

聊天系统 Chat System

课程版本: v6.0 主讲人: 东邪



扫描二维码关注微信/微博
获取最新面试题及权威解答

微信: [ninechapter](#)

微博: <http://www.weibo.com/ninechapter>

知乎: <http://zhuanglan.zhihu.com/jiuzhang>

官网: <http://www.jiuzhang.com>

- 设计微信
 - Work Solution
 - Real-time Service
 - Online Status: Pull vs Push
- 这节课之后您可以学会
 - 设计聊天系统的核心: Realtime Service
 - Pull 与 Push 的进一步比较分析
- 相关设计题
 - Design Facebook Messenger
 - Design WhatsApp
 - Design Facebook Live Comments



Telegram

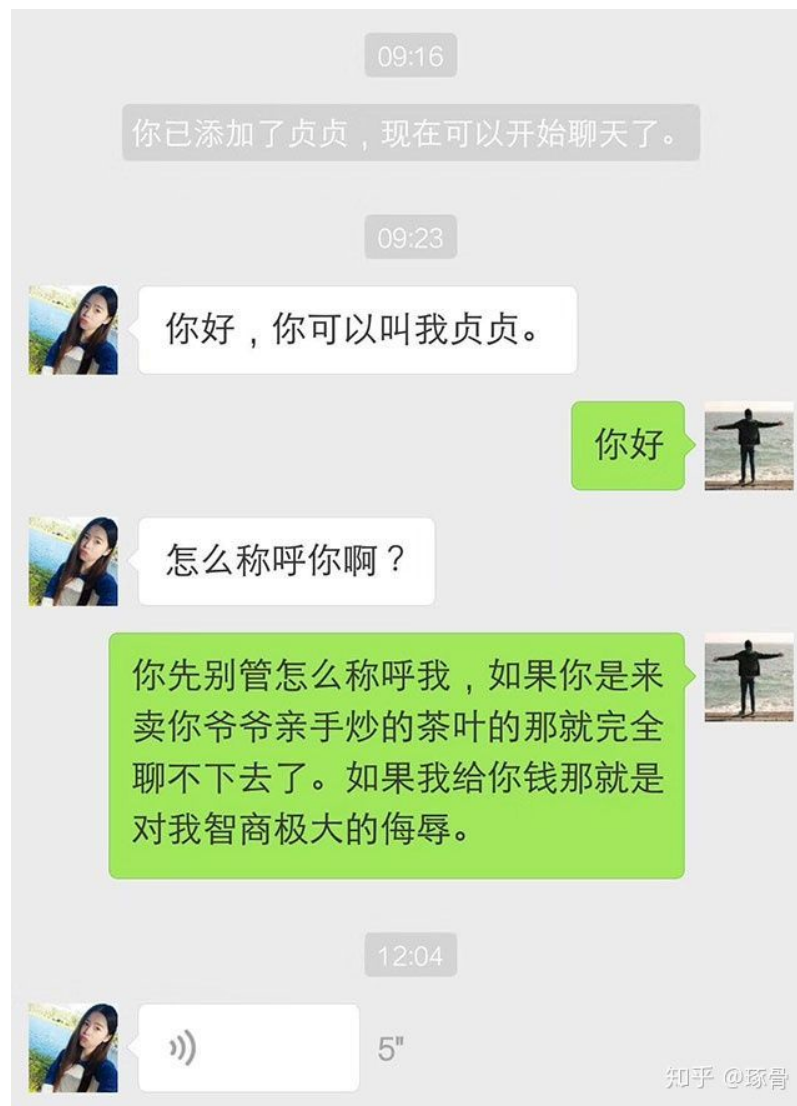


Interviewer: Design WeChat

设计微信



- 基本功能：
 - * 用户登录注册
 - * 通讯录
 - 两个用户互相发消息
 - 群聊
- 其他功能：
 - 限制多机登陆
 - 或者支持多机登陆
 - * 用户在线状态
 - WhatsApp, Messenger 等 APP 有此功能



Scenario - 设计多牛的系统？

- 微信
 - 10.8亿 月活跃用户
 - 日发送量 450 亿
 - ——数据来自 2019 年微信公开课PRO
- QPS:
 - Average QPS = $45B / 86400 \sim 520k$
 - Peak QPS = $520k * 3 \sim 1.5m$
- 存储:
 - 假设每条记录约30bytes的话, 大概需要 1.3T 的存储

微信的信息是否会经过服务器？

即，微信是否是点对点通信？



Telegram



Service 服务

Message Service 负责信息相关的存取

Realtime Service 负责信息的实时推送

Storage 存储

既然是聊天软件，自然需要一个 Message Table 了
我们需要在 Message Table 里存什么？

Message Table

下面这张表里，缺少了什么？

Message Table		
id	int	
from_user_id	int	谁发的
to_user_id	int	发给了谁
content	text	发了啥
created_at	timestamp	啥时候发的

如果按照上面的 Message Table, 那么要查询 A 与 B 之间的对话, 则需要如下的语句:

```
SELECT * FROM message_table  
WHERE from_user_id=A and to_user_id=B OR from_user_id=B and to_user_id=A  
ORDER BY created_at DESC;
```

问题1: WHERE 语句太复杂, 导致SQL效率低下

问题2: 如果是多人聊天群, 这种结构不可扩展

怎么办?

