

走进系统设计与新鲜事系统

Introducing System Design & News Feed System

课程版本:v6.0 本节主讲人:东邪



扫描二维码关注微信小程序/公众号
获取第一手求职资料

版权声明

九章的所有课程均受法律保护，不允许录像与传播录像
一经发现，将被追究法律责任和赔偿经济损失

讲师介绍

- 曾就职于亿级日活跃用户(DAU)的社交网络公司
- 多年基础架构(Infrastructure)经验
- 多年网站开发(Web Development)经验
- 北美顶尖IT企业Offer数10+



今日课程大纲

- 系统设计面试的形式
- 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 
- 从 News Feed Design 介绍什么是系统设计
- 系统设计面试的常见错误
- 系统设计面试的评分标准
- 系统设计的九阴真经——**4S 分析法**
- 后续课程安排

今天你可以学到

系统设计中常说的 (Tradeoff) 是什么

什么叫做 SOA (Service Oriented Architecture)

什么是 Pull Model & 什么是 Push Model

数据存储系统有哪些, 什么样的数据适合存在什么样的数据存储系统中

什么是异步任务和消息队列 (Message Queue)

什么是数据的可持久化 (Persistent)

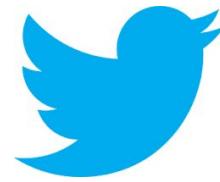
什么是去标准化 (Denormalize)

什么是惊群效应 (Thundering Herd)

有哪些与 News Feed 类似的系统设计问题?

- **设计某某系统 Design XXX System**

- 设计微博 Design Twitter
- 设计人人 Design Facebook
- 设计滴滴 Design Uber
- 设计微信 Design Whatsapp
- 设计点评 Design Yelp
- 设计短网址系统 Design Tiny URL
- 设计NoSQL数据库 Design NoSQL



- **设计某某系统中的某某功能**

- 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
- 设计删除一个 Tweet 的功能
- 设计邮件系统中将所有邮件标记为已读的功能

不需要写代码

需要写代码

系统设计 vs 面向对象设计



面试形式上有什么不同？哪种面试
需要写代码？

系统设计 vs 面向对象设计

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, Scalability, 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

<https://www.lintcode.com/ladder/8/>

注册 LintCode & 绑定九章账号即可开启, 参考答案在 www.jiuzhang.com/solutions 中可查

The screenshot shows the LintCode Ladder interface for learning system design. It features three main sections: 1 - Introduction to System Design & News Feed System, 2 - Database & User System, and 3 - Consistent Hashing and Design Tiny URL. Each section lists problems with their difficulty levels (Medium or Easy) and completion status. A sidebar on the right provides course details and statistics.

| Section | Problem ID | Difficulty | Status | Progress (%) |
|--|-------------------------|------------|-------------|--------------|
| 1 - Introduction to System Design & News Feed System | 501. Design Twitter | Medium | Completed | 17% |
| 2 - Database & User System | 519. Consistent Hashing | Easy | In Progress | 29% |
| | 538. Memcache | Medium | Completed | 31% |
| | 502. Mini Cassandra | Medium | In Progress | 29% |
| 3 - Consistent Hashing and Design Tiny URL | | | | |

System Design
Jiuzhang Users
Related problems for Nine Chapters online system design live course.
More information on:
<http://www.jiuzhang.com/>

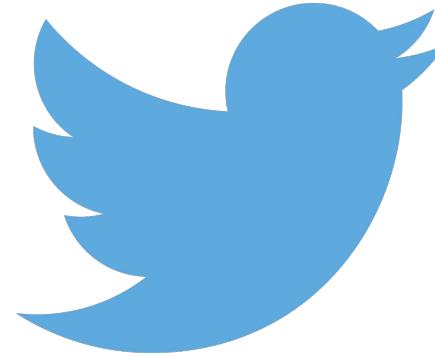
9 Levels 9 Unlocked 44 Problems

面试官：请设计推特

Interviewer: Please design twitter



你第一句对面试官说的话是什么？



- ① 用户有多少？
- ② 需要我设计哪些功能？
- ③ 质疑他的前提条件？

常见错误：关键词大师

Load Balancer, Memcache, NodeJS, MongoDB, MySQL,
Sharding, Consistent Hashing, Master Slave, HDFS, Hadoop ...

你想过没有：或许现在只有2个用户呢？

系统设计面试的评分标准

<http://www.jiuzhang.com/qa/1246/>

可行解 Work Solution 25% *work is better than perfect*

特定问题 Special Case 20%

分析能力 Analysis 25%

权衡 Tradeoff 15% *做A好? 什么时候B好?*

知识储备 Knowledge Base 15%

系统设计问题的 4S 分析法

Scenario, Service, Storage, Scale



- **Scenario** 场景

- 说人话: 需要设计哪些功能, 设计得多牛
 - Ask / Features / QPS / DAU / Interfaces

- **Service** 服务

query parse daily Active User

- 说人话: 将大系统拆分为小服务
 - Split / Application / Module

- **Storage** 存储

最重要的部分, 如何存, 如何取? 怎么快?

- 说人话: 数据如何存储与访问
 - Schema / Data / SQL / NoSQL / File System

- **Scale** 升级

数据变大, 该怎么办?

