常见的系统设计面试问题：

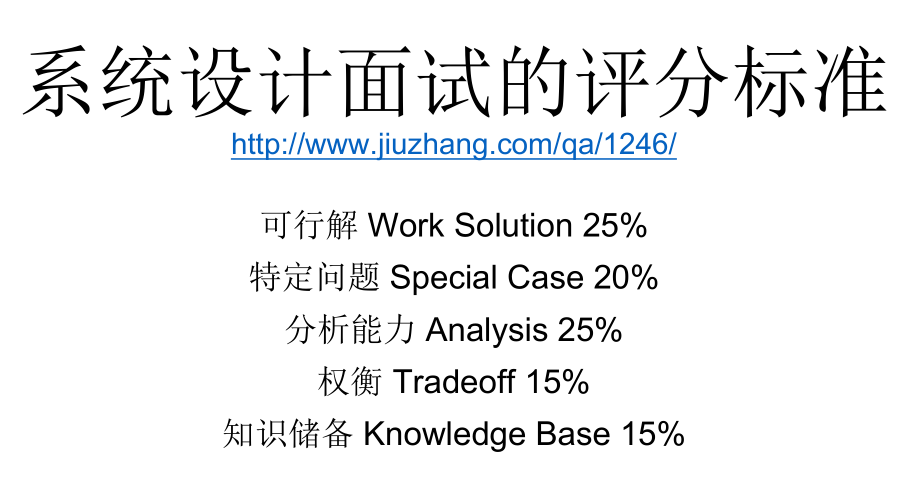


系统设计与面向对象设计的异同。

****

Tradeoff：权衡。

系统设计的评分标准：



4S分析法：

Scenario 场景：需要设计哪些功能，设计得多牛，QPS（每秒查询量），DAU（日活跃用户）等

Service 服务：将大系统拆分为小服务

Storage 存储：数据如何存储与访问。Schema SQL，NoSQL，File System

Scale 升级：解决缺陷，处理可能遇到的问题。

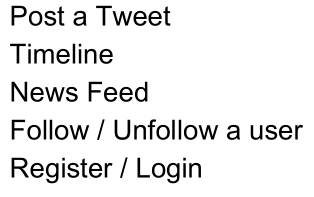
设计推特：

Scenario：

第一步：把Twitter的功能罗列出来



第二步：功能排序，选出核心功能



计算并发用户：（不要求很准确，猜测的用户只要合理就行）

日活跃用户 \* 每个用户平均请求次数/一天多少秒=150M \* 60/86400 ~100k

峰值（Twitter访问肯定每天都有1个或几个峰值段，比如早上起床，晚上睡觉，中午吃饭时间段）：平均日活跃用户\*3=300k

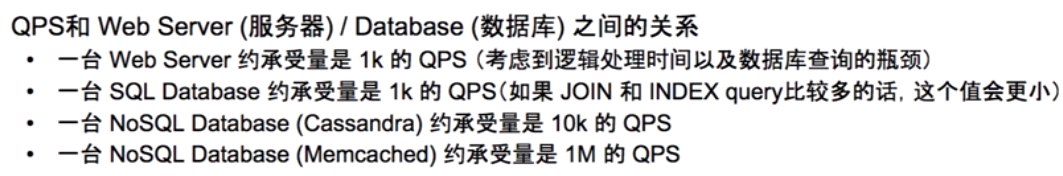
快速增长的产品：3个月最大峰值用户=峰值\*3

读频率 Read QPS：就是峰值，300k

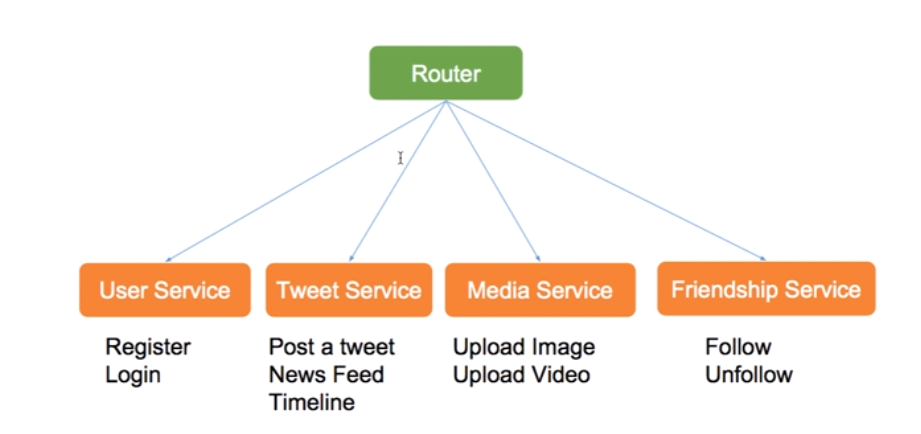
写频率Write QPS：峰值/60=5k，假设每天访问60次，但只写一次。

