# Android平台上关于IM的实践总结（

## 前言

IM通信在互联网发展到现在已经是码农的世界里人尽皆知的技术，特别在当下移动互联网迅猛发展的时代这种技术的开发也更加火热，其中老牌的代表作就有QQ和MSN，和最近新崛起的微信，默默，易信，来往等眼花缭乱的各种应用都把IM技术应用其中。我是Android开发人员，写这篇文章主要原因也是因为我自己从事开发以来主要做过的几款APP都是包含着IM通信，在不断的摸爬滚打的解决问题的过程中，积累了一些经验记录便将其记录到博客中作为自己一个阶段性的总结，也可以分享其他需要的开发者，作为一种参考实践的方案，当然，我也并非神马大神级别的人物，如果博客中存在错误或者读者看完后觉得有疑问，欢迎在文章下方留下你的建议或问题作为评论。

## 从socket到XMPP协议

一个简单完成的IM通信流程如同下面的模型里面1-2-3-4的步骤所示，两个移动设备之间的收发消息都需要服务器进行中转并做消息的转发。



整个IM流程中，服务器接收消息发送方的请求，并检查接收方当前是否在线，如果在线的话则直接向接收方发送消息，如果不在线则作为离线消息存储到数据库中，等待接收方上线的时候再将消息进行发送。

IM作为一门网络通信的技术，必然涉及到通信协议的问题，而在Android平台上做IM开发，目前我所涉及到的协议有socket和XMPP两种类型，当然，这两者并不是一种并列的关系，从严格意义上来说socket是Java所有网络实现通信的基石，在Java上所有任何网络协议通信，最终都要依赖与socket来实现，在Android平台上使用XMPP协议通信，其最终底层同样还是依赖于socket实现，最典型的参考例子就是smack了，估计绝大多数小企业在做IM开发的时候，都会使用已经把XMPP协议封装好smack，以此节省开发时间，smack本身就是用Java语言开发，并采用了mina作为核心实现的框架（mina本身就是一个Java的NIO通信框架，但是很久不更新了，原先mina的作者写了一个新的NIO通信框架叫做netty，应该是做为mina的替代者）。作为socket和XMPP协议实现IM通信其主要的优缺点如下：

|  |  |  |
| --- | --- | --- |
| 实现方式  指标 | Socket | XMPP |
| 开发效率 | 低。相当于自定义协议，需要编写相关的协议内容，例如，断线重连机制和消息体格式等等。开发周期相对较长。 | 高。已经有提供好的jar包供我们使用，很多需要考虑的东西已经有开源贡献者封装定制好，我们可以直接使用，不必重复制造车轮，缩短开发的周期。 |
| 灵活性 | 高。可以自己决定相关的机制，已经消息体的内容格式等等，根据自身的开发需求来进行决定，具有很高的自定义灵活性，但伴随而来的是开发周期延长的问题。 | 低，已经是约定熟成的协议通信方式，不能做任何更改，需要按照协议的规定来进行消息收发通信。 |
| 消息冗余性 | 可控。使用socket开发可以自定义消息体的格式，例如JSON、XML或者直接传递对象Protocol Buffer等。 | 不可控。XMPP协议是使用XML格式的消息体来收发消息。XML本身的格式就比较冗余，伴随而来的会存在解析效率相对较慢和耗电耗流量的问题。 |

因此，在实际的IM开发中，到底是选用XMPP协议和使用socket来实现，可以根据开发应用场景和他们的优劣来决定，当然，也不仅仅局限于IM开发，服务器给客户端做消息推送也可以作为参考。

## 长连接和数据丢包

IM开发中除了基本的消息收发，以及协议使用方式的选择外，还需要解决所有IM通信技术所面临的两个共性问题。

其一，保持客户端和服务器间的长连接。这是由于业务场景所需要， IM通信实现的基础是需要时时刻刻保持客户端在服务器上的在线状态，这样才能保证发送方的消息能够及时到接收方，特别是微信的设计概念出来后，IM类型的客户端已经没有在线和离线状态的区别了，这是和QQ最大的区别 ，在这种情况下对客户端实时在线的要求就变得更高。那么作为客户端如何实时保持在线状态呢？由于Java网络通信的最底层都要依赖于socket来实现，而socket又分为UDP和TCP两种类型，从理论上来讲，我们使用TCP类型的socket来做实现通信即可保证长连接的实现，不过这仅仅是理论上而已，为什么单靠TCP不行咧，这跟我们真是的网络环境和操作系统都有一定的关系，预知为毛，请接着往下看。

首先，在上一面那种简单的IM通信图中，仅仅列出了IM通信的服务器和客户端，但是在实际的网络环境是非常复杂的（如下图所示，黄色的闪电图形代表一个连接），我们发送的消息报文是需要经过很多不同的运营商线路，在各个交换机和路由器间穿梭才能最终到达接收方的设备上，下图是一个简单的网络消息传递图，黄色的闪电线连起来就是TCP所保持的长连接，理论上这个链接是一直保持通畅的，所以客户端只需建立连接后就可以撒手不管。

