

你知道的越多，你不知道的越多

点赞再看，养成习惯

本文 **GitHub** <https://github.com/JavaFamily> 已收录，有一线大厂面试题思维导图，也整理了很多我的文档，欢迎Star和完善，大家面试可以参照考点复习，希望我们一起有点东西。

前言

作为一个在互联网公司面一次拿一次Offer的面霸，打败了无数竞争对手，每次都只能看到无数落寞的身影失望的离开，略感愧疚（请允许我使用一下夸张的修辞手法）。

于是在一个寂寞难耐的夜晚，我痛定思痛，决定开始写互联网技术栈面试相关的文章，希望能帮助各位读者以后面试势如破竹，对面试官进行360°的反击，吊打问你的面试官，让一同面试的同僚瞠目结舌，疯狂收割大厂Offer！

所有文章的名字只是我的噱头，我们应该有一颗谦逊的心，所以希望大家怀着空杯心态好好学，一起进步。

捞一下

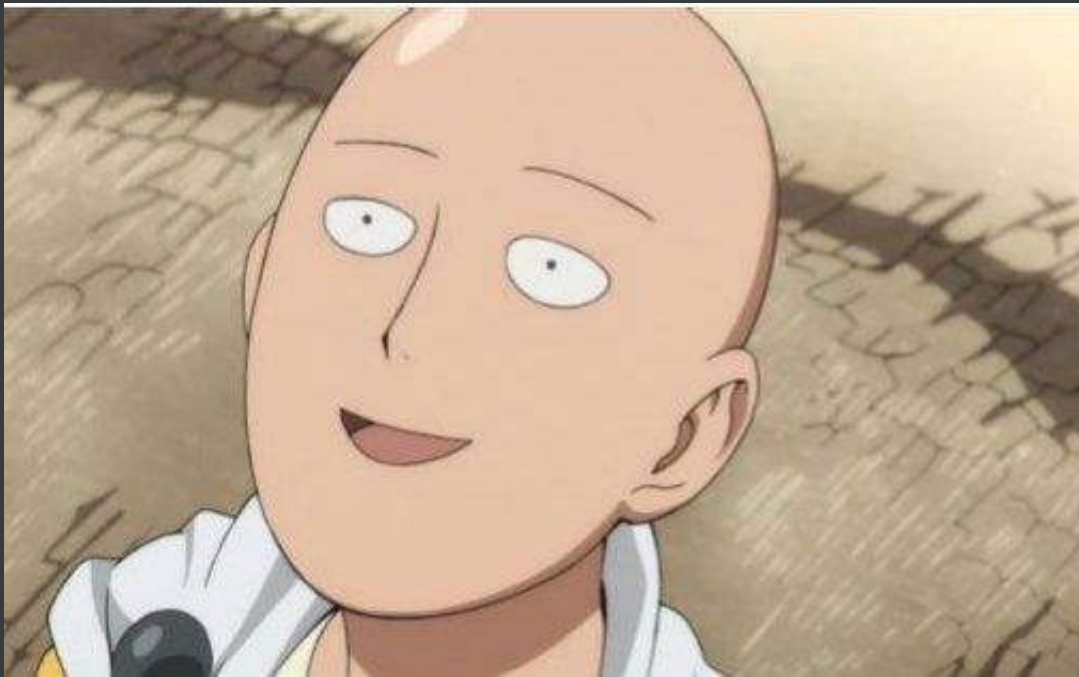
上一期，简单的介绍了一下**消息队列**的基础知识，里面有消息队列的应用场景，以及使用之后可能带来的问题，但是上期没对怎么解决这些问题做回答，因为要控制篇幅嘛（明明是自己觉得MQ写不了多少期，要多怼一期出来！渣男）

咳咳，我们言归正传，没看的朋友去看一下，有助于这期的阅读：

[《吊打面试官》系列-消息队列基础](#)

面试开始

一个风度翩翩，穿着格子衬衣的中年男子，拿着一个满是划痕的mac向你走来，看着锃亮的头，心想着肯定是尼玛顶级架构师吧！但是我们看过暖男敖丙的系列，腹有诗书气自华，虚都不虚。



没错小伙子还是我，上次话说一半你就溜了，这次我非得好好的问问你。

好的面试官，因为上次着急，敖丙的系列更新了所以赶回家去看了！

我信你个鬼，我们开始吧，上次说到了消息队列的消息重复消费，你能跟我介绍这是怎样子的场景么？

消息**重复消费**是使用消息队列之后，必须考虑的一个问题，也是比较严重和常见的问题，**帅丙**我在开发过程中，但凡用到了消息队列，我第一时间考虑的就是**重复消费**的问题。