增加 Thread Table

Thread 可以翻译为“会话”

Thread vs Message

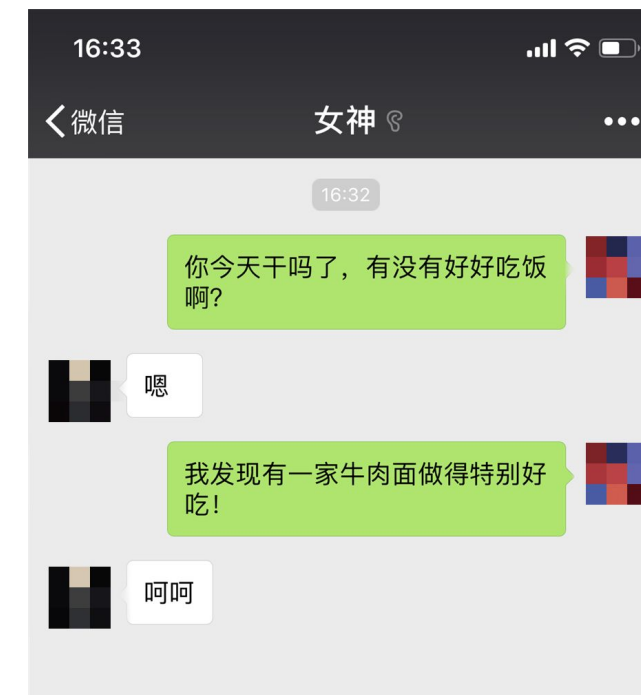
Inbox has a list of “Threads”

Thread has a list of “Messages”

Thread Table 里应该存一些什么？



A list of “Threads”



A list of “Messages”

Thread Table		
id	int	
participant_user_ids	text	比如 [1,2], 表示是1和2之间的对话
created_at	timestamp	
updated_at	timestamp	index=true

问题1: 如何取某个 Thread 下的所有 Message?

问题2: 这样设计的 Thread Table 有什么问题?

Message Table		
id	int	
thread_id	int	
user_id	int	谁发的
content	text	发了啥
created_at	timestamp	啥时候发的

如何取 Message

SELECT * FROM message_table

WHERE thread_id=12345 —— 筛选属于某个 Thread 的信息

ORDER BY created_at DESC —— 按照时间倒序排列

LIMIT 20 —— 取最近20条

有一些Thread信息是私有的

is_muted(是否被静音)
unread_count(未读信息数)

方法1: 拆成多张表

Thread - 存储基本信息

UserThread - 存储 User 在 Thread 上的私有信息

方法1: 拆成多张表

Thread Table		
id	primary key	bigint
last_message	text	
avatar	varchar	
created_at	timestamp	

User Thread Table		
id	primary key	bigint
user_id	foreign key	
thread_id	foreign key	
unread_count	int	
is_muted	boolean	
updated_at	timestamp	什么时候更新
joined_at	timestamp	什么时候加入对话

问题1: UserThread 的 Primary Key 还可以用什么?

问题2: 这种存储方法有什么弊端?

方法2: 合成一张表

只使用 UserThread

公有信息复制一份到每个人的 UserThread 里

User Thread Table		
user_id	foreign key	谁的 thread 信息
thread_id	varchar	可以是一个 uuid
participant_user_ids	text	如 “[1,2]”
is_muted	boolean	
unread_count	int	
last_message	text	
avatar	varchar	
created_at	timestamp	
updated_at	timestamp	

这样设计又有什么坏处？
(我们后面的内容将基于前面的方法1进行)

面试官问：如何查询 Thread id？

当用户 A 给用户 B 发消息的时候，可能并不知道他们之间的 thread_id 是什么，如何在服务器上查询？

在 Thread Table 中增加一个 participants_hash_code

该值由所有参与者的user_id排序之后hash得到, 即:

```
participants_hash_code = any_hashfunc(sorted(participants_user_ids))
```

不直接使用排序之后的 user_ids 是因为如果是群聊的话, 会太长

如果只需要考虑两人对话的话, 可以自定义一个格式: private::user1::user2

采用uuid之类的hash方式则不需要考虑hash collision的问题

- Message Table (NoSQL)
 - 数据量很大, 不需要修改, 一条聊天信息就像一条log一样
 - 问: sharding key (row key) 是什么?



Storage 存储 - 选择存储结构 Message Table

- Message Table (NoSQL)
 - 数据量很大, 不需要修改, 一条聊天信息就像一条log一样
 - 问: sharding key (row key) 是什么?
- 存储结构:
 - row_key = thread_id
 - column_key = created_at 因为要按照时间倒序
 - value = 其他信息

随便 你忙 多喝热水



男友三连



- Thread Table (SQL / NoSQL, 基于方法1)
 - Thread Table 存储公有的 Thread 信息
 - 如果使用 SQL 需要同时 index by
 - thread_id 用于查询某个对话的信息
 - participant_hash_code 用户查询某些用户之间是否已经有 thread
 - 问: 如果使用 NoSQL 该如何存储?