计算QPS的用处：





Service：



Storage 存储：

1. Select为每个Service选择存储结构
2. 细化表

数据库系统 Database：

1. 关系型数据库 Database：用户信息表 User Table，原因：技术成熟，支持多种形式的查询（可以根据Email，电话号码，姓名等查询）。SQL是精确的，它最适合于具有精确标准的定义明确的项目。典型的使用场景是在线商店和银行系统
2. 非关系型数据库 NoSQL Database：推文Tweets Follow/UnFollow，原因：Tweets增加快，查询简洁，NoSQL自带分布式。NoSQL是多变的，它最适合于具有不确定需求的数据。典型的使用场景是社交网络，客户管理和网络分析系统

SQL是精确的。它最适合于具有精确标准的定义明确的项目。典型的使用场景是在线商店和银行系统。

文件系统 File System：图片、视频文件

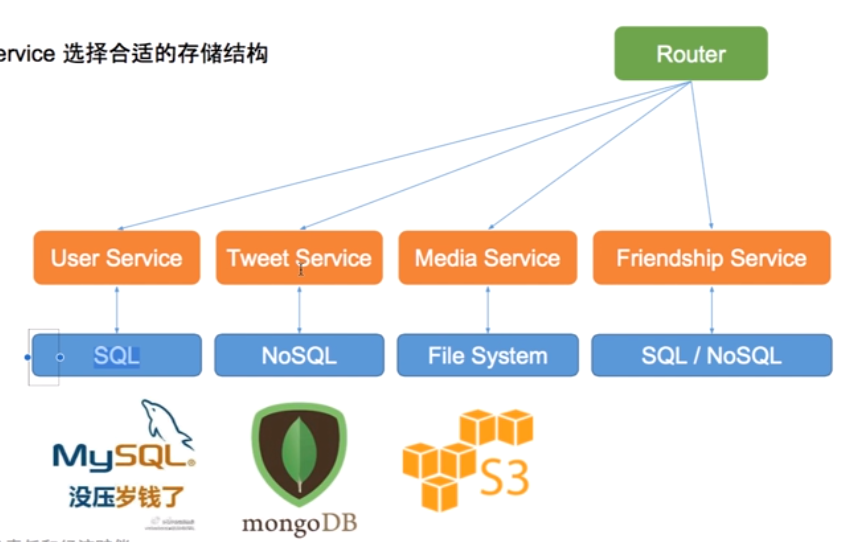
缓存系统 Cache：不支持数据持久化，效率高，访问速度快。

文件系统和数据库系统的关系：

联系：数据库系统是文件系统的一层包装，他们不是独立的关系，是依赖关系，数据库系统依赖于文件系统。

区别：数据库系统提供了更丰富的数据操作，很细；文件系统只提供简单的文件操作接口，很粗。例如，关系型数据库提供了SQL语言这样的丰富的查询语言，可以进行复杂的过滤，如快速查找20-24岁的学生信息。如果直接在文件系统上，则需要扫描完所有的学生数据才能找到。