就比如有这样的一个场景，用户下单成功后我需要去一个活动页面给他加**GMV**（销售总额），最后根据他的GMV去给他发奖励，这是电商活动很常见的玩法。

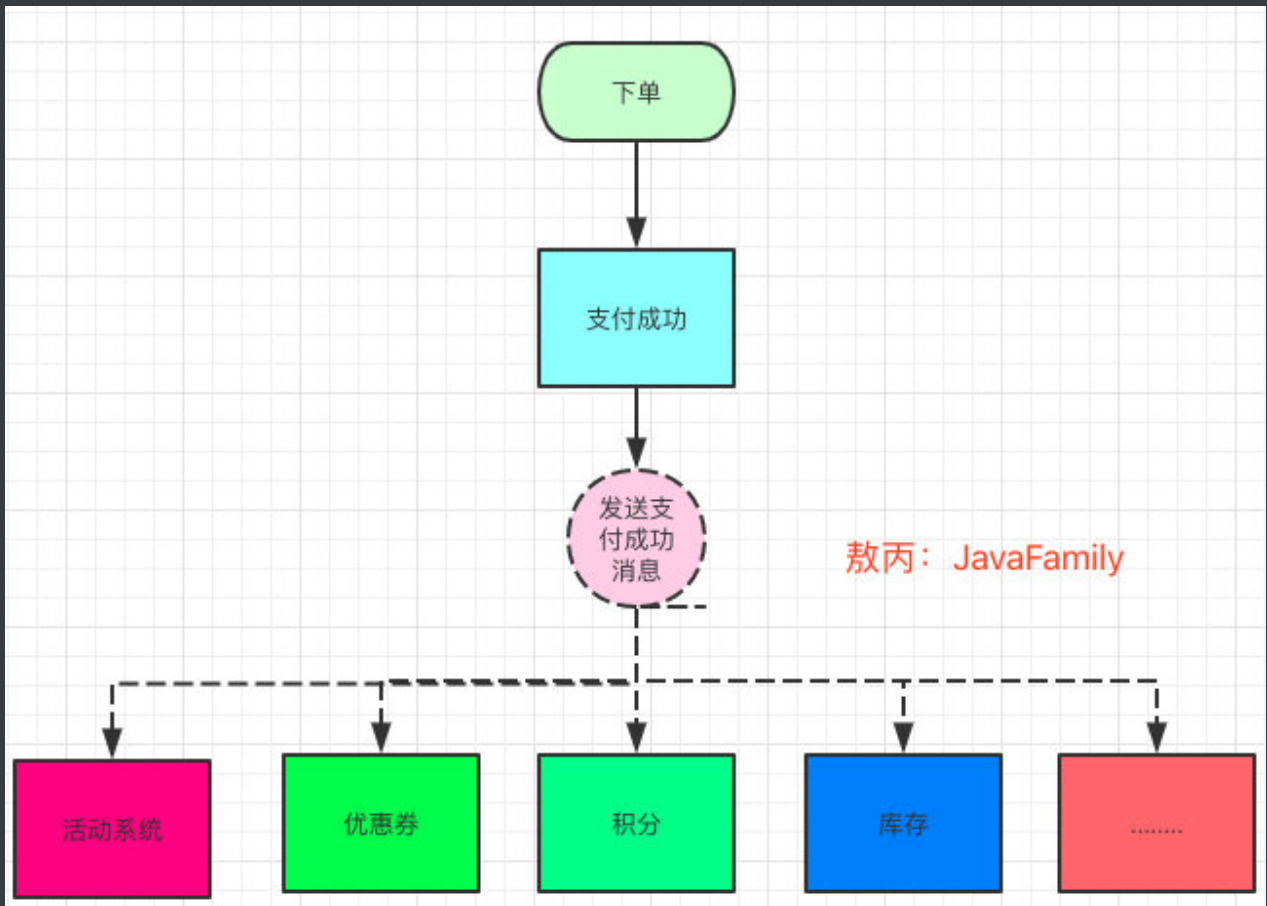
类似累计下单金额到哪个梯度给你返回什么梯度的奖励这样。



我只能告诉你这样的活动页面10000%是用异步去加的（别问我为什么，因为这个活动的后端是敖丙我做的😂），不然你想，你一个用户下一单就给他加一下，那就意味着对那张表就要操作一下，你考虑下双十一当天多少次对这个表的操作？这数据库或者缓存都顶不住吧。

而且大家应该也有这样的体会，你下单了马上去看一些活动页面，有时候马上就有了，有时候却延迟有很久，为啥？这个速度取决于消息队列的消费速度，消费慢堵塞了就迟点看到呗。

你下个单支付成功你就发个消息出去，我们上面那个活动的开发人员就监听你的支付成功消息，我监听到你这个订单成功支付的消息，那我就去我活动GMV表里给你加上去，听到这里大家可能觉得顺理成章。



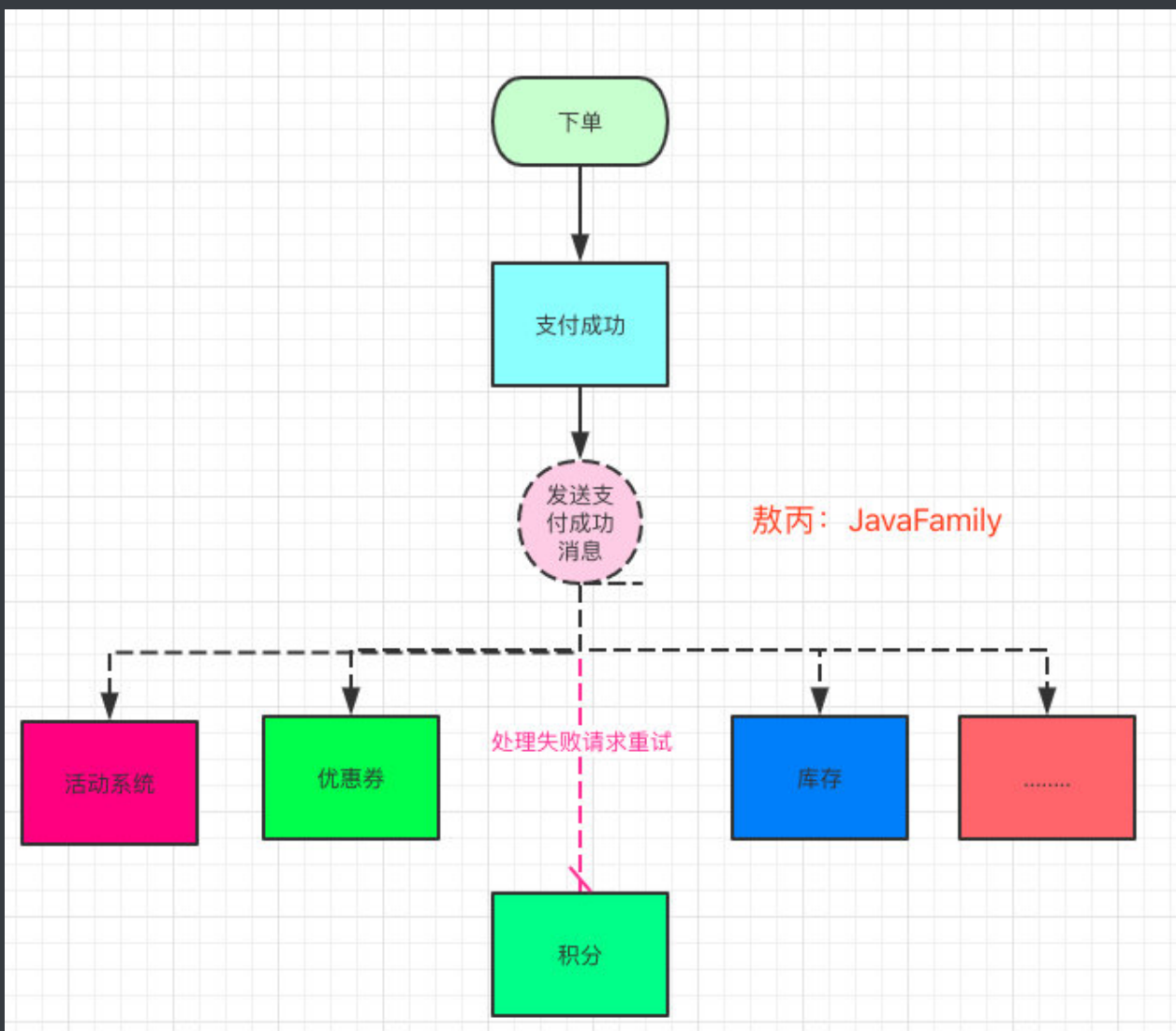
但是我告诉大家一般消息队列的使用，我们都是有**重试机制**的，就是说我下游的业务发生异常了，我会抛出异常并且要求你**重新发一次**。

