Atitit im技术包含要点

目录

[1. IM系统大致的选型过程：服务方式，网络通讯协议，数据通信协议选择、协议设计。 1](#_Toc17980)

[2. P2P还是服务器中转？ 1](#_Toc6640)

[2.1.1. 1P2P方式 2](#_Toc31629)

[2.1.2. 2服务器中转方式 2](#_Toc27068)

[3. 网络通讯技术？ 2](#_Toc10891)

[3.1. 基于TCP的长连接； 2](#_Toc14521)

[3.2. 基于HTTP短连接PULL的方式。 2](#_Toc1108)

[4. 《Web端即时通讯技术盘点：短轮询、Comet、Websocket、SSE》 3](#_Toc18246)

[4.1. 传统通信方式实现IM应用需要解决的问题 3](#_Toc13462)

[4.2. 关于Ajax短轮询： 3](#_Toc5089)

[4.3. 有关Comet技术的详细介绍请参见： 3](#_Toc25404)

[4.4. 有关WebSocket的详细介绍请参见： 4](#_Toc17576)

[4.5. SSE（服务器推送事件（Server-sent Events） 4](#_Toc11360)

[5. 该如何设计私有通信协议？ 4](#_Toc8690)

[5.1.1. 1序列化与反序列化 4](#_Toc2299)

[5.1.2. 2协议格式设计 4](#_Toc21202)

[5.1.3. 协议设计 5](#_Toc13802)

[5.1.4. 2快速重连 6](#_Toc21111)

[6. 其他不可忽视的问题 7](#_Toc21568)

[6.1.1. 1协议加密 7](#_Toc3528)

[6.1.2. 2快速连接（即掉线重连机制） 7](#_Toc5370)

[6.1.3. 3连接保持（即心跳机制） 7](#_Toc21169)

[6.1.4. 4消息可达（即QoS机制） 8](#_Toc5014)

[6.1.5. 5文件上传优化 8](#_Toc18166)

[7. Ref 8](#_Toc24091)

# IM系统大致的选型过程：服务方式，网络通讯协议，数据通信协议选择、协议设计。

# P2P还是服务器中转？

IM通讯方式无非两种选择：设备直连(P2P)和通过服务器中转。

### 1P2P方式

P2P多见于局域网内聊天工具，典型的应用有:飞鸽传书、天网Maze(你懂的)等。这类软件在启动后一般做两件事情：

* 进行UDP广播:发送自己信息和接受同局域网内其他端信息；
* 开启TCP监听:等待其他端进行连接。

详细的流程可以参考飞鸽传书源码。但是这种方式在有种种限制和不便:一方面它只适合在线的点对点消息传输，对离线，群组等业务支持不够。另一方面由于 NAT 的存在，使得不同局域网内机器互联难度大大上升，在某些网络类型(对称NAT)下无法建立连接。

### 2服务器中转方式

几乎所有互联网IM产品都采用服务器中转这种方式进行消息传输，相对于P2P的方式，它有如下的优点:

* 能够支持更多P2P无法支持或支持不好的业务，如离线消息，群组，聊天室服务；
* 方便业务逻辑的拓展和新旧版本的兼容。

当然它也有自己的问题:服务器架构复杂，并发要求高。

消息模型跟邮箱的邮件模型也很有渊源，都是存储转发。

上图展示了这一消息模型，消息被发出后，会先在后台临时存储；为使接收者能更快接收到消息，会推送消息通知给接收者；最后客户端主动到服务器收取消息。

# 网络通讯技术？

****IM主流网络通讯技术有两种:****

## 基于TCP的长连接；

## 基于HTTP短连接PULL的方式。

后者常见于WEB IM系统(当然现在很多WEB IM都是基于WebSocket实现)，它的优点是实现简单，方便开发上手，问题是流量大，服务器负载较大，消息及时性无法很好地保证，对大规模的用户量支持不够，比较适合小型的IM系统,如小网站的客户系统。  
  
基于TCP长连接则能够更好地支持大批量用户，问题是客户端和服务器的实现比较复杂。当然也还有一些变种，如下行使用MQTT进行服务器通知/消息的下发，上行使用HTTP短连接进行指令和消息的上传。这种方式能够保证下行消息/指令的及时性，但是在弱网络下上行慢的问题还是比较严重。早期的来往就是基于这种方式。

