软件复用 讨论课1

复用解决方案 谢志杰 1352975

* 长连接心跳机制
* 消息不遗漏
* 消息不重复
* 消息压缩

1. 长连接心跳机制

所谓的心跳包就是客户端定时发送简单的信息给服务器端告诉它我还在而已。代码就是每隔几分钟发送一个固定信息给服务端，服务端收到后回复一个固定信息如果服务端几分钟内没有收到客户端信息则视客户端断开。比如有些通信软件长时间不使用，要想知道它的状态是在线还是离线就需要心跳包，定时发包收包。发包方：可以是客户也可以是服务端，看哪边实现方便合理。一般是客户端。服务器也可以定时轮询发心跳下去。心跳包之所以叫心跳包是因为：它像心跳一样每隔固定时间发一次，以此来告诉服务器，这个客户端还活着。事实上这是为了保持长连接，至于这个包的内容，是没有什么特别规定的，不过一般都是很小的包，或者只包含包头的一个空包。

在TCP的机制里面，本身是存在有心跳包的机制的，也就是TCP的选项。系统默认是设置的是2小时的心跳频率。但是它检查不到机器断电、网线拔出、防火墙这些断线。而且逻辑层处理断线可能也不是那么好处理。一般，如果只是用于保活还是可以的。心跳包一般来说都是在逻辑层发送空的包来实现的。下一个定时器，在一定时间间隔下发送一个空包给客户端，然后客户端反馈一个同样的空包回来，服务器如果在一定时间内收不到客户端发送过来的反馈包，那就只有认定说掉线了。只需要send或者recv一下，如果结果为零，则为掉线。

但是，在长连接下，有可能很长一段时间都没有数据往来。理论上说，这个连接是一直保持连接的，但是实际情况中，如果中间节点出现什么故障是难以知道的。更要命的是，有的节点（防火墙）会自动把一定时间之内没有数据交互的连接给断掉。在这个时候，就需要我们的心跳包了，用于维持长连接，保活。在获知了断线之后，服务器逻辑可能需要做一些事情，比如断线后的数据清理呀，重新连接呀当然，这个自然是要由逻辑层根据需求去做了。总的来说，心跳包主要也就是用于长连接的保活和断线处理。一般的应用下，判定时间在30-40秒比较不错。如果实在要求高，那就在6-9秒。

# 应用场景：

互联网推送消息的方式很常见，特别是移动互联网上，手机每天都能收到好多推送消息，经过研究发现，这些推送服务的原理都是维护一个长连接（要不不可能达到实时效果），但普通的socket连接对服务器的消耗太大了，所以才会出现像MQTT这种轻量级低消耗的协议来维护长连接

推送的实现方式：

一：客户端不断的查询服务器，检索新内容，也就是所谓的pull 或者轮询方式

二：客户端和服务器之间维持一个TCP/IP长连接，服务器向客户端push

三：服务器又新内容时，发送一条类似短信的信令给客户端，客户端收到后从服务器中下载新内容，也就是SMS的推送方式

苹果的推送系统和googleC2DM推送系统其实都是在系统级别维护一个TCP/IP长连接，都是基于第二种的方式进行推送的。第三种方式由于运营商没有免费开放

这种信令导致了这种推送在成本上是无法接受的，虽然这种推送的方式非常的稳定，高效和及时。

2. 消息不遗漏

# 案例：QQ聊天过程中，使用的是UDP而非TCP

QQ开发的时候，没有epoll这种可以支持成千上万TCP并发连接的技术,所以他们使用了UDP,然后在UDP上面封装了一下,模拟了一下TCP,解决了大并发的问题,之后由于技术成熟了,虽然epoll这种技术出现了,还是没有改回使用TCP.