Storage 存储 - 选择存储结构 Thread Table

- 如果使用 NoSQL 存储 Thread Table 并同时支持按照 thread_id 和 participant_hash_code 进行查询, 我们需要两张表:
 - 表1: Thread Table
 - row_key = thread_id
 - column_key = null
 - value = 其他的基本信息
 - 表2: ParticipantHashCode Table
 - row_key = participant_hash_code
 - column_key = null
 - value = thread_id
- 因为这里用不到 range query, 也就用不到 column key, 因此也可以选择如 RocksDB 这样的纯 key-value 的 NoSQL。



- UserThread Table (NoSQL, 基于方法1)
 - UserThread Table 存储私有的 Thread 信息
 - 用什么做 sharding key(row_key) ?



- UserThread Table (NoSQL, 基于方法1)
 - UserThread Table 存储私有的 Thread 信息
 - 用什么做 sharding key(row_key) ?
- 存储结构:
 - row_key = user_id
 - column_key = updated_at 按照更新时间倒序
 - value = 其他信息

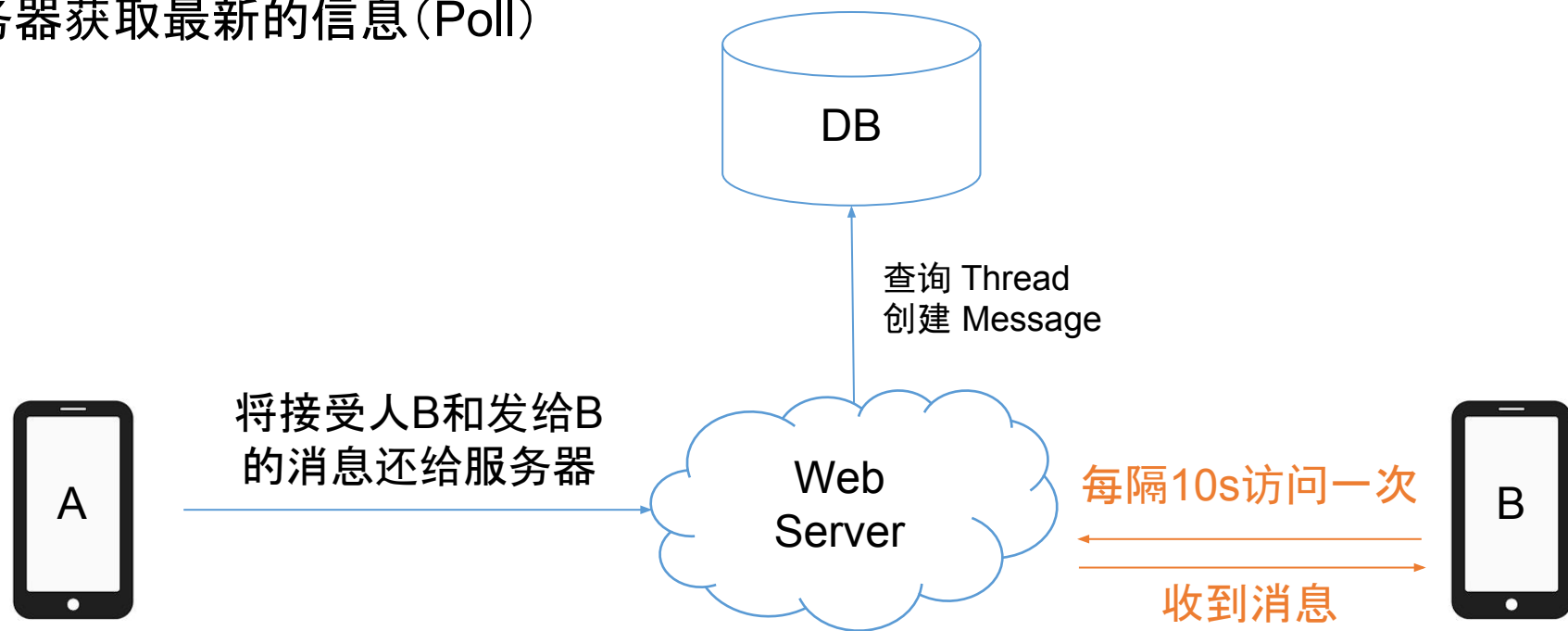


数据拆分的基本原则

按什么取(Query), 就按什么拆(Sharding)

一个可行解的流程

- 用户 A 发送一条消息给 用户 B
- 服务器收到消息, 查询是否有 A 和 B 的对话记录(Thread), 如果没有则创建对应的 Thread
- 根据 Thread id 创建 Message
- B 每隔 10s 访问一次服务器获取最新的信息(Poll)
- B 收到信息



Pull vs Poll

Pull: Client主动问 server 读/写 数据的行为

Poll: Client 每隔一段时间就问 server 读/写 数据的行为

Scale 拓展

有没有更好的信息更新方式？

聊天系统的 Scale 更多的会问你一些小功能的设计

Interviewer: How to speed up?

