，这个工程接收到商品的 id 后，根据我们写的商品详情页的模板 item.ftl 文件，以及根据 id 查出来的商品数据（动态数据），利用模板的 process（）生成商品详情页，输出到工程外部的一个目录。最后将这些生成的静态页面部署到服务器中，方便用户直接访问这些静态页，提高查询的效率。



2、你们项目中为什么要用 freemarker？

【答案解析】

商品详情页是消费者了解商品具体参数的主要途径，访问的频率比较高，同时商品

的数量比较大，如果我们必须对商品详情页进行优化，单独的将商品详情页抽离出来，以提高访问的速度。因为访问静态页不需要经过程序处理，可以提高处理速度、稳定性高。

对于商品详情页这块，也可以使用缓存 redis，这样也能提高查询效率，但是涉及到数据的同步，并且商品的数量比较大，不是特别适合，因此我们使用 freemarker 来生成静态页面 html。

3、你们用 freemarker 做了什么功能？

【答案解析】

4、freemarker 生成的静态化页面，如果商品的信息更改以后，会不会生成新的静态化化页面，freemarker 静态化页面的数据是从哪里调用出来的，如果不是从数据里面调的数据的，这个地方需要用到同步，和谁同步？

【答案解析】

（1）商品信息新增（审核通过）和修改之后，需要生成新的商品详情页；

（2）freemarker 静态化页面的数据是在创建静态化页面的时候获取到的，我们项目中是从数据库中查出来的；

（3）针对上个问题，也可以将数据放在缓存中，放在缓存中的时候，这个数据也需要我们通过 MQ 去同步缓存中的数据；

【扩展】

（1）Freemaker 中的模板编程

主要有四个部分构成：

文本：按照原样输出；

插值：由${a}，这是由后台传递过来，或者计算的值；

FTL 标签：和 HTML 标签相似，给 freemarker 的指示，不会打印在输出内容中，比如变量是否存在 if_exists 、!；字符串转化成小写、大写：lower_case、upper_case;

声明变量：<#assign>；逻辑判断<#if><#else><#elseif>；集合遍历:

```
<#list itemList! as item>
    ${item.title!}
<#list>
```

包含指令：<#include>；导入文件<#import>

注释：和 html 的类似，由<#--和-->来分隔的，不会在输出内容中显示

（3）这个静态页面是完全静态的吗？

针对我们生成的这个商品详情页不是完全静态的，对于商品的库存和评论两个部分不是静态的

商品详情页显示的库存是可销售库存。

商品的评论：对于评论这块，我们可以在静态页中使用局部动态，在商品评论这块，如果用户在对商品进行评论之后将提交到后天服务，然后在添加的业务里面把这个商品对应的评论信息利用 list 结构存放到 redis 中。用户在查看商品评论的时候，可以通过局部

批注 [53]: 这个延伸内容，涉及到的面试题：
Freemarker 中常用的标签都有什么？
商品详情页的模板是你写的吗？怎么写的？

批注 [54]: 库存结构中一般分为：可销售库存（S）、订单占用库存（O）、锁定库存（L）、虚库存（V）、调拨出占用库存（T）、调拨入库存（A）、不可销售库存（U）
S=I+V+A-O-U-L-T
针对这个库存问题，后面会重点详细说明
可销售库存：库存结构的最大组成部分，这个库存也是电商网站上显示的库存数量。网站前台显示的这个数据一般和 WMS（仓储系统）的数据保持同步，并做出判断：当可销售库存>0 时，这个商品可以购买，则会显示商品可销售，当可销售库存<0 时，这个商品就会显示缺货。我们平常的缺货不是指库房中没有库存了，而是指没有可供销售库存的情况。

ajax 去访问 redis。

七、ActiveMQ

1、介绍一下 activeMq？（实现原理）

2、还用过其他的 MQ 吗？分别有什么优缺点？你们为什么要使用 activeMQ？

3、你们项目中如何应用的 ActiveMQ？

4、ActiveMQ 是如何解决消息发送失败和消息消费失败的？

5、延伸

八、Nginx

1、什么是 nginx?