# 《[Web端即时通讯技术盘点：短轮询、Comet、Websocket、SSE](http://www.52im.net/thread-336-1-1.html" \t "http://www.52im.net/_blank)》

## **传统通信方式实现IM应用需要解决的问题**

我们认识到基于web实现IM软件依然要走浏览器请求服务器的模式，这这种方式下，针对IM软件的开发需要解决如下三个问题：

* 双全工通信：  
  即达到浏览器拉取（pull）服务器数据，服务器推送（push）数据到浏览器；
* 低延迟：  
  即浏览器A发送给B的信息经过服务器要快速转发给B，同理B的信息也要快速交给A，实际上就是要求任何浏览器能够快速请求服务器的数据，服务器能够快速推送数据到浏览器；
* 支持跨域：  
  通常客户端浏览器和服务器都是处于网络的不同位置，浏览器本身不允许通过脚本直接访问不同域名下的服务器，即使IP地址相同域名不同也不行，域名相同端口不同也不行，这方面主要是为了安全考虑。

## ****关于Ajax短轮询：****

找这方面的资料没什么意义，除非忽悠客户，否则请考虑其它3种方案即可。

## ****有关Comet技术的详细介绍请参见：****

《[Comet技术详解：基于HTTP长连接的Web端实时通信技术](http://www.52im.net/thread-334-1-1.html" \t "http://www.52im.net/_blank)》  
《[WEB端即时通讯：HTTP长连接、长轮询（long polling）详解](http://www.52im.net/thread-224-1-1.html" \t "http://www.52im.net/_blank)》  
《[WEB端即时通讯：不用WebSocket也一样能搞定消息的即时性](http://www.52im.net/thread-296-1-1.html" \t "http://www.52im.net/_blank)》  
《[开源Comet服务器iComet：支持百万并发的Web端即时通讯方案](http://www.52im.net/thread-330-1-1.html" \t "http://www.52im.net/_blank)》

## ****有关WebSocket的详细介绍请参见：****

比如[socket.io](http://www.52im.net/forum.php?mod=viewthread&tid=190&ctid=15" \t "http://www.52im.net/_blank)、[sockjs](https://github.com/sockjs/sockjs-client" \t "http://www.52im.net/_blank)

****有关SSE的详细介绍文章请参见：****  
《[SSE技术详解：一种全新的HTML5服务器推送事件技术](http://www.52im.net/thread-335-1-1.html" \t "http://www.52im.net/_blank)》

## SSE（服务器推送事件（Server-sent Events）

为了解决浏览器只能够单向传输数据到服务端，HTML5提供了一种新的技术叫做服务器推送事件SSE（关于该技术详细介绍请参见《[SSE技术详解：一种全新的HTML5服务器推送事件技术](http://www.52im.net/thread-335-1-1.html" \t "http://www.52im.net/_blank)》），它能够实现客户端请求服务端，然后服务端利用与客户端建立的这条通信连接push数据给客户端，客户端接收数据并处理的目的。从独立的角度看，SSE技术提供的是从服务器单向推送数据给浏览器的功能，但是配合浏览器主动请求，实际上就实现了客户端和服务器的双向通信。它的原理是在客户端构造一个eventSource对象，该对象具有readySate属性，分别表示如下：

0：正在连接到服务器；

# 该如何设计私有通信协议？

## **IM通信协议设计的考量**

目前行业里有一些开源的方案如 XMPP、MQTT 等。但是，这些开源的方案对后期产品的增长，用户量级的突发式爆增是非常不利的。 原因有几方面，一个是这些开源项目出现的较早，其实并没有考虑到移动互联网 2/3/4G 等复杂的网络情况，包括应对电信运营商在信令等方面的复杂设置，另外，目前鲜有对这些开源技术软件和服务把控度比较强的技术团队，难以进行源码级的扩展和修改，出现问题也难以及时解决。因此，基本上各大IM产品，如QQ、微信等肯定都是自研的协议。

### 1序列化与反序列化

移动互联网相对于有线网络最大特点是:带宽低，延迟高，丢包率高和稳定性差，流量费用高。所以在私有协议的序列化上一般使用二进制协议，而不是文本协议。  
  
常见的二进制序列化库有protobuf和MessagePack，当然你也可以自己实现自己的二进制协议序列化和反序列的过程，比如蘑菇街的TeamTalk。但是前面二者无论是可拓展性还是可读性都完爆TeamTalk(TeamTalk连Variant都不支持，一个int传输时固定占用4个字节)，所以大部分情况下还是不推荐自己去实现二进制协议的序列化和反序列化过程。