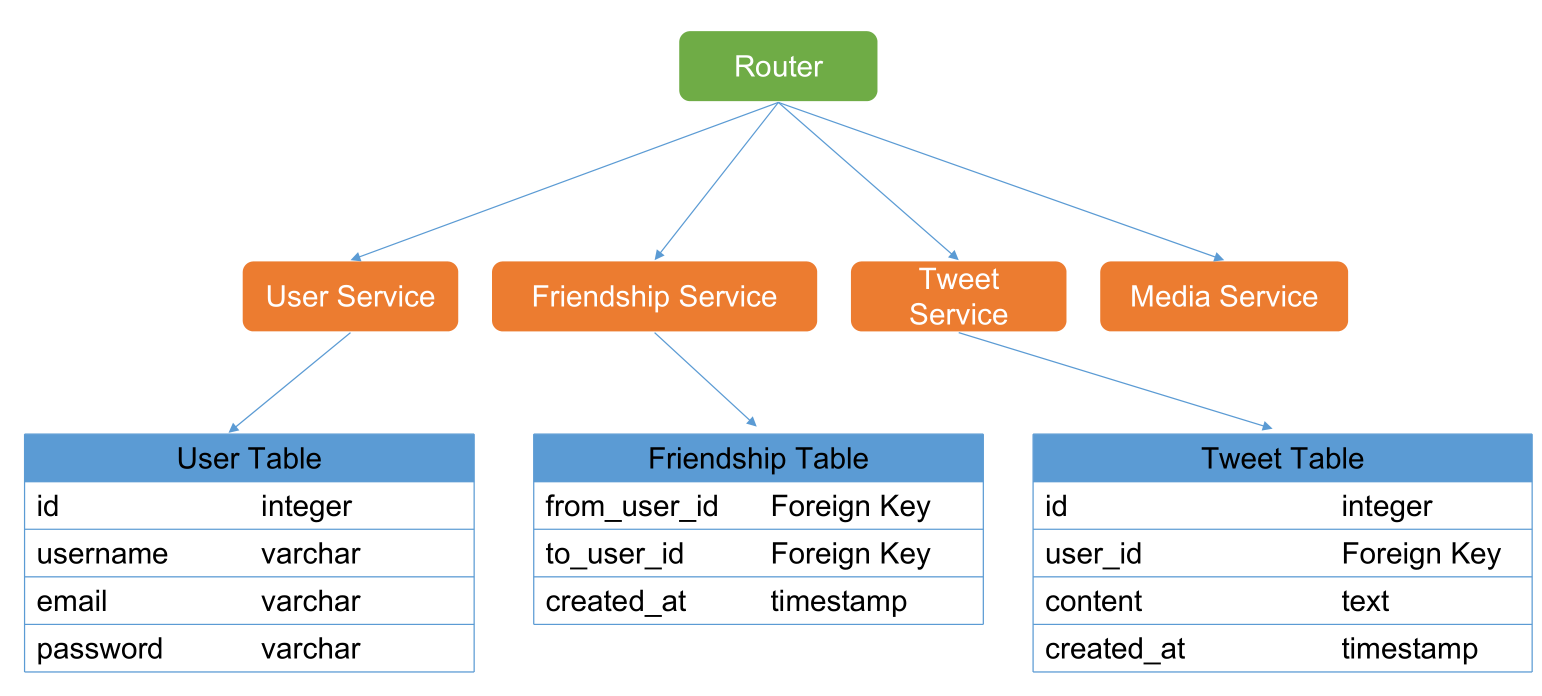
数据库系统读取的数据，大部分情况下（不考率Cache）都会从文件系统中读取，因此两个系统的读写效率（不考虑复杂查询）可以认为差不多。



