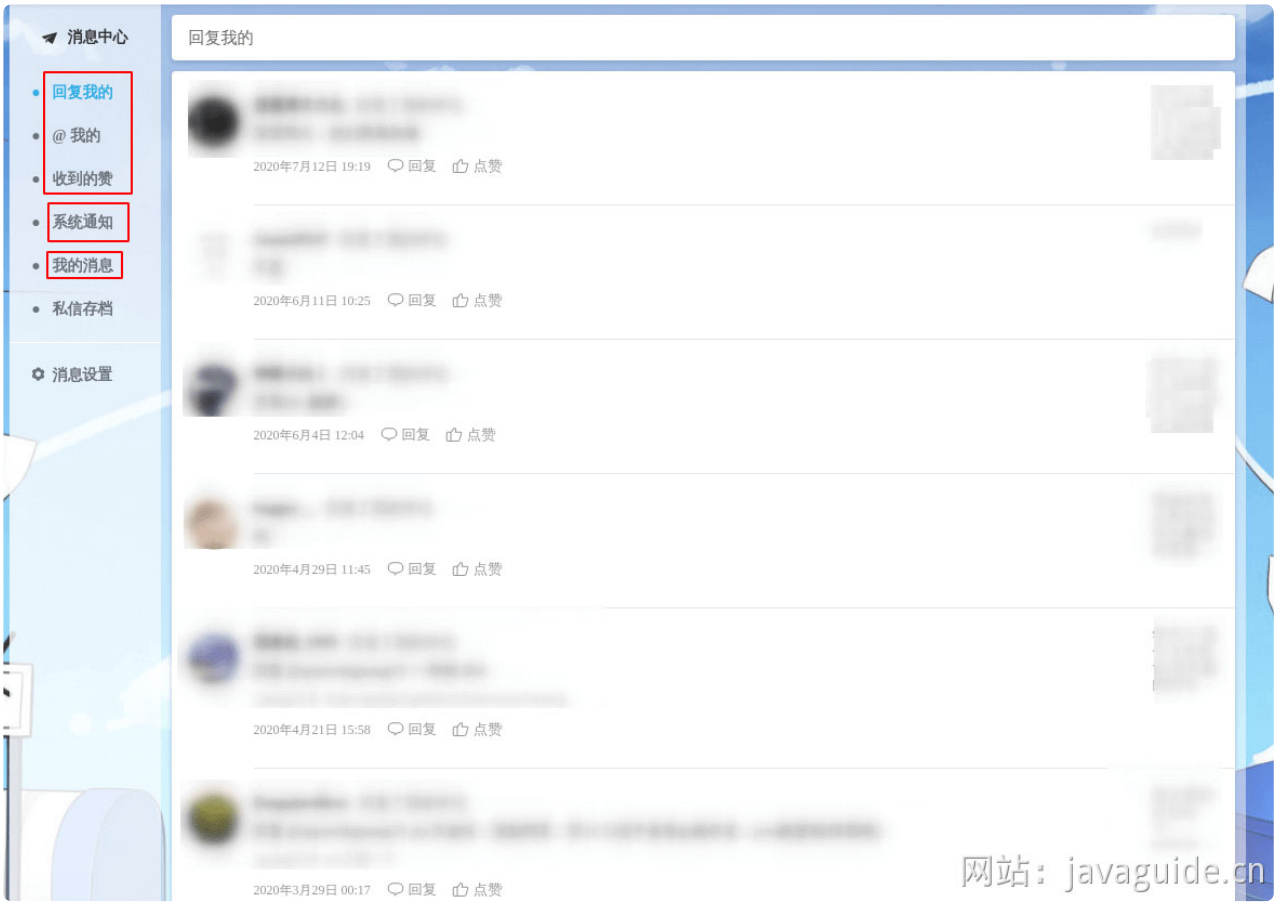
如何设计一个站内消息系统?

> 这篇文章是一位朋友投稿给我的，我简单完善了一下。

各位使用过简书，知乎或 B 站的小伙伴应该都有这样的使用体验：当有其他用户关注我们或者私信我们的行为时，我们会收到相关的消息。

虽然这些功能看上去简单，但其背后的设计是非常复杂的，几乎是一个完成的系统，可以称之为 **站内消息系统**。

我以 B 站举例（个人认为 B 站的消息系统是我见过的非常完美的，UI 也最为人性化的）：



可以看到 B 站把消息大致分为了三类：

1. 系统推送的通知(System Notice)；
2. 回复、@、点赞等用户行为产生的提醒(Remind)；
3. 用户之间的私信(Chat)。

这样设计不仅分类明确，且处于同一个主体的事件提醒还会做一个聚合，极大的提高了用户体验，不让用户收到太多分散的消息。

举个例子：比如你在某个视频或某篇文章下发表了评论，有 100 个人给你的评论点了赞，那么你希望消息页面呈现的是一个一个用户给你点赞的提醒，还是像以下聚合之后的提醒：

