OneStepOffer系统设计
第四讲

聊天系统与访问限制系统

聊天系统的使用场景

设计功能

- 基本功能
  - 登录注册
  - 通讯录
  - 两个用户互相发消息
  - 群聊
  - 用户在线状态
- 其它功能
  - 历史消息
  - 多机登录 multi devices

聊天系统的使用场景

系统限制

- 活跃
  - 1B 月活跃用户
  - 75% 日活跃 / 月活跃
  - 约750M日活跃用户

为了计算方便起见，我们来设计一个100M日活跃的WhatsApp

- QPS：
  - 假设平均一个用户每天发20条信息
  - Average QPS = 100M * 20 / 86400 ~ 20k
  - Peak QPS = 20k * 5 = 100k
- 存储：
  - 假设平均一个用户每天发10条信息
  - 一天需要发 1B，每条记录约30bytes的话，大概需要30G的存储

Service 服务

- Message Service ： 负责管理信息
- Real-time Service ： 负责实时推送信息给接受者

stortage存储方法

1. Message Table

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

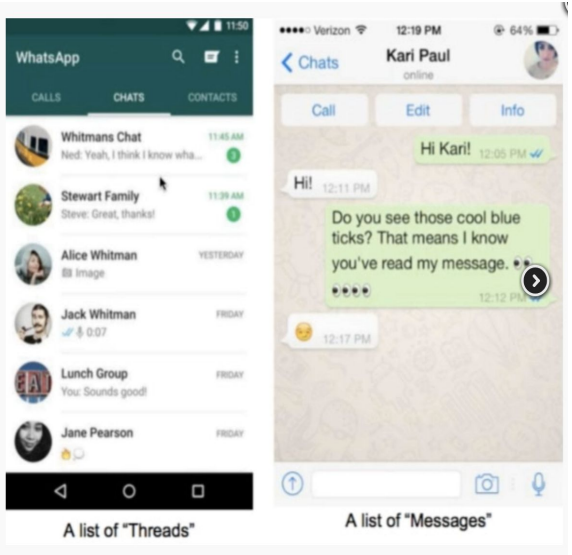
id	int	PK
from_user_id	int	发消息人
to_user_id	int	收消息人
content	text	消息内容
createtime	timestamp	发送/产生时间

只有mesaaage table的缺陷

- 效率太低。
  a. 如果单纯用sql获取内容，效率太低 -- 可以用kafka 等 message produder方式解决
- 无法支持群聊功能
  - 群聊需要加入新的表做join

改进 -- 引入thread

hread表示两个的对话如下图所示，左边是thread。右边是message。



A list of "Threads"

A list of "Messages"

Thread Table设计

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

Thread Table		
owner_id	int	谁的Thread
thread_id	int	
participant_ids	text	
is_muted	bool	
nickname	string	
created_at	timestamp	
updated_at	timestamp	index=true

流程梳理

- messagetable，存储每个人发的每条消息，以及对应的thread_id。即[id, thread_id,user_id,content, created_id]
- thread table， 存储每个thread的信息，每个人独享一份数据。即[owner_id,thread_id, participant_ids, is_muted, nickname, created_at, updated_at]
- 查询我的所有thread list——从thead_table里查询给定user_id的所有thread_id即可。因此thread_table用SQL
- 查询我的某个thread的大概信息——从thread_table里查询给定owner_id+thread_id的信息
- 查询我的某个thread的具体消息——从message_table中，根据thread_id查询所有的消息，并且返回.因此message_table用NoSQL
- 查询我与另一个用户是否有一个thread?

流程梳理

- 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
- Thread Table (SQL) —— 对话表
  - 需要同时 index by
    - Owner ID + Thread ID (primary key)
    - Owner ID + Updated time (按照更新时间倒叙排列)
  - NoSQL 对 secondary index 的支持并不是很好

Word Solution

用户如何发送消息？

- Client 把消息和接受者信息发送给 server
- Server为每个接受者（包括发送者自己）创建一条 Thread （如果没有的话）
- 创建一条message（with thread_id）

用户如何接受消息？

- 可以每隔10秒钟问服务器要一下最新的 inbox
- 如果有新消息就提示用户

Scale 扩展

sharding

- Message 是 NoSQL，自带 Scale 属性
- Thread 按照 user_id 进行 sharding

每隔十秒要inbox?太慢了，优化——Socket

Socket——TCP长连接

Push Service——提高Socket连接服务，可以与Client保持TCP长连接

- 当用户打开APP之后，就连接上Push Service中属于自己的socket
- 有人发消息时，Message Service收到消息，通过Push Service把消息发出去
- 如果一个用户长期不活跃（比如10分钟），就可以断开连接，释放掉网络断开
- 断开连接之后，如何收到消息？
  - 打开APP时主动pull + android GCM/IOS APNS
- Socket连接与HTTP连接的最主要区别；
  - HTTP连接下，只能客户端问服务器要数据
  - Socket连接下，服务器可以主动推送数据给客户端