2、Nginx 在项目中怎么用的？（nginx 的应用场景）

3、延伸

九、MyCat

十、Springboot

1、简单介绍一下 springboot？
2、使用 springboot 的好处？（自动配置的原理）

业务

一、商品管理模块

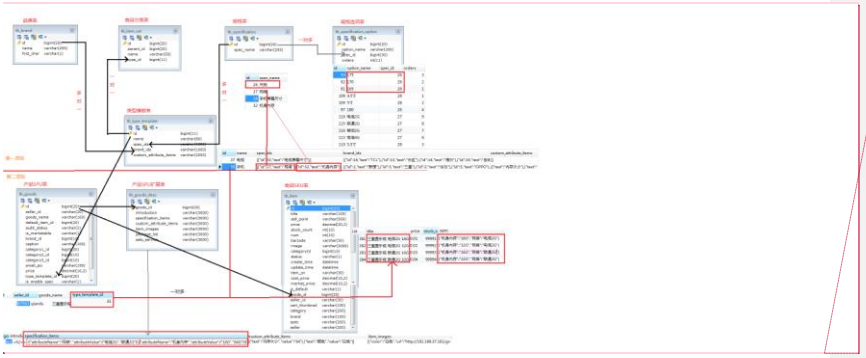
【扩展知识】

1、电商平台商品管理体系中的概念

SPU：标准化产品单元

SKU：库存量单位

2、电商平台商品管理体系的设计



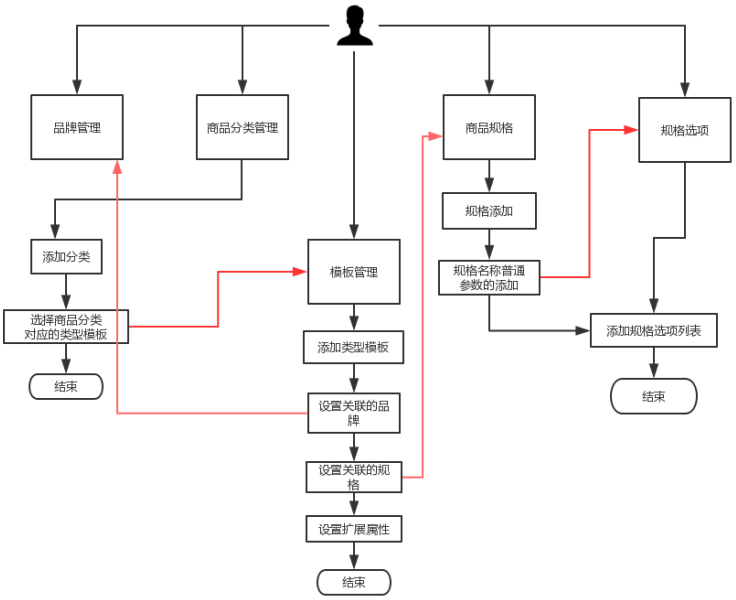
批注 [55]: 这个可以简单的理解为：属性值、特性相同的商品就可以称为一个 SPU，比如：iphone8 就是一个 SPU

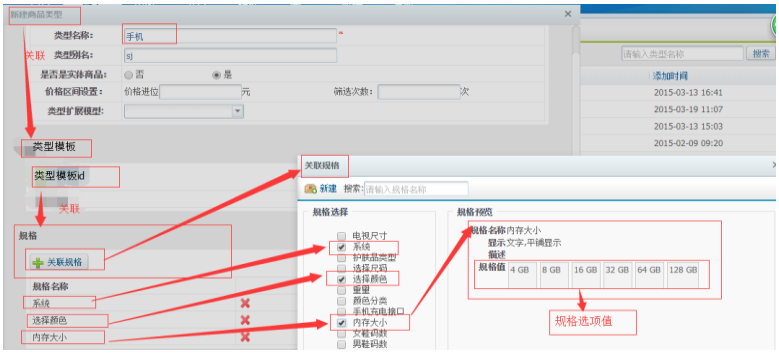
批注 [56]: SKU:保存库存控制的最小可用单位。在电商中可以简单的理解为：黑色 128G 的 iphone8 就是一个商品 SKU

批注 [57]: 这个图需要老师带学生分析

1、业务流程图

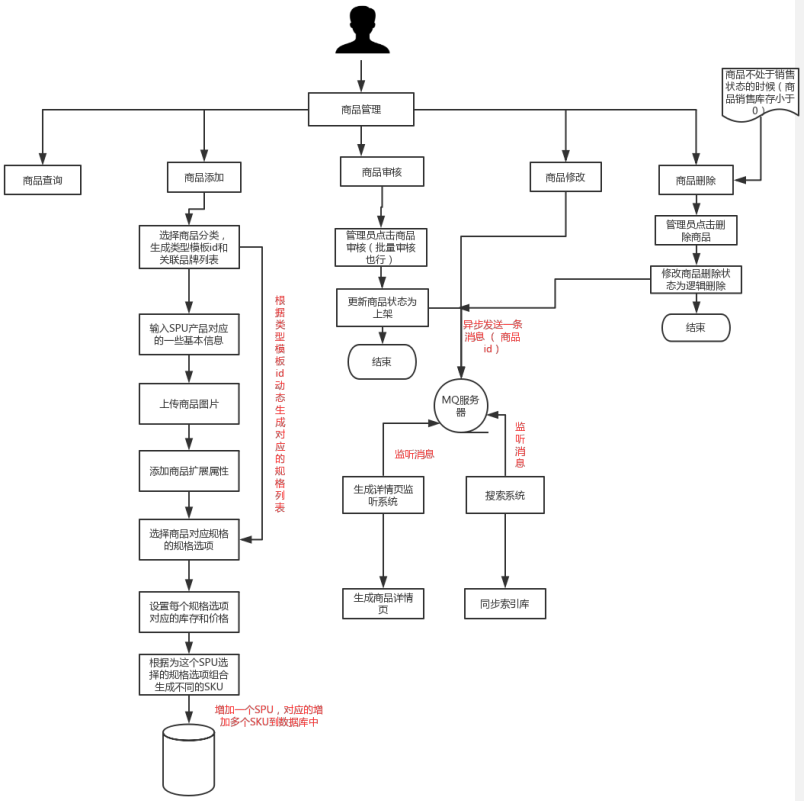
1、1 商品管理模块的品牌管理、商品分类管理、规格管理、规格选项管理、类型模板管理



