基于 Windows Socket 的网络通信中的心跳机制原理及其实现

周 凤 石

（沙洲职业工学院，江苏 张家港 215600）

摘 要：阐述了 Windows Socket 网络通信中的心跳机制及基本原理，在此基础上探讨了利用 Delphi 中 ServerSocket、ClientSocket 组件进行网络通信时，如何通过发送心跳包来检测客户端的非正常断开以及断开时服务端所作的相应处理。

关键词：套节字(Socket)；心跳包；KeepAlive；失效连接；非优雅断开

中图分类号：TP393.09 **文献标识码**：A **文章编号**：1009－8429(2009)03－0017－05

The Principle and Implementation of Heartbeat Packet based on Windows Socket Communication

ZHOU Feng-shi

(Shazhou Professional Institute of Technology, Zhangjiagang 215600, China)

Abstract: This paper describes the principle of Heartbeat Packet based on Windows Socket communication. It also discusses Server's respective actions to detect the non-graceful disconnection at the Client by sending off Heartbeat Packet with Server Socket and Client Socket in Delphi for net communication.

Key words: Socket; Heartbeat Packet; Keep Alive; Invalid Connection; Non-graceful disconnection

0 引言

在采用 TCP 连接的 C/S 结构的系统中，当通信的一方正常关闭或退出时，另一方能收到相应的连接断开的通知，然后进行必要的处理；但如果任意一方发生所谓的“非优雅断开”，如：意外崩溃、死机、拔掉网线或路由器故障时，另一方无法得知 TCP 连接已经失效，除非继续在此连接上不断地发送数据，经过若干时间后导致错误返回。但在很多时候，更希望服务器端和客户端都能及时有效地检测到网络连接的非正常断开，然后完成一些必要的清理工作并把错误报告给用户。

如何及时有效地检测到通信一方的非正常断开，采用的方法是通过通信的一方或双方发送心跳包来告诉对方网络通信是否正常或已断开。

1 心跳原理

在基于电路交换的网络中，有专用的控制信令通道，能够及时发现通路断开、故障，而 TCP/IP 网络中，链路的连通只在连接双方记录状态，物理通道内不存在一个实际的连接链路，通信的双方只能定时发送简单的信息给另一方，并根据超时来判断线路是长时间空闲还是已断开。这种通过每隔一定时间发送一个固定信息给对方，对方收到后回复一个固定信息，告诉对方“我还在”的方式非常类似于心跳，所发送的这种简单信息就称为“心跳包”。

心跳包的发送，通常有两种技术：一种是由用户在应用层实现的心跳包，另一种是由 TCP 协议层提供的 KeepAlive 。

收稿日期：2009-05-13

作者简介：周凤石（1965-），男，沙洲职业工学院电子信息工程系副教授。

2 应用层自己实现的心跳包

由应用程序自己发送心跳包来检测连接是否正常，大致的方法是：服务器在一个 Timer 事件中定时向客户端发送一个短小精悍的数据包，然后启动一个低级别的线程，在该线程中不断检测客户端的回应，如果在一定时间内没有收到客户端的回应，即认为客户端已经掉线；同样，如果客户端在一定时间内没有收到服务器的心跳包，则认为连接不可用。

以下代码给出在 Delphi 中使用 ServerSocket、ClientSocket 进行网络通信时，如何在服务端实现应用层心跳包：

```
//定义一个 SocketData 记录类型，用于保存客户端信息
Type SocketData = Record
     IP: string;                  //客户端 IP
     StartTime: Cardinal;       //每次向客户端发送心跳包的当前时间
     IsConnected: Boolean;     //是否正与服务端保持连接
   end;
   PSocketData = ^SocketData;   //定义一个指向 SocketData 的指针类型

//保存客户端信息，并创建线程，检测客户端是否有回应
procedure TForm1.ServerSktClientConnect(Sender: TObject;
   Socket: TCustomWinSocket);
var P: PSocketData;
begin
   New(p);
   p.IP := Socket.RemoteAddress;
   p.IsConnected := true;
   Socket.Data := p;
   if not Timer1.Enabled then
   begin
      MyThread := TCheckTimeOut.Create(ServerSkt, Panel1);
      Timer1.Enabled := true;
   end;
end;

//在定时器的 Timer 事件中，每隔 2 秒向所有客户端发送一次心跳包
procedure TForm1.Timer1Timer(Sender: TObject);
var
   i, ActConns: integer;
   CSocket: TCustomWinSocket;
begin
   ActConns := ServerSkt.Socket.ActiveConnections;
   caption := '连接数：' + inttostr(ActConns);
   for i := 0 to pred(ActConns) do
   begin
```

```
            CSocket := ServerSkt.Socket.Connections[i];
            CSocket.SendText('Msgtest');
            if PSocketData(CSocket.Data).IsConnected then
            begin
               PSocketData(CSocket.Data).StartTime := GetTickCount;
               PSocketData(CSocket.Data).IsConnected := false;
            end;
         end;
      end;

      //如果收到客户端的特定回应，由表示该客户处于连接状态
      procedure TForm1.ServerSktClientRead(Sender: TObject;
         Socket: TCustomWinSocket);
      var RecTxt: string;
      begin
         RecTxt := Socket.ReceiveText;
         if RecTxt = 'OK' then
         begin
            PSocketData(Socket.Data).IsConnected := true;
            Memo1.Lines.Add(RecTxt);
         end;
      end;

      //线程入口，用于检测客户端是否在规定时间内向服务端回应
      procedure TCheckTimeOut.Execute;
      begin
         while true do
         begin
            Synchronize(CheckConnect);
            if terminated then exit;
         end;
      end;

      //检查所有客户端，是否在 5 000(ms)内向服务端发送回应信息
      procedure TCheckTimeOut.CheckConnect;
      var i: integer;
      begin
         for i := 0 to FServerSocket.Socket.ActiveConnections - 1 do
            if not PSocketData(FServerSocket.Socket.Connections[i].Data).IsConnected then
               if (GetTickCount - PSocketData(FServerSocket.Socket.Connections[i].Data)^.StartTime) > 5 000
then
               begin
```

```
            Dispose(PSocketData(FServerSocket.Socket.Connections[i].Data));
            FServerSocket.Socket.Connections[i].Close;
        end;
end;
```