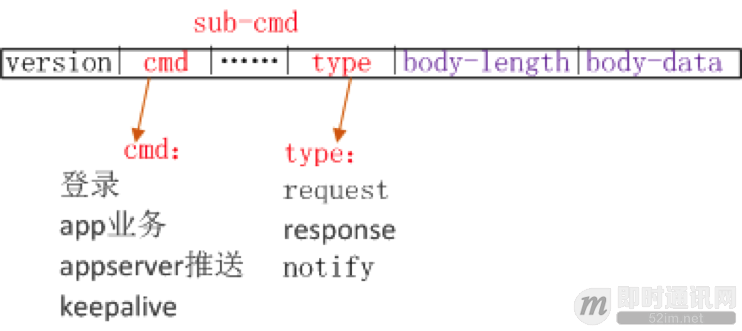
### 2协议格式设计

基于TCP的应用层协议一般都分为包头和包体(如HTTP)，IM协议也不例外。包头一般用于表示每个请求/反馈的公共部分，如包长，请求类型，返回码等。 而包头则填充不同请求/反馈对应的信息。  
  
****一个最简单的包头可以定义为：****

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | **struct** PackHeader  {      int32\_t     length\_;    //包长度      int32\_t     serial\_;    //包序列号      int32\_t     command\_;   //包请求类型      int32\_t     code\_;      //返回码  }; |

以心跳包为例，假设当前的serial为1，心跳包的command为10，那么使用MessagePack做序列化时:length=4，serial=1，command=10，code=0，每个字段各占一个字节，包体为空，仅需要4个字节。  
  
当然这是最简单的一个例子，面对真正的业务逻辑时，包体里面会需要塞入更多地信息，这个需要开发根据自己的业务逻辑总结公共部分,如为了兼容加入的协议版本号,为了负载均衡加入的模块id等。

### 协议设计

我们在进行协议设计时候，主要考虑扩展性、可调试性和异步处理。  
  
首要是扩展性。我们可以想象一下，消息平台里面，可能会有哪些扩展的需求？第一类有各种报文，比如登录、APP业务、appserver推送、keepalive等。最终消息推送的话肯定是业务的消息，对于业务包数据的变更我们要做到可扩展。其实也很简单，对于报文的话，比如登陆的话是一个，发消息的是另外一个。另外我们消息传送的话，有几类消息，可能我是个请求消息，另外可能是发出请求之后别人会返回我一个消息，还有一个就是由APP推过来的，在网络上面只有读和写或者收发，但是并不确认这个消息到底属于哪一个类型，所以这边有一个标识。如下图所示。  
  
  
  
一般的话可能还有更复杂的业务，比如有好友、匿名登陆，基于这一类设计的话，可能还会有一个协议的命令，比如登陆有登陆的协议，或者好友有好友的，对应好友可能有一些子命令，比如查看好友信息。如果类似复杂的业务可以把这些消息包规范好，另外像匿名登陆可以用协议族方式，一个大的协议族可以有子命令。另外业务数据包的话，可以通过变长的包体，只要跟业务方约定好，我们是基于怎样训练化的机制就可以了，至于里边放怎样的数据，其实随便，所以就可以做到很好的扩展。  
  
另外可调试性，因为消息的流转会有很多环节，APP端开始到gate，再到MQ等等，假设某个请求包处理失败，我们如何快速知道这个消息包已经流转到哪个步骤了？是APP端的问题，还是msg-gate的问题，还是msg-logic的问题，还是redis的问题？解决的方案比较简单，就给一个统一标识就可以了。如下图所示。  
  
58到家实时消息系统的协议设计等技术实践分享_3.png  
  
还有一个异步化的支持。这种消息通道最重要的是解决通道问题，所有消息处理不能是同步的，必须是异步的，你发一个消息出去，ABC三个包，你收到XYZ三个包之后，你怎么知道它是对应的，就是对应关系的话我们怎么处理，就是加一个ID，如下图所示。  
  
  
  
你可以维持一个发送包的上下文队列，当你收到包之后你从对应找到上下文做到处理。异步化处理的话两点，第一点是要有队列，另外一个就是必须要有一个回调，基本是队列加回调，因为收到小型包之后回去怎么做不确定，一般是这两个可以解决。

