

## 08 | 一个几乎每个系统必踩的坑儿：访问数据库超时

李玥 · 后端存储实战课



你好，我是李玥。

每一个创业公司，它的系统随着公司的发展一起成长的过程中，都难免会发生一些故障或者是事故，严重的会影响业务。搞技术的同学管这个叫：坑儿，分析解决问题的过程，称为：填坑儿。而访问数据库超时这个坑儿，是我见过的被踩的次数最多的一个坑儿，并且这个坑儿还在被不停地踩来踩去。

今天这节课，我和你分享一个典型的数据库超时案例。我也希望你通过和我一起分析这个案例，一是，吸取其中的经验教训，日后不要再踩类似的坑儿；二是，如果遇到类似的问题，你能掌握分析方法，快速地解决问题。最重要的是，学习存储系统架构设计思想，在架构层面限制故障对系统的破坏程度。

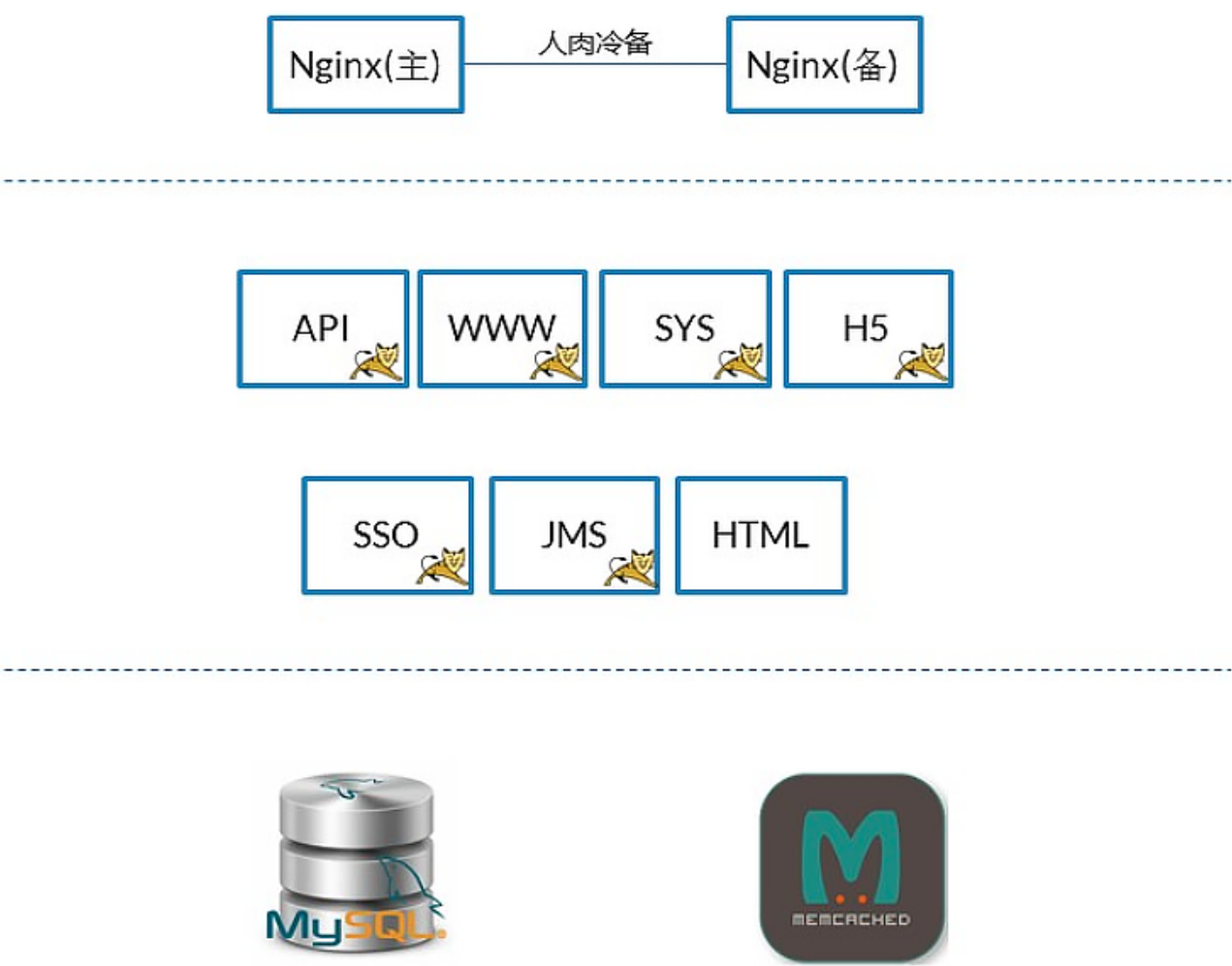
### 事故排查过程

我们一起来看一下这个案例。

每一个做电商的公司都梦想着做社交引流，每一个做社交的公司都梦想着做电商将流量变现。我的一个朋友他们公司做社交电商，当年很有前途的一个创业方向，当时也是有很多创业公司在做。

