引言

TCP长连接服务在传统的智能网应用中扮演着重要的角色。由于其传输的高效率，在智能网SCP和IP的各个模块之间，大量使用了这种服务。例如，SS7gateway与SCF、SCF与INES、INES与外部节点、CN与VN，等等。

相反，在各种WEB应用中，广泛使用TCP短连接服务。基于HTTP承载的各种应用协议，如HTML，XML，SOAP等，多数使用TCP短连接服务。原因有二：一是这些HTTP协议的数据包较大，传输所占的开销较大，连接建立的开销相对较小。此时使用长连接对性能的提升并不明显。二是相对于长连接而言，无论是对于客户端还是服务端，短连接的实现难度要低很多。

以网通水平业务平台SPGW为例，多数对外接口采用HTTP/XML/SOAP协议和短连接。然而，出于性能的考虑，还有两个接口采用TCP长连接：

l 东向接口（SPGW-DSMP）：SCCP协议使用二进制消息。

l 南向接口（SPGW-SMSC）：SPGW与短消息网关SMSC之间的SM7协议使用二进制消息。

从Java开发语言的角度，短连接的使用比较简单。因为Java的IO库已经提供了一个httpConnection类，成熟可靠，使用方便。但是，对于TCP长连接的使用，Java的IO库并没有直接的支持。本文将探讨对TCP长连接服务的一般需求和我们的实现考虑。

以下也简称TCP长连接服务为TCP服务。

需求

具有网络编程经验的人都知道，TCP程序的编写是“易学难精”。很容易编写一个TCP程序，具有一定的功能并且在少数正常情况下可以运行。但是，要想让它在各种网络条件、各种负荷情况下都能稳定运行，却不是一件简单的工作。具体说来，TCP长连接服务需要满足以下条件：

高性能

实现这一点的关键是，消息的接收操作必须是异步的。以SPGW与短信网关之间的消息流程为例，如下：

如上图所示，SPGW可以不等待上一消息的应答消息，就发送下一个短消息。因此在同一个TCP connection上，SM7消息的接收必须是异步的，否则就会阻塞后续消息的发送。

健壮性

健壮性要求，TCP长连接服务不仅要能适应良好的网络情况和低负荷，而且要能适应差的网络情况和高负荷。实现这一点需要做到：

l 应用级心跳：自动检测网络故障。

l 应用级重连：自动排除网络故障。

l 请求分发：需要将请求消息分发到消息队列或者独立的线程中，以免阻塞接收线程的工作。

l 统计与管理：可以查询统计TCP服务模块的工作情况。也可以通过某种标准的网络管理协议（如SNMP）来控制TCP连接的状态，如打开或者关闭。

应用友好性

l 同步的响应消息接收API：尽管TCP服务内部对消息的接收是异步的，但是，它需要向应用模块提供同步的响应消息接收API，以简化应用模块的开发。

l 明确的双向接口：一般的服务包只需要提供单向的API接口，由应用模块调用。但是，TCP服务包不同，除了被应用模块调用外，它还要对应用模块进行回调。例如，当接收到消息时，需要回调应用模块的方法，对消息进行定界和分发；在进行心跳检测时，需要回调应用模块的编解码方法。因此需要明确地定义TCP模块和应用模块之间的双向接口。

传统的实现方式

在我们以往的程序实现中，一般都是采用单connection，异步消息收发的方式。

Extra Node

Application

Main Thread

Transceive Thread

Message Queue

整个系统的逻辑结构大致如上。Application一般分为Main和Transceive两个线程。前者用于完成应用的逻辑，后者完成消息的收发。它们之间通过一个共享的Message Queue来通信。二者的流程使用伪码表示如下：

Main Thread：

while(true)

从Message Queue取输入消息

处理该消息（中间可能产生输出消息送到Message Queue）

Transceive Thread：

while(true)

if(connection 可写)

从Message Queue取输出消息

将该消息写到connection

if(connection 可读)

从connection读输入消息

将该消息送到Message Queue

对于Transceive Thread，判断connection是否可读写，是为了避免阻塞。这是通过socket API的select函数来完成的。

在实际的实现中，由于C语言的多线程存在移植性问题，以上两个线程一般合为一个：

Single Thread：

while(true)

执行Main Thread的操作

执行Transceive Thread的操作

优点

以上方式的优点是非常高效，这已经为我们以前的系统的性能表现所证明。

缺点

以上方式的缺点是Main Thread的编写比较麻烦。因为没有同步的消息接收API，我们需要使用FSM之类的机制将多条消息关联起来。当一条消息发出后，需要设置FSM实例的状态和定时器。当回应消息收到时，将回应消息投递到对应的FSM实例进行处理。使用FSM机制进行开发，对于一般的应用还是太复杂了。

另外，传统的实现方式不好解决TCP服务回调应用协议功能的问题，往往将应用协议的部分功能（如定界、编解码）集成到TCP模块中。结果随着应用协议类型的增加，TCP模块变得越来越臃肿和复杂。出现这一问题的部分原因是，与Java不同，C/C++语言不存在Interface（接口）这种语言结构，用于明确地定义两个模块之间的调用接口。