Scale TCP长链接容灾和心跳保活

长连接断开的原因：

进程被杀死

NAT超时

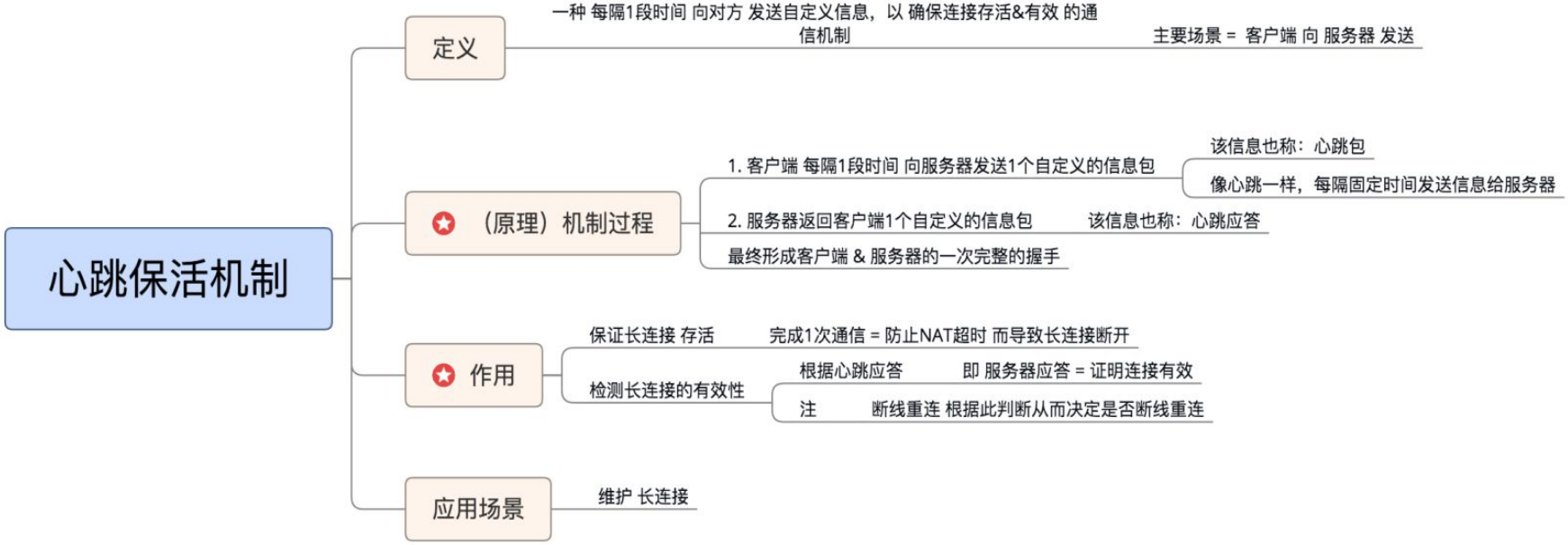
网络状态发生变化

其他不可抗因素（网络状态差、DHCP的租期等等 ）

维持长链接的方法:进程保活

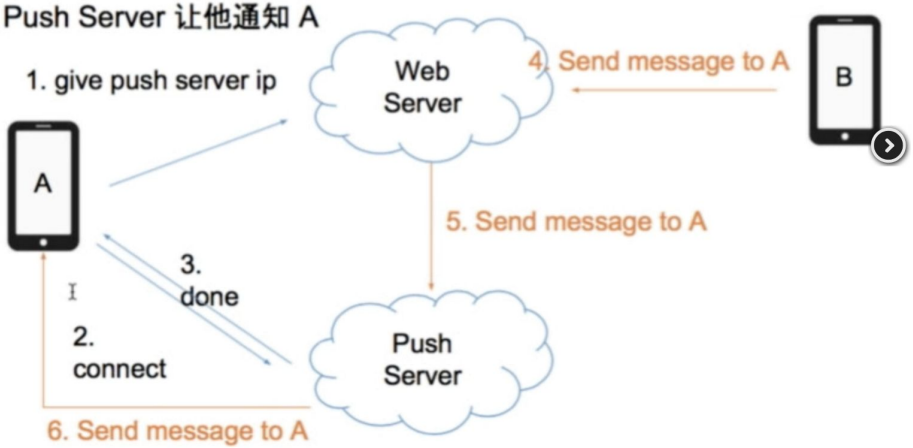
方法		具体描述	备注
提高进程优先级，降低进程被杀死概率	利用 Notification 提升权限	• 通过 setForeground 接口将后台 Service 设置为前台 Service，使进程的优先级由4提升为2 （原先 Android 中 Service 的优先级为4 ） • 从而使进程的优先级仅仅低于用户当前正在交互的进程，与可见进程优先级一致，使进程被杀死的概率大大降低	应用存在后台消息提醒功能 （因该方案必须在系统的通知栏发送一条通知，即 前台 Service 绑定 一条可见的通知，即用户是感知的）
	利用 Activity 提升权限	• 监控手机锁屏解锁事件，在屏幕锁屏时启动1个像素的 Activity • 在用户解锁时将 Activity 销毁掉。注该 Activity 需设计成用户无感知 • 通过该方案，可使进程的优先级在屏幕锁屏时间由4提升为最高优先级1	主要解决 第三方应用 & 系统管理工具在检测到锁屏事件后一段时间（一般为5分钟以内）内杀死后台进程、从而达到省电目的 的问题
进程被杀死后，进行拉活、重建	通过监听系统广播 拉活	• 在发生特定系统事件时，系统会发出响应的广播 • 通过在进程中"静态"注册对应的广播监听器，即可在发生响应事件时拉活	常用于拉活的广播：锁屏、拍照、网络切换频等