我这个活动这里发生错误，你要求重发肯定没问题。但是大家**仔细想一下**问题在哪里？

是的，不止你一个人监听这个消息啊，**还有别的服务也在监听**，他们也会失败啊，他一失败他也要求重发，但是你这里其实是成功的，重发了，你的钱不就加了两次了？

对不对？？是不是这个道理？？

还不理解？看下面↓



就好比上面的这样，我们的**积分系统处理失败了**，他这个系统肯定要求你**重新发送**一次这个消息对吧，积分的系统重新接收并且处理成功了，但是别人的活动，优惠券等服务也**监听了这个消息**呀，那不就可能出现活动系统给他加GMV加两次，优惠券扣两次这种情况么？

真实的情况其实重试是很正常的，服务的**网络抖动**，**开发人员代码Bug**，还有**数据问题**等都可能处理失败要求重发的。

嗯小伙子分析得很仔细嘛，那你在开发过程中是怎么去保证的呀？

一般我们叫这样的处理叫**接口幂等**。

幂等 (idempotent、idempotence) 是一个数学与计算机学概念，常见于抽象代数中。

在编程中一个幂等操作的特点是其任意多次执行所产生的影响均与一次执行的影响相同。

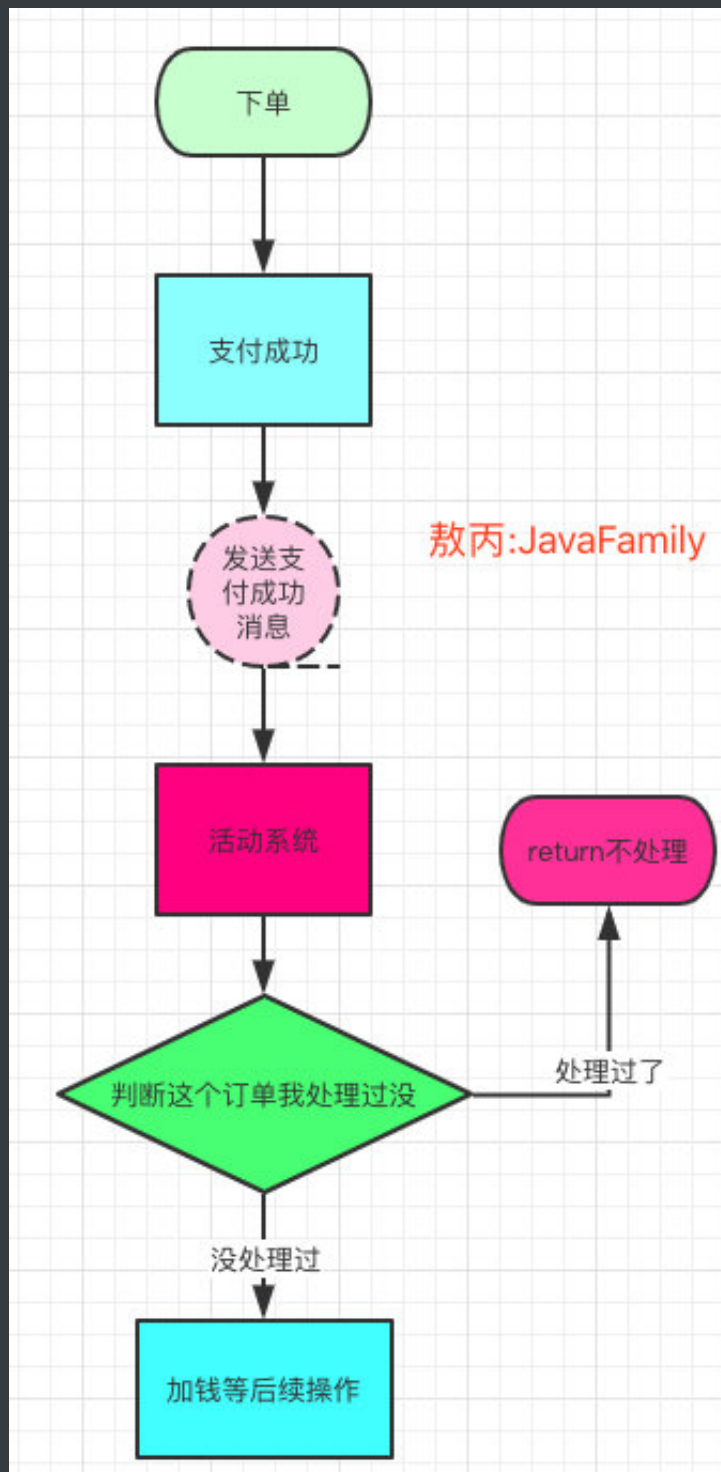
幂等函数，或幂等方法，是指可以使用相同参数重复执行，并能获得相同结果的函数。这些函数不会影响系统状态，也不用担心重复执行会对系统造成改变。

例如，“setTrue()”函数就是一个幂等函数,无论多次执行，其结果都是一样的.更复杂的操作幂等保证是利用唯一交易号(流水号)实现.