有一天他找到我，让我帮他分析一下他们系统的问题。这个系统从圣诞节那天晚上开始，每天晚上固定十点多到十一点多这个时段，大概瘫痪一个小时左右的时间，过了这个时段系统自动就恢复了。系统瘫痪时的现象就是，网页和 App 都打不开，请求超时。

这个系统的架构是一个非常典型的小型创业公司的微服务架构。系统的架构如下图：



整个系统托管在公有云上，Nginx 作为前置网关承接前端所有请求，后端按照业务，划分了若干个微服务分别部署。数据保存在 MySQL 中，部分数据用 Memcached 做了前置缓存。数据并没有按照微服务最佳实践的要求，做严格的划分和隔离，而是为了方便，存放在了一起。

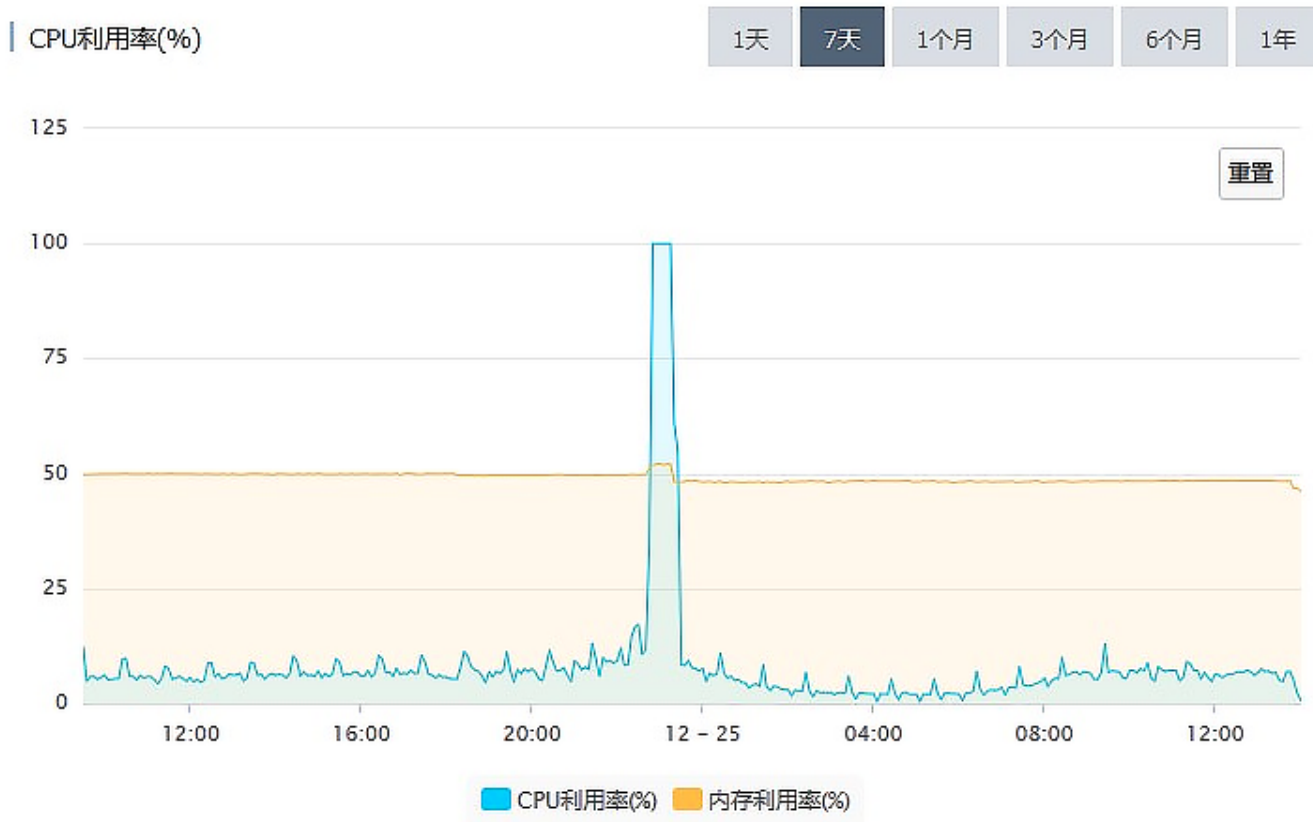
这样的存储设计，对于一个业务变化极快的创业公司来说，是合理的。因为它的每个微服务，随时都在随着业务改变，如果做了严格的数据隔离，反而不利于应对需求变化。