- 说人话: 解决缺陷, 处理可能遇到的问题
 - Sharding / Optimize / Special Case

QPS → DAU → Storage → Scale

twitter: 发表 | 删帖 | news feed

Work Solution
NOT Perfect Solution

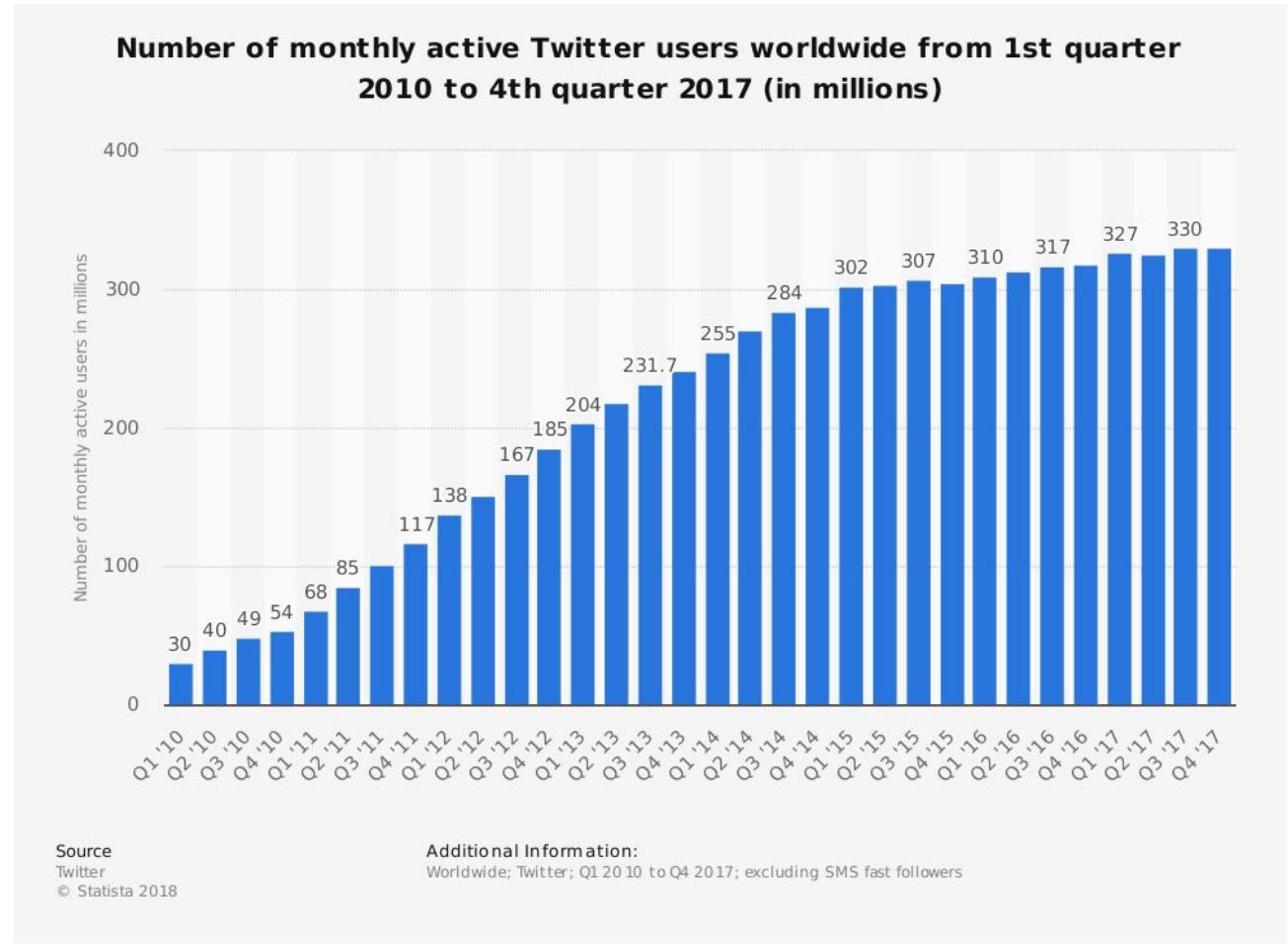
Scenario 场景

需要设计哪些功能，设计得多牛

1. Ask 问面试官
2. Analysis 分析

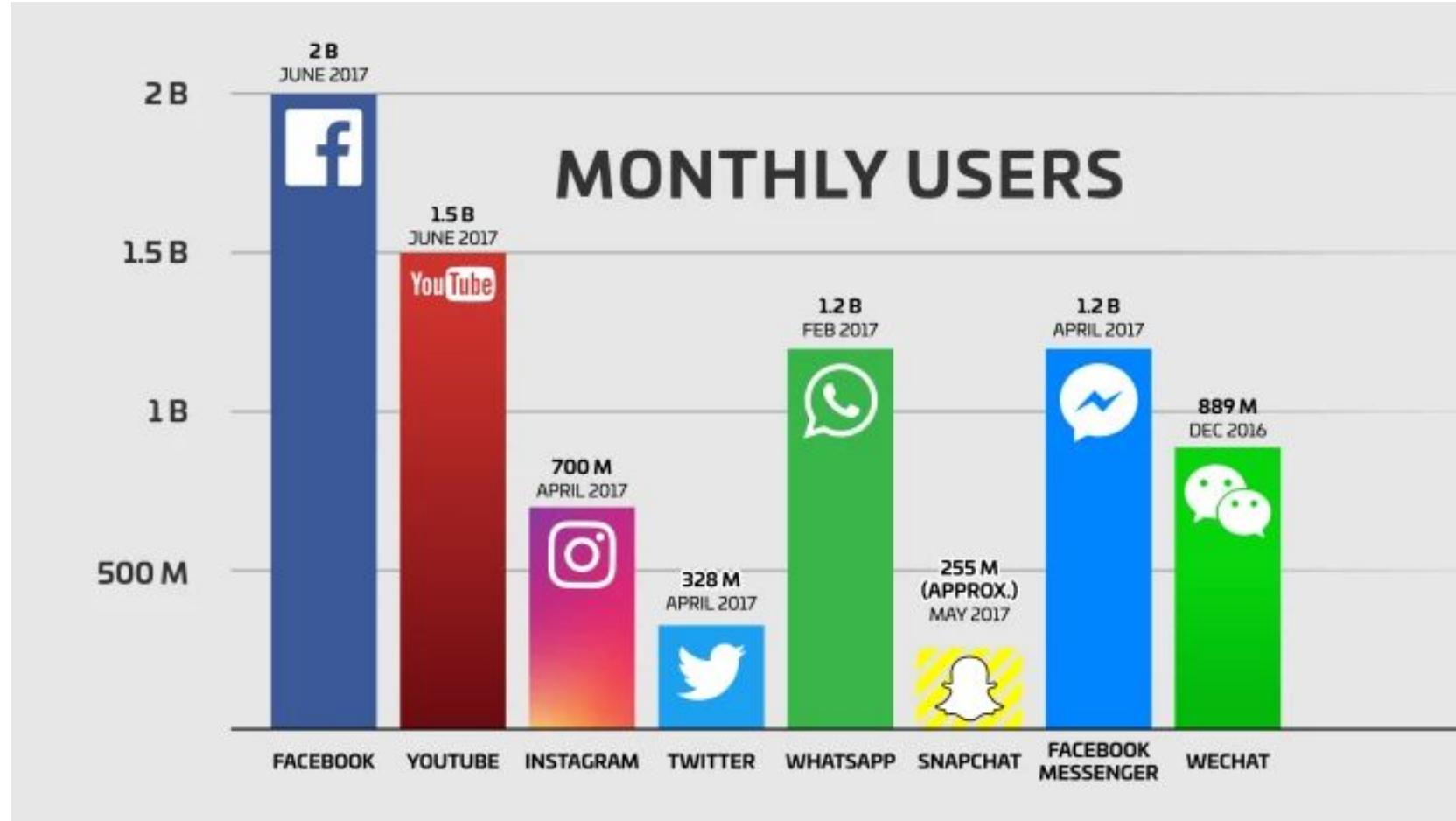


- 询问面试官：
news feed, timeline
- 需要设计哪些功能(也可以自己想)
- 需要承受多大的访问量？
 - 日活跃用户 Daily Active Users (DAU)
 $DAU = MAU / 2$
 - Twitter: MAU 330M, DAU ~170M+
 - Read more: <http://bit.ly/1Kml0M7>