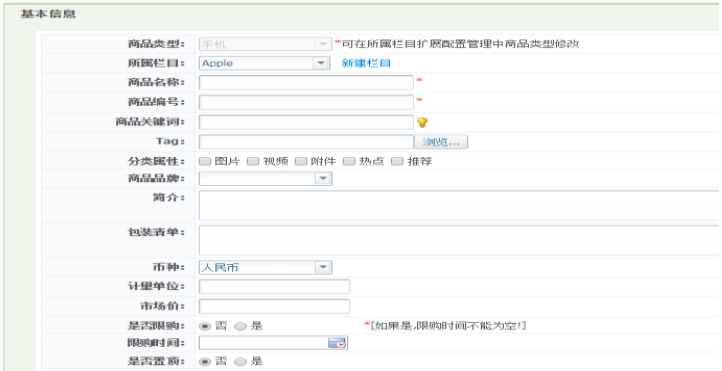


添加商品分类，因为商品分类关联了类型模板表，即可以针对这个类型模板关联对应的规格；

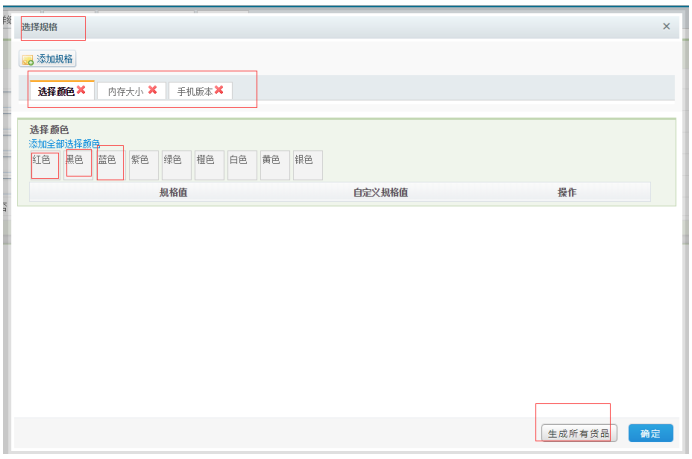
1、2　商品管理模块



第一步：选择商品分类，分类关联了模板，模板关联了规格；



第二步：针对这个 SPU 关联的模板就会显示出来，我们可以选择我们需要添加的商品的规格对应的都有什么规格选项值，然后生成所有的 sku；



第三步：下图就是对应的组合生成的不同的 SKU 对应不同的库存和价格

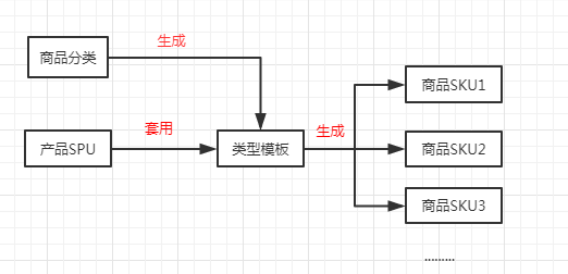
2、业务话术

【参考答案】我们这个电商平台是不支持商家入驻功能的，因此整个商品管理主要是我们这个电商后台系统的核心。主要的功能分为两个主要模块，第一个模块主要是管理员对于商品品牌、商品分类、模板管理、规格和规格参数进行管理的一些操作；第二个模块主要是针对有对商品操作权限的业务员对于商品的添加、审核、上下架、修改、删除、批量操作、模糊查询等功能。

针对商品这个模块来说，因为不同的分类对应不同的规格参数，同一个分类对应的规格参数相同，只是具体的规格选项值不同；同时，一个商品不同的规格参数值然后对应的价格和库存不一样，因此在整个商品管理模块中，我们针对不同的商品分类设计了不同的类型模板，这个类型模版中对应有多个规格参数，同时将规格参数和规格选项关联起来（类似于字典表）。这样的话我们就可以通过产品 SPU 对应的商品分类所关联的类型模板表，然后一个产品 SPU 就套用这个类型模板表生成多个商品 SKU 数据。

批注 [58]: 整个模块具体有什么功能模块的介绍

批注 [59]: 这块是电商模块中对于商品规格参数的设计思路，



商品添加：

1、先判断是否有对应的商品添加的权利，如果没有，直接显示没有操作的权限；

2、如果有，先选择商品所属的分类，然后商品分类选择之后会对应的生成类型模板和所关联的品牌；

3、添加产品 SPU 的一些基本信息（SPU 名称、副标题、商城价等）；

4、其次，上传产品 SPU 对应的图片，这块因为整个电商平台中需要保存大量的图片，因此我们使用了一个分布式文件存储系统 FastDFS 来存储商品的图片，这样的话后期也可针对容量进行水平扩展，不影响原来的使用，并且针对于高并发、高可用的问题，FastDFS 的容灾性、负载均衡也是个优势；

5、最后我们要选择是否启用规格，如果不启动规格的话，后台对应的商品 SKU 就只生成一条数据；如果启动规格的话，根据我们选择的不同的商品规格参数及规格属性值生成多个商品 SKU 数据，这多个商品 SKU 数据对应不同的商品价格和库存；

6、调用后台的商品添加接口/服务，然后对应的接口/服务中生成商品添加时的一些默认数据（添加时间、更新时间等），在商品添加的时候我们默认是下架状态。需要拥有审核功能的业务员针对商品中是否有一些不合法的数据进行审核。