每隔10秒钟收一次消息太慢了，聊天体验很差，不实时

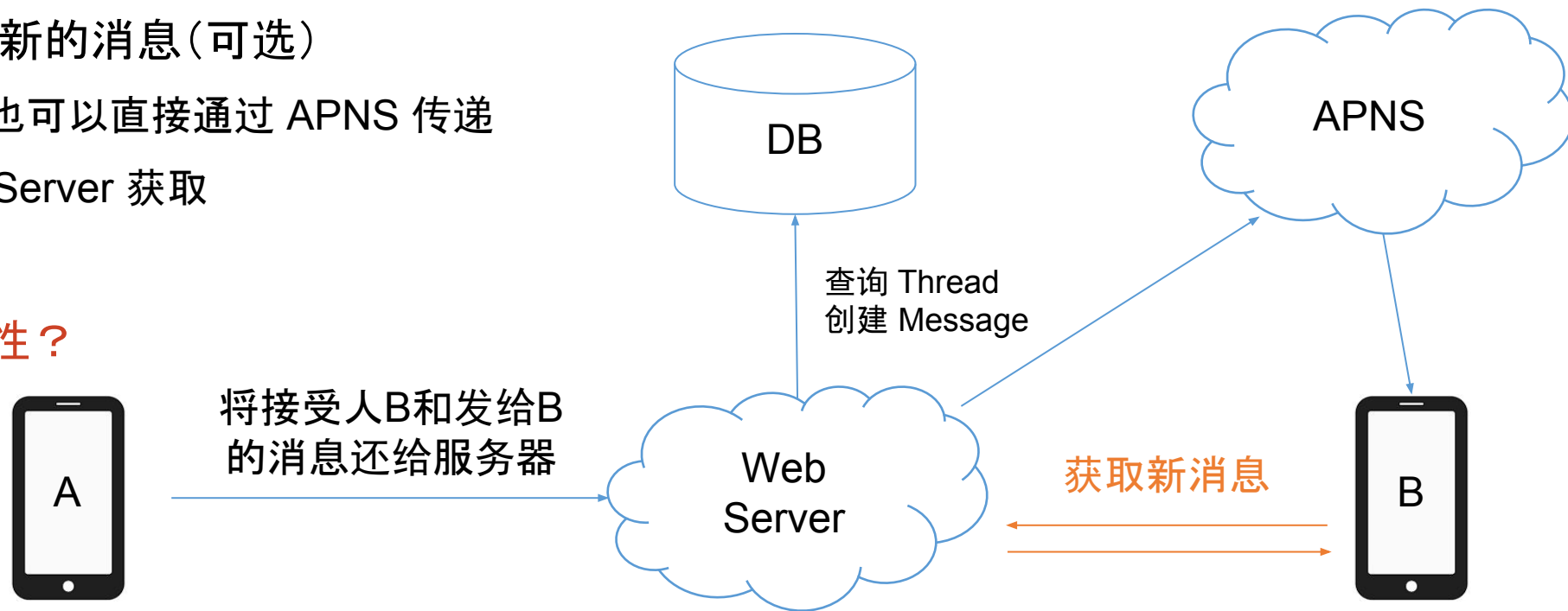
Push Notification

Android GCM (Google Cloud Messaging)

iOS APNS (Apple Push Notification Service)

手机自己的消息推送系统

- A 发送消息到 Web Server
- Web Server 创建信息存入数据库之后通知 APNS
- APNS 告诉 B 有新消息了
- B 去 Web Server 抓取下新的消息(可选)
 - 如果消息比较短的话, 也可以直接通过 APNS 传递
 - 无需 B 再次访问 Web Server 获取
- 问:这个方法有什么局限性?