程序 = 算法+数据结构

系统 = 服务+数据存储

细化表结构：



Friendship Table 中Foreign key是外键，表示from\_user\_id和to\_user\_id都是其他表的键。

新鲜事系统：例如Facebook/Twitter/朋友圈

核心因素：关注与被关注，每个人看到的新鲜事都是不同的。

新鲜事系统是Twitter设计的核心，它的存储有两种模式：Pull模式和Push模式。

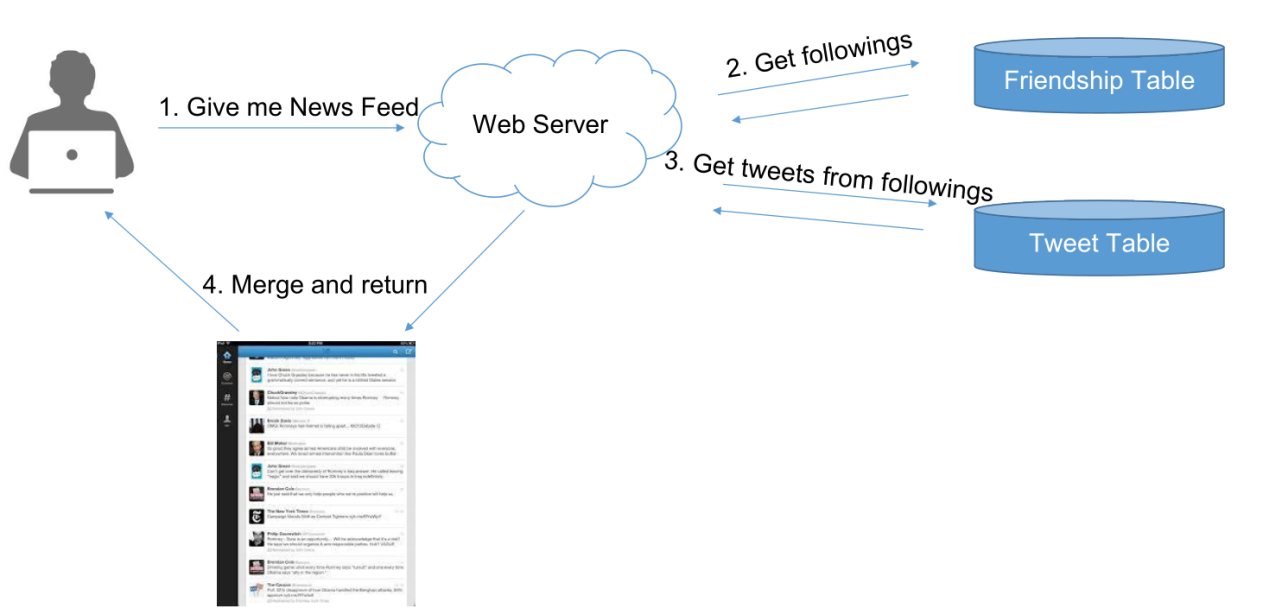
Pull模式：

算法：用户查看News Feed时，获取每个好友的前100条Tweets，合并出前100条News Feed，K路归并算法。

复杂度分析：假如有N个关注对象，则为N次DB Reads的时间+N路归并时间（可忽略，N路归并是在内存中进行，而DB Reads是读取数据库，从磁盘中读取，I/O的读取速度并内存读取低很多）。

发一个Tweet：1次DB Write时间。

Pull模式的原理图：



Pull模式的区别：N次DB Reads比较慢，且发生在用户获得News Feed的请求过程中（有些系统会进行预加载，用户翻了前100条，会预加载100-200条，依次类推）。

Pull模式下，也需要news feed table，因为如果用户要翻看之前已经看过的News feed时，不可能还重新计算，就要把之前计算的结果存储到数据库中，可以使用Nosql，类似于缓存。