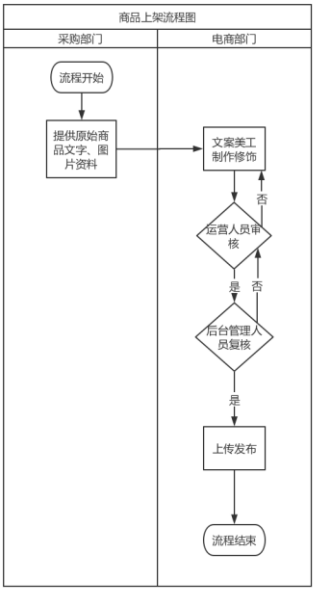
7、在这个功能中，因为产品 SPU 的信息比较多，我们针对这点，将产品 SPU 的一些经常查询的字段，比如产品的标题、名称、卖点等信息单独放在一张表中；而对于一些不经常查询的字段，比如描述、规格参数、扩展属性等信息放在另外一张表中，通过 goodsId 进行关联，这样设计的话，查询很方便；

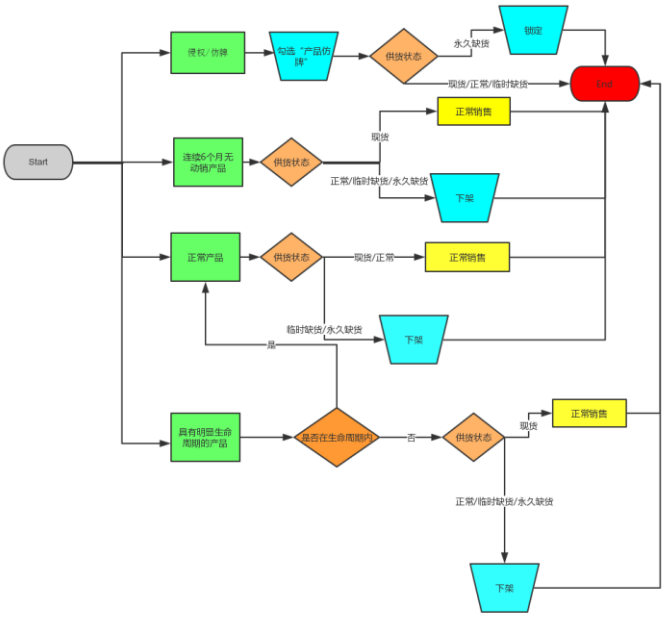
商品审核：

1、判断操作员是否有审核的权限，如果有，则针对商品的数据进行单个商品的审核或者批量审核；

2、商品审核的同时，修改商品的状态为已审核；

商品上下架：因为商品添加之后，默认是下架状态，如果商品想处于销售状态，必须将商品的状态改为上架状态





1、先判断是否有权限；

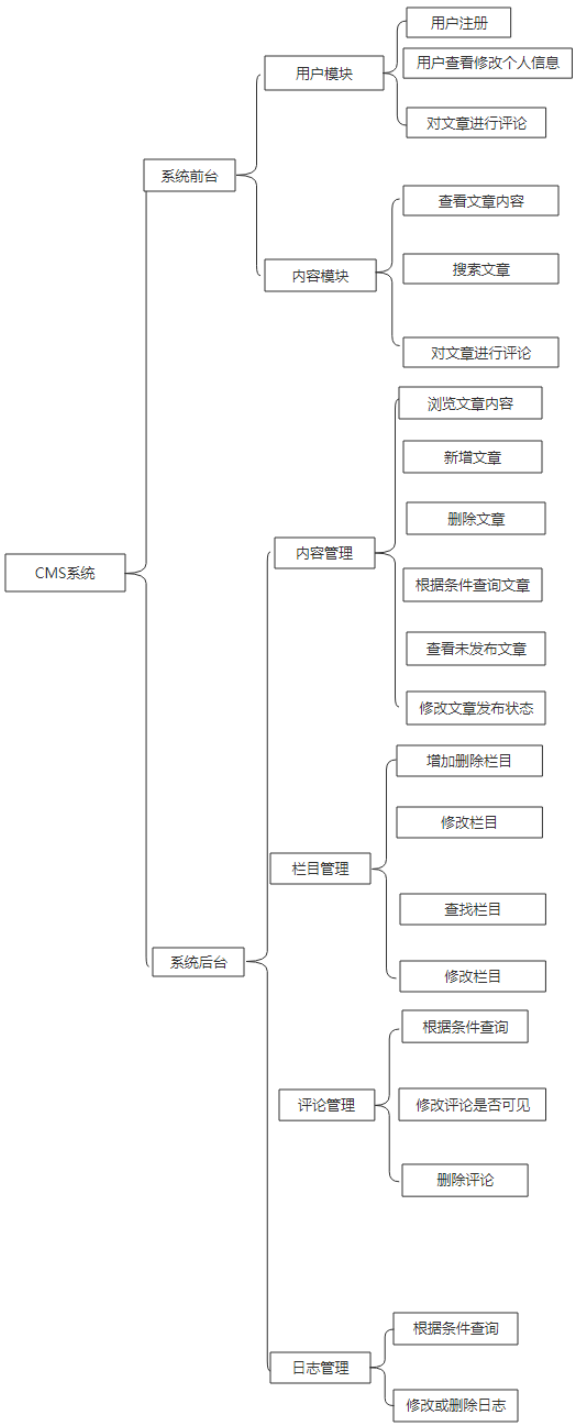
2、有权限的话，先判断商品是否已经审核通过，审核通过的话，才可以将商品改为上架状态；