听了我朋友对问题的描述，我的第一反应是，每天晚上十点到十一点这个时段，是绝大多数内容类 App 的访问量高峰，因为这个时候大家都躺在床上玩儿手机。初步判断，这个故障是和访问量有关系的，看下面这个系统每天的访问量的图，可以印证这个判断。



基于这个判断，排查问题的重点应该放在那些服务于用户访问的功能上。比如说，首页、商品列表页、内容推荐这些功能。

在访问量峰值的时候，请求全部超时，随着访问量减少，系统能自动恢复，基本可以排除后台服务被大量请求打死的可能性，因为如果进程被打死了，一般是不会自动恢复的。排查问题的重点应该放在 MySQL 上。观察下面这个 MySQL 的 CPU 利用率图，发现问题：



从监控图上可以看出来，故障时段 MySQL 的 CPU 利用率一直是 100%。这种情况下，MySQL 基本上处于一个不可用的状态，执行所有的 SQL 都会超时。


MySQL 这种 CPU 利用率高的现象，绝大多数情况都是由慢 SQL 导致的，所以我们优先排查慢 SQL。MySQL 和各大云厂商提供的 RDS 都能提供慢 SQL 日志，分析慢 SQL 日志，是查找类似问题原因最有效的方法。

一般来说，慢 SQL 的日志中，会有这样一些信息：SQL、执行次数、执行时长。通过分析慢 SQL 找问题，并没有什么标准的方法，主要还是依靠经验。

首先，你需要知道的一点是，当数据库非常忙的时候，它执行任何一个 SQL 都很慢。所以，并不是说，慢 SQL 日志中记录的这些慢 SQL 都是有问题的 SQL。大部分情况下，导致问题的 SQL 只是其中的一条或者几条。不能简单地依据执行次数和执行时长进行判断，但是，单次执行时间特别长的 SQL，仍然是应该重点排查的对象。

通过分析这个系统的慢 SQL 日志，首先找到了一个特别慢的 SQL。

这个 SQL 支撑的功能是一个红人排行榜，这个排行榜列出粉丝数最多的 TOP10 红人。

 复制代码

```
1 select fo.FollowId as vid, count(fo.id) as vcounts
2 from follow fo, user_info ui
3 where fo.userid = ui.userid
4 and fo.CreateTime between
5 str_to_date(?, '%Y-%m-%d %H:%i:%s')
6 and str_to_date(?, '%Y-%m-%d %H:%i:%s')
7 and fo.IsDel = 0
8 and ui.UserState = 0
9 group by vid
10 order by vcounts desc
11 limit 0,10
```

**这种排行榜的查询，一定要做缓存。**在这个案例中，排行榜是新上线的功能，可能忘记做缓存了，通过增加缓存可以有效地解决问题。

给排行榜增加了缓存后，新版本立即上线。本以为问题就此解决了，结果当天晚上，系统仍然是一样的现象，晚高峰各种请求超时，页面打不开。

再次分析慢 SQL 日志，排行榜的慢 SQL 不见了，说明缓存生效了。日志中的其他慢 SQL，查询次数和查询时长分布的都很均匀，也没有看出明显写的有问题的 SQL。

回过头来再看 MySQL CPU 利用率这个图。

CPU利用率(%)