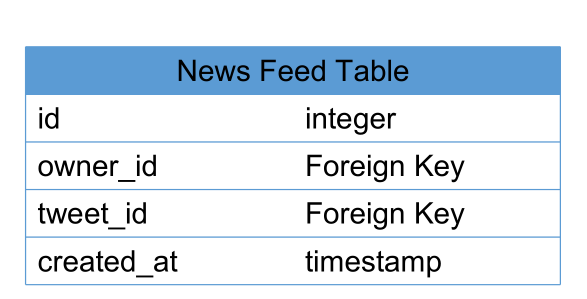
Push模式：

需要一个新的table，News Feed Table：

owner\_id：发tweet的人是谁

tweet\_id：tweet的内容

created\_at：发帖的时间



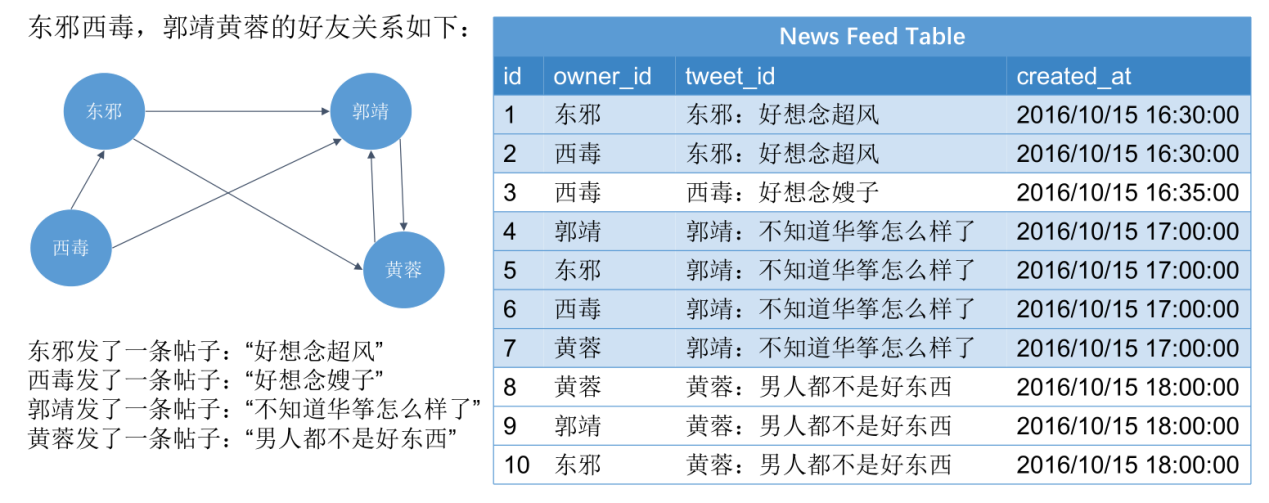
算法：

1. 为每个用户建一个List存储他的News Feed信息
2. 用户发一个Tweet之后，将该Tweet逐个推送到每个用户的News Feed List中，关键词：Fanout(写扩散)
3. 用户需要查看News Feed时，只需要从News Feed List中读取最新的100条就行了。

复杂度分析：

News Feed => 1次DB Read

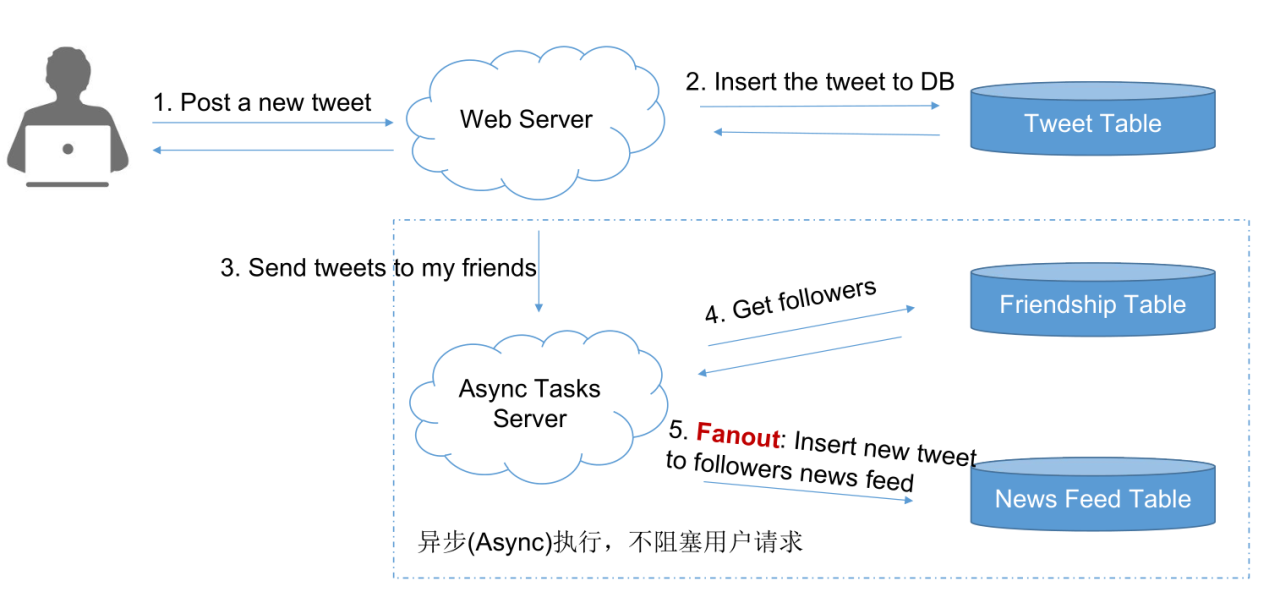
Post a tweet => N个粉丝，需要N次DB Writes，可以用异步任务在后台执行，无需用户等待。