3、当商品的可销售库存小于 1 的时候，或者商品停止销售的时候等情况（具体情况参考产品下架图），可以将商品设置为下架状态；

4、商品上架的同时，向 ActiveMQ 服务器发送一条消息，这个消息为商品的 ID，然后对应的商品搜索服务、生成商品详情页的服务都同时监听这个消息，当监听到这个消息后，再去更新索引库和生成商品详情页。

二、CMS 内容管理系统

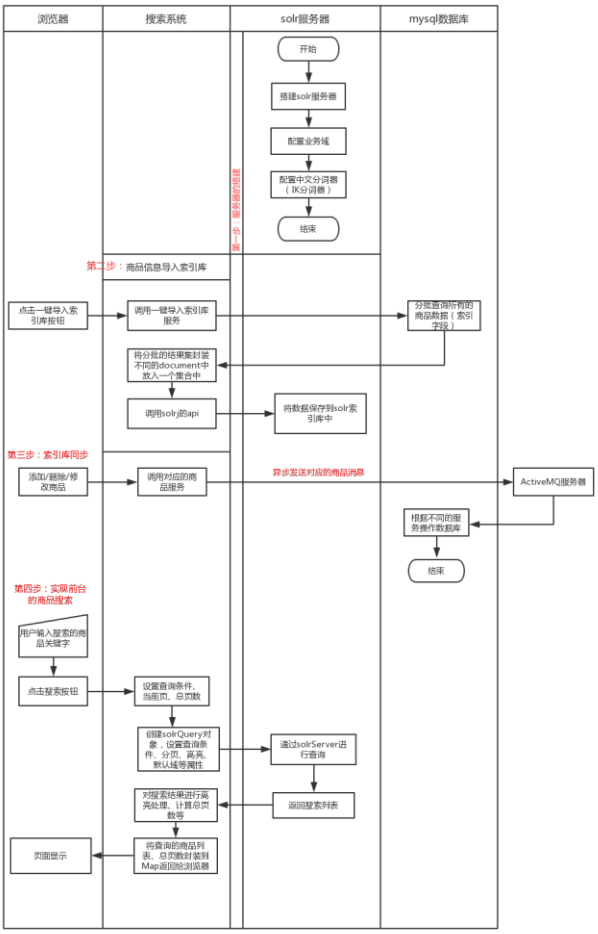
1、业务流程图


CMS系统
系统前台
用户模块
用户注册
用户查看修改个人信息
对文章进行评论
内容模块
查看文章内容
搜索文章
对文章进行评论
系统后台
内容管理
浏览文章内容
新增文章
删除文章
根据条件查询文章
查看未发布文章
修改文章发布状态
栏目管理
增加删除栏目
修改栏目
查找栏目
修改栏目
评论管理
根据条件查询
修改评论是否可见
删除评论
日志管理
根据条件查询
修改或删除日志