一般不是用注册用户，而是用月活跃用户来代表一个网站的用户数

17年数据



Scenario 场景 – 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 来个用户的[信息](#)
- News Feed
- Follow / Unfollow a user
- Register / Login



- 并发用户 Concurrent User
 - 日活跃 * 每个用户平均请求次数 / 一天多少秒 = $150M * 60 / 86400 \sim 100k$
 - 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

猜的

重要的不是计算结果
而是计算过程

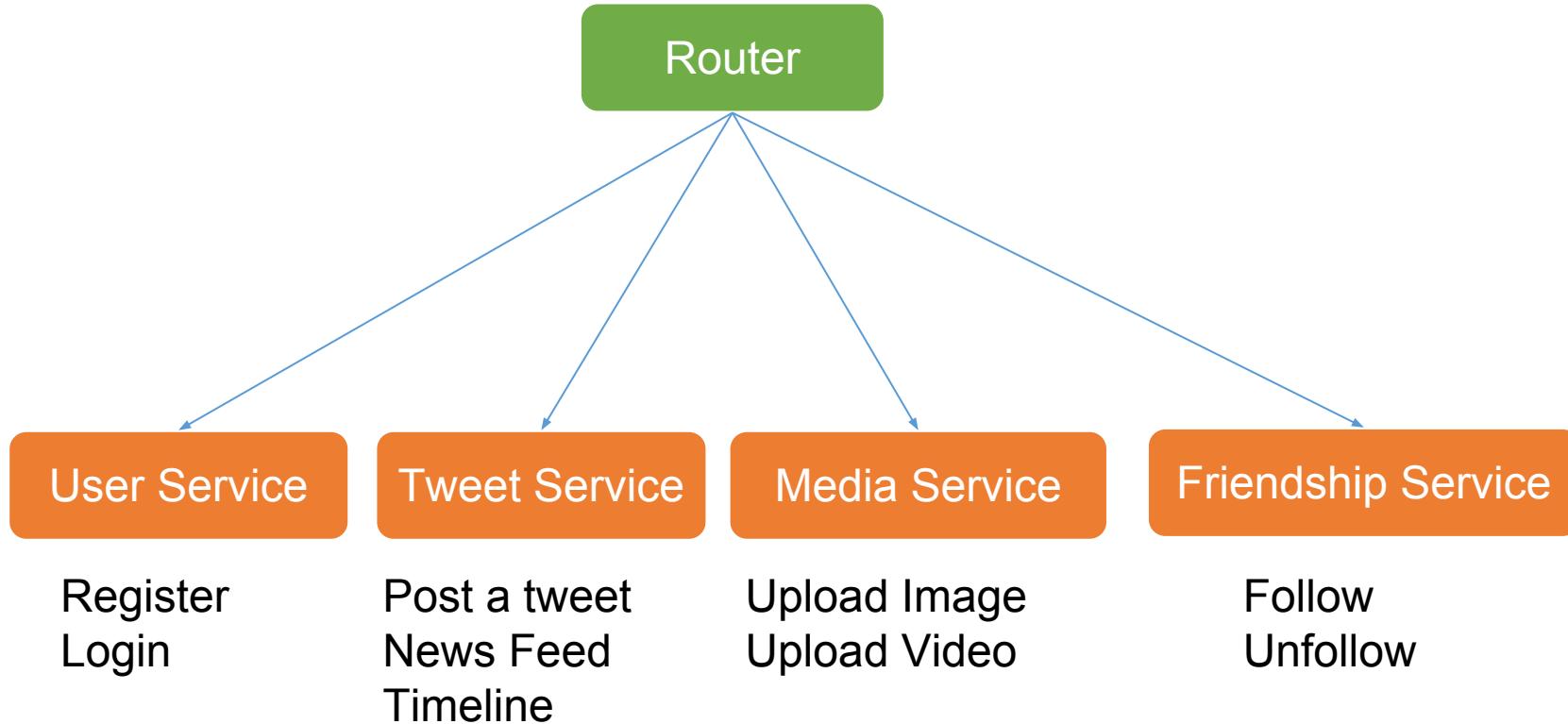
分析出 QPS 有什么用？

- $QPS = 100$
 - 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 服务垮掉了
- $QPS = 1m$
 - 需要建设一个1000台 Web 服务器的集群
 - 需要考虑如何 Maintenance(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)
写非常笨重的 query, not use join at all
 - 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

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



- 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 细分为若干个大的 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)
- 文件系统 File System
 - 图片、视频 Media Files
- 缓存系统 Cache
 - 不支持数据持久化 Nonpersistent
 - 效率高, 内存级访问速度