1天

7天

1个月

3个月

6个月

1年



把这个图放大后，发现一些规律：

1. CPU 利用率，以 20 分钟为周期，非常规律的波动；
2. 总体的趋势与访问量正相关。

那我们是不是可以猜测一下，对 MySQL 的 CPU 利用率的“贡献”来自两部分：红线以下的部分，是正常处理日常访问请求的部分，它和访问量是正相关的。红线以上的部分，来自某一个以 20 分钟为周期的定时任务，和访问量关系不大。

CPU利用率(%)

1天

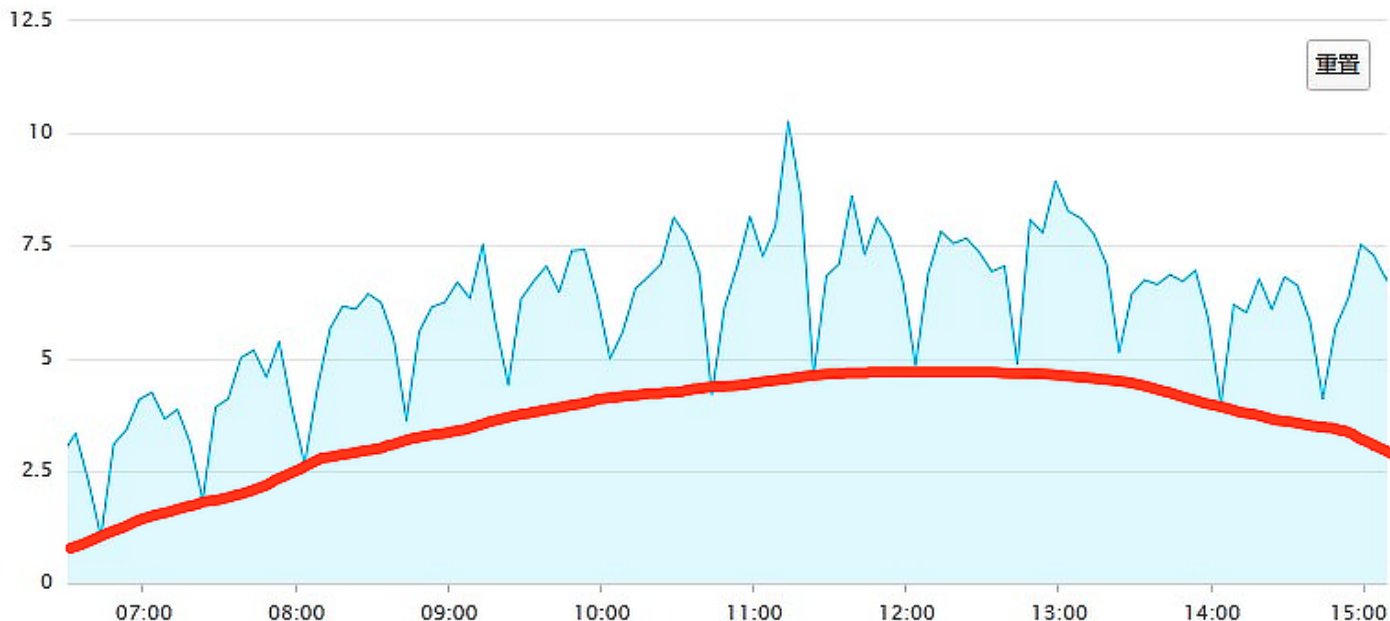
7天

1个月

3个月

6个月

1年



排查整个系统，没有发现有以 20 分钟为周期的定时任务，继续扩大排查范围，排查周期小于 20 分钟的定时任务，最终定位了问题。

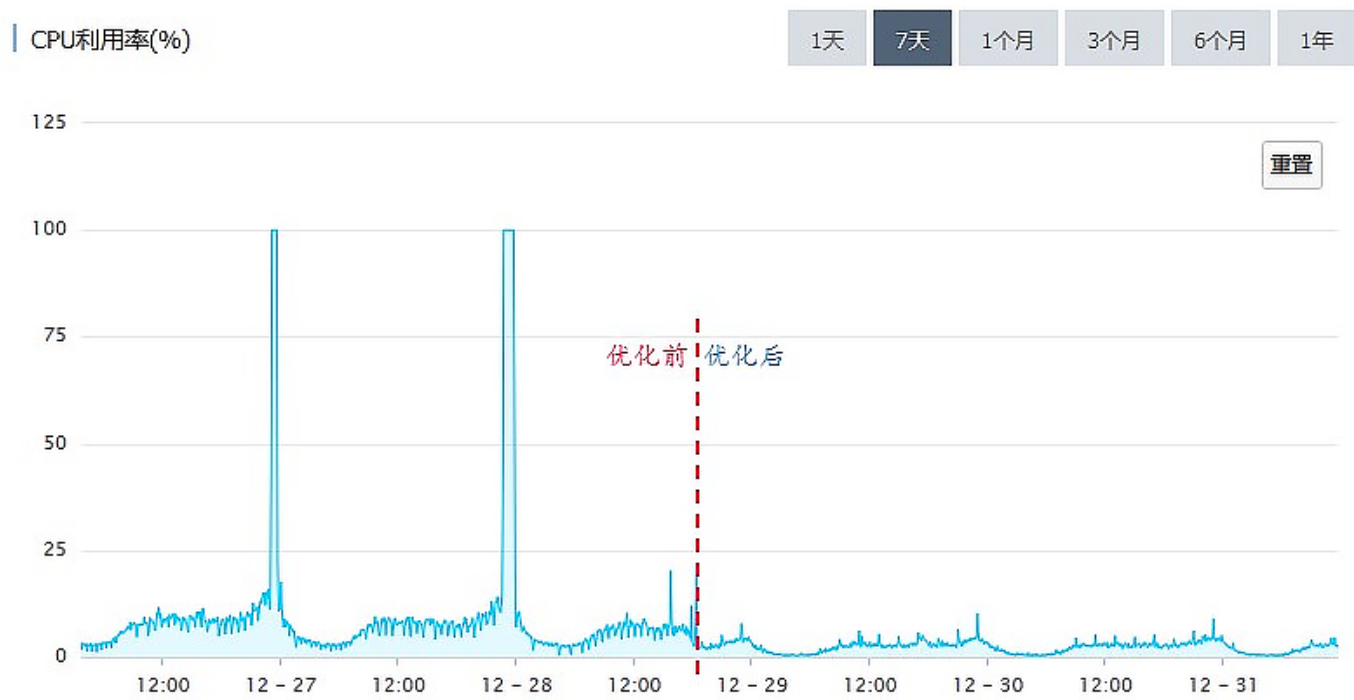
App 的首页聚合了非常多的内容，像精选商品、标题图、排行榜、编辑推荐等等。这些内容包含了很多的数据库查询。当初设计的时候，给首页做了一个整体的缓存，缓存的过期时间是 10 分钟。但是需求不断变化，首页需要查询的内容越来越多，导致查询首页的全部内容越来越慢。

通过检查日志发现，刷新一次缓存的时间竟然要 15 分钟。缓存是每隔 10 分钟整点刷一次，因为 10 分钟内刷不完，所以下次刷新就推迟到了 20 分钟之后，这就导致了上面这个图中，红线以上每 20 分钟的规律波形。