无法支持 Web 端

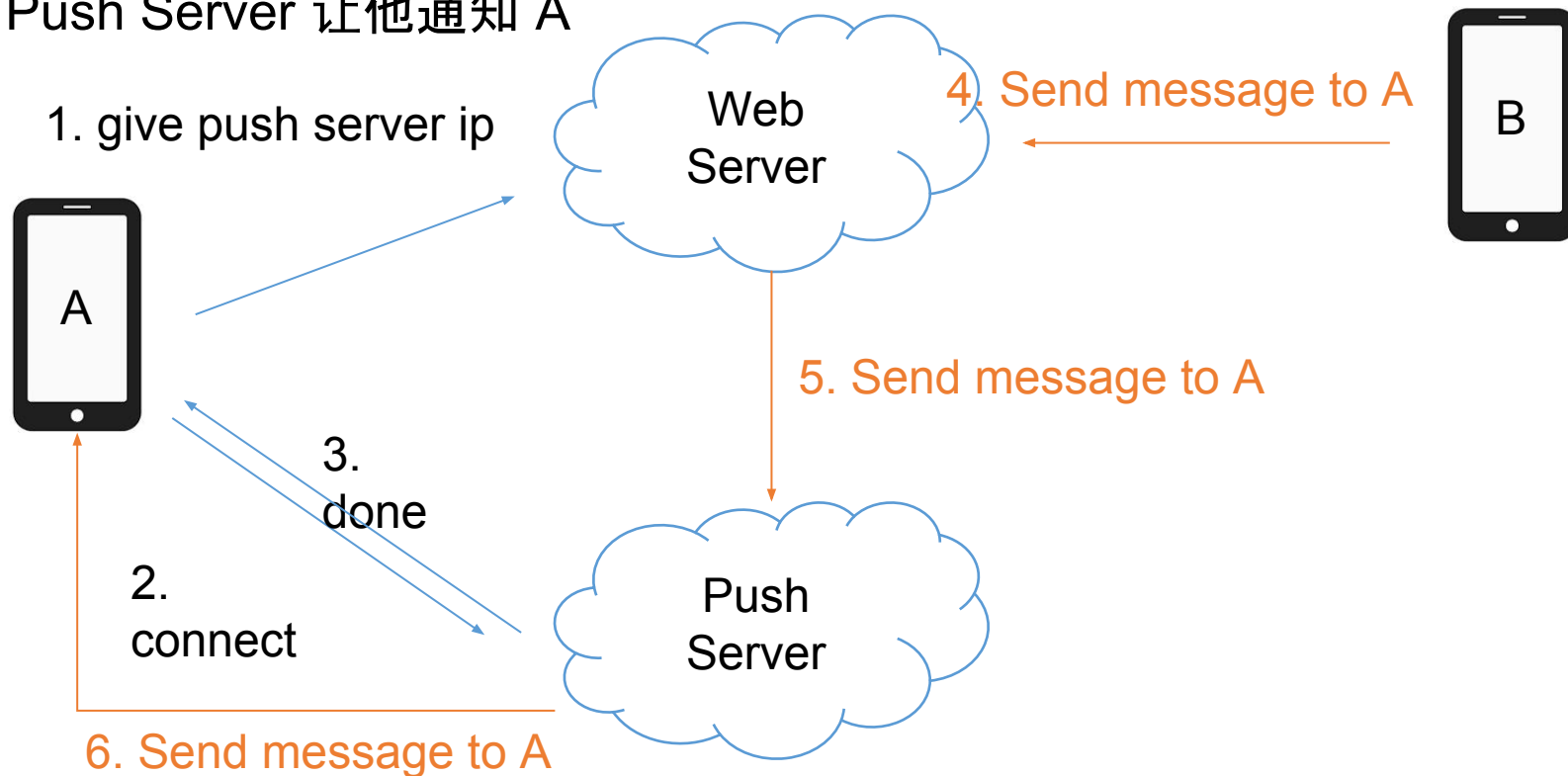
GCM / APNS 都是依赖于手机操作系统的
无法支持 Web 端或者桌面端的信息推送
如 Web 微信, Facebook Messenger