	通过监听第三方广播 拉活	• 类似接收 系统广播，但该方案接收的是：第三方应用 or SDK 广播 • 通过反编译第三方 Top 应用，如：手机QQ、微信等 & 对应应用的SDK（如微信）、推送SDK 友盟、个推等，找出它们外发的广播，在应用中进行监听 & 拉活	目前腾讯系、阿里系、百度系的全家桶主要采用该方式 （因为存在拳头产品：微信、支付宝、手机百度）
	通过 系统Service机制	• 在onStartCommand （）返回START_STICKY参数 • 当返回 START_STICKY时：在杀死服务后，则重建服务 & 调用onStartCommand()，但不会再次送入上一个intent， 而是用null intent来调用onStartCommand()，利用系统机制在 Service 挂掉后自动拉活	• onStartCommand （）必须返回一个整数 = 描述系统在杀死服务后应该如何继续运行 • 在进行该参数设置后，当Service因内存不足被kill后；当内存又有时，Service又被重新创建
	通过 JobScheduler 机制拉活	JobScheduler 接口作用：定时调用该进程 使得应用进行一些逻辑操作	Android5.0 以上版本提供

维持长链接方法:心跳保活



Scale 扩展流程

- 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



微信的系统设计

微信的系统设计和whatsapp类似 分为以下几步

图 1 微信消息模型

微信服务的消息队列

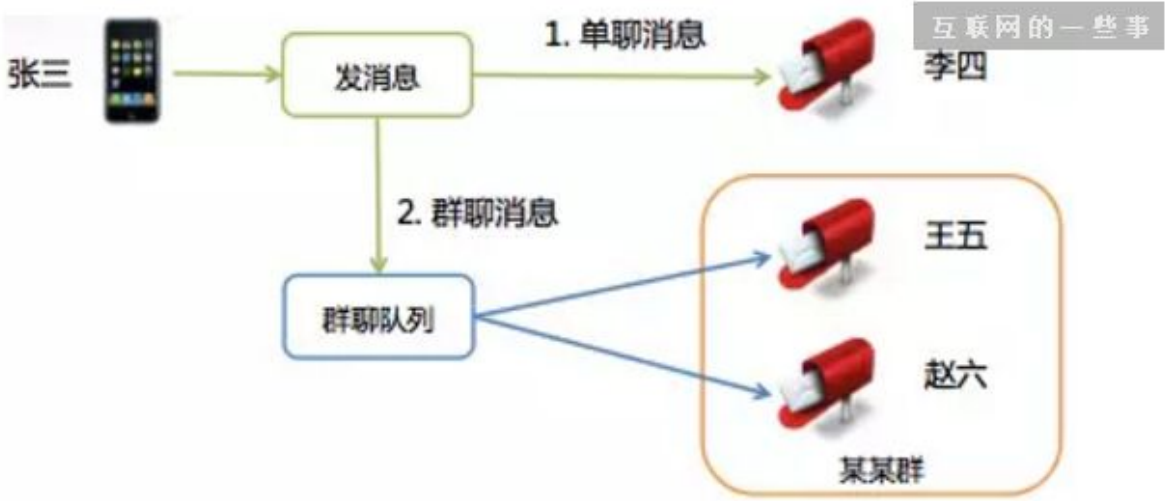


图 4 单聊和群聊消息发送过程