慢接口分析与优化

前言

在系统维护过程中，随着业务增长，业务复杂度增加，业务数据量的逐步增长，可能会出现一些请求响应时间较长的情况，给用户带来不便甚至灾难性的后果。因此，我们在系统设计期间充分考虑接口性能、在系统运行时期需要定期筛选出慢接口进行分析与优化，将业务系统的响应速度、并发能力及可用性维持在一个较高水位。

接口耗时的构成

Client

* **DNS Lookup**

域名解析耗时，一般都应该很短，小于20ms，内网的DNS服务器应该更小，如果耗时大了就需要考虑更换DNS服务器，如果无法解决，可以考虑直接做HOST来手动解析。

* **TCP Connect**

连接握手耗时，只有在新建连接的时候会产生，否则为0，如果相对耗时较大，且单独访问都经常耗时很大，那么问题基本在于网络线路，但如果是在大量访问下耗时较大，那么说明服务器或者中间网络设备无法应对，可能连接池已用完或者无法处理。

* **Request Send**

iddlerBeginRequest→ServerGotRequest

发送数据耗时，TCP独有，数据发送到对方后会有一个自动空响应，所以可以计算耗时，如果发送内容不是太大，但耗时较大，那么基本可以定性为网络问题，但一些特殊的协议如果可以多次不断的发送数据而不等待服务器响应，那么也有可能是Socket接收数据的缓冲区已满，遭遇排队。

* **Wait Response**

ServerGotReques→ServerBeginResponse

响应等待耗时，也就是从发送完成后，到服务器响应第1个字消耗的时间，如果网络没有问题，那么这将是服务器处理接口数据的精确耗时。

* **Response Receive**

ServerBeginResponse→GotResponseHeaders→ServerDoneResponse

响应接收耗时，这个耗时只是接收数据传输和读取的耗时，已经不是服务器的处理耗时范围了，信息量越大，耗时也一定越长。

Server
---

* **Cache读写**

以redis为例，redis作为内存数据库，拥有非常高的性能，单个实例QPS能达到10W左右，一般场景下建议使用cache协助进行接口性能优化。

* **DB读写**

以MqSql为例，DB读写往往代表着存在大量磁盘读写操作，复杂的查询往往会导致DB写入大量临时表数据。同时DB大多为单点服务，在接口性能优化时往往会投入大量精力进行慢查询优化。

* **调用外部服务**

大多数接口链路过程中还会同步调用外部服务，此时外部服务接口的性能及调用时的timeout时间、即时重试机制等会直接影响接口耗时。

* **系统内部逻辑处理**

系统进程处理的执行者为CPU，处理延迟即代表CPU没有及时运算应用程序代码。而导致CPU占用率较高的操作有：磁盘io、网络io、同步锁、jvm频繁GC等。当存在大量磁盘io、网络io长时间阻塞时会导致接口耗时增加。

优化方式
====

Cache篇

恰当使用缓存

对于一些不经常更新的数据，我们往往可以放入缓存提供频繁读取。这里的缓存包括分布式缓存、本地缓存。

* 分布式缓存

主体、供应商、汇率等主数据信息往往可以使用分布式缓存，通过job定时刷新、过期刷新、binlog/领域事件监听刷新等方式保证数据一致性。

* 本地缓存

相对分布式缓存，本地缓存建议存放数据量极小的不经常更新或基本不变的数据。如系统字典、系统业务配置等。

缓存延迟优化

以redis为例，当我们本身使用不当或运维不合理时，可能导致redis访问延迟变大。可能存在的原因：

* 使用了复杂度过高的命令，例如sort、sunion等等。
* 操作、存储大的bigkey。
* 大量数据集中过期。 集中过期导致redis响应慢的原因与redis过期策略有关。
* redis实例内存使用达到了上限。
* fork子进程导致响应变慢。ByteCache持久化方案：base rdb + aof。

DB篇

慢SQL优化

若系统存在单个慢SQL，可通过调整索引的设计/使用方式、调整分页方式、拆解/合并等处理方式进行优化，复杂/大表场景通过数据归档、分库分表、迁移ES/Hive等方式优化。

批量读写

循环内的查询、insert可能因数据量的增长导致接口性能急剧下降。将循环内的查询、insert前/后置成批量查询、批量insert可以大大降低数据库访问次数，缩减系统与DB之间的连接、关闭等消耗。此外，也可结合业务实际情况分析是否可以大批次拆分成小批次处理。

避免大事务

> 建议单次执行影响行数不得超过100000，总执行次数不得超过200次，总耗时<360s。
>
>
> * 慎用@Transactional
> * @Transactional范围足够小&&精确

单次执行影响行数、总执行次数较大的事务我们称之为大事务。大事务往往执行时间较长。

* 增删改操作会生成undo log，大量数据操作将生成大量的undo log，所在表上的其他查询及DML也将会受到影响，尤其是大事务发生回滚将会耗时漫长，严重影响业务。
* 同时SQL语句影响行数过多，容易造成从库延迟过大，甚至无法追平主库；
* 更新记录时间长，锁持有时间长，容易引发行级锁阻塞；

调用外部服务篇

拒绝无效等待

当外部服务接口抖动时，若程序一直处于等待状态，这可能会导致程序在等待过程中变得不响应。超时控制可以让程序在一定时间内完成操作。当外部服务接口持续劣化时，则需要采取其他方案进行改造，防止影响业务使用。

系统内部处理篇

快速失败

fail-fast。当接口收到不合法、无效的请求时，快速响应错误信息。快速失败可以节省程序处理成本，同时也能避免系统整体数据的污染。

异步执行

对于一些非核心业务逻辑，如果这部分逻辑执行时间长且不影响主业务流程，我们可以考虑“异步”执行这些逻辑。 具体来说，可以通过以下技术方案实现：

* 使用消息队列，让主业务逻辑快速返回，将非核心逻辑作为消息放入队列异步执行。
* 设计异步线程或定时任务，在主线程返回后，异步线程负责后台执行非核心逻辑。
* 利用事件编程模型，主业务逻辑触发事件，事件监听者异步响应事件执行非核心逻辑。
* 非核心逻辑作为微服务单独部署，主业务快速调用微服务，微服务后台异步执行逻辑。
* 使用reactor模式，主线程接收请求触发非核心逻辑，再通过多线程异步执行非核心处理。

并发优化

在设计接口时，若使用串行逻辑，即一个任务完成后再执行下一个任务，则任务只能顺序执行，整体吞吐量受到限制。 为提高吞吐量，可以考虑将程序的DB操作、外部服务访问等远程调用改为并行执行。在并行处理过程中若需串行访问共享资源，可通过锁或CAS算法来控制并发访问。

拒绝阻塞等待

当接口链路中需要调用一个外部系统的接口，且依赖其响应结果，但该接口耗时较长(例如超过10s)时，如果一直阻塞等待，接口性能会受到极大拖累且不合理。此时可以参考IO多路复用模型，等待外部系统响应的事件回调通知，再进行后续业务处理。

网络篇

流量控制

通过限流、降级等方式控制流量保护服务稳定运行。

压缩传输内容

当传输报文较大时，可以考虑压缩后传输，可减少网络带宽占用、提升传输速度。

总结
==

慢接口的本质是CPU处理不过来或者处理时间过长，在慢接口优化时我们要做的就是尽可能精简系统处理的流程或者数据，让CPU提效。接口耗时构成为接口链路的整个生命周期，所以其分析治理往往较为复杂，希望本文能提供一些优化思路，同时也为高性能的接口设计提供一些参考。

原文链接: https://juejin.cn/post/7359076801278656564