QQ整个应用过程中的网络情况：

登陆采用TCP协议和HTTP协议，用户与好友之间发送消息，主要采用UDP协议，内网传文件采用了P2P技术。总来的说：

* 登陆过程，客户端client 采用TCP协议向服务器server发送信息，HTTP协议下载信息。登陆之后，会有一个TCP连接来保持在线状态。
* 和好友发消息，客户端client采用UDP协议，但是需要通过服务器转发。腾讯为了确保传输消息的可靠，采用上层协议来保证可靠传输。如果消息发送失败，客户端会提示消息发送失败，并可重新发送。
* 如果是在内网里面的两个客户端传文件，QQ采用的是P2P技术，不需要服务器中转。

案例来源：知乎。

链接：https://www.zhihu.com/question/20292749/answer/24557541

链接：<https://www.zhihu.com/question/20292749/answer/18294472>

3. 消息不重复

案例：Metamorphosis

Metamorphosis是淘宝开源的一个Java消息中间件，他类似apache-kafka，但不是一个简单的山寨拷贝，而是做了很多改进和优化，项目的主页在淘蝌蚪上。

消息的重复包含两个方面，生产者重复发送消息以及消费者重复消费消息。

针对生产者来说，有可能发生这种情况，生产者发送消息，等待服务器应答，这个时候发生网络故障，服务器实际已经将消息写入成功，但是由于网络故障没有返回应答。那么生产者会认为发送失败，则再次发送同一条消息，如果发送成功，则服务器实际存储两条相同的消息。这种由故障引起的重复，meta是无法避免的，因为meta不判断消息的data是否一致，因为它并不理解data的语义，而仅仅是作为载荷来传输。

针对消费者来说也有这个问题，消费者成功消费一条消息，但是此时断电，没有及时将前进后的offset存储起来，则下次启动的时候或者其他同个分组的消费者owner到这个分区的时候，会重复消费该条消息。这种情况meta也无法完全避免。

Meta对消息重复的保证只能说在正常情况下保证不重复，异常情况无法保证，这些限制是由远程调用的语义引起的，要做到完全不重复的代价很高，meta暂时不会考虑。

4. 消息压缩

# 案例：游戏运行架构Pomelo

在实际编程中，为了减少数据传输带宽的消耗，提高传输效率，pomelo提供了对消息的压缩，包括基于字典的对route的压缩和基于protobuf的对具体传输数据的压缩。

# route问题

在pomelo编程中，pomelo中的route是用来确定消息的分发路径，将其交给相应的服务器和服务处理的。route分为两类，由客户端发给服务端消息时使用的route和服务端向客户端广播时使用的route。

前一种route是由服务器自动生成的，其中的字段就代表了对应的方法在服务端的位置。如“area.playerHandler.attack”则表示在“area”类型的服务器上的“playerHandler”提供的“attack”方法，其格式为".."。 路由信息过长，使得有效消息数据负载率大大降低。例如，在聊天应用中，如果用户的发言仅仅是一个字符，结果不得不携带一个route,"chat.chatHandler.send",这样使得有效数据负载率大大降低。

后一种route是服务端想客户端推送消息时使用，是客户端的路由信息，如“onMove”，“onAttack”等，其格式一般为"on"这些字段是由用户自己定义的。虽然可以定义很短的路由，但是那样会造成可读性变差，不利于代码阅读。

一般来说，当应用固定后，具体路由就不会再变动，因此可以考虑通过一种简单替换的方式对路由信息进行压缩。

# 基于dict的压缩

pomelo中实现了基于字典的route压缩，目前route压缩功能仅仅支持hybridconnector，sioconnector目前无法使用route压缩。

# 基于protobuf的传输数据压缩

在进行消息传输时，pomelo实现了基于protobuf的数据编码协议，与其他的编码协议如xml，json相比，protobuf有着更好的传输效率和压缩比率。在ordofpomelo项目中，使用protobuf进行数据编码后的消息大小只有基于Json的编码的20%左右。

在这部分，介绍了pomelo中实现的对数据的压缩。通过对数据的压缩提高了带宽的有效数据利用率，使得在一些带宽以及流量敏感的环境中，pomelo能够更好地工作。目前pomelo的route以及基于protobuf的数据压缩仅仅支持hybridconnector，对于sioconnector，是使用json作为通信格式的，目前不支持对其进行压缩。