文件系统和数据库系统的关系是什么？

是文件系统一层包装

数据库系统 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。

Storage 存储 – 数据如何存储与访问

• 第一步 Step 1: Select

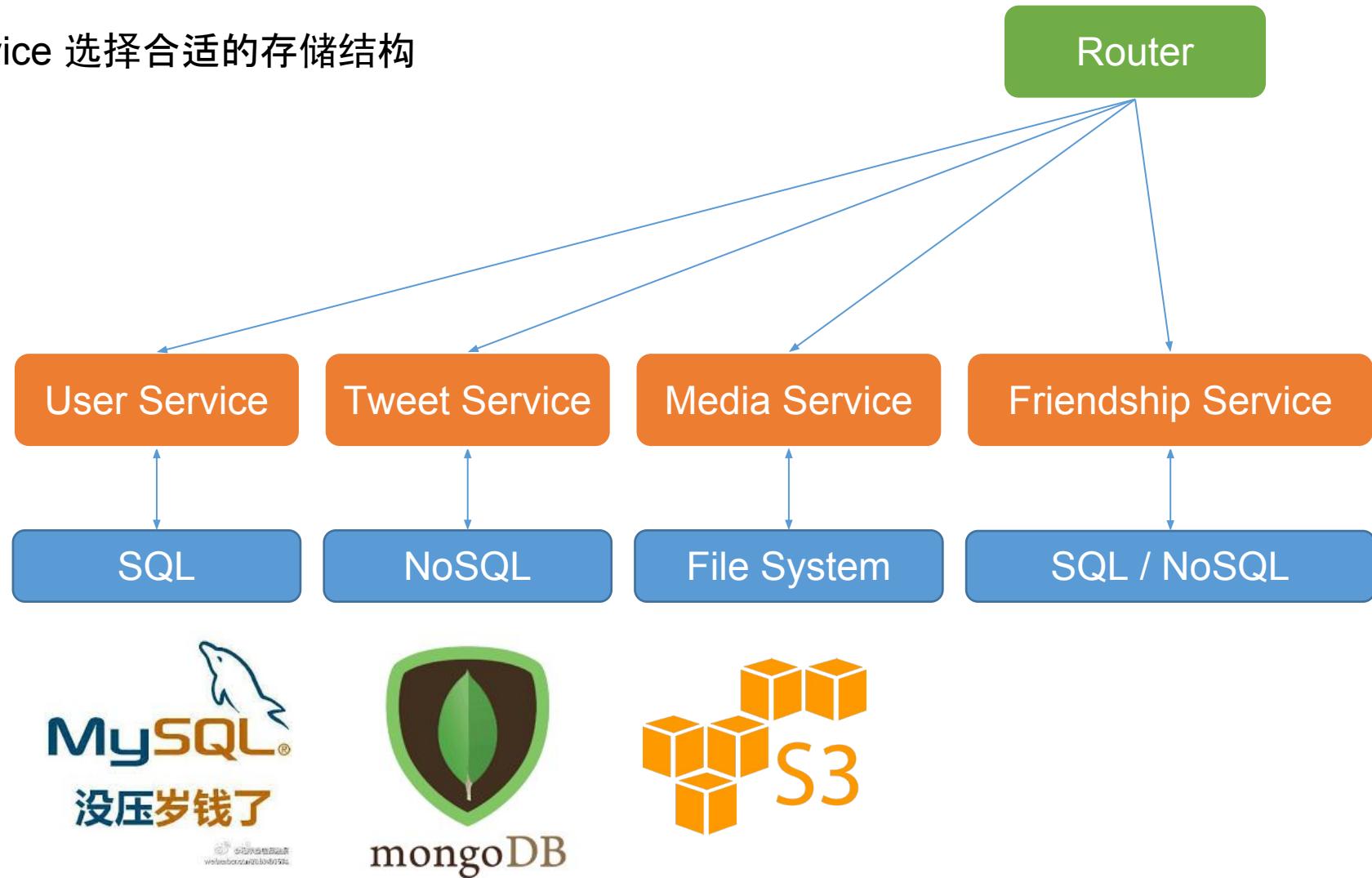
- 为每个 Application / Service 选择合适的存储结构

• 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



Interviewer: Please design schema



请设计数据库的表结构

Storage 存储 – 数据如何存储与访问

$pk = reference$ 用于唯一标识数据属性 not share

MySQL

| | id |
|---|------|
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| : | : |

sequential id

FK 外键

$User.id$

NoSQL

无主键分布式创建

Router

User Service

Friendship Service

Tweet Service

Media Service

User Table

| | |
|------------|---------|
| id | integer |
| $username$ | varchar |
| $email$ | varchar |
| $password$ | varchar |

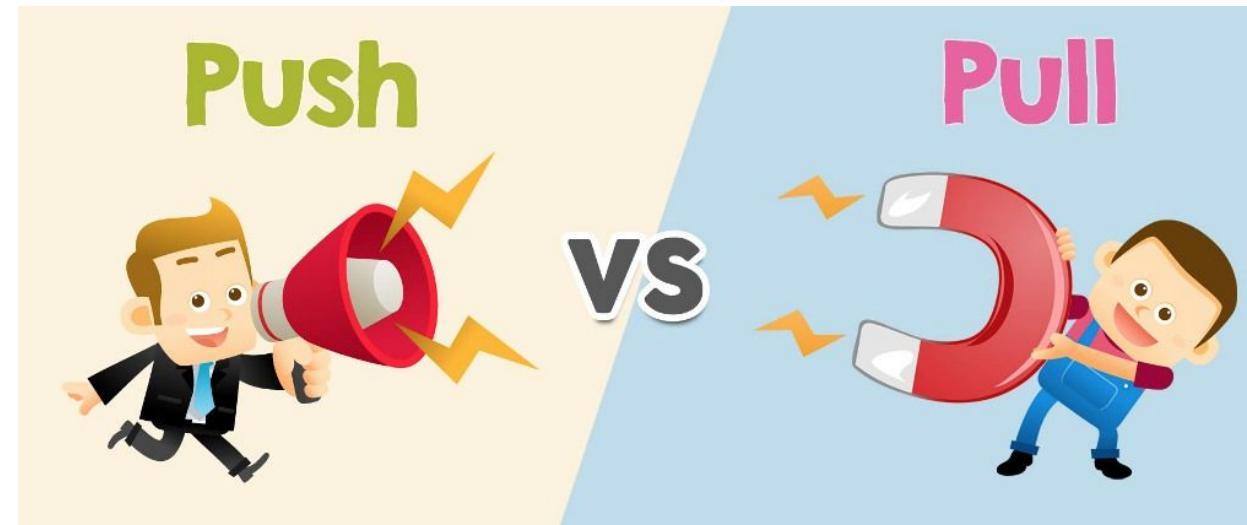
Friendship Table

| | |
|------------------|-------------|
| $from_user_id$ | Foreign Key |
| to_user_id | Foreign Key |
| $created_at$ | timestamp |

Tweet Table

| | |
|---------------|-------------|
| id | integer |
| $user_id$ | Foreign Key |
| $content$ | text |
| $created_at$ | timestamp |

Interviewer: News Feed 如何存取？



透扩散 / 写扩散

- 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
- 新鲜事系统的根本因素 ?
 - 关注与被关注
 - 每个人看到的新鲜事都是不同的