由于缓存刷新慢，也会很多请求无法命中缓存，请求直接穿透缓存打到了数据库上面，这部分请求给上图红线以下的部分，做了很多“贡献”。

找到了问题原因，做针对性的优化，问题很快就解决了。新版本上线之后，再没有出现过“午夜宕机”。





对比优化前后 MySQL 的 CPU 利用率，可以明显地看出优化效果。

## 如何避免悲剧重演

到这里问题的原因找到了，问题也圆满解决了。单从这个案例来看，问题的原因在于，开发人员犯了错误，编写的 SQL 没有考虑数据量和执行时间，缓存的使用也不合理。最终导致在忙时，大量的查询打到 MySQL 上，MySQL 繁忙无法提供服务。

作为系统的开发人员，对于这次事故，我们可以总结两点经验：

第一，在编写 SQL 的时候，一定要小心谨慎地仔细评估。先问自己几个问题：

你的 SQL 涉及到的表，它的数据规模是多少？

你的 SQL 可能会遍历的数据量是多少？

尽量地避免写出慢 SQL。

第二，能不能利用缓存减少数据库查询次数？在使用缓存的时候，还需要特别注意的就是缓存命中率，要尽量避免请求命中不了缓存，穿透到数据库上。



以上两点，是开发人员需要总结的问题。不过你想没想过，谁能保证，整个团队的所有开发人员以后不再犯错误？保证不了吧？那是不是这种的悲剧就无法避免了呢？

其实，还是有办法的。不然，那些大厂，几万开发人员，每天会上线无数的 Bug，系统还不得天天宕机？而实际情况是，大厂的系统都是比较稳定的，基本上不会出现全站无法访问这种情况。