通俗了讲就是你同样的参数调用我这个接口，调用多少次结果都是一个，你加GMV同一个订单号你加一次是多少钱，你加N次都还是多少钱。

但是如果**不做幂等**，你一个订单调用多次钱不就加多次嘛，同理你退款调用多次钱也就减多次了。

大致处理流程如下：



那怎么保证呢？

一般**帅丙**我是这么回答的：

帅气面试官您好，一般**幂等**，我会**分场景去考虑**，看是**强校验**还是**弱校验**，比如跟金钱相关的场景那就很关键呀，就做强校验，别不是很重要的场景做弱校验。

强校验：

比如你监听到用户支付成功的消息，你监听到了去加GMV是不是要调用加钱的接口，那加钱接口下面再调用一个加流水的接口，**两个放在一个事务，成功一起成功失败一起失败。**

每次消息过来都要拿着**订单号+业务场景这样的唯一标识**（比如天猫双十一活动）去流水表查，看看有没有这条流水，有就直接return不要走下面的流程了，没有就执行后面的逻辑。

之所以用**流水表**，是因为涉及到金钱这样的活动，有啥问题后面也可以去流水表**对账**，还有就是帮助开发人员定位问题。

有的小伙伴可能还是有点懵，然后**人才交流群**的小伙伴也说有些例子可以放一点伪代码，那这期开始能用代码写的我也写点。

```
/**
 * 强校验幂等伪代码演示 这都是简单的伪代码真实情况复杂一点 但是大的逻辑是这样
 *
 * @param orderId
 * @Author: 敖丙 JavaFamily
 */
public void process(String orderId) {
    try {
        // 查询这个订单是否存在这个活动加GMV的流水
        Object gmvFlow = getFlowByOrderId("addGmv" + orderId);
        if (Objects.isNull(gmvFlow)) {
            // 不存在流水 去加GMV和加流水 注意这两个在一个事务的 回滚就一起回滚了
            addGmvAndFlow(orderId);
        } else {
            // 存在流水证明加过了 返回就好了
            return;
        }
    } catch (Exception e) {
        // 发送异常 触发消息队列框架重试机制
    }
}
```

Tip：**GitHub** <https://github.com/JavaFamily> 上有进群方式和个人联系方式，说实话在这个群，哪怕您不说话，光看聊天记录，都能学到东西（美团王炸，三歪（Java3y），并夕夕等的大佬都在）。

弱校验：

这个简单，一些不重要的场景，比如给谁发短信啥的，我就把这个id+场景唯一标识作为**Redis**的key，放到缓存里面失效时间看你场景，**一定时间内的**这个消息就去Redis判断。

用KV就算消息丢了可能这样的场景也没关系，反正丢条**无关痛痒**的通知短信嘛（你敢说你没验证码短信

丢失的情况？）。



还有很多公司的弱校验用**token**啊什么的，反正花样很多，但是**重要的场景一定要强校验**，真正查问题的时候没有在磁盘持久化的数据，心里还是空空的，就像你和女朋友分开的时候的心里状态一样。（我单身的怎么知道这种感觉？猜的）

你们有接触过消息顺序消费这样的场景么？你怎么保证的？

没有！over！

乖，你肯定不能说没有啊，就算真的没有，你看过**敖帅丙**的文章都要说有！

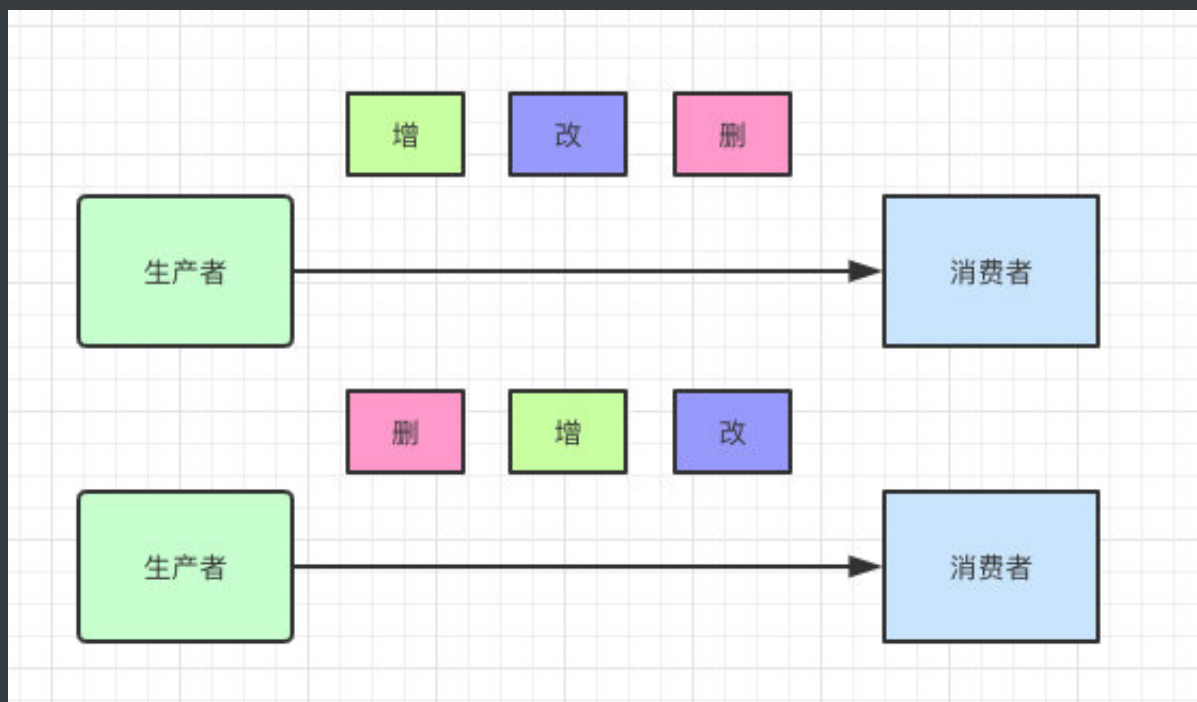
Tip：但是说实话**顺序消费**这里很难介绍，我上周到这周问了很多身边的师兄开发过程中这样的场景不多，我跟三歪也讨论了几次，网上更多的都是介绍binlog的同步，好像更多的场景就没了。

一般都是**同个业务场景下不同几个操作的消息同时过去**，本身顺序是对的，但是你发出去的时候同时发出去了，消费的时候却乱掉了，这样就有问题了。

我之前做电商活动也是有这样的例子，我们都知道数据量大的时候数据同步压力还是很大的，有时候数据量大的表需要同步几个亿的数据。（并不是主从同步，主从延迟大的话会有问题，可能是从数据库或者主数据库同步到**备库**）