2、业务话术

三、搜索系统

1、业务流程图



2、业务话术

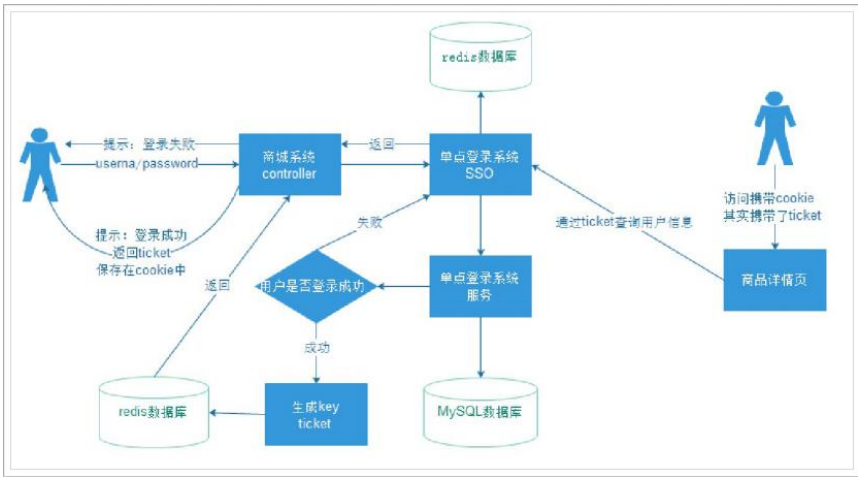
四、商品详情页

1、业务流程图

2、业务话术

五、单点登录

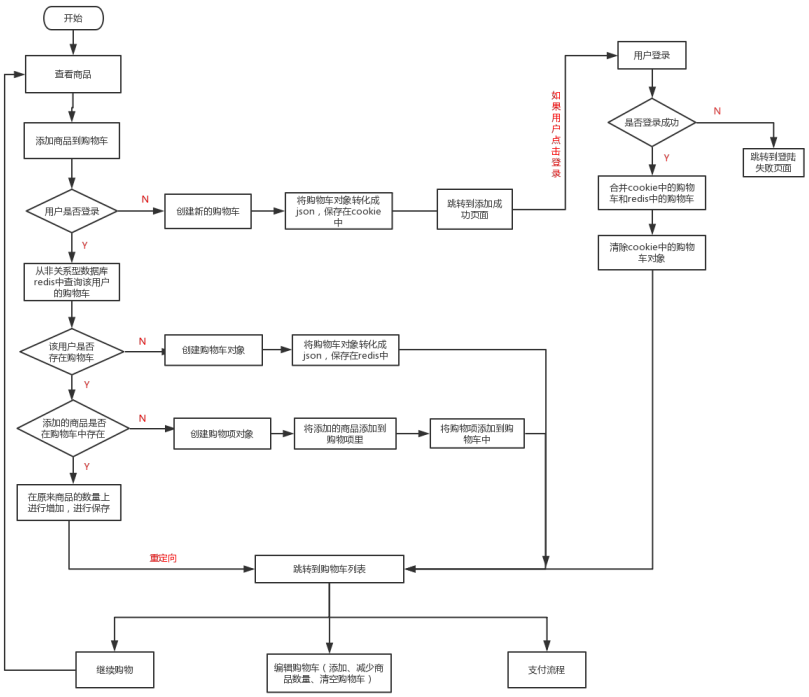
1、业务流程图



2、业务话术

六、购物车

1、业务流程图



2、业务话术

为了提高客户体验度、增大商品的购买量，我们项目对于购物车的设计主要分为两个模块：未登录和已登录两种状态。未登录的时候将购物车的信息放在 cookie 中。

如果已经登录，将购物车列表放入 redis 非关系型数据库中

saveCartListToRedis(username,cartList);

key:username

value：cartList　(对应 redis 中的 hash 类型)

好处：减少与数据库的交互

坏处：可能会产生数据丢失，这里可以使用 redis 的持久化方式 aof，避免数据丢失

cartservice 层（生成者，服务提供方）

方法 1：addGoodsToCartList(List<Cart> cartList, Long itemId, Integer num)

1、判断以前是否有购物车列表 cartList（也就是判断这个用户的购物车是否有东西）

1.1 如果有，在原来的购物车列表 cartList 上进行添加

1.2 如果没有，创建 cartList

2、根据商品 id，查找商品，

1.1 判断商品在数据库中是否存在

--不存在，抛出异常

--存在，则判断商品是否处于销售状态

--非销售状态，抛出异常

3、根据 item（sku 表）得到商家 id

4、根据商家 id 和 cartlist 判断购物车列表中是否有该商家的购物车（商家购物车 Cart）

Cart cart = searchCartListBySellerId(cartList, sellerId);

4.1 如果查不到商家购物车，则在购物车列表中添加商家购物车

a:判断加入购物车的数量大于 0；

b:构建商家购物车，

封装商家购物车数据

构建购物车明细，封装数据

将购物车明细添加到商家购物车中

c:将商家购物车添加到购物车列表中

4.2 如果有商家购物车

4.2.1 根据 SKU 的 id 在购物车明细列表中查询明细对象

TbOrderItem searchOrderItemByItemId(List<TbOrderItem> orderItemList,Long itemId)

a:如果在购物车明细列表中没有查到购物车明细

构建购物车明细，将购物车明细添加到购物车对象中

b:如果查到了，

则在原来购物车明细的基础上对数量进行增加，总金额增加

如果数量操作之后小于等于 0，则移除；如果移除后 cart 的明细数量为 0,则移除 Cart

5、返回购物车列表给消费者

PS: 购物车列表：CartList<Cart>

Cart:商家购物车 商家购物车里面有购物车明细列表 List<TbOrderItem>

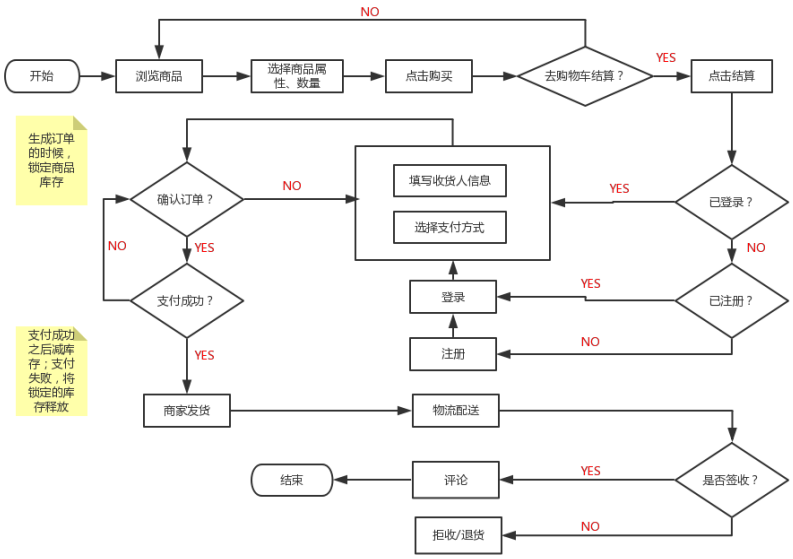
TbOrderItem:购物车明细

一个购物车列表 包含多个商家购物车

一个商家购物车 包含多个购物车明细

七、订单

1、业务流程图



2、业务话术

当用户点击结算的时候，这块我们用 springmvc 的拦截器先判断用户是否登录，如果用户没有登录，则跳转到登陆页面，让用户去登录，登录成功之后，跳转到订单结算页面（这块拦截器在跳转到登录页面的时候，把之前的请求地址保存下来，作为参数进行跳转，在登录成功之后，查看是否有请求参数，如果有就跳转到对应的 url，如果值为 null，跳转到首页）；如果用户登录了，则跳转到订单结算页面；