我相信你大概率会选择后者。

我认为对于很多应用来说，这样的设计都是非常合理的，接下来我写写我对于消息系统的设计。

系统通知(System Notice)

系统通知一般是由后台管理员发出，然后指定某一类（全体，个人等）用户接收。基于此设想，可以把系统通知大致分为两张表：

1. **t_manager_system_notice（管理员系统通知表）** ：记录管理员发出的通知；
2. **t_user_system_notice（用户系统通知表）** ： 存储用户接受的通知。

t_manager_system_notice（管理员系统通知表） 表结构如下：

字段名	类型	描述
system_notice_id	LONG	系统通知
title	VARCHAR	标题
content	TEXT	内容

type	VARCHAR	发给哪些用户：单用体用户 all，vip 用用位小伙伴可以根据自
state	BOOLEAN	是否已被拉取过，过，就无需再
recipient_id	LONG	接受通知的用户的为单用户，那么 re户的 ID;否则 re
manager_id	LONG	发布通知的管
publish_time	TIMESTAMP	发布时

t_user_system_notice（用户系统通知表）结构如下：

字段名	类型	描述
user_notice_id	LONG	主键 I
state	BOOLEAN	是否已
system_notice_id	LONG	系统通知
recipient_id	LONG	接受通知的用
pull_time	TIMESTAMP	拉取通知的

当管理员发布一条通知后，将通知插入 t_manager_system_notice 表中，然后系统定时的从 t_manager_system_notice 表中拉取通知，然后根据通知的 type 将通知插入 t_user_system_notice 表中。

如果通知的 type 是 single 的，那就只需要插入一条记录到 t_user_system_notice 中。如果是全体用户，那么就需要将一个通知批量根据不同的用户 ID 插入到 t_user_system_notice 中，这个数据量就需要根据平台的用户量来计算。

🧅 举个例子：管理员 A 发布了一个活动的通知，他需要将这个通知发布给全体用户，当拉取时间到来时，系统会将这一条通知取出。随后系统到用户表中查询选取所有用户的 ID，然后将这一条通知的信息根据所有用户的 ID，批量插入 t_user_system_notice 中。用户需要查看系统通知时，从 t_user_system_notice 表中查询就行了。

👉 需要注意的是：

1. 因为一次拉取的数据量可能很大，所以两次拉取的时间间隔可以设置的长一些。
2. 拉取 t_manager_system_notice 表中的通知时，需要判断 state，如果已经拉取过，就不需要重复拉取，否则会造成重复消费。
3. 有的小伙伴可能有疑问： 某条通知已经被拉取过的话，在其后注册的用户是不是不能再接收到这条通知？是的。但如果你想将已拉取过的通知推送给那些后注册的用户，也不是特别大的问题。只需要再写一个定时任务，这个**定时任务可以将通知的 push_time 与用户的注册时间比较一下，重新推送**即可。

认真思考的小伙伴应该也发现了，当用户量比较大比如上千万的时候，如果发送一个全体用户的通知需要挨个插入数据到一张表的话，是不靠谱的！

常见的解决办法，有两种方式:

1. 每位用户单独有一张或者几张专门用来存放站内消息的表，根据 `hash(userId)` 作为表名后缀。
2. 对于系统通知类型，只存放一条数据到 t_user_system_notice 表，用户自己拉取数据然后再判断消息是否已经读取过即可。

并且，当一条通知需要发布给全体用户时，我们还应该考虑到用户的活跃度。因为如果有些用户长期不活跃，我们还将通知推送给他（她），这显然会造成空间的浪费。 所以在选取用户 ID 时，我们可以将用户上次登录的时间与推送时间做一个比较，如果用户一年未登陆或几个月未登录，我们就不选取其 ID，进而避免无谓的推送。

以上就是系统通知的设计了，接下来再看看较难的提醒类型的消息。

事件提醒(EventRemind)

之所以称提醒类型的消息为事件提醒，是因为此类消息均是通过用户的行为产生的，如下：

- xxx 在某个评论中@了你；
- xxx 点赞了你的文章；
- xxx 点赞了你的评论；
- xxx 回复了你的文章；
- xxx 回复了你的评论；
- ……

诸如此类事件，我们以单词 action 形容不同的事件（点赞，回复，@（at））。
可以看到除了事件之外，我们还需要了解用户是在哪个地方产生的事件，以便当我们收到提醒时，
点击这条消息就可以去到事件现场，从而增强用户体验，我以事件源 source 来形容事件发生的地方。

