# 系统设计篇

四火老师的博客： https://www.raychase.net/6364

系统设计中快速估算技巧： https://www.raychase.net/6280

搞定面试中的系统设计题: https://jiajunhuang.com/articles/2019\_04\_29-system\_design.md.html

一个国外的课程 System Design for Tech Interviews https://www.hiredintech.com/courses/system-design

上面的课程的解读 https://blog.csdn.net/u013007900/article/details/79008993 https://blog.csdn.net/u013007900/article/details/79049187

ResumeJob的知乎文章，比较具体 https://zhuanlan.zhihu.com/p/77857433

九章算法的知乎文章 https://zhuanlan.zhihu.com/p/70743436 https://zhuanlan.zhihu.com/p/162937258

guide哥的系统设计的文章 https://mp.weixin.qq.com/s/1Jl8ee0jJoFdGMfIhl\_qaQ

## 系统设计基础知识点

网站服务器 web server - 一台性能比较好的web server， 大概每秒可以服务1000次访问请求

数据库 Database - 系统设计中最重要的考点

文件系统 File System

缓存 Cache - Cache是一个相对概念 - 可以在内存上 - 可以在磁盘上 - 可以在CPU里 - 可以在服务端 - 可以在客户端 - 我们在系统设计中说的Cache的时候默认是Memcache，即内存中的Cache - 我们通常把经常访问的数据放在Cache里来加速访问速度 - Cache因为空间受限制，因此需要淘汰掉一些不常用的数据，常见的淘汰算法有LRU

## 系统设计介绍

### 系统设计面试的形式

系统设计的题目往往是一个开放性的题目

比如：设计微博，设计微信

或者是设计某某系统中的某某功能。比如设计一个功能实现对用户访问频率的限制，设计一个功能实现统计某个具体事件的历史发生次数。

### 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

形式上： 面向对象设计手把手的 Coding 系统设计高屋建瓴的“扯淡”

考察的知识点上： 面向对象设计：Class, Object, Method, Inheritance, Interface … 系统设计考的是：Database, Schema, SQL, NoSQL, Memcached, File System, Distributed System, Latency, Scalbility, Master Slave, Load Balancer, Web Server, Message Queue, Sharding, Consistent Hashing, QPS …

典型题： 面向对象设计：电梯设计，游戏设计 系统设计：短网址系统设计，新鲜事系统设计

### 从 News Feed Design 介绍什么是系统设计

系统设计中首先要明确需求，面试官的问题可能很大，但是让你设计的东西未必会很多，设计的难度也未必会很大。极有可能从易到难引导你先设计一些简单的结构。

第一步，先细化问题

### 系统设计面试的常见错误

### 系统设计面试的评分标准

可行解 Work Solution 25% 特定问题 Special Case 20% 分析能力 Analysis 25% 权衡 Tradeoff 15% 知识储备 Knowledge Base 15%

### 系统设计的九阴真经—— 4S 分析法

Scenario, Service, Storage, Scale

没有思路时，可以采取这四个点

* Scenario 场景
  + 说人话：需要设计哪些功能，设计得多牛
  + Ask / Features / QPS / DAU / Interfaces
  + DAU日活用户
* Service 服务
  + 说人话：将大系统拆分为小服务
  + Split / Application / Module
* Storage 存储
  + 说人话：数据如何存储与访问
  + Schema / Data / SQL / NoSQL / File System
* Scale 升级
  + 说人话：解决缺陷，处理可能遇到的问题
  + Sharding / Optimize / Special Case

#### scenairo 场景

需要设计哪些功能，设计得多牛 1. Ask 问面试官 2. Analysis 分析

哪些功能，QPS是多少

**ASK**

询问面试官： - 需要设计哪些功能（也可以自己想） - 需要承受多大的访问量？ - 日活跃用户 Daily Active Users (DAU) - Twitter: MAU 330M, DAU ~170M+ - Read more: http://bit.ly/1Kml0M7

通常MAU是DAU的两倍

**MAU：月活跃用户**

需要设计哪些功能

第一步 Step 1：Enumerate - 说人话：把Twitter的功能一个个罗列出来 - Register / Login - User Profile Display / Edit - Upload Image / Video  *- Search*  - Post / Share a tweet - Timeline / News Feed - Follow / Unfollow a user

第二步 Step 2：Sort - 说人话：选出核心功能，因为你不可能这么短的时间什么都设计 - Post a Tweet - Timeline - News Feed - Follow / Unfollow a user - Register / Login