- 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

啥区别

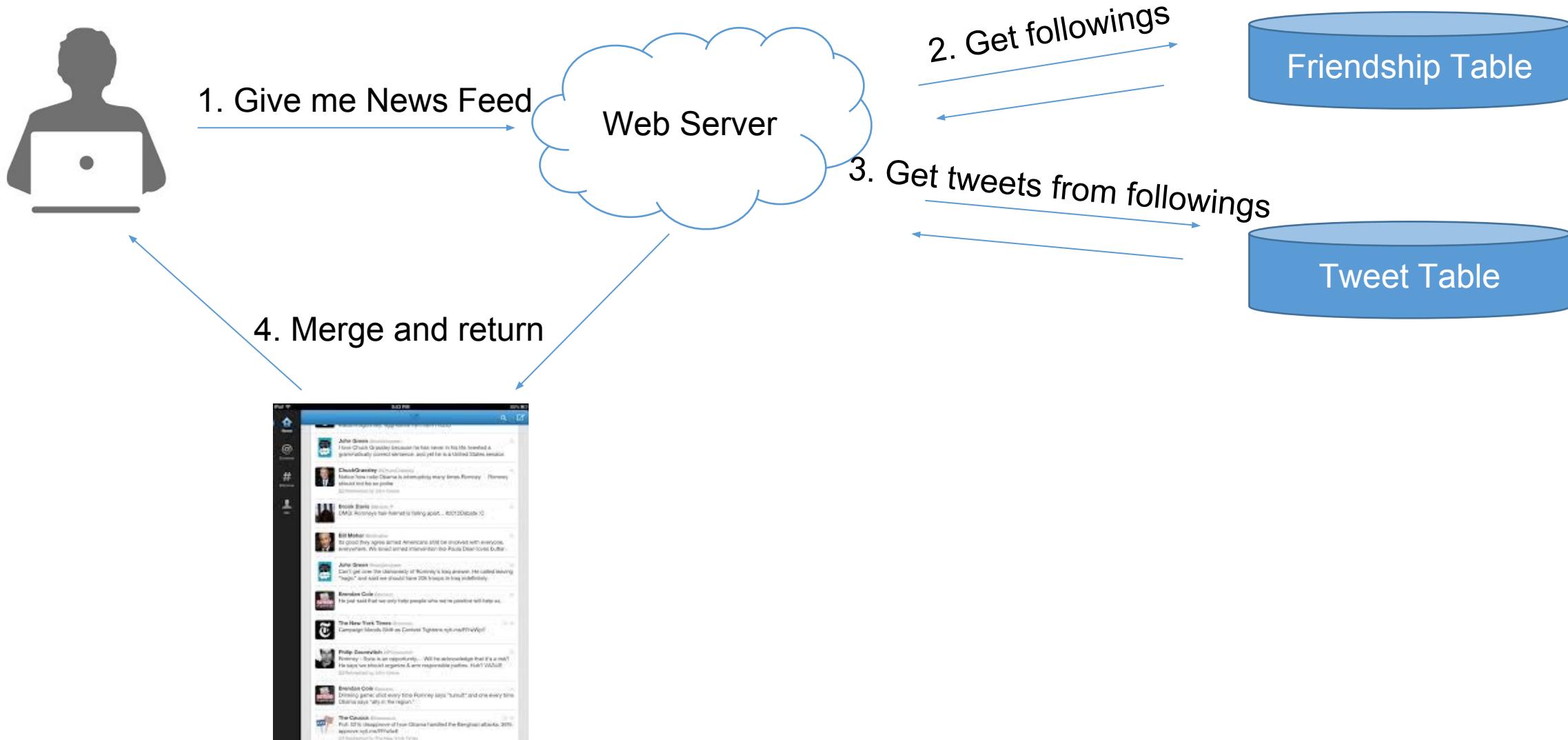
需要这个的时候主动去要

Pull = 主动撩妹



为什么 N 路归并算法
的耗时可以忽略?





Interviewer: Pull模型有什么缺陷么？



Pull 的缺陷是什么？

用户想要获得才开始算慢

Storage 存储 – Pull Model

- `getNewsFeed(request)`
 - `followings = DB.getFollowings(user=request.user)`
 - `news_feed = empty`
 - `for follow in followings:`
 - `tweets = DB.getTweets(follow.to_user, 100)`
 - `news_feed.merge(tweets)`
 - `sort(news_feed)`
 - `return news_feed[:100] # 返回前100条`

卡住

N次DB Reads非常慢
且发生在用户获得News
Feed的请求过程中

- `postTweet(request, tweet)`
 - `DB.insertTweet(request.user, tweet)`
 - `return success`

Denormalized 叫去标准化。通俗的解释是，通过在不同的 Table 中存储同一份数据的（也就是说至少一份是冗余数据）的形式，来加速数据的查询。因为当数据只存储在一个固定的 Table A 的时候，其他 Table B 访问时如果需要同时取得关联的 Table A 的数据，则需要进行 join 操作之类的，会比较慢。一个例子就是在，统计有多少人点赞了一个帖子，可以通过 `select count(*) from like_table where post_id=<id>` 的方式来获取，但是也可以在 post table 中新增一个 `like_count`，每次点赞就 +1。这里 `like_count` 就是一个 denormalized field，因为是可以通过 `select count(*)` 在 like table 中获得的。

denormalize

- 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

Async task

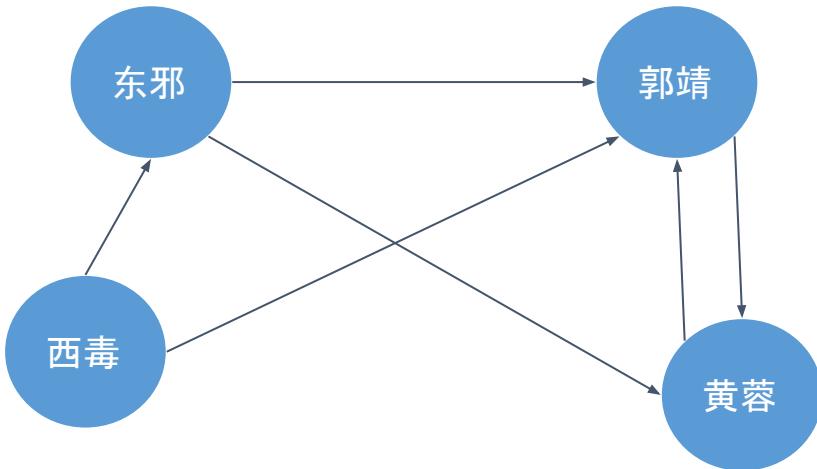
| News Feed Table | |
|-----------------|-------------|
| id | integer |
| owner_id | Foreign Key |
| tweet_id | Foreign Key |
| created_at | timestamp |

which is PK
Fk is a field in another table

Push = 坐等被撩



东邪西毒，郭靖黄蓉的好友关系如下：



东邪发了一条帖子：“好想念超风”

西毒发了一条帖子：“好想念嫂子”

郭靖发了一条帖子：“不知道华筝怎么样了”

黄蓉发了一条帖子：“男人都不是好东西”

