31 | 如何应对接口级的故障？

2018-07-07 李运华

31 | 如何应对接口级的故障？
朗读人：黄洲君　10′17″ | 4.71M

异地多活方案主要应对系统级的故障，例如，机器宕机、机房故障、网络故障等问题，这些系统级的故障虽然影响很大，但发生概率较小。在实际业务运行过程中，还有另外一种故障影响可能没有系统级那么大，但发生的概率较高，这就是今天我要与你聊的如何应对接口级的故障。

接口级故障的典型表现就是系统并没有宕机，网络也没有中断，但业务却出现问题了。例如，业务响应缓慢、大量访问超时、大量访问出现异常（给用户弹出提示“无法连接数据库”），这类问题的主要原因在于系统压力太大、负载太高，导致无法快速处理业务请求，由此引发更多的后续问题。例如，最常见的数据库慢查询将数据库的服务器资源耗尽，导致读写超时，业务读写数据库时要么无法连接数据库、要么超时，最终用户看到的现象就是访问很慢，一会访问抛出异常，一会访问又是正常结果。

导致接口级故障的原因一般有下面几种：

- 内部原因

程序 bug 导致死循环，某个接口导致数据库慢查询，程序逻辑不完善导致耗尽内存等。

- 外部原因

黑客攻击、促销或者抢购引入了超出平时几倍甚至几十倍的用户，第三方系统大量请求，第三方系统响应缓慢等。

解决接口级故障的核心思想和异地多活基本类似：优先保证核心业务和优先保证绝大部分用户。

降级

降级指系统将某些业务或者接口的功能降低，可以是只提供部分功能，也可以是完全停掉所有功能。例如，论坛可以降级为只能看帖子，不能发帖子；也可以降级为只能看帖子和评论，不能发评论；而 App 的日志上传接口，可以完全停掉一段时间，这段时间内 App 都不能上传日志。

降级的核心思想就是丢车保帅，优先保证核心业务。例如，对于论坛来说，90% 的流量是看帖子，那我们就优先保证看帖的功能；对于一个 App 来说，日志上传接口只是一个辅助的功能，故障时完全可以停掉。

