```
{ * 外部api支持，百度翻译服务                                                 * }
{ * https://github.com/PassByYou888/CoreCipher                              * }
{ * https://github.com/PassByYou888/ZServer4D                                * }
{ * https://github.com/PassByYou888/zExpression                              * }
{ ************************************************************************* }

program BaiduTranslateService;

{$APPTYPE CONSOLE}

{$R *.res}


uses
  System.SysUtils,
  System.Classes,
  System.Variants,
  PascalStrings, CommunicationFramework, CommunicationFramework_Server_Indy, CommunicationFramework_Server_CrossSocket,
  DoStatusIO, CoreClasses, DataFrameEngine, UnicodeMixedLib, MemoryStream64, JsonDataObjects,
  ZDBLocalManager, ZDBEngine,
  BaiduTranslateAPI in 'BaiduTranslateAPI.pas';

(*
  百度翻译服务器使用delphi xe10.1.2所编写
  如果要在linux下使用，请更换delphi xe10.2.2或以上版本，如果我们在平台下拉项会找不到linux，就新建一个console工程，将代码复制过去即可

  百度翻译的http查询是在线程中干的
  一个客户端假如同时发送1000个查询请求，不会发生1000个线程，而是一个查询完成后，接下来才会查询下一个
  这套服务器有安全机制，限定为500ip同时查询

  注意：这套服务器模型使用了数据库，并且没有DataStoreService这类有热备功能组件(主要是我不想把一个小翻译服务搞太庞大)
  当你使用Ctrl+F2关闭服务器时，相当于断电，ZDB有安全回写机制，稳妥的做法是，先把客户端全部关闭完，2秒以后，再用ctrl+f2

  如果发生数据库损坏，就是不可恢复的，只能直接删除History.ox，重开服务器即可恢复
*)

var
  MiniDB: TZDBLocalManager;

type
  TMyServer = class(TCommunicationFramework_Server_CrossSocket)
  public
    procedure DoClientConnectAfter(Sender: TPeerIO); override;
    procedure DoClientDisconnect(Sender: TPeerIO); override;
  end;

  PDelayReponseSource = ^TDelayReponseSource;

  TDelayReponseSource = record
    serv: TMyServer;
    id: Cardinal;
    sourLan, destLan: TTranslateLanguage;
    s: TPascalString;
    Hash64: THash64;
  end;

procedure TMyServer.DoClientConnectAfter(Sender: TPeerIO);
begin
  DoStatus('id: %d ip:%s connected', [Sender.id, Sender.PeerIP]);
  Sender.UserVariants['LastIP'] := Sender.PeerIP;
  inherited DoClientConnectAfter(Sender);
end;

procedure TMyServer.DoClientDisconnect(Sender: TPeerIO);
begin
  DoStatus('id: %d ip: %s disconnect', [Sender.id, VarToStr(Sender.UserVariants['LastIP'])]);
  inherited DoClientDisconnect(Sender);
end;

procedure cmd_BaiduTranslate(Sender: TPeerIO; InData, OutData: TDataFrameEngine);
var
  sp: PDelayReponseSource;
begin
  // 这里实现的BaiduTranslate命令也是一种高级服务器技术示范
  // 如果有些代码读起来感觉头大，牵扯太大，那是因为你没有关心过ZDB这套数据库引擎
  // 如果不关心zdb也没事，直接跳过zdb环节，参阅BaiduTranslateWithHTTP的使用即可

  // 处于对服务器的安全考虑
```

```
  // 如果在线ip操过500个，又同时发出查询，就返回错误
  // 因为只有ip在线超过500人，同时500人又都在发出翻译请求才会触发这样的条件
  if BaiduTranslateTh > BaiduTranslate_MaxSafeThread then
    begin
      OutData.WriteBool(False);
      exit;
    end;

  // 开启延迟响应模式，ZS延迟的技术体系和作用，请自行在标准演示中了解相关Demo
  Sender.PauseResultSend;

  // 我们创建一个回调的数据结构，用于延迟的安全释放，不会泄漏
  new(sp);
  sp^.serv := TMyServer(Sender.OwnerFramework);
  sp^.id := Sender.id;
  // 发自客户端的翻译数据
  sp^.sourLan := TTranslateLanguage(InData.Reader.ReadByte);                                          // 翻译的源语言
  sp^.destLan := TTranslateLanguage(InData.Reader.ReadByte);                                          // 翻译的目标语言
  sp^.s := umlTrimSpace(umlCharReplace(umlDeleteChar(InData.Reader.ReadString, #13), #10, #32)); // 这里是做预裁剪，将换行全部替换成空格
  sp^.Hash64 := FastHash64PascalString(@sp^.s);                                                       // 高速hash

  // 从cache数据库查询我们的翻译
  // 因为超过200万条翻译，就必须给百度交钱
  MiniDB.QueryDB(
    False,      // 查询结果写入到返回表
    True,       // 查询的返回表是内存表，如果是False就是一个实体的文件表
    True,       // 从最后开始查询
    'History', // 查询的目标数据库名称
    '',         // 返回表的名称，因为我们不输出，这里给空
    True,       // 查询完成时，释放返回表
    0,          // 查询完成时，释放返回表的延迟时间，单位是秒
    0.1,        // 碎片积累时间，当查询有很多反馈时，每积累到这个时间，就触发反馈事件，便于批量操作，在积累时间中，数据都存在于内存
    0,          // 查询执行时间,0是无限
    0,          // 最大的查询条目匹配数量，0是无限
    1,          // 最大的查询结果反馈，我们只查一条我们的翻译cache
      procedure(dPipe: TZDBPipeline; var qState: TQueryState; var Allowed: Boolean)
    var
        p: PDelayReponseSource;
        j: TJsonObject;
        cli: TPeerIO;
    begin
        // 查询过滤器回调
        p := dPipe.UserPointer;

        j := qState.DBEng.GetJson(qState);