Analysis & Predict

并发用户 Concurrent User - 日活跃 \* 每个用户平均请求次数 / 一天多少秒 = 150M \* 60 / 86400~ 100k - 峰值 Peak = Average Concurrent User \* 3 ~ 300k - 快速增长的产品 Fast Growing - MAX peak users in 3 months = Peak users \* 2 读频率 Read QPS (Queries Per Second) - 300k 写频率 Write QPS - 5k

除了算平均请求数，还要算峰值

**分析出QPS有什么用**

QPS = 100 - 用你的笔记本做 Web 服务器就好了 QPS = 1k - 用一台好点的 Web 服务器就差不多了 - 需要考虑 Single Point Failure QPS = 1m - 需要建设一个1000台 Web 服务器的集群 - 需要考虑如何 Maintainance（某一台挂了怎么办）

QPS和 Web Server (服务器) / Database (数据库) 之间的关系 - 一台 Web Server 约承受量是 1k 的 QPS （考虑到逻辑处理时间以及数据库查询的瓶颈） - 一台 SQL Database 约承受量是 1k 的 QPS（如果 JOIN 和 INDEX query比较多的话，这个值会更小） - 一台 NoSQL Database (Cassandra) 约承受量是 10k 的 QPS - 一台 NoSQL Database (Memcached) 约承受量是 1M 的 QPS

单台web server能到1k是非常理想的状况 写了Join会很慢，要写轻量级的SQL

#### service 服务

将大系统拆分为小服务 1. Replay 重放需求 2. Merge 归并需求

**将大系统拆分为小服务**

User Service - Register - Login Tweet Service - Post a tweet - News Feed - Timeline Friendship Service - Follow - Unfollow Media Service - Upload Image - Upload Video

第一步 Step 1: Replay - 重新过一遍每个需求，为每个需求添加一个服务 第二步 Step 2: Merge - 归并相同的服务 什么是服务 Service? - 可以认为是逻辑处理的整合 - 对于同一类问题的逻辑处理归并在一个 Service 中 - 把整个 System 细分为若干个小的 Service

#### Storage 存储

数据如何存储与访问 1. Select 为每个 Service 选择存储结构 2. Schema 细化表结构

**数据如何存储与访问**

数据库系统 Database - 关系型数据库 SQL Database - 用户信息 User Table - 非关系型数据库 NoSQL Database - 推文 Tweets - 社交图谱 Social Graph (followers) - 例如存储在mongoDB 文件系统 File System - 图片、视频 Media Files - 例如S3 缓存系统 Cache - 不支持数据持久化 Nonpersistent - 效率高，内存级访问速度

第一步 Step 1: Select - 为每个 Application / Service 选择合适的存储结构 第二步 Step 2: Schema - 细化数据表结构

* 程序 = 算法 + 数据结构
* 系统 = 服务 + 数据存储

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。 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。

你可以先这么记着，nosql就是类似json存储，sql就是类似excel存储 不同类别的数据 或 不相近属性的数据 分表存储，也可以是功能拆分。

**数据系统vs文件系统**

关系：数据库系统是文件系统的一层包装，他们不是独立的关系，是依赖的关系。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；文件系统只提供了简单的文件操作接口，很粗。 以关系型数据库(Relational Database) 为例，提供了 SQL 语句这样的丰富的查询语言，可以一些复 杂的 filter，如快速找出学生信息表中，所有 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后才能找到。

数据库系统中读取的数据，大部分情况下（除了被 cache 的），都还是会到文件系统上去读取出来的。 因此两个系统的读写效率（不考虑复杂查询）可以认为是差不多的。

**请设计数据库的表结构**

User Table id integer username varchar email varchar password varchar

Friendship Table from\_user\_id Foreign Key to\_user\_id Foreign Key created\_at timestamp

Tweet Table id integer user\_id Foreign Key content text created\_at timestamp

#### News Feed 如何存取？

什么是新鲜事 News Feed？ - 你登陆 Facebook / Twitter / 朋友圈 之后看到的信息流 - 你的所有朋友发的信息的集合 有哪些典型的新鲜事系统？ - Facebook - Twitter - 朋友圈 - RSS Reader 新鲜事系统的核心因素？ - 关注与被关注 - 每个人看到的新鲜事都是不同的

**Pull Model**

算法 - 在用户查看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 Merge K Sorted Arrays

复杂度分析 - News Feed => 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) - Post a tweet => 1次DB Write的时间

为什么 N 路归并算法的耗时可以忽略？

假设一共 N 个关注的好友，每个好友100条信息。多路归并的时间复杂度是 O(100NlogN)，这里使用了一个大小为 N 的堆。初始化这个堆需要 O(N) 的时间，归并出 100N 条信息，每一条需要 O(logN) 的时间。值得注意的是，读 DB 的时间是获取了 O(100N) 条数据，这个从时间复杂度上来说要更低，但是因为 DB 的操作事实上是在对文件进行操作，而文件的访问比内存访问慢几百上千倍。