常见的实现降级的方式有：

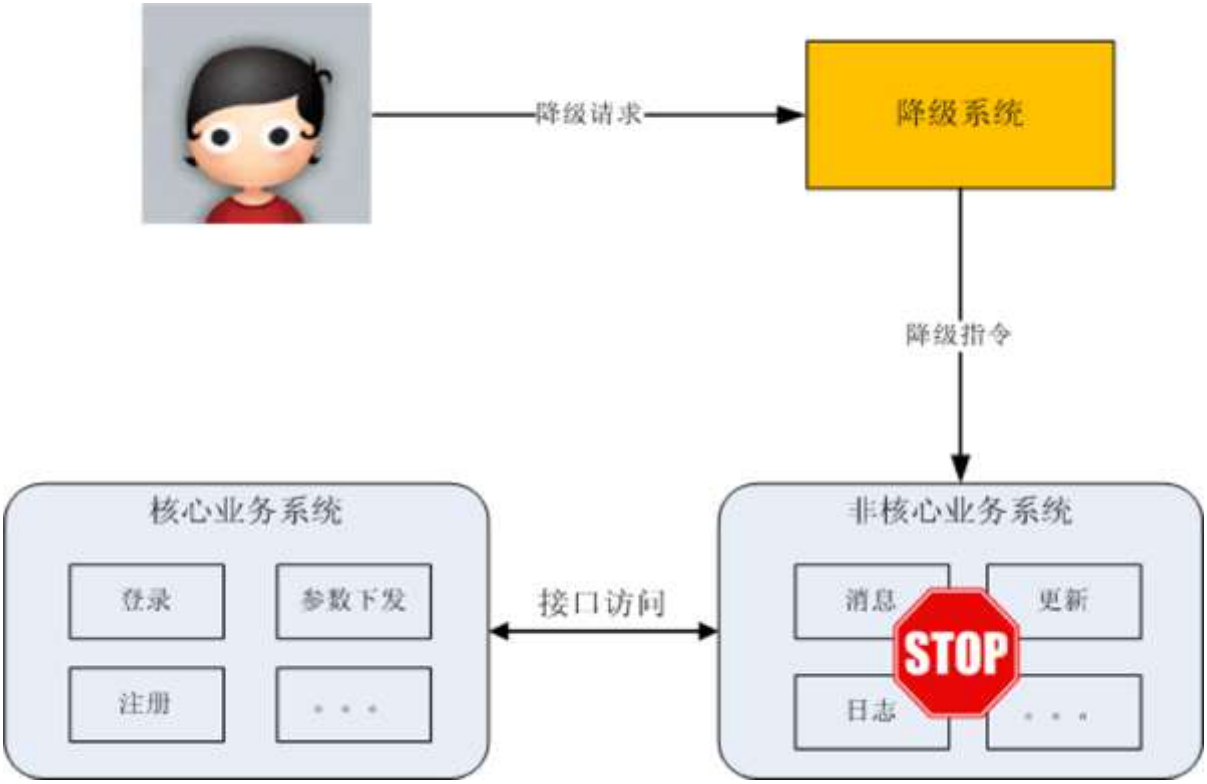
- 系统后门降级

简单来说，就是系统预留了后门用于降级操作。例如，系统提供一个降级 URL，当访问这个 URL 时，就相当于执行降级指令，具体的降级指令通过 URL 的参数传入即可。这种方案有一定的安全隐患，所以也会在 URL 中加入密码这类安全措施。

系统后门降级的方式实现成本低，但主要缺点是如果服务器数量多，需要一台一台去操作，效率比较低，这在故障处理争分夺秒的场景下是比较浪费时间的。

- 独立降级系统

为了解决系统后门降级方式的缺点，将降级操作独立到一个单独的系统中，可以实现复杂的权限管理、批量操作等功能。其基本架构如下：



熔断

熔断和降级是两个比较容易混淆的概念，因为单纯从名字上看好像都有禁止某个功能的意思，但其实内在含义是不同的，原因在于降级的目的是应对系统自身的故障，而熔断的目的是应对依赖的外部系统故障的情况。

假设一个这样的场景：A 服务的 X 功能依赖 B 服务的某个接口，当 B 服务的接口响应很慢的时候，A 服务的 X 功能响应肯定也会被拖慢，进一步导致 A 服务的线程都被卡在 X 功能处理上，此时 A 服务的其他功能都会被卡住或者响应非常慢。这时就需要熔断机制了，即：A 服务不再请求 B 服务的这个接口，A 服务内部只要发现是请求 B 服务的这个接口就立即返回错误，从而避免 A 服务整个被拖慢甚至拖死。

熔断机制实现的关键是需要有一个统一的 API 调用层，由 API 调用层来进行采样或者统计，如果接口调用散落在代码各处就没法进行统一处理了。

熔断机制实现的另外一个关键是阈值的设计，例如 1 分钟内 30% 的请求响应时间超过 1 秒就熔断，这个策略中的“1 分钟”“30%”“1 秒”都对最终的熔断效果有影响。实践中一般都是先根据分析确定阈值，然后上线观察效果，再进行调优。

限流

降级是从系统功能优先级的角度考虑如何应对故障，而限流则是从用户访问压力的角度来考虑如何应对故障。限流指只允许系统能够承受的访问量进来，超出系统访问能力的请求将被丢弃。

虽然“丢弃”这个词听起来让人不太舒服，但保证一部分请求能够正常响应，总比全部请求都不能响应要好得多。

限流一般都是系统内实现的，常见的限流方式可以分为两类：基于请求限流和基于资源限流。

- 基于请求限流

基于请求限流指从外部访问的请求角度考虑限流，常见的方式有：限制总量、限制时间量。

限制总量的方式是限制某个指标的累积上限，常见的是限制当前系统服务的用户总量，例如某个直播间限制总用户数上限为 100 万，超过 100 万后新的用户无法进入；某个抢购活动商品数量只有 100 个，限制参与抢购的用户上限为 1 万个，1 万以后的用户直接拒绝。限制时间量指限制一段时间内某个指标的上限，例如，1 分钟内只允许 10000 个用户访问，每秒请求峰值最高为 10 万。