这种情况我们都是怼到队列里面去，然后慢慢消费的，那问题就来了呀，我们在数据库同时对一个Id的数据进行了增、改、删三个操作，但是你消息发过去消费的时候变成了改，删、增，这样数据就不对了。

本来一条数据应该删掉了，结果在你那却还在，这不是**出大问题**！



两者的结果是不是完全不一样了 ↑

那你怎么解决呢？

我简单的说一下我们使用的**RocketMQ**里面的一个简单实现吧。

Tip: 为啥用**RocketMQ**举例呢，这玩意是阿里开源的，我问了下身边的朋友很多公司都有使用，所以读者大概率是这个的话我就用这个举例吧，具体的细节我后面会在**RocketMQ**和**Kafka**各自章节说到。

生产者消费者一般需要保证顺序消息的话，可能就是一个业务场景下的，比如订单的创建、支付、发货、收货。

那这些东西是不是一个订单号呢？一个订单的肯定是一个订单号的，那简单了呀。

一个**topic**下有多个队列，为了保证发送有序，**RocketMQ**提供了**MessageQueueSelector**队列选择机制，他有三种实现：

```
public interface MessageQueueSelector {
    MessageQueue select(List<MessageQueue> var1, Message var2, Object var3);
}
```

Choose Implementation

- Anonymous in send1() in OrderMessageTest (com.lei.record)
- SelectMessageQueueByHash (org.apache.rocketmq.client.producer.selector)
- SelectMessageQueueByMachineRoom (org.apache.rocketmq.client.producer.selector)
- SelectMessageQueueByRandom (org.apache.rocketmq.client.producer.selector)

我们可使用**Hash取模法**，让同一个订单发送到同一个队列中，再使用同步发送，只有同个订单的创建消息发送成功，再发送支付消息。这样，我们保证了发送有序。

RocketMQ的topic内的队列机制,可以保证存储满足**FIFO**（First Input First Output 简单说就是指先进先出）,剩下的只需要消费者顺序消费即可。

RocketMQ仅保证顺序发送，顺序消费由消费者业务保证!!!

这里很好理解，一个订单你发送的时候放到一个队列里面去，你同一个的订单号Hash一下是不是还是一样的结果，那肯定是一个消费者消费，那顺序是不是就保证了？

真正的顺序消费不同的中间件都有自己的不同实现我这里就举个例子，大家思路理解下。

Tip：我写到这点的时候人才群里也有人问我，一个队列有序出去，一个消费者消费不就好了，我想说的是**消费者是多线程**的，你消息是有序的给他的，你能保证他是有序的处理的？还是一个消费成功了再发下一个**稳妥**。

你能跟我聊一下分布式事务么？

分布式事务在现在遍地都是分布式部署的系统中几乎是必要的。

我们先聊一下啥是**事务**？

分布式事务、事务隔离级别、ACID我相信大家这些东西都耳熟能详了，那什么是事务呢？

概念：

一般是指要做的或所做的事情。

在计算机术语中是指访问并可能更新数据库中各种数据项的一个程序执行单元(unit)。

事务通常由高级数据库操纵语言或编程语言（如SQL，C++或Java）书写的用户程序用户程序的执行所引起，并用形如**begin transaction**和**end transaction**语句（或函数调用）来界定。

事务由事务开始(**begin transaction**)和事务结束(**end transaction**)之间执行的全体操作组成。

特性：

事务是恢复和并发控制的基本单位。

事务应该具有4个属性：**原子性、一致性、隔离性、持久性**。这四个属性通常称为**ACID特性**。

原子性（atomicity）：一个事务是一个不可分割的工作单位，事务中包括的操作要么都做，要么都不做。

一致性 (consistency)：事务必须是使数据库从一个一致性状态变到另一个一致性状态。一致性与原子性是密切相关的。

隔离性 (isolation)：一个事务的执行不能被其他事务干扰。即一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰。

持久性 (durability)：持久性也称永久性 (permanence)，指一个事务一旦提交，它对数据库中数据的改变就应该是永久性的。接下来的其他操作或故障不应该对其有任何影响。

那有同学还是不理解，敖丙我总结了一下就是：**事务就是一系列操作，要么同时成功，要么同时失败。**然后会从事务的 **ACID** 特性（原子性、一致性、隔离性、持久性）展开叙述。

事务就是为了保证一系列操作可以正常执行，它必须同时满足 **ACID** 特性。

那什么是分布式事务呢？

大家可以想一下，你下单流程可能涉及到10多个环节，你下单付钱都成功了，但是你优惠券扣减失败了，积分新增失败了，前者公司会被薅羊毛，后者用户会不开心，但是**这些都在不同的服务怎么保证大家都成功呢？**

聪明，**分布式事务**，你看你都会抢答了！

Tip：真实的应用场景可能比我介绍的场景复杂数倍，我只是为了举例方便一下大家理解所以用了很简单的例子。

我接触和了解到的分布式事务大概分为：

- 2pc（两段式提交）
- 3pc（三段式提交）
- TCC（Try、Confirm、Cancel）
- 最大努力通知
- XA
- 本地消息表（ebay研发出的）
- 半消息/最终一致性（RocketMQ）

这里我就介绍下最简单的**2pc（两段式）**，以及大家以后可能比较常用的**半消息事务**也就是**最终一致性**，目的是让大家理解下分布式事务里面**消息中间件的作用**，别的事务都大同小异，都有很多优点。