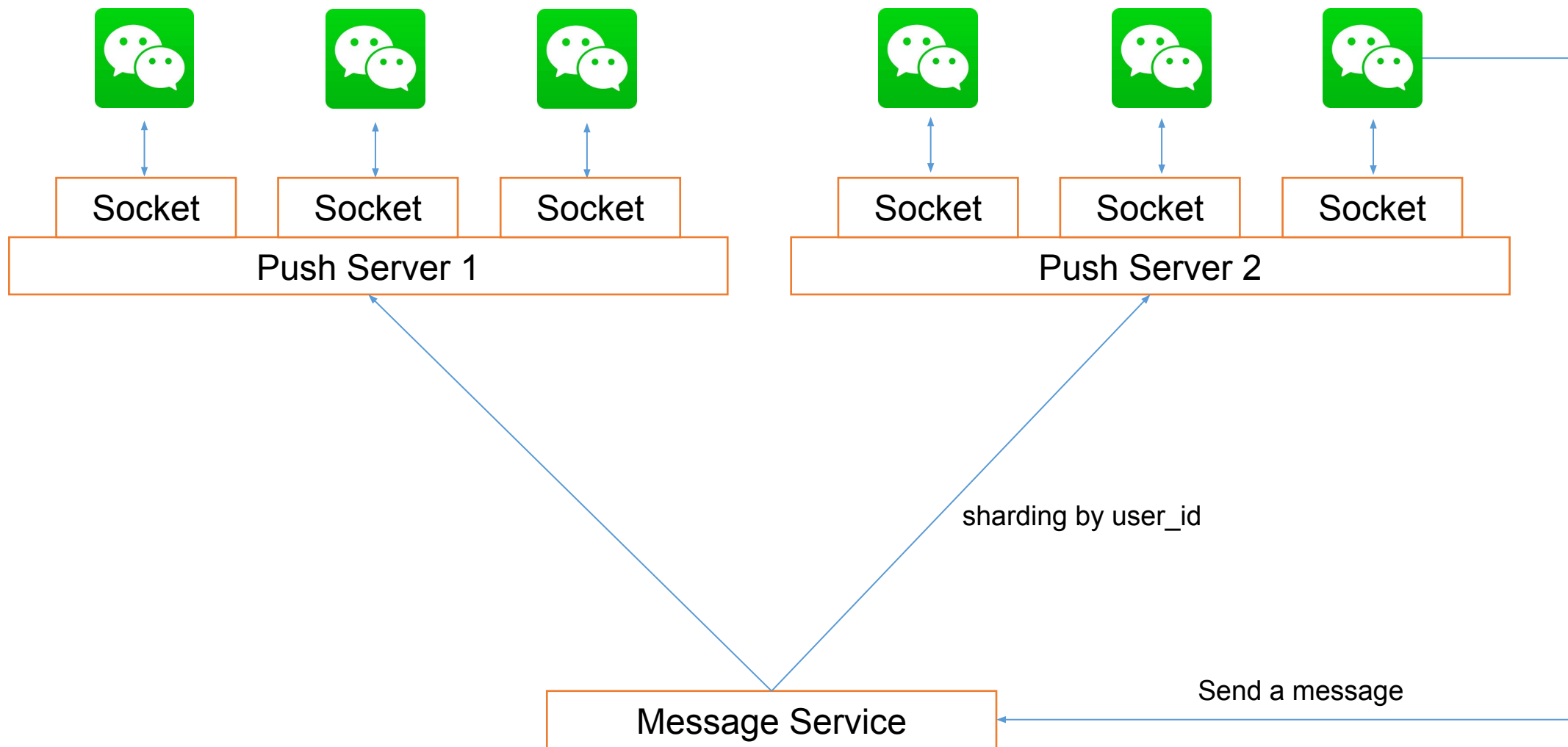
Socket

让服务器可以主动向客户端推送数据的技术

HTTP 只支持客户端向服务器获取数据

- 用户A打开App后, 问 Web Server 要一个 Push Service 的连接地址
- A通过 socket 与push server保持连接
- 用户B发消息给A, 消息先被发送到服务器
- 服务器把消息存储之后, 告诉 Push Server 让他通知 A
- A 收到及时的消息提醒





Q: WebSocket 和 Socket 是什么联系和区别？

A: Socket 是很早就有的技术，Web Socket 是在 H5 之后才诞生的技术，专门用于让浏览器支持被服务器推送信息所用。Socket 是更通用和强大的可以在任何地方使用的。WebSocket 只在浏览器上使用。

Q: 如果 Push Server 宕机了怎么办？用户还收得到信息么？

A: 因为 Socket 是一个双向连接，如果 Push Server 宕机了，Client 端是知道链接已经断开了的，因此 Client 端上只需要有一个 backup 的逻辑，让 client fallback 到每隔 10s 拉一次数据的 poll 机制就可以了。