靠的是什么？靠的是架构。

优秀的系统架构，可以在一定程度上，减轻故障对系统的影响。针对这次事故，我给这个系统在架构层面，提了两个改进的建议。

第一个建议是，上线一个定时监控和杀掉慢 SQL 的脚本。这个脚本每分钟执行一次，检测上一分钟内，有没有执行时间超过一分钟（这个阈值可以根据实际情况调整）的慢 SQL，如果发现，直接杀掉这个会话。

这样可以有效地避免一个慢 SQL 拖垮整个数据库的悲剧。即使出现慢 SQL，数据库也可以在至多 1 分钟内自动恢复，避免数据库长时间不可用。代价是，可能会有些功能，之前运行是正常的，这个脚本上线后，就会出现问题。但是，这个代价还是值得付出的，并且，可以反过来督促开发人员更加小心，避免写出慢 SQL。

第二个建议是，做一个简单的静态页面的首页作为降级方案，只要包含商品搜索栏、大的品类和其他顶级功能模块入口的链接就可以了。在 Nginx 上做一个策略，如果请求首页数据超时的时候，直接返回这个静态的首页作为替代。这样后续即使首页再出现任何的故障，也可以暂时降级，用静态首页替代。至少不会影响到用户使用其他功能。

这两个改进建议都是非常容易实施的，不需要对系统做很大的改造，并且效果也立竿见影。

当然，这个系统的存储架构还有很多可以改进的地方，比如说对数据做适当的隔离，改进缓存置换策略，做数据库主从分离，把非业务请求的数据库查询迁移到单独的从库上等等，只是这些改进都需要对系统做比较大的改动升级，需要从长计议，在系统后续的迭代过程中逐步地去实施。

## 小结

这节课，我和你一起分析了一个由于慢 SQL 导致的全站故障的案例。在“破案”的过程中，有一些很有用的经验，这些经验对于后续你自己“破案”时会非常有用。比如说：

1. 根据故障时段在系统忙时，推断出故障是跟支持用户访问的功能有关。
2. 根据系统能在流量峰值过后自动恢复这一现象，排除后台服务被大量请求打死的可能性。
3. 根据 CPU 利用率曲线的规律变化，推断出可能和定时任务有关。

在故障复盘阶段，除了对故障问题本身做有针对性的预防和改进以外，更重要的是，在系统架构层面进行改进，让整个系统更加健壮，不至于因为某一个小的失误，就导致全站无法访问。

我给系统提出的第一个自动杀慢 SQL 的建议，它的思想是：系统的关键部分要有自我保护机制，避免外部的错误影响到系统的关键部分。第二个首页降级的建议，它的思想是：当关键系统出现故障的时候，要有临时的降级方案，尽量减少故障带来的影响。

这些架构上的改进，虽然并不能避免故障，但是可以很大程度上减小故障的影响范围，减轻故障带来的损失，希望你能仔细体会，活学活用。

## 思考题

课后请你想一下，以你个人的标准，什么样的 SQL 算是慢 SQL？如何才能避免写出慢 SQL？欢迎你在留言区与我交流互动。

感谢你的阅读，如果你觉得今天的内容对工作有所帮助，也欢迎把它分享给你的朋友。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

## 精选留言 (38)



冯玉鹏

2020-03-14

老师，慢SQL 我感觉也没有个人标准，个人的标准也要分场景，业务复杂度等；如果作为常规的用户业务系统，超过1秒就是慢SQL；但是如果是类似生成报表的服务，选择在业务低峰期，从库执行等策略，时间长点也不是不能接受。

避免慢SQL：第一点肯定想到的是合适的索引，毕竟SQL执行速度的快慢关键还是语句需要扫描数据的行数，如尽量不要使用 对where 条件列进行计算的做法让MySQL查询优化器不知道如何选择索引，特定业务 可以设置联合索引让需要查询返回的列都在索引中避免回表操作。

第二：排序也是可能完成慢SQL的因素，尤其是数据量大，需要使用外部排序的时候又可以与磁盘IO性能扯上关系等，常见的问题还有limit m,n m很大又无法使用索引的时候