### 2快速重连

另外一个TCP的快速重连，连接之后只是说明我们可以收发包消息了，但是连接之前要做一些验证，比如是不是合法用户等。  
  
我们解决快速重连的方法就是射线保持，不会清掉就没必要重复创建了。由于无线网络特别不稳定，射线保持也会引入其他的问题。如果断开，射线还在这里的话有两类消息，一类消息从APP-server推送到APP端的时候，因为TCP已经断开了，对应的其实你是发不出去消息的，这种情况下怎么办。  
  
可能会导致往一群司机推送订单的时候，这些司机就断开了，推送会失败，现在的做法会把射线清掉。因为这个情况没法重建TCP链接，因为是从server端到APP端的。  
  
另外一类，APP端往app-server推送消息的时候，因为我们这边TCP连接之后肯定有一些机制保持连接，一种通过心跳，另外再发消息的时候，我会重新建立TCP连接，会出现一个问题，重建连接的时候，我发送连接请求的服务器的话，可能不是上一台，之前连了一台服务器，现在网络点开重连，选的是另外一台服务器，这个是有问题的，导致用户在两台上边登陆了，这个问题怎么解决。  
  
第一个问题，我们其实不好再重建TCP连接，因为是APP－server到APP的推送，好的做法就是把这个射线清掉，因为我们维护了APP端到gate的连接，这个是需要做的。另外一个就是像这种APP端同时登陆两个gate的时候，首先第一步把上一个清掉，重新再建立新的TCP连接。这个东西会有问题就是说网络不稳定，出现APP端跟好几个接入层进行了连接，优化的方案就是会在APP端做一些处理，比如可以记录上一次我和哪一台服务器连接，下一次网络有连接的时候就走上一台的，这样就不会导致登陆两个服务器的情况。

# 其他不可忽视的问题

上面的内容就是一个IM系统大致的选型过程：服务方式，网络通讯协议，数据通信协议选择、协议设计。但是实际开发过程中还有大量的问题需要处理。

### 1协议加密

为了保证协议不容易被破解，市面上几乎所有主流IM都会对协议进行加密传输。常见的流程和HTTPS加密相似:建立连接后，客户端和服务器进行进行协商，最终客户端获得一个当前Sessino的秘钥，后续的数据传输都通过这个秘钥进行加解密。一般出于效率的考虑都会采用流式加密，如RC4。而前期协商过程则推荐使用RSA等非对称加密以增加破解难度。

### 2快速连接（即掉线重连机制）

对iOS APP而言，因为没有真后台的存在，APP每次启动基本都需要一次重连登录(短时间内切换除外)，所以如何快速重连、重登就非常重要。  
****常见优化思路如下:****

* 本地缓存服务器IP并定期刷新。移动网络调优可以参考《[iOS端移动网络调优的8条建议](http://www.52im.net/thread-134-1-1.html" \t "http://www.52im.net/_blank)》；
* 合并部分请求。如加密和登录操作可以合并为同一个操作，这样就可以减少一次不必要的网络请求来回的时间；
* 简化登录后的同步请求，部分同步请求可以推迟到UI操作时进行，如群成员信息刷新。

### 3连接保持（即心跳机制）

一般APP实现连接保持的方式无非是采用应用层的心跳，通过心跳包的超时和其他条件(网络切换)来执行重连操作。那么问题来了:为什么要使用应用层心跳和如何设计应用层心跳。众所周知TCP协议是有KEEPALIVE这个设置选项，设置为KEEPALIVE后，客户端每隔N秒(默认是7200s)会向服务器发送一个发送心跳包。  
  
****但实际操作中我们更多的是使用应用层心跳。原因如下:****

* KEEPALIVE对服务器负载压力比较大(服务器大大是这么说的...)；
* socks代理对KEEPALIVE并不支持；
* 部分复杂情况下KEEPALIVE会失效，如路由器挂掉，网线(移动端没有网线...)直接被拔除。

****移动端在实际操作时为了节约流量和电量一般会在心跳包上做一些小优化：****

* 精简心跳包，保证一个心跳包大小在10字节之内；
* 心跳包只在空闲时发送；
* 根据APP前后台状态调整心跳包间隔 (主要是安卓)。

### 4消息可达（即QoS机制）

在移动网络下，丢包，网络重连等情况非常之多，为了保证消息的可达，一般需要做消息回执和重发机制。参考易信，每条消息会最多会有3次重发，超时时间为15秒，同时在发送之前会检测当前连接状态，如果当前连接并没有正确建立，缓存消息且定时检查(每隔2秒检查一次，检查15次)。所以一条消息在最差的情况下会有2分钟左右的重试时间，以保证消息的可达。  
  