但是从性能和耗费资源的角度来看，使用socket是非常耗费资源的，特别是保持长连接的TCP类型socket，所以在实际的运营商设备上，这个长连接保持一小段时间后就会被断开，这样做也是运营商为了节省和充分利用设备的资源，而中间有一环的链接断开之后，双方就再也无法保持长连接且进行通信，因为如果仅仅依赖于TCP的长连接，此时服务器和客户端是并不知道链接已经被断开的，所以就会造成下面将要说到的数据丢包问题。除了运营商自己会将设备的长连接断开外，还有包括其他原因也将导致链接断开，例如，运营商的设别断电或故障等、移动设别接收端本身设备断点掉线等诸多复杂的因素存在导致直接依赖于TCP的长连接方案不可行，其次例如Android操作系统本身也是会对TCP的长连接做处理，诸如之前所说的，TCP的socket长连接是非常耗费资源的，这种资源相对于移动设备来说更加宝贵，所以在一定时间的TCP长连接闲置后，系统也会自动清除设备上的长连接，以此节省移动设别上的能耗。为了彻底解决长连接的问题，IM技术实际上引入了心跳包的方案，说起来这种实现方案也并不复杂，就是客户端每个一会儿就要向服务器发送一个请求报文（这个报文可以是空的，跟服务器那边约定好），目的是为了告诉服务器“我是在线的”，而服务器在接收心跳请求报文之后同样需要反馈一个消息告诉客户端“我知道了”，通过应用层上的不断进行消息收发来维持长连接的存在，当客户端没有一直没有收到服务器的反馈时默认已经掉线，进行相关的业务逻辑处理后将再重连服务器，同样服务器没有收到客户端发送来的心跳包时，可以默认客户端已经掉线，将服务器上的通信socket进行关闭回收资源。

其二，客户端发送数据的丢包问题，在第一个原因中降到运营商和各种各样的网络情况都会导致发送数据的丢包，陷入以下的这种场景A和B之间同样经历1-2-3-4的一个消息收发流程，但是不幸的是由于各种原因，B返回的消息在IM服务器转发给A的时候丢失了，此时A并不知道B回复了他消息，而服务器也不知道A没有收到B的回复，这种情况下所带给用户的体验是及其差劲的。



那么要如何应对这种丢包问题呢？目前想到的一种简单粗暴的可行方案就服务器每次向客户端发送一条消息都要为这条消息做一个id的标记，客户端收到此消息后需要返回一个回执给服务器，作为标记当前客户端已经收到消息的证明，如果服务器发送出消息而迟迟没有收到服务器的回执，就把这些消息作为离线消息存放到数据库中，等到客户端重现上线的时候再将离线消息推送给客户端。通过这种方式就可以很好的解决了消息包发送途中丢包而客户端和服务器都不知情的问题。体验一下子就彪上去了，咔咔咔。

## 优化性能

由于IM通信的实时要求性比较高，伴随着长连接所耗费的资源也相对比较多，所以在移动平台上的IM通信客户端想要性能和体验上得去，性能优化是必不可少的，对此为Android平台上IM优化做一些小小的总结，当然不一定很全面，如果你有更好的建议欢迎在评论里面做补充哈。

|  |  |
| --- | --- |
| 问题 | 优化点 |
| 耗电问题 | 1. 发送心跳采用AlarmManager，对于部分尝试定制的Android系统不太适合，例如：小米的系统，Alarm的周期最小允许设置为5分钟； 2. JobScheduler（Android5.0平台提供）监控耗电情况； 3. 针对不同网络进行数据发送的优化，移动网络下载速度大于上传速度；2G一次发送数据包不要太大；3G4G一次发送多个更省电。上传buffer大小：2G 1024字节，3G 10k。下载buffer大小：2G 1024字节， 3G 30K。 4. 频繁的网络操作，在网络状况良好的进行，详情可以查看ConnectivityManager在Android5.0平台上新提供的API接口NetworkCallback、OnNetworkActiveListener。 |
| 流量问题 | 1. 减小心跳包的大小，降低心跳频率，动态更新心跳包发送频率，例如，在通信繁忙的时候，可以减少心跳包的发送等等。 2. 自定义封包的格式和大小，例如：XMPP协议使用XML作为报文格式就显得很冗余，耗电又耗流量。 3. 压缩消息体，例如采用gzip压缩算法。 |
| 丢包问题 | 暂时只有一个办法，就是客户端给服务器发送回执。 |

## 小结

以上的内容仅仅时平时开发过程中积累的一部分知识和经验的总结，希望他能帮助到其他在寻找资料开发者，同时，也感谢很多开源项目的贡献者以及热爱分享的博客撰写者，因为有他们的贡献和分享才有这篇文章的诞生，才有我在开发过程中的进步。**（请勿随意用于商业用途）**