磁盘和内存的访问差异是1000倍

**Pull模型有什么缺陷么？**

现算，比较慢

N次DB Reads非常慢 且发生在用户获得News Feed的请求过程中

#### Push Model

算法 - 为每个用户建一个List存储他的News Feed信息 - 用户发一个Tweet之后，将该推文逐个推送到每个用户的News Feed List中 - 关键词：Fanout - 用户需要查看News Feed时，只需要从该News Feed List中读取最新的100条即可

复杂度分析 - News Feed => 1次DB Read - Post a tweet => N个粉丝，需要N次DB Writes - 好处是可以用异步任务在后台执行，无需用户等待

News Feed Table id integer owner\_id Foreign Key tweet\_id Foreign Key created\_at timestamp

**Push模型有缺陷么？**

记住一句话 "Disk is cheap"，不要怕浪费数据库存储，为了加速查询，多存一些东西是没关系的。

**Pull VS Push**

热门Social App的模型 - Facebook – Pull - Instagram – Push + Pull - Twitter – Pull - 朋友圈 - ? 误区 - 不坚定想法，摇摆不定 - 不能表现出Tradeoff的能力 - 无法解决特定的问题

广告是Pull模型： 广告的投放人群很多，Push 模式很费时间，无法实时生效。且广告主可以筛选和改变投放人群，因此 Push 模型并不能很好的支持这种动态的改动。

#### 4S

用过前3个步骤的分析，我们已经得到了一个可行方案 Scenario 场景 - 和面试官讨论 - 搞清楚需要设计哪些功能 - 并分析出所设计的系统大概所需要支持的 Concurrent Users / QPS / Memory / Storage 等 Service 服务 - 合并需要设计功能，相似的功能整合为一个Service Storage 存储 - 对每个 Service 选择合适的存储结构 - 细化数据表单 - 画图展示数据存储和读取的流程 得到一个 Work Solution 而不是 Perfect Solution 这个Work Solution 可以存在很多待解决的缺陷

#### Scale 扩展

How to Scale? 系统如何优化与维护 1. Optimize 优化 2. Maintenance 维护

**如何优化系统**

第一步 Step 1: Optimize - 解决设计缺陷 Solve Problems - Pull vs Push - 更多功能设计 More Features - Like, Follow & Unfollow, Ads - 一些特殊情况 Special Cases - 鹿晗关晓彤搞挂微博, 僵尸粉

第二步 Step 2: Maintenance - 鲁棒性 Robust - 如果有一台服务器/数据库挂了怎么办 - 扩展性 Scalability - 如果有流量暴增，如何扩展

**解决Pull的缺陷**

最慢的部分发生在用户读请求时（需要耗费用户等待时间） - 在 DB 访问之前加入Cache - Cache 每个用户的 Timeline - N次DB请求 → N次Cache请求 (N是你关注的好友个数) - Trade off: Cache所有的？Cache最近的1000条？ - Cache 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Tweets，然后取出结果的前100条 - 有Cache News Feed的用户༚ 归并N个用户的在某个时间戳之后的所有Tweets

课后作业：对比MySQL 和 Memcached 的 QPS - Memcached QPS / MySQL QPS ~ 100 ~ 1000

DB和Cache之间相差一百倍到一千倍的差距

**解决Push的缺陷**

浪费更多的存储空间 Disk - 与Pull模型将News Feed存在内存(Memory)中相比 - Push模型将News Feed存在硬盘(Disk)里完全不是个事儿 - Disk is cheap 不活跃用户 Inactive Users - 粉丝排序 Rank followers by weight (for example, last login time) 粉丝数目 followers >> 关注数目 following - Lady Gaga问题 - 无解？完全切换回Pull？ - rade off: Pull + Push vs Pull

粉丝 Followers 80 M - Justin Bieber 95 M on Instagram - 谢娜 100M on Weibo Push 的挑战 - Fanout 的过程可能需要几个小时！ 面试时错误的回答方案 - 既然 Push 不行，那我们就切换到 Pull 吧！ - 说起来好容易啊！ - 正确的思路 - 尝试在现有的模型下做最小的改动来优化 - 比如多加几台用于做 Push 任务的机器，Problem Solved! - 对长期的增长进行估计，并评估是否值得转换整个模型

**Push结合Pull的优化方案**

Push 结合 Pull 的优化方案 - 普通的用户仍然 Push - 将 Lady Gaga 这类的用户，标记为明星用户 - 对于明星用户，不 Push 到用户的 News Feed 中 - 当用户需要的时候，来明星用户的 Timeline 里取，并合并到 News Feed 里