        Allowed := (TTranslateLanguage(j.I['sl']) = p^.sourLan) and (TTranslateLanguage(j.I['dl']) = p^.destLan) and
        (p^.Hash64 = j.U['h']) // 我们用hash来提高遍历速度
        and (p^.s.Same(TPascalString(j.s['s'])));

        if Allowed then
        begin
            cli := p^.serv.ClientFromID[p^.id];

            // 在延迟技术体系中，客户端可能发送完请求就断线了
            // 如果断线，cli就是nil
            if cli <> nil then
            begin
                cli.OutDataFrame.WriteBool(True);
                cli.OutDataFrame.WriteString(j.s['d']);
                cli.ContinueResultSend;
            end;

            // 这里退出以后，后台查询会自动结束，因为我们只需要一条反馈
        end;
    end,
    procedure(dPipe: TZDBPipeline)
    var
      p: PDelayReponseSource;
    begin
      p := dPipe.UserPointer;
      // 如果找到了一条反馈，dPipe.QueryResultCounter就会是1，现在我们释放刚才申请的内存
      if dPipe.QueryResultCounter > 0 then
        begin
          dispose(p);
          exit;
        end;
```

```
    // 如果在Cache数据库中没有找到，我们调用百度api，并且将翻译结果保存到cahce数据库
    BaiduTranslateWithHTTP(False, p^.sourLan, p^.destLan, p^.s, p, procedure(UserData: Pointer; Success: Boolean; sour
, dest: TPascalString)
      var
        cli: TPeerIO;
        n: TPascalString;
        js: TJsonObject;
        p: PDelayReponseSource;
      begin
        p := UserData;
        cli := TPeerIO(PDelayReponseSource(UserData)^.serv.ClientFromID[PDelayReponseSource(UserData)^.id]);
        // 在延迟技术体系中，客户端可能发送完请求就断线了
        // 如果断线，cli就是nil
        if cli <> nil then
          begin
            cli.OutDataFrame.WriteBool(Success);
            if Success then
              begin
                cli.OutDataFrame.WriteString(dest);

                // 将查询结果记录到数据库
                // 因为超过200万条翻译，就必须给百度交钱
                js := TJsonObject.Create;
                js.I['sl'] := Integer(p^.sourLan);
                js.I['dl'] := Integer(p^.destLan);
                js.U['h'] := FastHash64PascalString(@p^.s);
                js.F['t'] := Now;
                js.s['ip'] := cli.PeerIP;
                js.s['s'] := p^.s.Text;
                js.s['d'] := dest.Text;

                MiniDB.PostData('History', js);

                // 在ubuntu服务器模式下，无法显示中文
                {$IFNDEF Linux}
                DoStatus(js.ToString);
                {$IFEND}
                disposeObject(js);
              end;

            // 继续响应
            cli.ContinueResultSend;
          end;
        dispose(p);
      end);
  end).UserPointer := sp;
end;

var
  server_1, server_2: TMyServer;

begin
  MiniDB := TZDBLocalManager.Create;
  // 因为创建文件形式的数据库，对于这种经常ctrl+f2的强退，数据库很容易损坏
  MiniDB.InitDB('History');

  server_1 := TMyServer.Create;

  // 如果在Ubuntu下使用indy服务器，这里必须指定绑定的回环地址
  // if server_IPv4.StartService('0.0.0.0', 59813) then

  // 新版本的CrossSocket已经修复了在Ubuntu下ipv4+ipv6同时侦听一个端口问题
  // 我么使用空字符同时侦听ipv4+ipv6的59813
  if server_1.StartService('', 59813) then
      DoStatus('start service with ipv4:59813 success')
  else
      DoStatus('start service with ipv4:59813 failed!');

  // 但是，我们仍然可以同时开启多个服务，同时侦听ipv6,ipv4和各个不同端口，然后再将指令触发点指向同一个地方
  // 这样干可以用于任何一个外部服务器接口，DIOCP,Cross,Indy,ICS等等服务器接口均能以这种方式实现多侦听的集中式服务

  // 如果在linux出现ipv6侦听错误，要么自己装ipv6服务和模块，要么就无视它
  server_2 := TMyServer.Create;
  if server_2.StartService('::', 59814) then
      DoStatus('start service with ipv6:59813 success')
  else
      DoStatus('start service with ipv6:59813 failed!');

  server_1.RegisterStream('BaiduTranslate').OnExecuteCall := cmd_BaiduTranslate;
```

```
  server_2.RegisterStream('BaiduTranslate').OnExecuteCall := cmd_BaiduTranslate;

  // 15秒空闲时断开链接
  server_1.IdleTimeout := 15000;
  server_2.IdleTimeout := 15000;

  server_1.QuietMode := True;
  server_2.QuietMode := True;

  while True do
    begin
      MiniDB.Progress;
      server_1.ProgressBackground;
      server_2.ProgressBackground;

      // 绿色环保，避免多余开销
      if server_1.Count + server_2.Count > 0 then
          System.Classes.CheckSynchronize(10)
      else
        begin
          System.Classes.CheckSynchronize(100);
        end;
    end;

end.
```