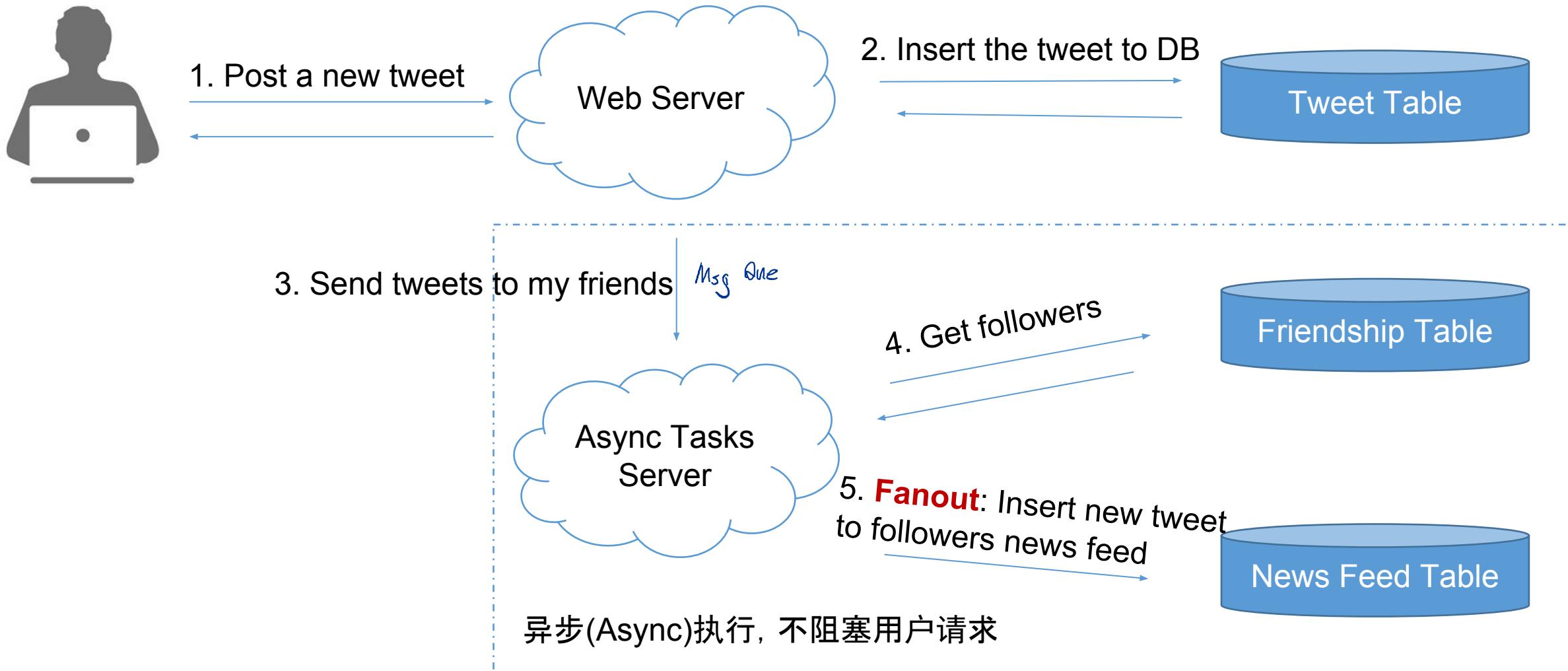
| News Feed Table | | | |
|-----------------|----------|--------------|---------------------|
| id | owner_id | tweet_id | created_at |
| 1 | 东邪 | 东邪:好想念超风 | 2016/10/15 16:30:00 |
| 2 | 西毒 | 东邪:好想念超风 | 2016/10/15 16:30:00 |
| 3 | 西毒 | 西毒:好想念嫂子 | 2016/10/15 16:35:00 |
| 4 | 郭靖 | 郭靖:不知道华筝怎么样了 | 2016/10/15 17:00:00 |
| 5 | 东邪 | 郭靖:不知道华筝怎么样了 | 2016/10/15 17:00:00 |
| 6 | 西毒 | 郭靖:不知道华筝怎么样了 | 2016/10/15 17:00:00 |
| 7 | 黄蓉 | 郭靖:不知道华筝怎么样了 | 2016/10/15 17:00:00 |
| 8 | 黄蓉 | 黄蓉:男人都不是好东西 | 2016/10/15 18:00:00 |
| 9 | 郭靖 | 黄蓉:男人都不是好东西 | 2016/10/15 18:00:00 |
| 10 | 东邪 | 黄蓉:男人都不是好东西 | 2016/10/15 18:00:00 |

黄蓉登陆“射雕APP”之后可以看到的所有帖子通过一句 SQL 查询可以拿到：

select * from news_feed_table where owner_id=黄蓉 order_by created_at desc limit 20;

composit index owner_id + desc created_ts

Storage 存储 – Push 原理图



Interviewer: Push模型有缺陷么？



Push 的缺陷是什么？

① 粉丝数太多

Storage 存储 – Push Model

- getNewsFeed(request)
 - return DB.getNewsFeed(request.user)
- postTweet(request, tweet_info)
 - tweet = DB.insertTweet(request.user, tweet_info)
 - AsyncService.fanoutTweet(request.user, tweet)
 - return success
- AsyncService::fanoutTweet(user, tweet)
 - followers = DB.getFollowers(user)
 - for follower in followers:
 - DB.insertNewsFeed(tweet, follower)