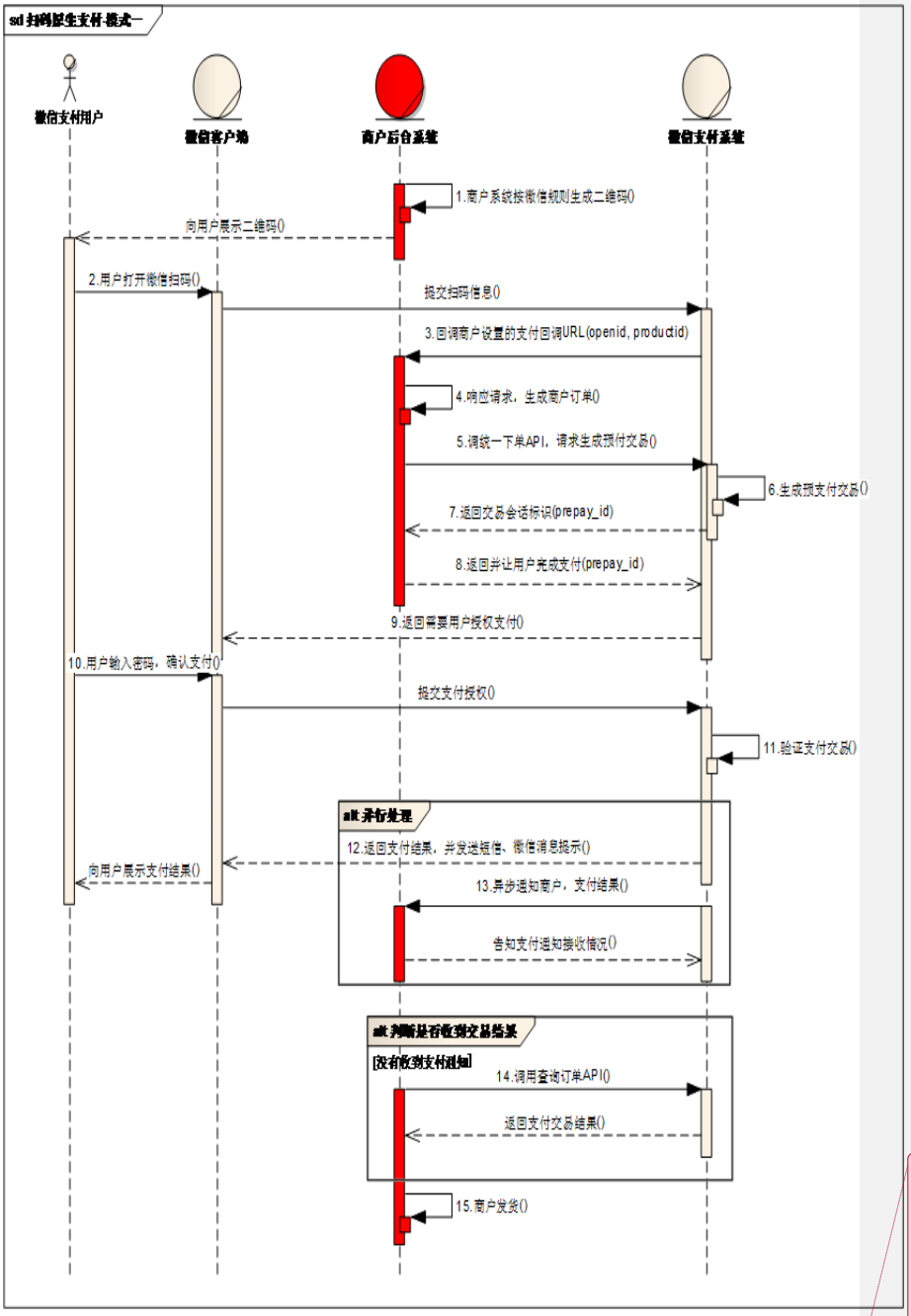
当跳转到订单结算页面的时候，首先对收货人地址进行管理，其次选择支付方式，一期的时候可以先提供了微信扫码支付，确认订单信息，然后提交数据到后台，生成对应的订单商品表、订单表和订单物流表（当订单生成的时候，我们要调用对应的库存系统针对订单的商品数量进行锁定，即锁定库存）。当订单创建成功之后，自动跳转到成功页面（将订单数据和到货时间传递过去）。

这块我们设置的订单的有效时间为 2 天/两个小时（这个时间可以自己定，只要合理就行），因为我们设置了一个定时任务，利用 Quartz 每分钟去检查订单的有效性，如果订单是无效订单，把订单状态修改为关闭订单，当订单为无效订单的时候，我们要释放原来锁定的库存。

批注 [60]: 无效订单：创建时间是两天前/两小时前；订单状态是未付款状态

八、支付

1、业务流程图（微信扫码支付）



业务流程说明：

（1）商户后台系统根据微信支付规定格式生成二维码，展示给用户扫码。

批注 [61]: 携带的参数：
sign:数字签名，根据微信官方提供的密钥和一套算法生成的一个加密信息，就是为了保证交易的安全性
appid:微信分配的公众账号 ID
mchid:商户号
product_id:商品 id
time_stamp:时间戳，一般是系统当前时间
nonce_str:随机字符串