第三：多表联合查询的时候，尽量使用小表驱动大表。

第四：避免大事务，这也是发生死锁常见的雷区，尽量减小事务粒度，尽量注意不同事务对表操作的顺序一致，大事务其实也包含着批量操作的隐式事务，如一个update 影响100万行数据。

第五：见过的关于架构方面的慢SQL问题 1~数据量到达一定规模后，单机性能容易受限导致数据库响应慢；2~读写分离，从库提供读服务，错误的认为从库只需要提供查询服务采用了达不到性能指标的机器，其实是主库承受的数据更新压力，从库一个不落的都要承受，还要更多的提供查询服务

作者回复: 🍌🍌🍌🍌🍌

共 4 条评论>

👍 108



一步

2020-03-17

上面那个小型创业公司的微服务架构，想知道有关 Nginx 的主备是怎么实现的？

作者回复: 我们例子里面当时它采用的就是人肉冷备，主节点出问题的时候人肉切换到备用节点上。

其实更合理的做法是做通过负载均衡器或者域名把流量均匀的打到多个NGINX节点上，配合探活机制，当某个节点有问题的时候，自动摘掉这个节点。

共 2 条评论>

👍 17



小唐

2020-06-11

请问老师，我有两个问题请教。1. 为什么后台服务被大量请求打死的话无法自动恢复呢？  
2. 案例中用的什么cache，怎么refresh的？

作者回复: 1. 一般“服务被打死”，比较常见的情况是内存溢出、栈溢出或者进程直接挂掉，这些情况都是不能自动恢复的。

2. 案例中用的是Memcached，刷新的策略也是根据不同业务有不同的策略。

共 2 条评论 >

👍 13



Halo

2020-04-03

老师，我现在想做一个Mysql的本地熔断方案。就是监控对每一个表的操作语句，通过机器数量在配置中心配置每个服务的访问频次、访问时间等。比如Mysql的TPS是4000，我们有10台机器，平均下来每个服务的上限为400/s。碰到超限、或者超慢的情况就熔断、告警。可以整体监控，也可以对热点表进行监控，这种方案是否可行？

作者回复: 必须可行啊。但要注意一下配置中心的高可用。别出现因为配置中心宕机，导致不能熔断了。

共 2 条评论 >

👍 11



leslie

2020-03-23

合理使用日志系统、通过合理监控获取必要时信息和做报警提醒，每天定时排查日志中记录的问题；例：cpu使用率、IO使用率等。

指定一套完善的开发规范：严禁开发去写，上线之前做code review;规范制度+code review+架构审核，基本能避免大多数问题的发生。



👍 11



美美

2020-03-15

个人感觉，算不算慢SQL首先取决于这条SQL有没有正确的命中索引。如果可以正确的命中索引，那么从业务上是否正确。如果业务匹配，且正常命中索引，那应该不算是慢查询。看完本章还有一点疑惑，就是当第一个慢查询SQL处理完成后，MySQL的CPU使用率已经降到了20%以下。那么即便会有周期性的SQL执行，但是以这个利用率不足以整体导致服务不可用吧。

作者回复: 20%左右那个是闲时的图，忙时依然是100%....



6

**Regis**

2020-03-16

老师讲的非常好，给出了查找问题的思路和解决问题思路，对于项目经验少的很有用。后续课程里面有没有涉及对于视频类的大文件存储方式和使用方式的课程？这部分数据知道使用对象存储进行存储比较好，但是对于有这种大文件的存储的系统架构方面还是不清晰，老师要是了解给指导一下可以吗

作者回复: 后面有一节课是专门讲对象存储的，敬请期待。



5

**myrfy**

2020-03-14

慢SQL要以业务场景来区分。例如做即时通讯或者消息类等有实时性要求的，可能2秒就算慢查询了，但是读从库做大数据分析的场景，可能跑一个小时也不算慢。另外，对于请数量大的时候，如果存在多个请求会加锁，即使一个查询是毫秒级别的，上百个查询访问一个热数据加锁也会有很大的问题，所以，没有慢查询的具体标准，影响到业务，拖慢了服务的，就算慢查询。