**如何定义明星**

不是明星不能在线动态计算，要离线计算 - 为 User 增加一个 is\_superstar 的属性 - 一个用户被标记为 superstar 之后，就不能再被取消标记

User Table id integer username varchar email varchar password varchar is\_superstar boolean

**Pull vs Push**

为什么既然大家都用Pull，我们仍然要学习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 - 如果你没有很大的流量，Push是最经济最省力的做法 - 系统设计面试也并不是期望你答出最优的解决方法，而是从你的分析当中判断你对系统的理解和知识储备。

什么时候用 Push？ - 资源少 - 想偷懒，少写代码 - 实时性要求不高 - 用户发帖比较少 - 双向好友关系，没有明星问题（比如朋友圈）

什么时候用 Pull ? - 资源充足 - 实时性要求高 - 用户发帖很多 - 单向好友关系，有明星问题

**通用问题**

* 数据库服务器挂了怎么办？How to maintenance?
* 用户逐渐怎么怎么办？How to scale?
  + 服务器顶不住压力怎么办？
  + 数据库顶不住压力怎么办？
* 以上两个问题，将在第二节课 Database 的专题中涉及！

### 系统设计面试总结

Ask before design 问清楚再动手设计 不要一上来就冲着一个巨牛的方案去设计 切忌不要做关键词大师

No more no less 不要总想着设计最牛的系统 要设计够用的系统

Work solution first 先设计一个基本能工作的系统，然后再逐步优化 Done is better than perfect! —— Mark Zuckerberg

Analysis is important than solution 系统设计没有标准答案 记住答案是没用的 通过分析过程展示知识储备 权衡各种设计方式的利弊

### 拓展问题

**拓展问题1：果取关问题**

如何实现 follow & unfollow ? 除了在数据库中创建/删除记录，还需要做什么？

如何实现 follow 与 unfollow? - Follow 一个用户之后，异步地将他的 Timeline 合并到你的 News Feed 中 - Merge timeline into news feed asynchronously. - Unfollow 一个用户之后，异步地将他发的 Tweets 从你的 News Feed 中移除 - Pick out tweets from news feed asynchronously. 为什么需要异步 Async？ - 因为这个过程一点都不快呀 异步的好处？ - 用户迅速得到反馈，似乎马上就 follow / unfollow 成功了 异步的坏处？ - Unfollow 之后刷新 News Feed，发现好像他的信息还在 - 不过最终还是会被删掉的

**拓展问题2： 如何存储likes**

如何在 News Feed 中同时得到每个帖子被点赞、评论和转发的次数？

Tweet Table id integer user\_id Foreign Key content text created\_at timestamp num\_of\_likes \* integer num\_of\_comments \* integer num\_of\_retweets \* integer

Like Table \* id integer user\_id Foreign Key tweet\_id Foreign Key created\_at timestamp

De-normalize

Normalize vs Denormalize

Normalize 获得点赞数的方式： SELECT COUNT \* FROM like\_table where tweet\_id=xxx; 优点：标准化，最准确。 缺点：炒鸡慢，会增加 O(N) 个 SQL Queries（对于某一页的 Tweets，每个都得来这么一句查询） Denormalize 获得点赞数的方式：

当有人点赞的时候： UPDATE like\_table SET num\_of\_likes = num\_of\_likes + 1 where tweet\_id = xxx 当有人取消赞的时候： UPDATE like\_table SET num\_of\_likes = num\_of\_likes - 1 where tweet\_id = xxx 想要获得一个 Tweet 的点赞数时，因为 num\_of\_likes 就存在 tweet 里，故无需额外的 SQL Queries

**鹿晗公布恋情会怎样**

惊群现象 Thundering Herd

数据分片机制不管用 因为访问的是同一个数据，sharding 机制无论如何都会 sharding 到同一个机器上。此时 sharding 不能做到分摊流量的作用。

什么是惊群？

我们通常会使用缓存来作为数据库的“挡箭牌”，优化一些经常读取的数据的访问速度。即，在访问这些 数据时，会先看看是否在缓存中，如果在，就直接读取缓存中的数据，如果不在，就从数据库中读取 之后，写入缓存并返回。 那么在高并发的情况下，如果一条非常热的数据，因为缓存过期或者被淘汰算法淘汰等原因，被踢出 缓存之后，会导致短时间内（<1s），大量的数据请求会出现缓存穿透 (Cache miss)，因为数据从 DB 回填到 Cache 需要时间。从而这些请求都会去访问数据库，导致数据库处理不过来而崩溃，从而影响 到其他数据的访问而导致整个网站崩溃。

（关于缓存的几种场景，我要额外去看下）

录屏+手写

window:surface pencil Mac: iPad+apple pencil+airServer