面试官：如何支持群聊？

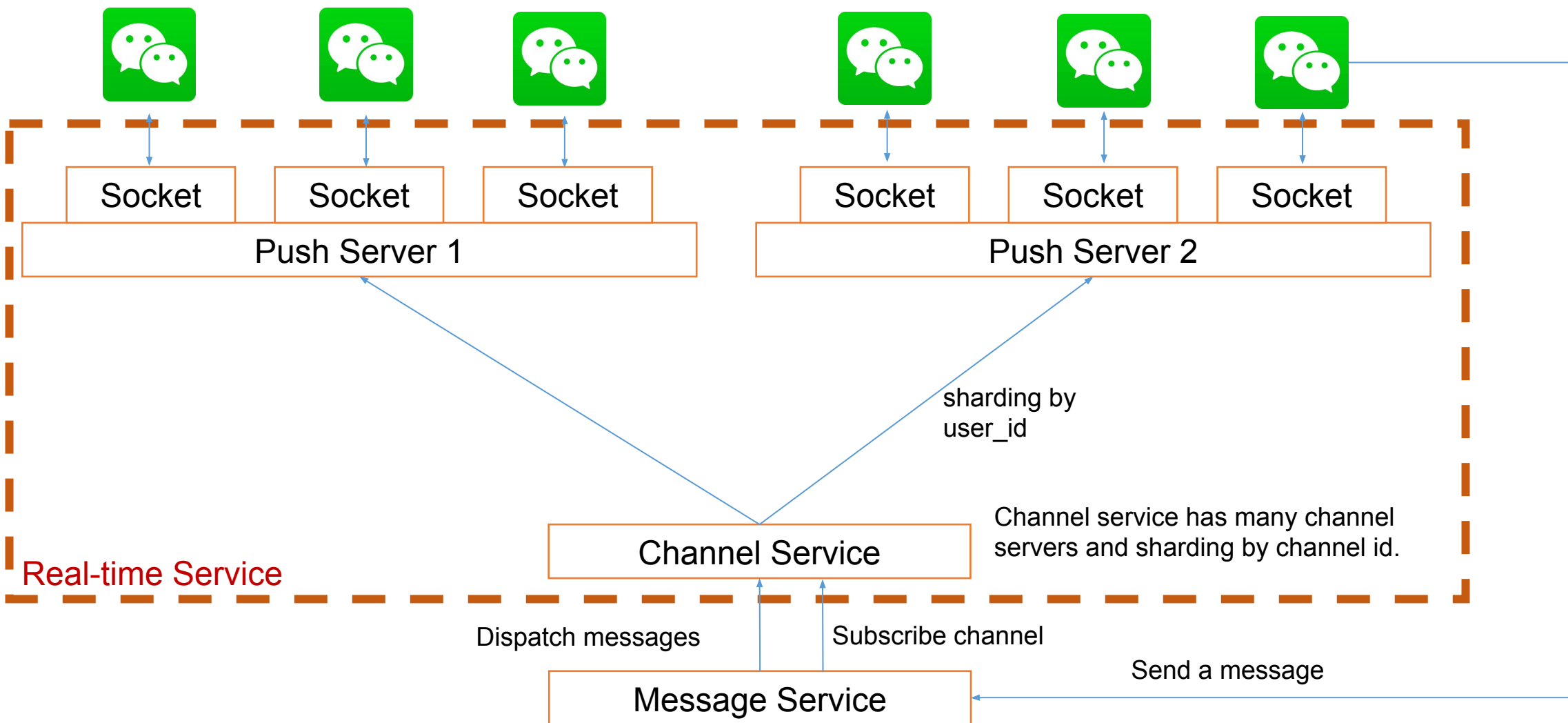
Group Chat



- 假如一个群有500人(1m用户也同样道理)
- 如果不做任何优化, 需要给这 500 人一个个发消息
- 但实际上 500 人里只有很少的一些人在线(比如10人)
- 但Message Service仍然会尝试给他们发消息
 - Message Service (web server) 无法知道用户和Push Server的socket连接是否已经断开
 - 至于 Push Server 自己才知道
- 消息到了Push Server 才发现490个人根本没连上
- Message Service 与 Push Server 之间白浪费490次消息传递



- 解决
 - 增加一个Channel Service(频道服务)
 - 为每个聊天的Thread增加一个Channel信息
 - 对于较大群, 在线用户先需要订阅到对应的 Channel 上
 - 用户上线时, Web Server (message service) 找到用户所属的频道(群), 并通知 Channel Service 完成订阅
 - Channel就知道哪些频道里有哪些用户还活着
 - 用户如果断线了, Push Service 会知道用户掉线了, 通知 Channel Service 从所属的频道里移除
 - Message Service 收到用户发的信息之后
 - 找到对应的channel
 - 把发消息的请求发送给 Channel Service
 - 原来发500条消息变成发1条消息
 - Channel Service 找到当前在线的用户
 - 然后发给 Push Service 把消息 Push 出去



Q: Channel Service 中的数据是什么结构？

A: key-value 的结构。key 为 channel name, 可以是一个字符串比如 “#personal::user_1”。value 是一个 set 代表哪些人订阅到了这个 channel 下。

Q: Channel Service 用什么数据存储？

A: 根据上面所提到的 key-value 结构以及 value 需要是一个 set, Redis 是一个很好的选择。

Q: 如何知道一个用户该订阅到哪些 Channels？

A: 首先用户需要订阅自己的 personal channel, 如 #personal::user_1, 与该用户有关的私聊信息都在这个 channel 里发送。小于一定人数的群聊可以依然通过 personal channel 推送, 超过一定人数的群聊, 可以采用 lazy subscribe 的方式, 在用户打开 APP 且群处于比较靠前的位置的时候才订阅, 用户没有主动订阅的群聊靠 Poll 的模式获取最新消息。

Q: 用户关闭 APP 以后还能收到提醒么？

A: 如果真的关闭了 APP 是不行的。所以很多 APP 会常驻后台, 保证至少 Poll 模式还能工作即可。

聊天系统 Chat System 的基本考点如下：

- 能够设计出 Message, Thread, UserThread 这几个最主要的数据库表单
- 能够使用 NoSQL 存储以上数据表单
- 能够设计基于 Socket 技术的 Realtime Service (或者叫 Push Service)
- 能够针对群聊通过订阅 Channel 的模式优化群聊的 Push

大部分的考点依然最基本的数据库相关知识！

拓展问题1：多机登录

如何限制多台客户端同时登录一个账户？

(默认是允许多机登陆的, 因为可以同时存在多个 session)

拓展问题1: 多机登录

考虑我们的使用场景:

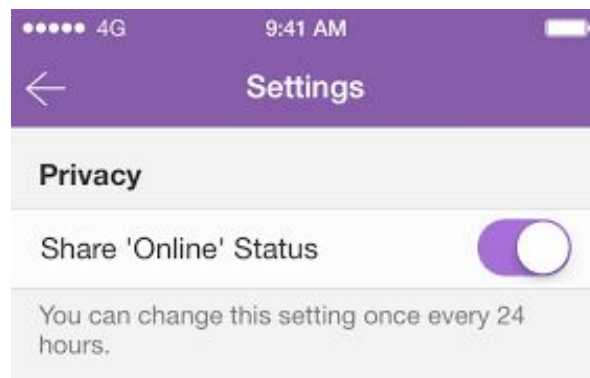
1. 不允许两个手机同时登录微信
2. 允许手机和桌面客户端或Web微信同时登录

解决办法: 在 session 中记录用户的客户端信息

用户尝试从新的客户端登录时

- 如从手机登陆时, 查询是否已经有其他手机处于登陆状态
 - 如果没有, 则创建新的 session
 - 如果有, 将对应的 session 设为 expire 或者删除, 并发送 push notification 让已经登录的手机 logout
 - 如果 Push Notification 失败也没有关系
 - 该手机会在下次访问任何API的时候发现自己已经logout了并跳转至登入界面