东邪发了一个帖子后，东邪默认自己关注自己，所以表中创建一条记录，owner\_id为东邪。

西毒关注了东邪，表中创建了一条记录，owner\_id为西毒，两条记录的内容是相同的，同样的内容存储了两次，这个叫denormalized，非规范化添加数据的冗余副本来提高数据库的可读性。

Push模式的流程图：



Push模型的缺陷：

1. 粉丝数目可能很大，导致Fanout过程很长，从而导致用户刷到新鲜事有延迟
2. 浪费系统资源为很多僵尸粉创建新鲜事记录
3. 明星发帖会在短时间内为系统带来很大的处理压力



广告的插入是Pull模型，广告商发布广告时不可能给每个受众都插入广告记录，而且广告商会优化广告的投放，肯定不适合在发布时就给所有受众都插入一条记录，而是会根据用户的不同，在用户浏览新鲜事时选择发给或不发给用户。

Scale 扩展：

Step 1：Optimize 优化

1. 解决设计缺陷

Pull模式的缺陷解决：

最慢的部分发生在用户读请求：

* 1. 在DB访问之前加入Cache，通常用Memcached，Cache类似一个链表，每个用户都在Cache中有一个key，缓存每个用户最近例200条的tweets，如果用户新发了tweet，就把链表头的删除，把新tweet插入到尾部，类似LRU缓存。

缓存每个用户的Timeline(用户的tweets)

N次DB请求转变为N次Cache请求，不用Cache所有的Tweets，而且Cache最近的200条左右

缓存每个用户的News Feed：

没有Cache News Feed的用户：归并N个用户最近的100条News Feed

有Cache News Feed的用户：归并N个用户的某个时间戳之后的所有Tweets

Push模式的缺陷：

浪费了更多的存储空间 Disk

与Pull模式将News Feed存在内存中相比，Push模型将News Feed存在硬盘上完全不是个事儿，Disk is Cheap。

不活跃用户：

经常登陆的粉丝，优先写。

粉丝数目远大于关注数目

例如 Lady Gaga在Twitter有8000万用户，谢娜在微博有1亿用户，Fanout的过程可能需要几个小时。

尝试在现有的模型做最小的改动，例如加几台机器做Push任务

对长期的增长进行估计，并评估是否值得转换整个模型

Push+Pull的优化方案：

1. 普通用户仍然Push
2. 将Lady Gaga这类用户标记为明星用户。对于明星用户，不Push到用的News Feed中。
3. 当用户需要的时候，首先从News Feed Table中获取和自己有关的News Feed，然后根据自己关注的明星用户，来明星用户的Timeline里取，并合并到News Feed中。

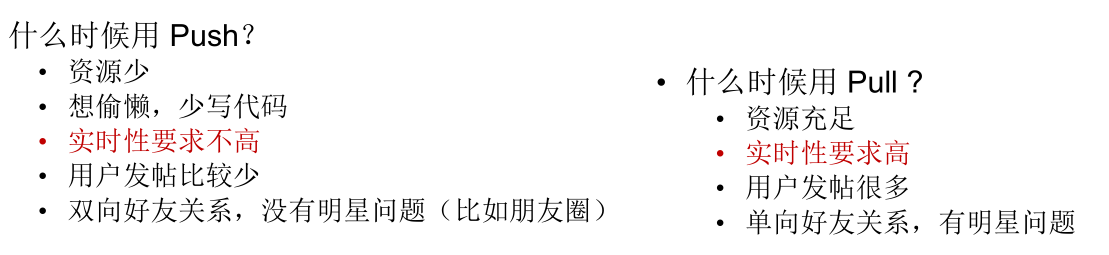
定义明星用户：不能直接根据粉丝数目>100万来判断是否是明星用户，例如，假如邓超开始是明星用户，他发一个帖子，因为是明星，所以系统不Push，结果帖子内容糟糕，导致粉丝取关，变成了普通用户，这时如果你上线并且之前是邓超的粉丝，在获取News Feed时，用于此时的邓超是普通用户，所以不会去Pull邓超的帖子，而之前系统又没有把邓超的帖子写入到News Feed Table中，结果导致你丢失了帖子。

