ZServer4D 异步通讯模型编程引导

该文档主要介绍 ZS 在服务器和客户端使用异步方式的思路和程序范式
首先,我们来看看非异步机制的同步,也就是我们常说的阻塞机制(发一条命令后会等反馈)
下图是分别是发送一个字符串再等一个字符串反馈,以及发送一个数据结构再等一个数据结构的反馈.

```
// wait send cmd
function WaitSendConsoleCmd(Cmd: SystemString; ConsoleData: SystemString; Timeout: TTimeTick): SystemString;
procedure WaitSendStreamCmd(Cmd: SystemString; StreamData, ResultData: TDataFrameEngine; Timeout: TTimeTick);
```

在服务器中,阻塞机制都被 ban 掉了,我们可以看见和编译 WaitSendXXX 的方法,但是无法使用,因为阻塞机制会让服务器变得非常不稳定,所有的服务器编程必须全异步的工作模式.

在客户端中,WaitSendXX 这种阻塞机制是开放的,
每个通讯接口(CrossSocket,DIOCP,ICS,Synapse,INDY)都可以使用 WaitSendXXX 的阻塞式通讯
大概这样更符合人类习惯吧.

异步数据反馈

异步反馈是指发送一个数据出去,会收到一个反馈,我常将这样的机制以”响应式”来描述.
响应式数据收发,在 ZS 只有两种收发,一种是字符串,一种是数据结构(TDataFrameEngine)
响应式会卡队列,假如前面的响应数据没有收到,后面的就不会发出去.
下图是异步方式的发送和反馈字符串以及发送数据结构和反馈反馈数据结构

```
// send console cmd and result proc
procedure SendConsoleCmdP(Cmd, ConsoleData: SystemString; OnResult: TConsoleProc); overload;
procedure SendConsoleCmdP(Cmd, ConsoleData: SystemString; Param1: Pointer; Param2: TObject; OnResult: TConsoleParamProc); overload;
procedure SendConsoleCmdP(Cmd, ConsoleData: SystemString; Param1: Pointer; Param2: TObject; OnResult: TConsoleParamProc; OnFailed: TConsoleFailedProc); overload;
// send stream cmd and result proc
procedure SendStreamCmdP(Cmd: SystemString; StreamData: TMemoryStream64; OnResult: TStreamProc; DoneAutoFree: Boolean); overload;
procedure SendStreamCmdP(Cmd: SystemString; StreamData: TDataFrameEngine; OnResult: TStreamProc); overload;
procedure SendStreamCmdP(Cmd: SystemString; StreamData: TDataFrameEngine; Param1: Pointer; Param2: TObject; OnResult: TStreamParamProc); overload;
procedure SendStreamCmdP(Cmd: SystemString; StreamData: TDataFrameEngine; Param1: Pointer; Param2: TObject; OnResult: TStreamParamProc; OnFailed: TStreamFailedProc); overload;
```

异步数据反馈编程范式 1 讲解

这是常用的响应式编程范式,发一个数据,反馈会触发一个异步事件,我们再到该事件编程
异步事件的变量引用是自动指针

```
procedure TDRClientForm.DelayResponseBtnClick(Sender: TObject);
var
  SendDe: TDataFrameEngine;
  a: Integer;
begin
  // 异步方式发送，并且接收Stream指令，反馈以proc回调触发
  a := 123;
  SendDe := TDataFrameEngine.Create;
  client.SendStreamCmdP('DelayResponse', SendDe,
    procedure(Sender: TPeerClient; ResultData: TDataFrameEngine)
    begin
      // 这里的事件在触发时,其实DelayResponseBtnClick已经执行完毕,变量a也已经不再存在,至少它脱离了正常程序范围
      // 在异步事件触发时,a在一个未被破坏的堆栈空间中,这是脱离正常使用范围的,因为这是异步事件
      // 不要在异步事件中引用外面的local变量,尽量用全局变量,或则使用para方式的异步事件,将变量以指针方式指定传递,参考DelayResponse2BtnClick实现
      while ResultData.Reader.NotEnd do
          DoStatus('server response:%s', [ResultData.Reader.ReadString]);
      // 那么你知道变量a的引用是copy还是指针吗?
      // 答案是指针,delphi的匿名函数会自动化的引用的外部变量成为一个指针,当你引用时,是在访问一个未知区域的东西
      // 这里打印出来的a是456
      DoStatus(a);
    end);
  disposeObject([SendDe]);
  a := 456;
end;
```

异步数据反馈编程范式 2 讲解

```
procedure TDRClientForm.DelayResponse2BtnClick(Sender: TObject);
type
  TMyDefine = record
    a, b, c: Integer;
  end;

  PMyDefine = ^TMyDefine;

var
  SendDe: TDataFrameEngine;
  p: PMyDefine;
begin
  // 由于异步操作,客户端往往难以用正常流程来编写,因此,我们经常会需要用到交换结构
  // PMyDefine就是交换结构,它维系了异步程序的数据一致性
  new(p);
  p^.a := 1;
  p^.b := 2;
  p^.c := 3;

  SendDe := TDataFrameEngine.Create;
  client.SendStreamCmdP('DelayResponse', SendDe, p, nil,
    procedure(Sender: TPeerIO; Param1: Pointer; Param2: TObject; SendData, ResultData: TDataFrameEngine)
    var
      p2: PMyDefine;
    begin
      // 如果客户端未离线,并且受到服务器反馈,触发该事件
      // 该事件触发时,DelayResponse2BtnClick已经调用结束,这时候我们不能直接访问p变量,因为堆栈已经被破坏,我们需要重新获取PMyDefine的指针数据到p2

      p2 := Param1;

      DoStatus('a:%d', [p2^.a]);
      DoStatus('b:%d', [p2^.b]);
      DoStatus('c:%d', [p2^.c]);

      while ResultData.Reader.NotEnd do
          DoStatus('server response:%s', [ResultData.Reader.ReadString]);

      // 如果对方离线,该事件不会触发,我们刚才申请的PMyDefine内存也会丢失
      dispose(p2);
    end,
    procedure(Sender: TPeerIO; Param1: Pointer; Param2: TObject; SendData: TDataFrameEngine)
    var
      p2: PMyDefine;
    begin
      p2 := Param1;
      // 正在等反馈中,断线了,触发该事件
      DoStatus('未收到反馈,异常断线');
      // 由于不会触发成功接收反馈事件,所以需要在这里释放PMyDefine
      dispose(p2);
    end
    );
  disposeObject([SendDe]);
end;
```

正常情况下请避免使用 delphi 的匿名函数方式事件,开一个异步事件的方法,虽然这样做很麻烦,但是程序条理更加清晰和严谨,更易于阅读

```
sendDE := TDataFrameEngine.Create;
sendDE.WriteString(AI_UserKey);
sendDE.WriteString(p^.RequestFile);
sendDE.WriteInt64(0);
sendDE.WriteInt64(p^.EndPos);
Client.SendStreamCmdM('GetFileMD5', sendDE, p, nil, GetMD5Result);
disposeObject(sendDE);
```

GetMD5Result 是一个独立的方法回调,而不是匿名函数

```
procedure TInstallTask.GetMD5Result(Sender: TPeerIO; Param1: Pointer; Param2: TObject; InData, ResultData: TDataFrameEngine); ...
```

By.qq600585

2019-9