异步执行

followers的数
目可能很大



Pull vs Push



哪一种方式更好？
你喜欢撩妹还是被撩？



- 热门Social App的模型

- Facebook – Pull
- Instagram – Push + Pull
- Twitter – Pull
- 朋友圈 - ?^{pull}

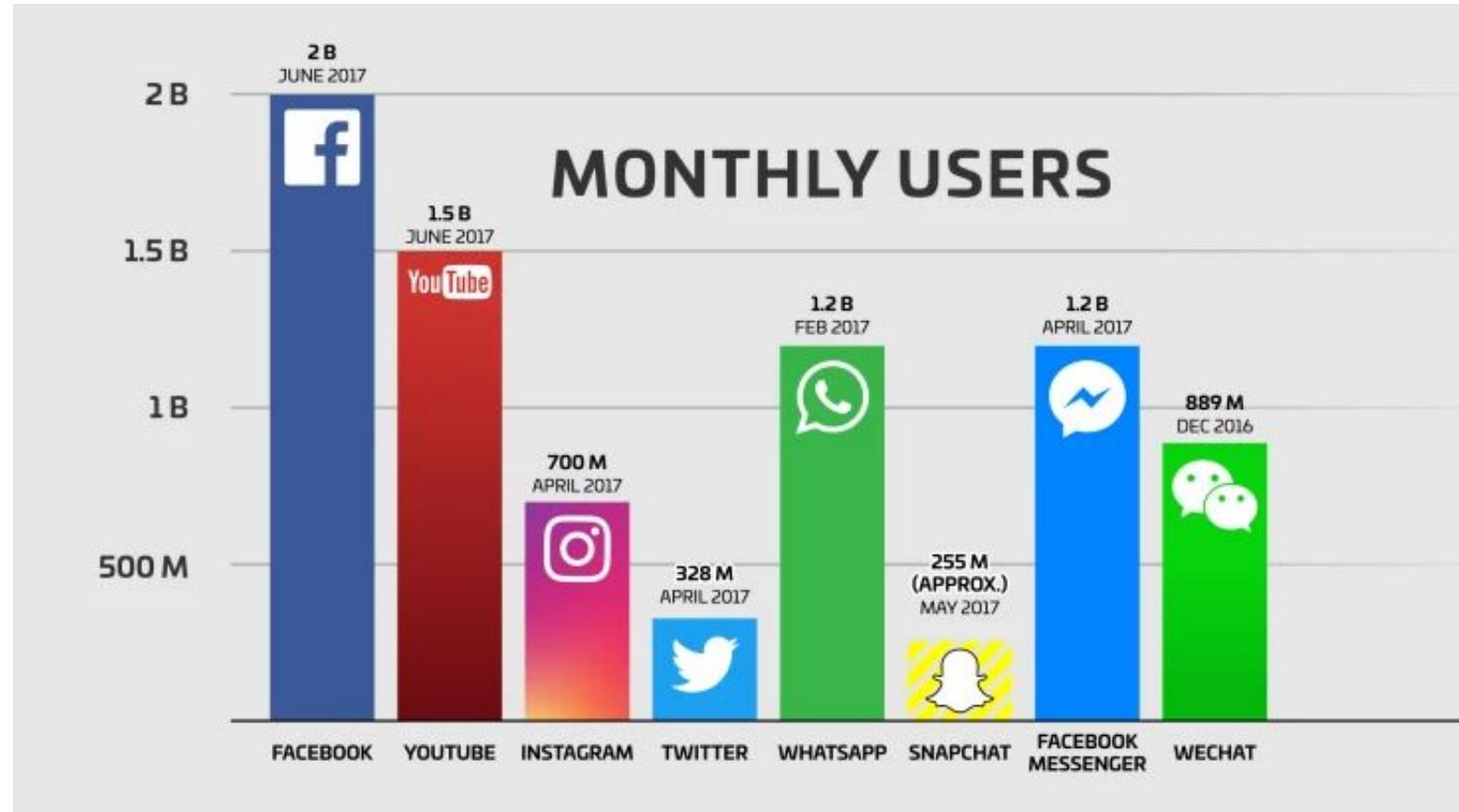
- 误区

- 不坚定想法, 摆摆不定
- 不能表现出Tradeoff的能力
- 无法解决特定的问题



猜猜微信朋友圈用的是什么模型？

朋友圈广告是 pull 模型



- 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

- 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

Scale 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 不用每次都归并
 - 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

Scale 扩展 – 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？
 - Trade off: Pull + Push vs Pull

Scale 扩展 – Lady Gaga

- 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

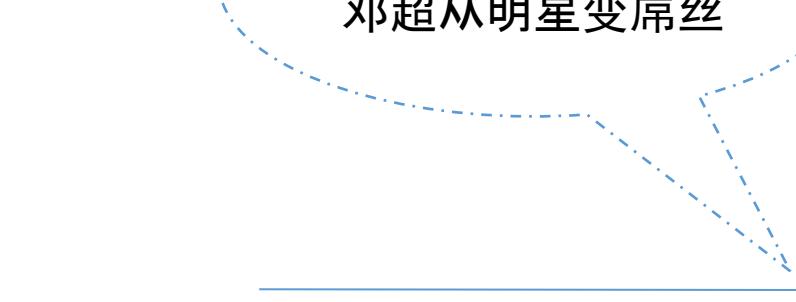


- Push 结合 Pull 的优化方案
 - 普通的用户仍然 Push
 - 将 Lady Gaga 这类的用户, 标记为明星用户
 - 对于明星用户, 不 Push 到用户的 News Feed 中
 - 当用户需要的时候, 来明星用户的 Timeline 里取, 并合并到 News Feed 里

如何定义明星？

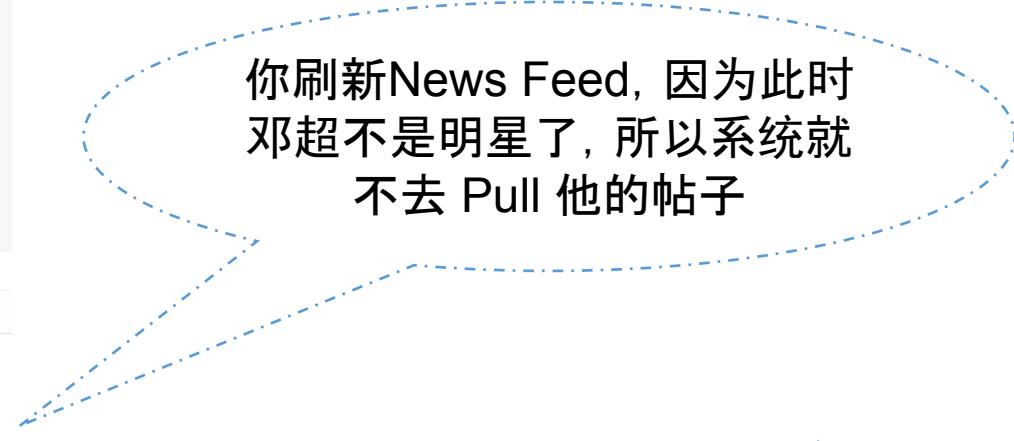
单纯的用 followers > 1m 是否有问题？

许多粉丝果取关
邓超从明星变屌丝



时间轴

你刷新News Feed, 因为此时
邓超不是明星了, 所以系统就
不去 Pull 他的帖子



邓超发了一个帖
子, 因为是明星,
所以系统不Push



娱乐 影视评论 修改

如何看待邓超微博刷屏掉粉16万? / 修改

27日晚, 邓超的微博突然在1小时内连转近80条微博, 转发微博内容均为普通观众对《恶棍天使》的好评, 转发词统一为“碗得服”配“秋田犬”表情。《恶棍天使》...显示全部 ▾

关注问题

写回答



添加评论



分享



邀请回答



举报

...

