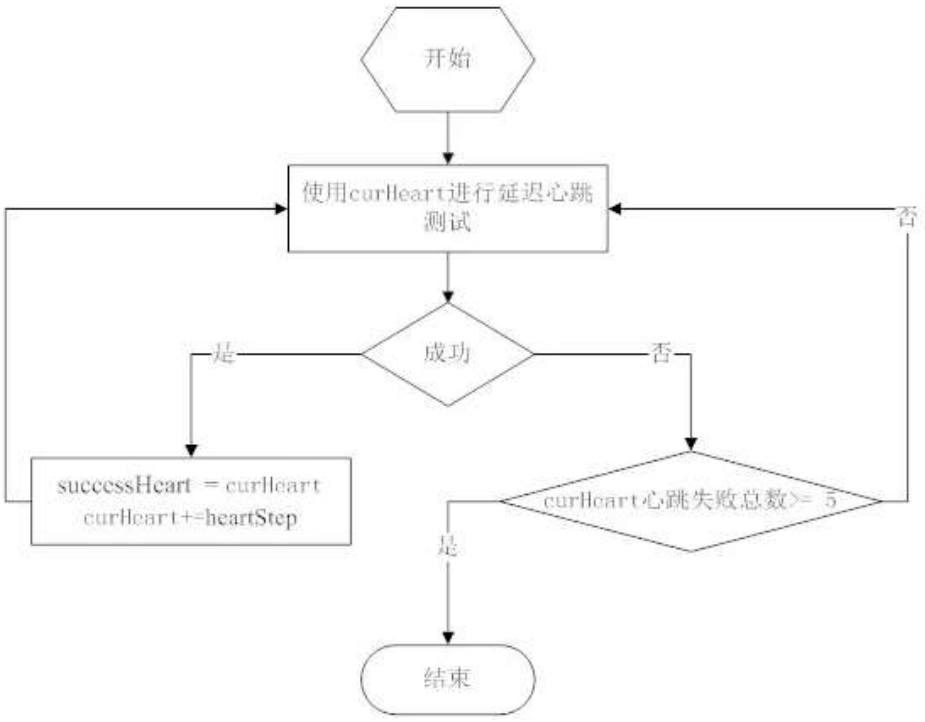
讨论课 01 – 复用解决方案

用户登录后始终在线，考虑低带宽/不稳定网络
-长连接心跳机制
-消息不遗漏
-消息不重复
-消息压缩
-……

解决方案:
1. 长连接心跳机制

在服务器和客户端之间进行长连接时有可能出现连接意外断开但服务器仍不知道的情况，此时服务器所分配给该连接的系统资源和网络开销都被浪费了，因此需要通过心跳机制来保证服务器与客户端之间的连接是有效的。但是心跳的周期也是一个重要的问题，周期过短，会使得心跳很频繁，流量消耗增大，对于移动客户端来说，还会使得耗电量增加（没收集到具体数据表明耗电量会增加多少）；而周期过长，则不能及时检测到连接的断开。

以微信的长连接心跳机制为例，老版本的微信心跳周期是 4.5 分钟，而现在微信采用了动态心跳，以获取特定用户网络下心跳的最大值。下图是微信动态心跳的自适应流程示意图：



通过上述自适应流程可以得到最合适的心跳周期（会减去 10 秒避开临界值），而当连续五次心跳确认失败后会重新进行上述过程以获得新的合适的心跳周期。

我们的项目中使用的是 NIO2，其有 SO_KEEPALIVE 选项，可以实现长连接，当该值为 true 时，每隔一段时间会检测确认与客户端的连接是否还在。
2. 消息不遗漏与不重复

消息不遗漏与不重复实际上可以通过同一种方法来解决，就是给消息编上序号，每次发

送消息后序号自增加 1，服务器响应消息时需要确定是否是需要的序号，若确认无误，则返回确认信息。客户端的消息发送之后先缓存起来，等到收到对应的确认就从缓存队列里将其移除，若超时还未收到确认则重发，多次重发失败后丢弃该条消息。这样就能同时实现消息的不遗漏和不重复。在项目中我们所使用的就是这样一种思想。

3. 消息压缩

为了减少数据传输带宽的消耗，提高传输效率，需要对消息进行压缩。而压缩算法有很多种，需要根据压缩比和压缩/解压时间综合选择。

下图是 Google 几年前发布的一组测试数据：

Algorithm	% remaining	Encoding	Decoding
GZIP	13.4%	21 MB/s	118 MB/s
LZO	20.5%	135 MB/s	410 MB/s
Zippy/Snappy	22.2%	172 MB/s	409 MB/s

可以看出 GZIP 的压缩率最高，不过其对 CPU 的消耗比其他算法要多，压缩和解压速度也慢；LZO 的压缩率居中，比 GZIP 要低一些，但是压缩和解压速度明显要比 GZIP 快很多，其中解压速度快的更多；Zippy/Snappy 的压缩率最低，而压缩和解压速度要稍微比 LZO 要快一些。

因为没有找到更多的数据，因此就上面三种压缩算法讨论，我认为在极端差的网络条件下（如带宽很低，很不稳定等），应该牺牲时间尽力压缩数据，这时 GZIP 是首选，当网络条件较好时，可以采用 LZO，特别是对于较大的内容，因为 LZO 的压缩效果能赶上 GZIP 的一半多，但其压缩和解压速度可以达到 GZIP 的四倍左右。