无论是限制总量还是限制时间量，共同的特点都是实现简单，但在实践中面临的主要问题是比较难以找到合适的阈值，例如系统设定了 1 分钟 10000 个用户，但实际上 6000 个用户的时候系统就扛不住了；也可能达到 1 分钟 10000 用户后，其实系统压力还不大，但此时已经开始丢弃用户访问了。

即使找到了合适的阈值，基于请求限流还面临硬件相关的问题。例如一台 32 核的机器和 64 核的机器处理能力差别很大，阈值是不同的，可能有的技术人员以为简单根据硬件指标进行数学运算就可以得出来，实际上这样是不可行的，64 核的机器比 32 核的机器，业务处理性能并不是 2 倍的关系，可能是 1.5 倍，甚至可能是 1.1 倍。

为了找到合理的阈值，通常情况下可以采用性能压测来确定阈值，但性能压测也存在覆盖场景有限的问题，可能出现某个性能压测没有覆盖的功能导致系统压力很大；另外一种方式是逐步优化，即：先设定一个阈值然后上线观察运行情况，发现不合理就调整阈值。

基于上述的分析，根据阈值来限制访问量的方式更多的适应于业务功能比较简单的系统，例如负载均衡系统、网关系统、抢购系统等。

- 基于资源限流

基于请求限流是从系统外部考虑的，而基于资源限流是从系统内部考虑的，即：找到系统内部影响性能的关键资源，对其使用上限进行限制。常见的内部资源有：连接数、文件句柄、线程数、请求队列等。

例如，采用 Netty 来实现服务器，每个进来的请求都先放入一个队列，业务线程再从队列读取请求进行处理，队列长度最大值为 10000，队列满了就拒绝后面的请求；也可以根据 CPU 的负载或者占用率进行限流，当 CPU 的占用率超过 80% 的时候就开始拒绝新的请求。

基于资源限流相比基于请求限流能够更加有效地反映当前系统的压力，但实践中设计也面临两个主要的难点：如何确定关键资源，如何确定关键资源的阈值。通常情况下，这也是一个逐步调优

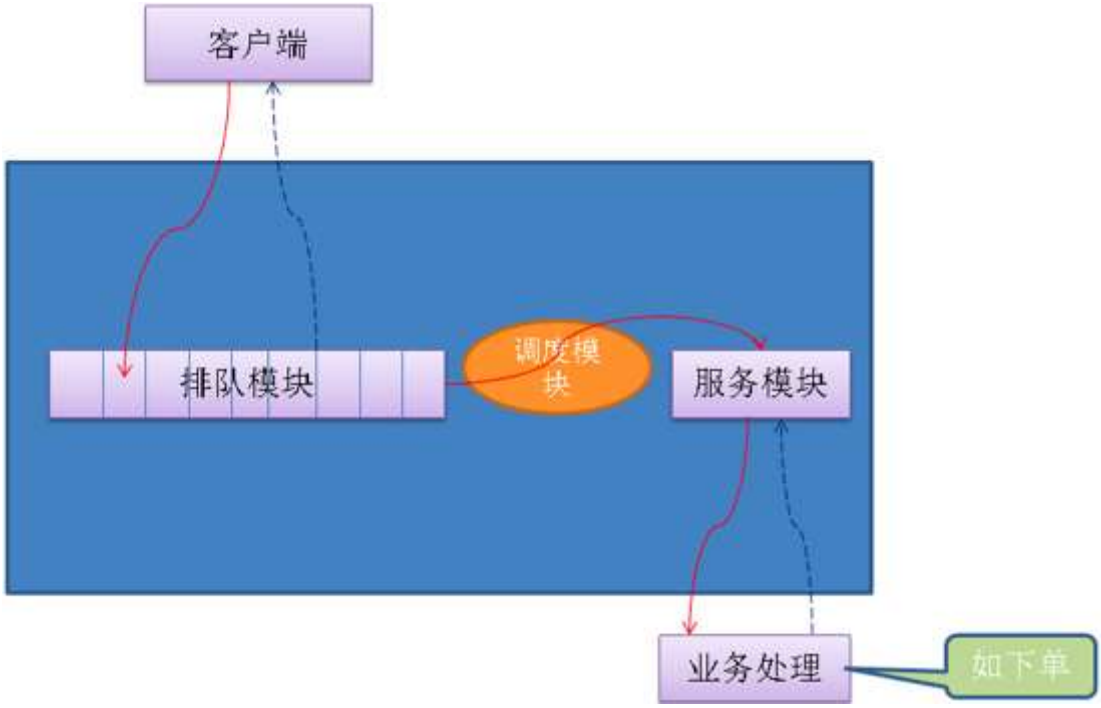
的过程，即：设计的时候先根据推断选择某个关键资源和阈值，然后测试验证，再上线观察，如果发现不合理，再进行优化。