如何定义明星

- 是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 |

Scale 扩展 – 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法
 - 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

- 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

Conclusion

4S

Scenario, Service, Storage, Scale



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

后续课程安排

九章邀你华山论剑

<http://www.jiuzhang.com/course/2/>



学员福利

九章QA板块访问权限(助教老师答疑, 最新面经, 内推资源)

学员专属QQ群(助教老师答疑, 同学一起打鸡血)

LintCode 系统班阶梯训练 1 年使用权限

QA

谢谢大家



扫描二维码关注微信小程序/公众号
获取第一手求职资料

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？

拓展问题1: 果取关问题

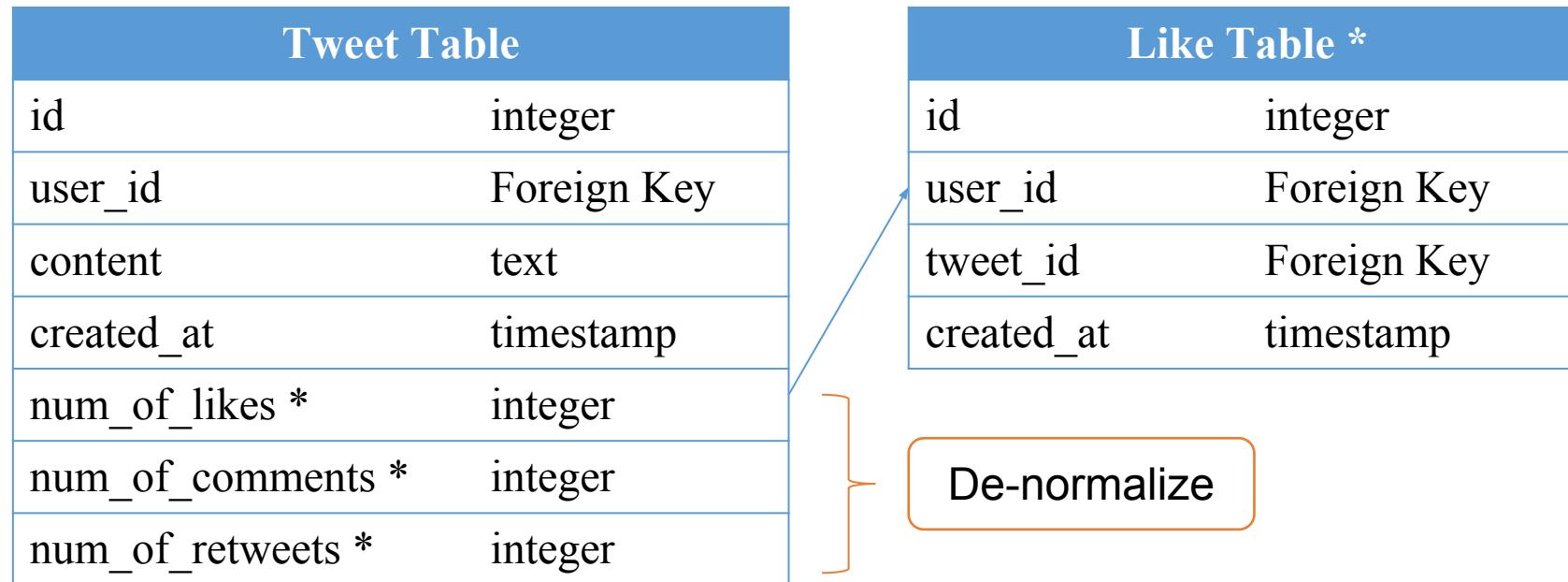
- 如何实现 follow 与 unfollow?
 - Follow 一个用户之后, 异步地将他的 Timeline 合并到你的 News Feed 中
 - Merge timeline into news feed asynchronously.
 - Unfollow 一个用户之后, 异步地将他发的 Tweets 从你的 News Feed 中移除
 - Pick out tweets from news feed asynchronously.
- 为什么需要异步 Async?
 - 因为这个过程一点都不快呀
- 异步的好处?
 - 用户迅速得到反馈, 似乎马上就 follow / unfollow 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？



拓展问题2: 如何存储 Likes?



拓展问题2: 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



请问此时数据分片机制
(Sharding) 是否管用？

拓展问题3:惊群现象 Thundering Herd

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。

解决办法及参考资料

Memcache Lease Get - 《Scaling Memcache at Facebook》<http://bit.ly/1jDzKZK>

Facebook 如何解决惊群效应的：<https://bit.ly/1Q3t3P7>

Redis 防雪崩架构设计 <https://bit.ly/2KFneb5>

- Cache是什么？
 - 你可以认为相当于算法中的HashMap
 - 是Key-value的结构
- Cache存在哪儿？
 - Cache存在内存里
- 常用的Cache工具/服务器有什么？
 - Memcached
 - Redis
- 为什么需要用Cache？
 - Cache因为是内存里，所以存取效率比DB要高
- 为什么不全放Cache里？
 - 内存中的数据断电就会丢失
 - Cache 比硬盘贵

- News Feed 和 Timeline 的定义和区别?
 - News Feed: 新鲜事, 我朋友+我发的所有帖子按照某种顺序排列的整合(比如按照时间排序)
 - 用户打开Twitter之后首先看到的界面就是News Feed界面, 这些 tweets 来自你关注的用户
 - Timeline: 某个用户发的所有帖子
 - 用户点开某个人的页面之后, 看到这个人发的所有帖子
 - 在有的系统中, 这两个概念的定义会完全反过来, 这里我们统一按照上面的定义。
- 什么是消息队列
 - 简单的说就是一个先进先出的任务队列列表
 - 做任务的worker进程共享同一个列表
 - Workers从列表中获得任务去做, 做完之后反馈给队列服务器
 - 队列服务器是做异步任务必须有的组成部分
- 哪些工具可以做消息队列
 - 最常用的有 RabbitMQ, ZeroMQ, Redis, Kafka

其他问题

NewsFeed Table 中 Pull Model 和 Push Model 的区别？

<http://www.jiuzhang.com/qa/2074/>

<http://www.jiuzhang.com/qa/2031/>

<http://www.jiuzhang.com/qa/1741/>

NewsFeed 如何实现 Pagination?

<http://www.jiuzhang.com/qa/1839/>

Twitter Pull 模型里用cache来存timeline时，关于保持实时性的问题

<http://www.jiuzhang.com/qa/962/>