对于客户端而言，只要在收到服务端的心跳包后，简单地发送一个回应信息即可，代码略。

3 TCP 的 KeepAlive 保活机制

因为要考虑到一个服务器通常会连接多个客户端，因此由用户在应用层自己实现心跳包，代码较多且稍显复杂，而利用 TCP/IP 协议层为内置的 KeepAlive 功能来实现心跳功能则简单得多。

不论是服务端还是客户端，一方开启 KeepAlive 功能后，就会自动在规定时间内向对方发送心跳包，而另一方在收到心跳包后就会自动回复，以告诉对方我仍然在线。

因为开启 KeepAlive 功能需要消耗额外的宽带和流量，所以 TCP 协议层默认并不开启 KeepAlive 功能，尽管这微不足道，但在按流量计费的环境下增加了费用，另一方面，KeepAlive 设置不合理时可能会因为短暂的网络波动而断开健康的 TCP 连接。并且，默认的 KeepAlive 超时需要 7,200,000 MilliSeconds，即 2 小时，探测次数为 5 次。对于很多服务端应用程序来说，2 小时的空闲时间太长。因此，我们需要手工开启 KeepAlive 功能并设置合理的 KeepAlive 参数。

以下代码给出在 Delphi 中使用 ServerSocket、ClientSocket 进行网络通信时，如何在服务端通过添加 KeepAlive 功能来自动实现心跳机制。

```
//定义心跳常量
Const
  IOC_IN = $80000000;
  IOC_VENDOR = $18000000;
  IOC_out = $40000000;
  SIO_KEEPALIVE_VALS = IOC_IN or IOC_VENDOR or 4;
  DATA_BUFSIZE = 8192;

//定义 KeepAlive 数据结构
Type
  TTCP_KEEPALIVE = packed record
    onoff: integer;
    keepalivetime: integer;
    keepaliveinterval: integer;
  end;

// 开启 KeepAlive 保活机制，每隔 3 秒向客户端发送一次心跳包
procedure TForm1.ServerSocket1ClientConnect(Sender: TObject; Socket: TCustomWinSocket);
var
  opt: integer;
  klive, outKlive: TTCP_KEEPALIVE;
begin
  opt := 1;
```

```
    if setsockopt(Socket.SocketHandle,SOL_SOCKET, SO_KEEPALIVE, @opt, SizeOf(opt)) <> 0 then
    begin
      Showmessage('setsockopt KeepAlive Error!');
    end;
    klive.onoff := 1;
    klive.keepalivetime := 3000;
    klive.keepaliveinterval := 1;
  if WSAIoctl( Socket.SocketHandle, SIO_KEEPALIVE_VALS, @klive,
                SizeOf(TTCP_KEEPALIVE), @outKlive,
                SizeOf(TTCP_KEEPALIVE), @opt,0,nil) = SOCKET_ERROR then
    begin
      Showmessage('WSAIoctl KeepAlive Error!');
    end;
  end;
```

其中，Windows Socket API 函数 Setsockopt 用于设置套接口的选项，而此处则用来开启 KeepAlive 功能，WSAIoctl 函数则用于设置 KeepAlive 超时及心跳包发送次数。

由于是在 ServerSocket 的 OnClientConnect 事件中使用 Socket 参数来设置每个连接上来的客户端的 KeepAlive，所以上述代码同样支持多客户连接的情况。

在开启了 KeepAlive 后，一旦客户端死机、拔网线等“非优雅”退出，就会触发服务端的 OnClientError 事件，然后在此事件中完成必要的“善后”处理工作：

```
  procedure TForm1.ServerSocket1ClientError(Sender: TObject;Socket: TCustomWinSocket; ErrorEvent: TErrorEvent;  var ErrorCode: Integer);
  begin
    ErrorCode:=0;
    Showmessage('客户端 '+Socket.RemoteAddress+'非正常退出，断开连接');
    ……          //“善后”处理工作
    Socket.Close;
  end;
```

4 结束语

实践证明，利用 TCP 本身支持的 KeepAlive 功能实现断线检测，比用户自己在应用层实现检测更方便有效，而且探测时带宽消耗很小。在没有数据传输时依赖 KeepAlive 确保断线检测，在传输数据时 TCP 会通过超时判断是否断线。

在网络通信应用系统开发中，根据实际需要也可以在客户端开启 KeepAlive，对服务端的非正常断开进行及时有效地检测。

参考文献：

[1] Anthony Jones. Network Programming for Microsoft Windows, Microsoft Press; 2nd Edition, 2002.

[2] 王艳平, 张越. Windows 网络与通信程序设计[M]. 北京: 人民邮电出版社, 2007.

[3] 夏靖波, 杜华桦, 王晓东. Windows 网络程序设计[M]. 西安: 西安电子科技大学出版社, 2008.