当然也都有**种种弊端**：

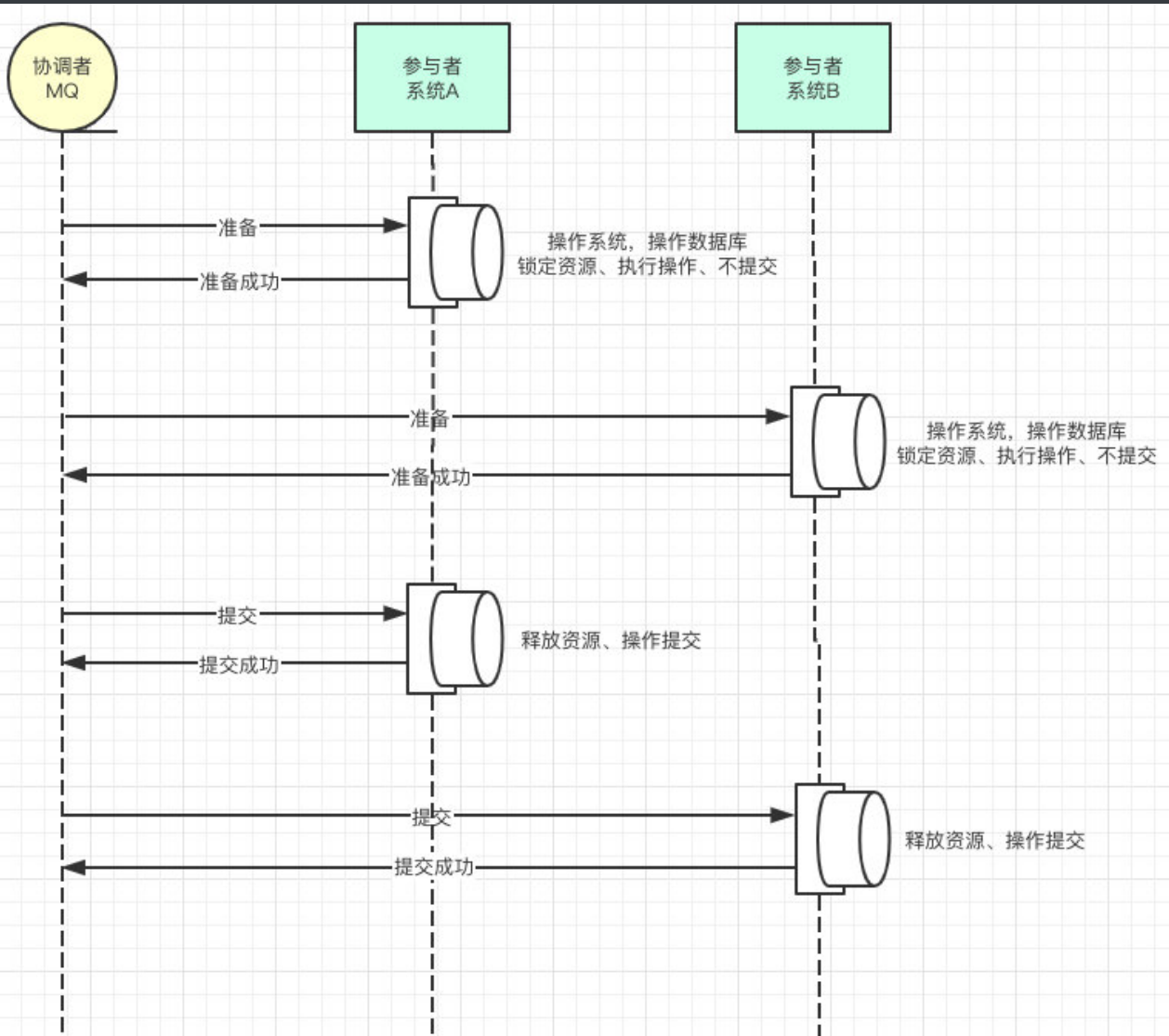
例如**长时间锁定数据库资源**，导致系统的**响应不快**，**并发上不去**。

网络抖动出现**脑裂**情况，导致事物参与者，不能很好地执行协调者的指令，导致**数据不一致**。

单点故障：例如事物协调者，在某一时刻宕机，虽然可以通过选举机制产生新的Leader，但是这过程中，必然出现问题，而TCC，只有强悍的技术团队，才能支持开发，**成本太高。**

不多BB了，我们开始介绍这个两个事物吧。

2pc（两段式提交）：

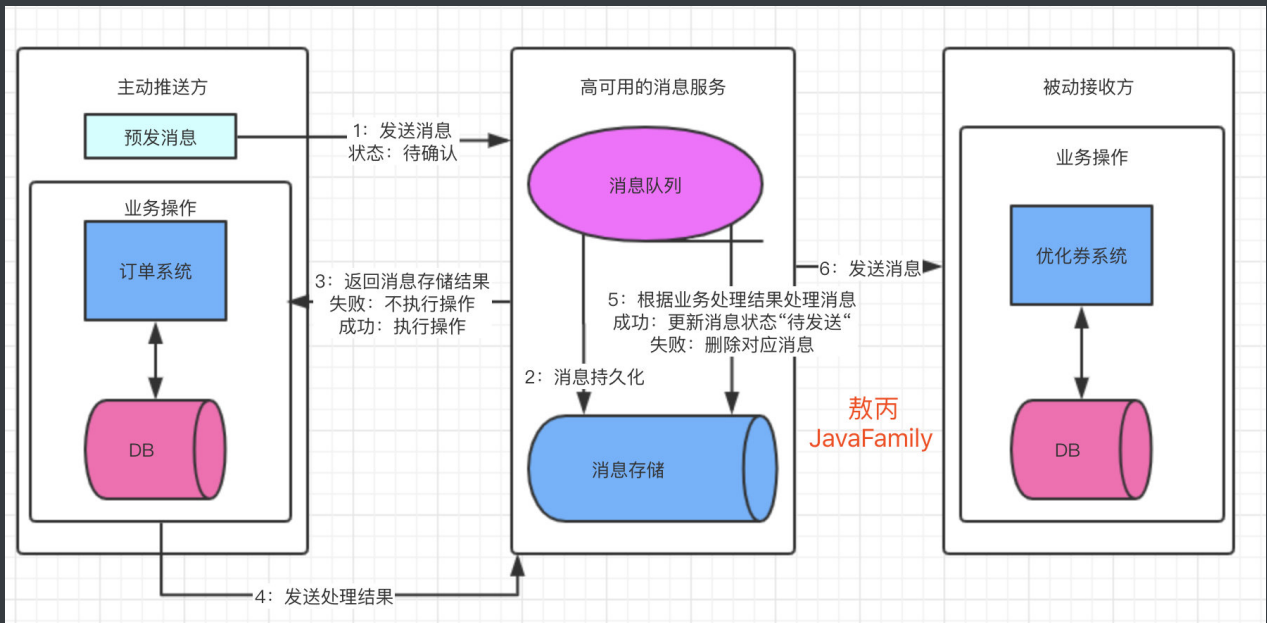


2pc（两段式提交）可以说是分布式事务的最开始的样子了，像极了**媒婆**，就是通过消息中间件协调多个系统，在两个系统操作事务的时候都锁定资源但是不提交事务，等两者都准备好了，告诉消息中间件，然后再分别提交事务。