- 当 action 为点赞，source 为文章时，我就知道：有用户点赞了我的某篇文章；
- 当 action 为点赞，source 为评论时，我就知道：有用户点赞了我的某条评论；
- 当 action 为@（at）， source 为评论时，我就知道：有用户在某条评论里@了我；
- 当 action 为回复，source 为文章时，我就知道：有用户回复了我的某篇文章；
- 当 action 为回复，source 为评论时，我就知道：有用户回复了我的某条评论；

由此可以设计出事件提醒表 t_event_remind，其结构如下：

字段名	类型	描述
event_remind_id	LONG	消息 I
action	VARCHAR	动作类型，如点赞等
source_id	LONG	事件源 ID，如评论
source_type	VARCHAR	事件源型："Comment
source_content	VARCHAR	事件源的内容，比如回复的评论
url	VARCHAR	事件所发生的地
state	BOOLEAN	是否已
sender_id	LONG	操作者的 ID，即谁了你
recipient_id	LONG	接受通知的用
remind_time	TIMESTAMP	提醒的时

消息聚合

消息聚合只适用于事件提醒，以聚合之后的点赞消息来说：

- 100 人 {点赞} 了你的 {文章 ID = 1}：《A》；
- 100 人 {点赞} 了你的 {文章 ID = 2}：《B》；
- 100 人 {点赞} 了你的 {评论 ID = 3}：《C》；

聚合之后的消息明显有两个特征，即： **action** 和 **source type**，这是系统消息和私信都不具备的，
所以我个人认为事件提醒的设计要稍微比系统消息和私信复杂。

如何聚合?

稍稍观察下聚合的消息就可以发现：某一类的聚合消息之间是按照 source type 和 source id 来分组的，
因此我们可以得出以下伪 SQL：

```SQL
SELECT * FROM t_event_remind WHERE recipient_id = 用户ID
AND action = 点赞 AND state = FALSE GROUP BY source_id , source_type;
```

当然，SQL 层面的结果集处理还是很麻烦的，所以我的想法先把用户所有的点赞消息先查出来，然后在程序里面进行分组，这样会简单不少。

拓展

其实还有一种设计提醒表的做法，即按业务分类，不同的提醒存入不同的表，这样可以分为:

1. 点赞提醒表
2. 回复提醒表
3. at(@)提醒表。

我认为这种设计比第一种的更松耦合，不必所有类型的提醒都挤在一张表里，但是这也会带来表数量的膨胀。 所以各位小伙伴可以自行选择方案。

私信

站内私信一般都是点到点的，且要求是实时的，服务端可以采用 Netty 等高性能网络通信框架完成请求。

我们还是以 B 站为例，看看它是怎么设计的：

B 站的私信部分可以分为两部分：

1. 左边的与不同用户的聊天室；
2. 与当前正在对话的用户的对话框，显示了当前用户与目标用户的所有消息。

按照这个设计，我们可以先设计出聊天室表 t_private_chat，因为是一对一，所以聊天室表会包含对话的两个用户的信息：

字段名	类型	描述
private_chat_id	LONG	聊天室
user1_id	LONG	用户 1 的
user2_id	LONG	用户 2 的
last_message	VARCHAR	最后一条消息

这里 user1_id 和 user2_id 代表两个用户的 ID，并无特定的先后顺序。

接下来是私信表 t_private_message 了，私信自然和所属的聊天室有联系，且考虑到私信可以在记录中删除（删除了只是不显示记录，但是对方会有记录，撤回才是真正的删除），就还需要记录私信的状态，以下是我的设计：

字段名	类型	描述
private_message_id	LONG	私信 I
content	TEXT	私信内
state	BOOLEAN	是否已
sender_remove	BOOLEAN	发送消息的人是否把 天记录中删
recipient_remove	BOOLEAN	接受人是否把这条消 删除了
sender_id	LONG	发送者
recipient_id	LONG	接受者
send_time	TIMESTAMP	发送时

消息设置

消息设置一般都是针对提醒类型的消息的，且肯定是由用户自己设置的。所以我想到一般有以下设置选项：

1. 是否开启点赞提醒；
2. 是否开启回复提醒；
3. 是否开启@提醒；

下面是 B 站的消息设置：

可以看到 B 站还添加了陌生人选项，也就是说如果给你发送私信的用户不是你关注的用户，那么视之为陌生人私信，就不接受。

以下是我对于消息设置的设计：

字段名	类型	描述
user_id	LONG	用户 I
like_message	BOOLEAN	是否接收点
reply_message	BOOLEAN	是否接收回
at_message	BOOLEAN	是否接收 a
stranger_message	BOOLEAN	是否接收陌生

总结

以上就是我对于整个站内消息系统的大概设计了，我参考了很多文章的内容以及很多网站的设计，但实际项目的需求肯定与我所介绍的有很多出入，所以各位小伙伴可以酌情参考。