5

**观弈道人**

2020-03-14

作者文中描述的问题可以理解成就是缓存更新慢，导致的缓存穿透

共 1 条评论 >



3

**mickey**

2020-04-17

请问老师，杀掉慢 SQL 的脚本应该怎样写呢？



2

**Yezhiwei**

2020-04-09

第一个建议杀掉慢SQL主要还是对慢查询的SQL吧？如果是 update SQL 会带来更多的问题哈

共 1 条评论 >



2



发条橙子。

2020-03-15

老师 针对两个问题排查后处理的方法能不能给个思路

1. 缓存热点数据：因为使用连表查询等复杂语句在数据量大的时候会产生慢差。是否该考虑修改查询语句或者上搜索（es / 阿里open search）然后再加一道缓存 缓存的读写策略采用旁路策略。

2. 像这种定时任务应该大部分公司都会有很多，一般都是放到凌晨来执行，经常会有人问当数据量大的时候 这种定时任务是否可行。所以像数据量非常大（京东这种级别数据）定时任务扫表是否还可行 有没有其他的解决思路

希望老师给些思路 谢谢🙏

作者回复: 这两个问题我在后续的课程中都会讲到一些解决的经验和方法。

这种大查询，首先肯定是要用缓存，但要根据实际情况选择合适的缓存更新策略。

数据量特别大的统计分析，一般选择放到其它分析型数据库或者数据仓库中去执行，或者使用流计算来解决。

共 3 条评论 >



2



webmin

2020-03-14

1. 以你个人的标准，什么样的 SQL 算是慢 SQL？

SQL的快慢是个相对标准，与数据量和设备性能有相关性，需要建立监控机制，了解SQL在正常情况下的执行时间的基线是多少，偏离基线超过阈值时可以认为是慢了。

2. 如何才能避免写出慢 SQL？

抓大放小，针对查询量大和查询大数据集的SQL

2.1 先通过SQL执行计划看看是否使用了与预想一至的索引和SQL的执行路径是否与预想的一至；（需要在生产库上看执行计划，现在的DB大都是用成本法对SQL进行优化，数据量不一样会导致执行路径不同）

2.2 利用好测试DB，在无法模拟生产数据量的情况下，也需要按一定比例在测试DB在灌入数据，通过实际执行测量执行时间。



2



沐

2022-01-11

作为系统开发人员，

一是在编写SQL的时候，一定要仔细评估

1、SQL涉及到的表，数据规模是多少

2、SQL会遍历到的数据量

3、尽量避免写出慢SQL

二是能不能利用缓存减少数据库查询次数，在使用缓存的时候要注意缓存的命中率，避免缓存穿透

同时要在架构层面，减轻故障对系统的影响

一是上线定时监控和杀掉慢SQL的脚本

二是简单的静态首页作为降级方案



1



而立斋

2020-09-24

重构:用自己的话，重述内容

对于一次系统高峰时段出现的问题，从排查分析到解决，到复盘总结，过程的一次演练。

根据出现的时间段，分析出是用户请求超时导致的结果，进而对系统中的慢sql进行分析，分析出慢sql之后进行修复，从数据库cpu使用率上分析出定时任务的存在，并分析出定时任务的周期，至此问题解决。但在复盘的时候从架构的层次进行了更为本质的分析，并给出数据库慢sql的预处理模式，数据库分离的建议以及页面降级预案。

共 1 条评论 >



1



Monday

2020-08-02

总感觉这篇文章在哪里听过读过，折腾了好久，终于找到出处了，卖桃者说 2020-03-20期推荐过这篇好文。哈哈，现在再次读来，倍感亲切



1



小袁

2020-03-17

不仅仅要扫描长时间执行的sql语句，还要扫描长事务。有些python库不开启自动提交，会导致长事务占据表的元数据锁，从而导致更多的问题。

共 2 条评论 >



1





肥low

2020-03-14

老师好 你讲的例子好像就发生在昨天，慢SQL的话执行时间超过1秒就算是了，长的主要原因有：语句复杂，比如各联表，group by我就被搞死过一次；还有就是没有建立合适的索引，总而言之，如果对数据库如MySQL BTree有较深入的理解的话，肯定不会写出这么慢的SQL来

共 1 条评论 >



1



ifelse

2022-12-07 来自浙江

MySQL 这种 CPU 利用率高的现象，绝大多数情况都是由慢 SQL 导致的。--记下来



Catcher

2022-10-23 来自北京

老师，想问下问什么缓存更新慢的监控指标是这样的？更新缓存15分钟，使用缓存5分钟，低谷期也会是一条直线呀？而不是图中这样，低谷期很短，基本上就是上升回落上升回落