但是我不知道大家看到问题所在没有？

是的你可能已经发现了，如果A系统事务提交成功了，但是B系统在提交的时候网络波动或者各种原因提交失败了，其实还是会失败的。

最终一致性：



整个流程中，我们能保证是：

- 业务主动方本地事务提交失败，业务被动方不会收到消息的投递。
- 只要业务主动方本地事务执行成功，那么消息服务一定会投递消息给下游的业务被动方，并最终保证业务被动方一定能成功消费该消息（消费成功或失败，即最终一定会有一个最终态）。

不过呢技术就是这样，各种极端的情况我们都需要考虑，也很难有完美的方案，所以才会有这么多的方案三段式、TCC、最大努力通知等等分布式事务方案，大家只需要知道为啥要做，做了有啥好处，有啥坏处，在实际开发的时候都注意下就好了，系统都是根据业务场景设计出来的，离开业务的技术没有意义，离开技术的业务没有底气。

还是那句话：没有最完美的系统，只有最适合的系统。

面试结束

小伙子看不出来啊，还是有点东西的嘛，这几个点都回答的不错，明天你能跟我聊一下RocketMQ么？

敖丙这章花了这么多时间，不确定他写不写的完，心疼他。好想给他点赞啊，消息回溯也在单独介绍消息中间件的时候介绍吧，这章篇幅有点长了。

总结


这章其实我写的时间比之前的秒杀还要久，因为顺序消息这个场景我不知道怎么讲出来大家容易懂一点，最后就参考了网上的，顺序消息的实际应用场景没别的那么广泛，跟3y也聊了好几次，最后定了这个binlog的场景。

总之就是这期创作源泉有点枯竭，这章是真的难写，包括分布式事务在实际开发过程中也是很复杂的环节，需要用的时候光是做设计都要很久，反正我的流程图长得一匹。

我每次都想着写得**通俗易懂**一点，这篇即使是这样我觉得还是不够通俗易懂，但是消息的场景就是这样，还有大家加我也不要一上来就问我很多扣细节的点，**自己多点思考我觉得可能帮助比我告诉你答案好很多吧？**

絮叨


敖丙我呀，这周**有牌面**哟，上了**CSDN**的原力计划榜单，而且奖金高达50块！！

 《吊面试官》系列-Redis常见面试题（带答案）

敖、丙

钱不多但是很开心，跟老妈聊到她也觉得我出息了，刚好她生日，以前我们这一家人就是那种不过生日的，不过呀今年我工作了，而且**有牌面**的我拿了的奖金就很关键，偷偷叫表弟悄悄去给她买了蛋糕和礼物🎁，嘻嘻，开心。🎂

DISS

#1楼 2019-11-13 08:54 

多年经验告诉你，面试的时候要谦虚，装傻，特别是大公司，你要是表现的什么都懂，比面试你的人还强，肯定不会录用你的

这是**博客园**的一个网友在我文章下面的评论，说实话不知道**大家怎么看的**，我只想说：呵呵！傻*

我不知道这个多年的经验到底是怎么样子的多年的经验，我本来其实不准备说出来的，因为我发现我群里很多都是还没毕业的**大学生**或者**应届生**，那就假设我读者还有很多这样的学生，他们都没**社会经验**我怕他们被这样的人给误导了。

我记得我在群里说过：

学生没出社会

我们就是社会了

以身作则各位

对 价值观一定要 导向正确

好了。关于学习。我要给丙丙。动力了。日常催更。

我可以80%肯定的告诉大家他这个观点就是扯淡，还有那20%我是**认同他的谦虚那个观点**，但是**谦虚难道不应该是我们对待事物最基本的态度嘛？**

但是**面试装傻**这个观点？还有什么**不会要比你强的人**这个观点？技术人我相信也有面试官也在看我的文章，你们在面试的时候，我想遇到厉害的人巴不得招入麾下，为自己冲锋陷阵吧。

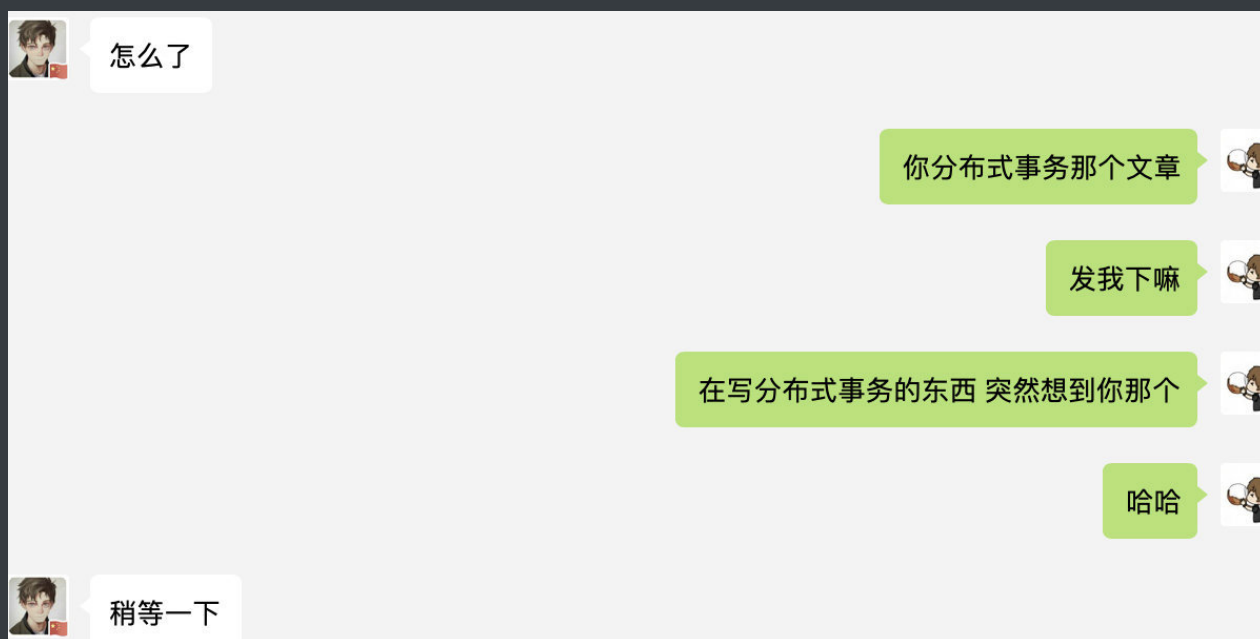
而且**正常面试**的时候你是1-3年的经验，面试你的基本上都是3年以上的，然后依次顺推，当然也有很多很厉害的Leader（我前东家Leader95年的，字节跳动某产品线很强的Leader96的等等）等大家工作了你就会发现有些东西**没有时间积累**是学不到的，你要做的只是一步一个脚印踏实走好就好了。