排队

排队实际上是限流的一个变种，限流是直接拒绝用户，排队是让用户等待一段时间，全世界最有名的排队当属 12306 网站排队了。排队虽然没有直接拒绝用户，但用户等了很长时间后进入系统，体验并不一定比限流好。

由于排队需要临时缓存大量的业务请求，单个系统内部无法缓存这么多数据，一般情况下，排队需要用独立的系统去实现，例如使用 Kafka 这类消息队列来缓存用户请求。

下面是 1 号店的“双 11”秒杀排队系统架构



其基本实现摘录如下：

> 【排队模块】
> 负责接收用户的抢购请求，将请求以先入先出的方式保存下来。每一个参加秒杀活动的商品保存一个队列，队列的大小可以根据参与秒杀的商品数量（或加点余量）自行定义。
> 【调度模块】
> 负责排队模块到服务模块的动态调度，不断检查服务模块，一旦处理能力有空闲，就从排队队列头上把用户访问请求调入服务模块，并负责向服务模块分发请求。这里调度模块扮演一个中介的角色，但不只是传递请求而已，它还担负着调节系统处理能力的重任。我们可以根据服务模块的实际处理能力，动态调节向排队系统拉取请求的速度。
> 【服务模块】

> 负责调用真正业务来处理服务，并返回处理结果，调用排队模块的接口回写业务处理结果。

小结

今天我为你讲了接口级故障的四种应对方法，分别是降级、熔断、限流和排队，希望对你有所帮助。

这就是今天的全部内容，留一道思考题给你吧，如果你来设计一个整点限量秒杀系统，包括登录、抢购、支付（依赖支付宝）等功能，你会如何设计接口级的故障应对手段？

欢迎你把答案写到留言区，和我一起讨论。相信经过深度思考的回答，也会让你对知识的理解更加深刻。（编辑乱入：精彩的留言有机会获得丰厚福利哦！）

精选留言

空档滑行 15

1.对于用户服务，在抢购期间可以准备降级策略，压力过大时保证用户登录的可用，注册和修改信息可以做降级处理

2.抢购下单涉及到订单，库存，和商品查询。可通过请求排队来限流，超出库存的请求直接返回。

为了应对库存和商品服务可能发生的故障，可以提前对商品数据和库存数据做缓存，如果对端服务故障，本地也可以提供服务

3.支付依赖第三方系统，合理设置熔断策略，如支付平均时长超过限制可提示用户稍晚做支付

2018-07-08

作者回复

分析全面👍👍

2018-07-09

凡凡 4

1.登录接口在流量特别大的时候，可以适当降级，较少参与人数，另外一点是登录一般有有效期（尤其对于web客户端），可以适当延长登录有效期。2.抢购接口采用队列方式，无法支撑时，也可以适当限流。另外一点，秒杀一般还会有一个秒杀结果查询的接口，也可以降级或者减小请求频次。3.支付接口，一般第三方支付对接入方有流量限制，可以提前申请扩大限制，同时做好降级准备。

