衡量一个网站流量的指标有PV UV IP等，也可以大概预估一个网站架构是否能够支撑多大的PV UV IP。

网站请求都是并发式的，一个请求一个线程，线程数量由硬件决定。此外IIS可以支撑10w+的请求，所以一般不会成为瓶颈。

互联网架构从最初的单机版到现在的微服务，经历了如下演化历程：

1.单机版

不够时：

升级硬件

水平分拆（第二代）

2.分拆-单机不够时

文件服务：硬盘-带宽

数据服务：硬盘 cpu 内存

应用服务：cpu 内存

根据不同服务的特点，针对性升级

广义的分布式

3.缓存-分拆不够时

缓存是系统性能优化的第一步

使用简单

效果明显

适用面广

缺点：实时性等

二八原则

80%查询

查询中80%都是那20%的数据

缓存就是解决这80%的重复查询

缓存穿透、雪崩

请求大时，还是会有很多请求直达服务器，导致撑不住

【问题：缓存穿透、雪崩，分布式缓存及其应用场景】

## [缓存穿透、缓存击穿、缓存雪崩区别和解决方案](https://www.cnblogs.com/sbj-dawn/p/11116673.html)

# [深入理解分布式缓存设计](https://zhuanlan.zhihu.com/p/55303228)

4.具体定位

木桶理论，找出短板

如何定位

看硬件资源使用率

日志，详细记录各环节

数据库查看sql执行记录

4.1瓶颈在应用服务器

多台（集群）

分布式，rpc webapi协作

【rpc 跨语言通信，可以的，thrift】

.net Remoting则要求客户端和服务端都是.net

[.NET Remoting学习笔记（一）概念](https://www.cnblogs.com/lonelyxmas/p/8989036.html)

集群：集群又涉及到请求分发的方式的问题

DNS-随机、就近分配、轮询

不是自己的，不灵活

负载均衡服务器

硬件负载均衡 F5 黑盒子

软件

LVS:linux virtual server，工作在第4层，效率高、但不灵活

Nginx

工作在第7层，http，可以拿到需要的所有信息，配置灵活

【Nginx的实操】

4.2 数据库撑不住了，读写分离

读80%写20%

将20%的增删改放在主库

从库负责查询，并同步主库的数据，从库可以集群

数据同步：发布订阅

【发布订阅与同步链的比较】

两者等同？

4.3 缝缝补补1

CDN，特别适合于图片、视频等带宽高的

【CDN】

反向代理？

**正向不知道客户端，反向不知道服务端**

**正向代理，代理客户端；反向代理，代理服务端**

【反向代理，CDN与反向代理如何配合】

# [反向代理和CDN是什么，作用是什么](https://blog.csdn.net/wanderlustLee/article/details/84110051)

# [CDN和反向代理](https://www.cnblogs.com/linix/p/9482247.html)

分布式文件系统 TFS GFS NFS

4.4 缝缝补补2

特殊组件解决特殊问题

全文检索 搜索引擎

Nosql数据库 redis 秒杀

【ES实操】

【redis实操】

水平扩展，多个单元做相同的事，增加单元就能提升处理能力

4.5垂直切分：按业务进行拆分

数据共享问题

消息队列

【业务之间通信有哪些方式】

消息队列：异步

PRC：同步

【消息队列实操】

4.6 分布式系统，去掉切分后的重复部分

分布式：以前一台电脑完成的多个步骤，分到不同的服务器共同完成

便于：服务模块的重用、高可用、动态伸缩

缺点：

一次请求调用多个服务，风险是指数级的；

数据一致性；

调用复杂性、代码复杂性；

分布式不是好东西，分布式设计的第一要务就是不要分布式，真的有必要，再考虑（性能、承载量）

【微服务是不是更不要随便用，这么火是为什么，分布式与微服务各自适用于什么体量】

【分布式与SOA、微服务的异同】

# [对微服务、集群、SOA、分布式的理解](https://blog.csdn.net/qq_34395857/article/details/83780873)

4.7 微服务

微服务：平台架构的风格

是分布式的进化；

分布式：是随着业务演进，被迫出现的。

微服务是主动式的，拆分粒度更细

微服务=分布式事务+分布式锁+容器

最终如何应对大数据高并发？

具体情况具体分析，根据现有水准进化

至少大概知道什么问题，用什么方式解决

然后一定要擅长至少其中一个。

【选择哪个收益更大】

技术视野+实操经验