那些人不管年轻与否能坐在那面试你**肯定有他的原因**，那你有什么才华，你**尽情施展**，他没那个度量包容你的优秀，这样的公司不去也罢，但是技术人这样的真的很少，程序员是一群很崇拜能力的人。

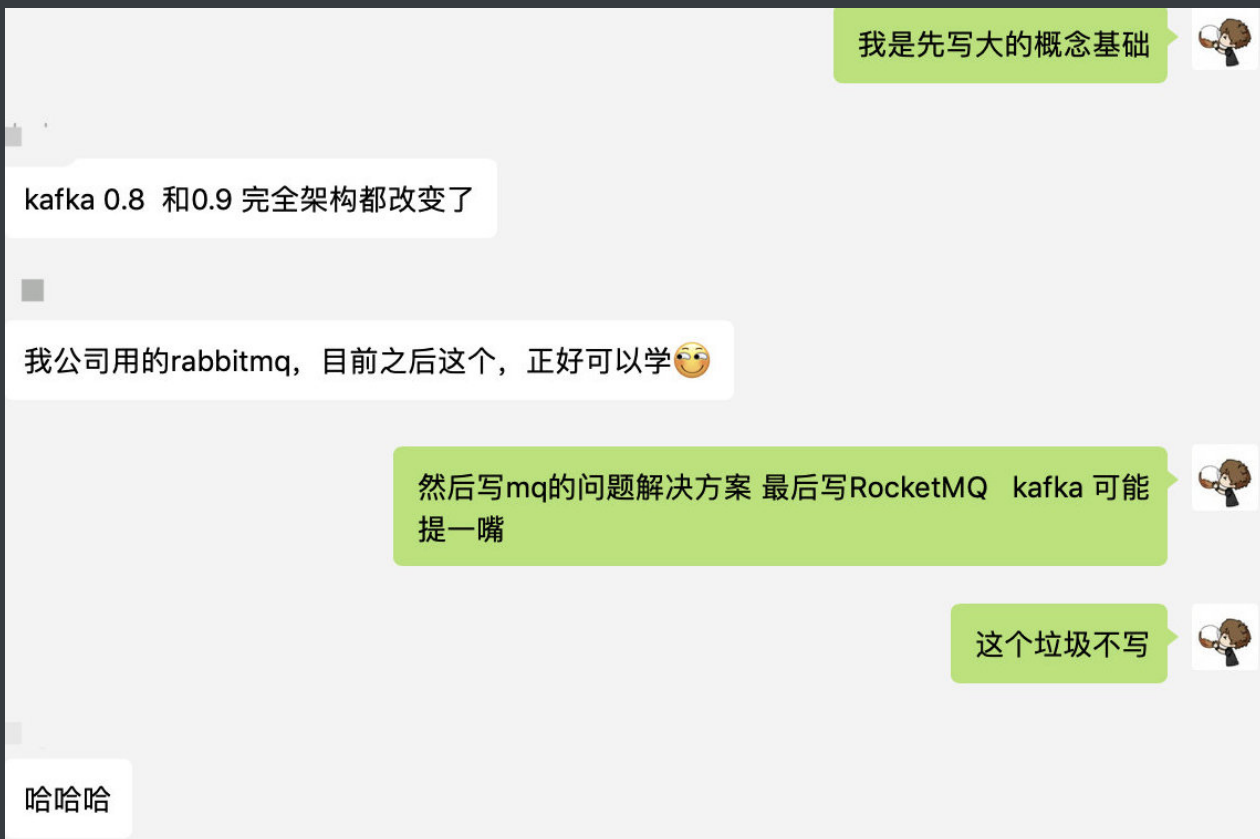
所以**面试你有啥都秀出来**，把你的才华尽情的展示出来，风就在那，你只管飞翔。

鸣谢

涉及到分布式事务的环节我参考了前大神同事：**鲁班**（花名）的技术分享，很感谢他的文章给的思路，还有问题的解析！



每次写我都会在群里问大家，下次大家都在我的交流群里面也可以多给我点意见，谢谢了。



看到没，就很民主。（敖丙你个渣男，呸，自己不会就不写！）

Tip: **GitHub** <https://github.com/JavaFamily> 上有进群方式和个人联系方式，说实话在这个群，哪怕您不说话，光看聊天记录，都能学到东西（美团王炸，三歪（Java3y），并夕夕等的大佬都在）。

点关注，不迷路

好了各位，以上就是这篇文章的全部内容了，我是敖丙，励志做一名让大家都记得住的博主，能看到这里的人呀，都是人才。

我后面会每周都更新几篇一线互联网大厂面试和常用技术栈相关的文章，非常感谢人才们能看到这里，如果这个文章写得还不错，觉得「敖丙」我有点东西的话 **求点赞**👍 **求关注**❤️ **求分享**👥 对暖男我来说真的 **非常有用**!!!

白嫖不好，创作不易，各位的支持和认可，就是我创作的最大动力，我们下篇文章见！

敖丙 | 文 【原创】

如果本篇博客有任何错误，请批评指教，不胜感激！

文章每周持续更新，可以微信搜索「**三太子敖丙**」第一时间阅读和催更（比博客早一到两篇哟），本文 **GitHub** <https://github.com/JavaFamily> 已经收录，有一线大厂面试点思维导图，也整理了很多我的文档，欢迎Star和完善，大家面试可以参照考点复习，希望我们一起有点东西。