2018-07-08

作者回复

分析更好，支付接口依赖第三方应该是熔断，然后做好容错措施，例如提示用户10分钟后来支付

2018-07-09

climber 4

1.抢购页面最大程度静态化，一般用户开始前会尝试刷新页面，查询压力要比下单压力大很多
2.抢购页面要求登陆后访问，一般人不会抢购开始那一刻才进来，错开登陆压力
3.活动未开始，不允许点击抢购按钮。对请求做轻量分析，对于请求过频繁或者可疑ua等做拉黑，为防误杀要求输验证码
4.抢购下单接口采用排队+限流，降低压力的同时保证公平性，如抢购1000件，只放2000人进来，其他返回排队人数过多。进来的请求全部入列，固定数量的队列消费者控制订单生成速率以及走到支付流程的速率。支付是下单流程核心功能，降级不应该降级掉。队列做削峰，保证支付系统不会压力过大。
请指正～

2018-07-08

作者回复

分析的很好，支付如果依赖第三方的话，需要考虑熔断，然后做补偿或者容错措施，例如提示用户10分钟后再来支付

2018-07-09

qpm 2

1、在抢购开始前，可以通过降级来保证登陆服务。例如提前半小时限制注册、修改用户信息的服务。
2、整个流量的洪峰应该在于下单。下单可以同时基于限流和排队两种方案，这个需要大概估计到。譬如说100W人抢购100件商品，可以在整点时对下单操作做99%的限流，把1000K的访问问题改写为10k，10k里面再进行排队。
3、支付属于第三方调用，使用熔断机制（譬如说下单后保留30分钟的支付的时间，等待支付宝服务恢复）

未实际开发过实际抢购系统，期望老师和其他同学提供好的方案借鉴学习

2018-07-07

作者回复

基本思路就是这样的，另外，登录也可以降级

2018-07-09

Tom 1

功能：登录、抢购、支付。
接口故障应对方法：降级、熔断、限流和排队。
三个功能是有依赖关系的，登录了才能抢购，抢购了才能支付。因此如果从依赖关系考虑，总体访问量过大无法满足时，降级时优先保留登录，其次是抢购，再其次是支付。但是支付是成交关键，并且访问数量比登录抢购少很多，可以说不是一个数量级，因此我觉得支付优先级是最高的。
其中只有支付涉及外部调用，因此只需为支付提供熔断方案。
抢购功能提供排队和限流方案，对于每个参与抢购活动的商品根据商品数量设定队列长度，限流也根据商品数量进行限流。
支付失败要怎么处理？

2018-07-09

作者回复

假如降级时优先保证登录，但是用户登录进来后发现抢购不了，其实体验也不好，而且已经抢购了的用户可能无法支付，这样体验更不好，甚至会引起投诉，因此抢购类降级是优先降登录会好些，保留抢购和支付，保证进来的用户能够完成业务流程。

支付失败真没什么好办法了，因为这是核心链路的核心功能。

2018-07-09

100kg 1

老师，我有个双机那节的问题，如果采用了mysql 双主架构，对于库存这样的字段，在并发较高时，如果两个mysql同时 update 了库存（采用乐观锁，库存本身作为版本号），这样会不会导致货物被两个人购买但库存却只减了1呢？该怎么解决呢？求赐教

2018-07-08

作者回复

会的，库存不能用双主

2018-07-09

衣申人 1

整点前降级其他非核心功能，登录采用限流，抢购采用排队加限流，支付对支付宝要有熔断，可加限流。完毕！

2018-07-08

作者回复

没问题就降级，有点过分了呀，产品经理要打你😂
另外，支付最好也不要限流，支付限流的话，用户体验很不好，还不如前面限流

2018-07-09

正是那朵玫瑰 0

降级是整个服务都停掉么？如果只停掉部分接口功能，是不是要对接口都要设置一个开关，这样做是不是对业务入侵很大？还有服务停掉依赖这个服务的消费方调用就会不错了，难道消费方也要做相应处理？

2018-08-03

作者回复

有一定侵入，但可以做到框架中，并不会有太多影响；也可以做到网关中，直接停掉URL

2018-08-03

食指可爱多 0

“每一个参加秒杀活动的商品保存一个队列，队列的大小可以根据参与秒杀的商品数量（或加点余量）自行定义。” 我之前也思考过类似于淘宝秒杀或12306这种高并发系统，应该不会所以资源在一个队列里排队，这样整个系统并发度无法水平伸缩，那么请问老师一个技术细节问题，每个商品一个队列这个有啥推荐的实现方案吗？