Java实现

功能与结构

TCP服务模块的功能是：

1. 消息收发功能：提供了两个发送API：

l sendRecv：发送请求消息，并且同步等待响应消息。

l send：发送消息并立即返回。用于无需等待响应消息的场合。

2. 心跳功能：通过设置心跳属性，如心跳模式、心跳间隔、是否发送心跳、是否显示心跳等，TCP模块自动执行心跳检查。

3. 重连功能：对于Tcp客户端，通过设置重连属性，如是否重连，重连间隔等，TCP模块在连接断开后，自动执行重连操作。

4. 请求分发：将请求消息分发到独立的线程中，并自动调用应用模块的方法进行处理。

5. 统计与管理：可以查询统计TCP服务模块的工作情况。也可以通过SNMP和JMX来控制TCP连接的状态，如打开或者关闭。

线程分配

为了同时满足高性能和易用性的要求，TCP模块充分利用了Java语言对多线程的良好支持。它包括如下线程：

HeartBeat thread

S

Tcp服务模块

S

Recv thread

S

App 模块

S

Send thread1

S

Send thread2

S

Process request thread3

S

sendRecv

S

send

S

processRequest

S

Reconnect thread

S

Send thread：发送线程就是应用线程。也就是说，TCP模块在应用线程中完成发送操作。

Recv thread：每个TCP连接都启动一个接收线程，用于接收来自对端的消息。

HeartBeat thread：每个TCP连接都启动心跳线程，用于定期向对端发送心跳消息，并检查是否及时收到响应。

Reconnect thread：TCP模块启动一个重连线程，用于定期检查连接的状态，并试图重连关闭的连接。

Process request thread：请求处理线程是应用线程。当接收线程收到一个请求消息，它会启动一个请求处理线程，将请求消息投递给该线程进行处理。

接口

下面简要列出TCP模块提供给应用模块的ITcpService接口：

l 打开连接：open(String ip, int port)

l 关闭连接：void close()

l 发送数据包：void send(byte[] data)

l 发送数据包并等待响应：byte[] sendRecv(byte[] data, int timeout)

l 设置应用协议接口：void setTcpMessage(ITcpMessage tcpMessage);

下面简要列出应用模块提供给TCP模块的ITcpMessage接口：

l 判断给定的消息是否为请求消息：boolean isRequest(byte[] data)

l 判断给定的消息是否为心跳消息：boolean isHeartBeat(byte[] data)

l 判断给定的消息是否有效：boolean isValid(byte[] data)

l 取消息的长度：int getLength(byte[] data)

l 获取给定的消息的Key（消息的Key用于关联一对请求和响应）：String getKey(byte[] data)

l 编码心跳请求消息：byte[] EncodeHeartBeatRequest()

l 编码心跳响应消息：byte[] EncodeHeartBeatResponse(byte[] request);

l 处理请求消息：void processRequest(byte[] data);

l 执行重连操作：void reconnect();

l 设置TcpService对象：void setTcpService(ITcpService tcpService);

消息的定界

在ITcpMessage接口中，需要关注的是getLength方法：

int getLength(byte[] data) throws TcpServiceException

这一方法非常关键。表面上的功能是取消息的长度，实际上TCP模块使用此方法对消息进行定界（delimiter）。

所有使用TCP长连接服务的应用协议都需要考虑如何对消息进行定界的问题。这是因为TCP连接上每次收到的数据包不一定正好对应一条完整的消息，可能需要对数据包进行拆分/合并操作。这个处理过程烦琐而容易出错，主要由TCP模块完成。但是应用模块需要实现合适的getLength方法，才能得到健壮的定界结果。

参数

data――迄今为止，接收到的所有未处理的数据包

所有其它方法都可以认为data参数是一个完整的消息。但是此方法不同。data中可能包含1条完整消息、1条消息的部分、或者多条消息。

返回值

返回消息的长度

正数：表示已经确定消息的长度

0：表示数据不够，不能确定消息的长度。例如，某些协议将前4个字节作为一个整数，保存消息的长度。如果数据还不满4个字节，则不能确定消息的长度；某些协议以\n作为消息

的结束字符，如果数据中没有\n，则不能确定消息的长度。

负数(-n)：表示数据中的前n个字节无效，应该从第N+1个字节开始，重新定界。例如，某些协议以字节FE作为消息的开始字节，则所有FE之前的字节都是无效字节。

异常

throws TcpServiceException 如果消息的格式不合法，则抛出此异常。

例如，如果已经确定消息的长度，但长度值为负值或者超过了应用协议规定的长度上限。这说明data数据包是无效的。

一旦抛出此异常，则TCP模块清空接收缓冲区。

总结

与对HTTP短连接的支持相比，Java的标准IO库对TCP长连接的支持远远不够。我们的TCP长连接服务实现是对Java的标准IO库功能的一个补充。