（2）用户打开微信“扫一扫”扫描二维码，微信客户端将扫码内容发送到微信支付系统。

（3）微信支付系统收到客户端请求，发起对商户后台系统支付回调 URL 的调用。调用请求将带 productid 和用户的 openid（用户标识）等参数，并要求商户系统返回数据包。

（4）商户后台系统收到微信支付系统的回调请求，根据 productid 生成商户系统的订单。

（5）商户系统调用微信支付【统一下单 API】（这块通过 httpclient 来请求微信统一下单接口）请求下单，获取交易会话标识（prepay_id）

（6）微信支付系统根据商户系统的请求生成预支付交易，并返回交易会话标识（prepay_id）。

（7）商户后台系统得到交易会话标识 prepay_id（2 小时内有效）。

（8）商户后台系统将 prepay_id 返回给微信支付系统。

（9）微信支付系统根据交易会话标识，发起用户端授权支付流程。

（10）用户在微信客户端输入密码，确认支付后，微信客户端提交支付授权。

（11）微信支付系统验证后扣款，完成支付交易。

（12）微信支付系统完成支付交易后给微信客户端返回交易结果，并将交易结果通过短信、微信消息提示用户。微信客户端展示支付交易结果页面。

（13）微信支付系统通过发送异步消息通知商户后台系统支付结果。商户后台系统需回复接收情况，通知微信后台系统不再发送该单的支付通知。

（14）未收到支付通知的情况，商户后台系统调用【查询订单 API】。

（15）商户确认订单已支付后给用户发货。

九、库存系统

1、可销售库存（S）

可销售库存数（S）是即网站前台显示的库存数值，也是库存结构的最大组成部分。大部分电子商务企业中，前台网站会与后台 WMS（存储管理系统）保持数据同步，并作出判断。当“可销售库存>0”时，这一商品可供购买，前台网站则会显示产品可销售；而一旦“可销售库存<0”时，前台网站则会显示商品缺货。一般所说的缺货并不等于库房中没有库存了，而是指没有可供销售库存情况。

而顾客选购完商品，确认订单时，前台网站系统会首先向后台订单系统发出请求，检查订单产品数量与当前可销售库存数量。若可销售库存数量>订单产品数量，则通知前台网站可进行交易并产生，一张新的预购订单后，该客户预购的商品数量则转化成库存预分配数据以暂时冻结此部分为库存，以便用于后续的发货，系统中可用库存数量减少。否则会通知前台库存不足，提醒客户。

2、订单占用库存（O）

当生成订单时，可用库存数量不断减少，订单占用库存数量增多，变化的数量即订单中的产品数量。

设立订单占用库存的原因在于：订单的生成和库房的发货在时间上是异步的。这

样做的优点在于：保证已经生成订单的库存，这部分客户可以顺利收货；而且客户在下订单时，能够保证有产品发货。若不设立订单占用库存，则会产生客户下订单后，出现库存无货可发的风险。而处理订单时，针对的只是已经被订单所占用的库存，与前台的销售无关。订单出库后，系统中扣减的也只是订单所占用库存。

3、锁定库存（L）

在销售中，经常会使用的一种促销方式是降价，这一方式的效果会非常好，成功的降价促销可以在很短时间内将商品一售而空，可销售库存直接转化为订单占用库存。

但是有一些情况下，销售方并不希望这么快就将所有的库存都售出。有的时候是因为所有库存全部作降价促销的成本很高，有的时候是防止竞争对手的恶意采购，更多的情况下，则是希望将这一产品的降价作为引子，带动网站的流量和整体销售，这就需要将促销分批次进行。

为达到以上的目的，会采用锁定库存（Locked Inventory）的方式。库存被锁定后，无法直接销售。促销进行一段时间后，可用库存为 0，无法继续销售，必须在解除锁定后才能转化为可销售库存，继续进行销售。

4、虚库存（V）

有一些商品，虽然库存中并没有，或者说没有很多，但是供应渠道非常通畅，可以在很短的时间内送到库房中，变为库存；另外一些产品，销售量少，库存的管理难度大，只有当产生订单后，才向供应室采购。这部分不在实际的库存中，但是可以很快采购到的货品就叫虚库存。

虚库存的存在，是为了使前台网站的可销售数量大于实际可销售数量。

5、调拨出占用库存（T）

在很多 B2C 网站中库房不止一个。多个库房的设置，主要是规模大了以后，库存量很大，很难在一个单独的库房中存储。另外也可能在离客户比较近的聚居地设置库存，以满足当地客户需求。

这样在各个库房之间，必然存在着库存的分派和调拨。当产生调拨计划后，调出地库房的某一部分库存就会被占用，这部分就会被称为调拨占用库存。和订单占用库存的性质有点类似。

6、调拨中占用库存（A）

库存的调拨，必然会存在一段时间，库存既不存在于调拨出库房，也不存在于调拨入库房，这一部分库存就像漂在空中一样，称为调拨中库存。

7、不可销售库存（U）

产品由于破损、性能故障、型号标错等原因导致产品无法销售，在系统中必须有相应的状态。这部分库存被称为不可销售库存。