2018-07-26

作者回复

用kafka，用商品id做队列名称就可以了

2018-07-26

zbest 0

每期都看看评论，看别人怎么分析“课后作业”，收货也挺大🤓🤓🤓

2018-07-10

作者回复

卧虎藏龙啊😄

2018-07-11

feifei 0

首先是通过压测的手段得倒到系统的最大的在线用户数，然后设置阈值，限流在80%
抢购分为多步进行，首先有验证吗！可以减小并发的秒杀数；抢购还有队列，比如商品最大1000，队列限制在1200，最大可接受1200请求，其他直接提示秒杀结束
秒杀到的商品，需要支付，可使用排队进行支付

2018-07-09

王维 0

首先保证系统的核心功能可用，在这里登录和抢购是系统的核心功能，所以要提高这两块的优先级别。
其次对于登录，可以采用排队机制，合理限制参加抢购的人数。
最后，如果调用支付接口拖慢了系统，可以采用熔断机制，从而保证系统的可用性。

2018-07-09

筱龍 0

老师，请问能否基于springcloud或者dubbo来给我们讲讲这些架构，算是最佳实践，多谢

2018-07-09

作者回复

spring cloud文档都有说明的哦，专栏的目的在于教会大家架构设计的方法论，然后你在具体实践过程中可以按照方法论去操作，也许是用spring cloud，也许是dubbo，也许是自研，但方法论是不变的

2018-07-09

W要改个网名 0

我认为可以这样设计：
登录可以使用降级或者排队策略（但我可能觉得不一定会使用，因为实际场景一般是用户都提前登录好了）；

抢购可以使用排队策略，再加上资源限流方式，主要基于cpu和内存；

支付可以使用排队策略，加上限流策略，如果支付宝响应慢，可以降低通过的个数；如果中途有人支付失败，需重新发起支付；如果有人放弃支付，可调用刚才排队队列中的相应请求再去支付，直到商品被抢购完成；

当然还有高性能[-]高可用方面的问题
以上是我的拙见……

2018-07-08

作者回复

支付一般不建议排队和限流，用户体验很不好，还不如前面登录和抢购就限制。

2018-07-09

正是那朵玫瑰 0

我觉得秒杀系统最应该做的就是限流，比如1元卖iphone，100台，有1000万人参与秒杀，通过边缘计算只放200人过去秒杀系统，其他用户全部返回秒杀结束，我想200的并发怎么样也扛得住！

2018-07-08

作者回复

分析的太简单了，需要根据业务场景逐一分析，例如登录，抢购，支付分别应该如何做

2018-07-09

张玮(大圣) 0

如果时间充裕，就单独把这个整点系统与固有系统隔离出来搞，

如果不允许，切涉及一些固有系统逻辑，就得用到本篇讲到的方法了，登录，抢购，支付都属核心，
1. 这里能降级的额是其他外围功能，

2. 抢购阶段，队列蓄洪客户请求，限流变种

3. 支付，防止由于第三方依赖系统拖垮，做好熔断处理

2018-07-07

作者回复

关键时刻登录也要限流

2018-07-09

星火燎原 0

整点限量抢购核心业务应该是登录和抢购，抢购完了放入购物车不一定马上支付，所以在系统负载较高的时候可以让支付接口做降级处理，过了整点再恢复。抢购接口一般采取商品对应一个抢购队列，队列到上限之后拒绝流量进来，系统根据自身负载情况去消息队列进行流量的拉取，大致思路如上所述，还有什么遗漏还请老师指教

2018-07-07

作者回复

一般不建议对支付做降级，用户体验很不好，还不如登录和抢购阶段限流，这是有心理学理论支撑的，用户没抢到前，如果抢不到他会认为自己运气不好，但如果用户抢到了没法支付，他会觉得自己损失了，会触发“损失厌恶”心理😁

2018-07-09