解决办法：不动态去计算你关注的用户是否是明星，而是离线计算，在User表中，定义一个字段is\_superstar，当一个用户被定义为明星后，就不能再被取消标记。

Pull模式 VS Push模式

大部分社交网站都是Pull，但仍然需要学习Push

系统设计不是选择一个最好的方案，而是选择一个最合适的方案。



双向好友和单向好友：

微博：你关注了一个明星，这就是单向好友关系；一个明星可能会有百万个粉丝的关注，一对多的关系，明星发一条讯息需要通知到百万个用户。push扛不住啊

微信：你和某人加了好友，这个就是双向好友关系。你们之间发送消息是一对一的关系。

Pull 和 Push适用的场景不同，需要根据场景来的。"

1. 更多功能设计
2. 一些特殊情况：僵尸粉，鹿晗关晓彤搞挂微博

Step 2：Maintenance 维护

鲁棒性：如果一台服务器/数据库挂了怎么办？

扩展性：如果流量暴增，如何扩展？

系统设计的总结：

先问清楚再设计

不要一上来就冲着一个巨牛的方案取设计

切忌不要做关键词大师

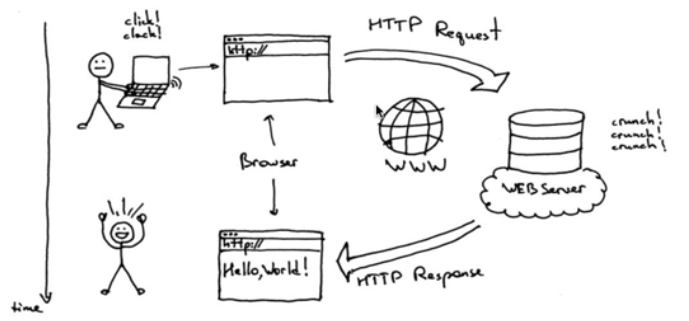
No more no less，设计够用的系统

系统设计没有标准答案。

网站服务器(Web Server)：[www.google.com](http://www.google.com)等网站背后的一台或者若干台机器，提供HTTP/HTTPS服务

HTTP/HTTPS叫超文本协议，主要用于网页浏览器和网站服务器之间通讯用的协议，HTTP是明文传输，HTTPS是加密传输。

一台性能较好的Web Server，每秒可以支持大约1000次访问（大概的估算值）。



数据库(Database)：存储数据的仓库，一般在内网中访问，一般和Web Server打交道。

一个合理的架构中，Web Server和Database Server是不同的机器

数据库适合存储结构化数据（数据库中的一个表对应面向对象中的一个类）

文件系统(File System)：

数据库系统是基于文件系统而存在的。

非结构化的数据直接存储在文件系统中，如\*.avi，\*.jpg。

缓存(Cache)：是一个相对概念

1. 可以在内存中
2. 可以在磁盘中
3. 可以在CPU里（L1 Cache/L2 Cache）
4. 可以在服务器端
5. 可以在客户端（如浏览器cache）

系统设计中Cache默认是内存中。内存中的Cache可以理解为哈希表，是一种Key-Value结构。

常见的Cache软件是Memcached。

通常把经常访问的数据放在Cache中，因为空间受限制，所以需要淘汰掉一些不常用的数据，常见的淘汰算法，LRU。

Memcached和Redis的区别：

memcached所有的值均是简单的字符串，redis作为其替代者，支持更为丰富的数据类型。

因为redis有数据的持久化，需要读写磁盘，所以memcached的速度比redis快很多。

redis支持数据的备份，即master-slave模式的数据备份。

使用底层模型不同，它们之间底层实现方式 以及与客户端之间通信的应用协议不一样。redis自己构建了VM 管理机制。

value大小不同，redis最大可以达到512MB，而memcache只有1MB。