因为重发的存在，接受端偶尔会收到重复消息，这种情况下就需要接收端进行去重。通用的做法是每条消息都戴上自己唯一的message id(一般是uuid)。

### 5文件上传优化

IM消息(包括SNS模块)内包含大量的文件上传的需求，如何优化文件的上传就成了一个比较大的主题。  
  
****常见有下面这些优化思路:****

* 将上传流程提前:音频提供边录边传。朋友圈的图片进行预上传，选择图片后用户一般会进行文本输入，在这段时间内后台就可以默默将选好的图片进行上传；
* 提供闪电上传的方式:服务器根据MD5进行文件去重；
* 优化和上传服务器的连接(参考快速连接)，提供连接重用的功能；
* 文件分块上传:因为移动网络丢包严重，将文件分块上传可以使得一个分组包含合理数量的TCP包，使得重试概率下降，重试代价变小，更容易上传到服务器；
* 在分包的前提下支持上传的pipeline，避免不必要的网络等待时间；
* 支持断点续传。

# 群聊的实现

较重要的变化就是这些功能催生了对异步队列的需求。例如，微博私信需要跟外部门对接，不同系统间的处理耗时和速度不一样，可以通过队列进行缓冲；群聊是耗时操作，消息发到群后，可以通过异步队列来异步完成消息的扩散写等等。

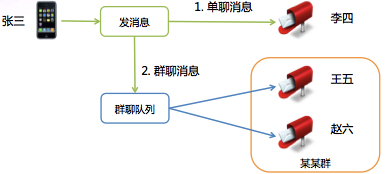
  
图 4 单聊和群聊消息发送过程

图4是异步队列在群聊中的应用。微信的群聊是写扩散的，也就是说发到群里的一条消息会给群里的每个人都存一份（消息索引）。

# 消息存储和下发方式

## **推送消息**

基于长连接的IM服务的消息大量通过PUSH进行下发，但如果用户设备不在线或网络丢包则无法接收。

## **缓存消息**

服务端为了保证更高的消息拉取性能一般都要把热点数据落入缓存中，保证用户可以快速的拉取最新的若干条记录，而不用去持久层进行查询。服务器需要考虑缓存消息的队列长度以及对应的过期时间等，保证缓存的良性存储率。

## **持久化消息**

光依靠推送和缓存是不够的，消息必须要进行持久化，保证缓存不可用的情况下仍然可以拉取消息，一般都是落入到数据库或HBase中，需要规划好分库、分表规则和RowKey的设计，防止热点数据造成的存储不均。

# 如何保证消息安全性

## **通信安全**

即消息传输层面的安全，在传输过程中，通过协议和加密算法进行保证，使传输过程中的消息是不可逆的。恶意用户即使抓到网络传输的包也没有办法破译出来。

## **会话安全**

加密级别最好做到会话级别，即一个用户的一次长链接中使用固定的密钥，即使同一个用户多次登录，或者在不同时间点登录，加密的密钥都是不一样的。

## **数据存储安全**

这里主要涉及：

* 如何实现数据的高效存储和访问 涉及到序列化和反序列化的算法和数据结构
* 如何实现尽量小的存储空间 可以考虑配合一定的压缩算法进一步减少序列化后的数据的大小
* 数据的加密存储 需要有统一的密钥管理体系来保证数据的加密存储以及密钥的安全性，保证被拖库后数据仍然是安全的

## **数据脱敏**

一般的IM中的用户账号体系都是采用自增ID来设计的，这样的坏处是可以被猜测已经容易受到攻击。 可以通过在业务层做一个mapping来避免暴露用户的核心信息，如uid、手机号等。具体算法可以使用混淆算法或自定义混淆数据库里的数据做映射，一般都是使用无法猜测的定长字符串的形式生成映射库里的数据。

## **防止重放**

数据在整个传输的过程中可能存在安全风险，比方第三方的攻击，以及数据在网络流转过程中被拷贝和重放的潜在安全风险，这些在设计过程中都需要被规避掉。

# Ref

新手入门贴：史上最全Web端即时通讯技术原理详解-网页端IM开发/专项技术区 - 即时通讯开发者社区!

移动端IM开发需要面对的技术问题-IM开发/专项技术区 - 即时通讯开发者社区!

亿级架构IM平台的技术难点解析 - 知乎