拓展问题2：如何支持用户在线状态显示？



是否可以用 Push Server 中的 Socket 的连接情况来代表用户是否在线？

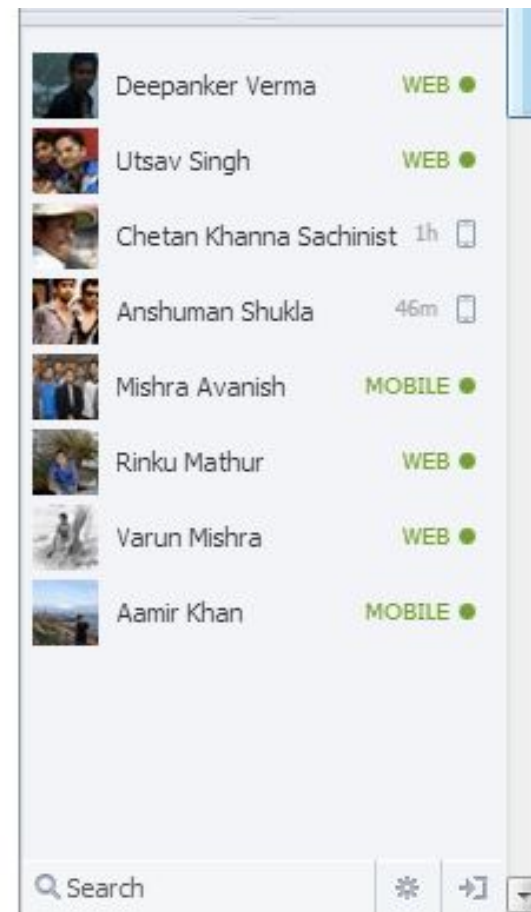
是否可以用 Push Server 中的 socket 连接情况？

缺陷1: 如果用户的网络不稳定, 会导致连接时断时连

缺陷2: 如果在 Push Service 中使用数据库来存储在线信息, Push Service 的结构会变得复杂, 通用性会变差, 依赖会增多。

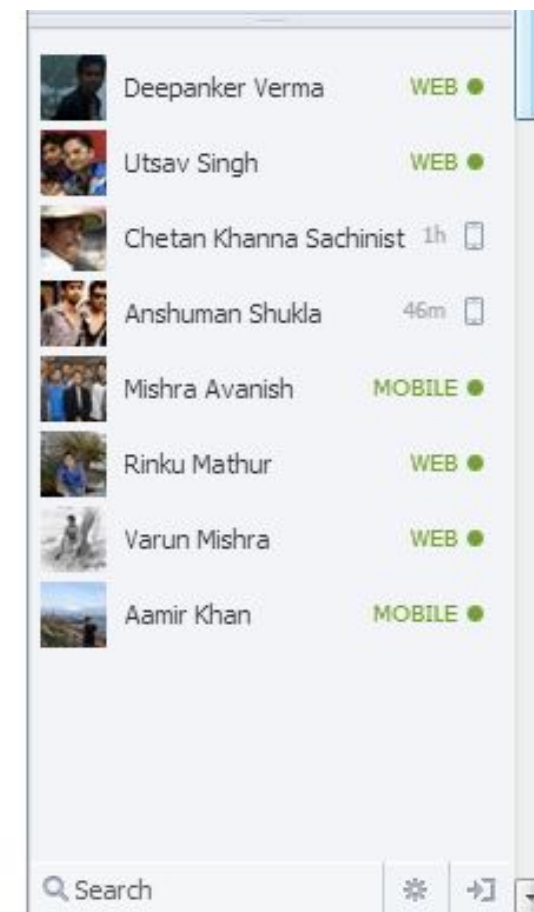
拓展问题2: Online Status

- 使用数据库存储 online status 的信息
- 使用 Web Server 直接访问数据库的获取该信息
- 问: OnlineStatus Table 中存储什么信息?



拓展问题2: Online Status

- 使用数据库存储 online status 的信息
- 使用 Web Server 直接访问数据库的获取该信息
- 问: OnlineStatus Table 中存储什么信息?
- 存如下一些信息足够: $\langle \text{user}, \text{last_updated_at}, \text{client_info} \rangle$
 - 类似于在打车软件设计中, 我们提到的司机在线状态的更新



是 Pull 还是 Push?




是用户主动告诉服务器我在线，还是服务器询问用户是否在线？

是 Pull, 每隔3-5s pull 一次(heartbeat)

原因:

1. Pull 更简单, 依赖更少(不依赖于 Push Service), 代码量更少
2. 在告诉服务器我在线的时候, 还可以顺带更新所有好友的在线状况, 用于客户端显示
 - a. 更新好友在线状态如果用 Push 的方式来做存在很多问题, 比如用户如果掉线了, 还需要由 Push Server 通知 Web Server 来更新在线状态, 然后再通过 Web Server 通知所有的好友他掉线了。

总而言之就是 Pull 更简单, Push 更复杂, **对实时性要求不高**的时候, 用 Pull 更好。

	pull?channel=p_1312800249&seq... 3-edge-chat.facebook.com	200
	pull?channel=p_1312800249&seq... 3-edge-chat.facebook.com	200
	pull?channel=p_1312800249&seq... 3-edge-chat.facebook.com	200

可以打开 Facebook 验证一下
Facebook Messenger 每隔 3-5s 会发送
一次 pull 请求