企业数字化转型的 delphi 中间件

目录

数据库连接池

firedac 和 unidac 是 delphi 的二大数据库引擎。firedac 是 delphi 自带的，unidac 是三方的商业控件。unidac 可以直连数据库（不需要安装数据库客户端），并且支持 delphi 所有版本。在此，我们不争论二者的长短，而是二者皆可使用，想用哪个都可以，通过编译开关来区分使用。

```
{$IFDEF unidac}
db.unidac, db.unidacpool,
{$ENDIF }
{$IFDEF firedac}
db.firedac, db.firedacpool,
```

```
  {$ENDIF }
```

配置数据库连接参数

database.conf

```
;数据库连接参数，支持多帐套，增加 ini 节，即是增加一个数据库帐套，可以复制现有的
;no--帐套号，name--帐套名称，type--数据库类型，user--用户名，pwd--密码，port--默认 0，ip--数据库 ip，db--数据库名称
;type 数据库类型：  MSSQL,MySQL,Ora,PG
[database1]
no=1
name=cs
type=MSSQL
user=sa
pwd=sql
ip=127.0.0.1
port=8829
db=yndb
library=
```

配置池参数

Server.conf

```
;数据库池初始化建立的连接数量
dbpoolsize=10
```

创建连接池

在程序初始化的时候，创建连接池。

```
TDBPool.start;
```

连接池使用

```
procedure select(req, res: TSerialize);
var
  db: tdb; //数据模块
  pool: tdbpool; //连接池
  i: Integer;
begin
  try
    try
      pool := GetDBPool(req.asStr['dbid']); //参数 dbid--帐套号
      db := pool.Lock;//从池中获取可用连接
      db.dsp.DataSet := db.qry;
      for i := 1 to req.asByte['cnt'] do
      begin
        db.qry.Close;
        db.qry.SQL.Clear;
```

```
            db.qry.SQL.Text := req.asStr['sql' + i.tostring];
            res.AsVariant['ds' + i.ToString] := db.dsp.Data;
        end;
        res.asInt['status'] := 200;
        res.asStr['message'] := 'success';
    except
        on E: Exception do
        begin
            res.asInt['status'] := 500;
            res.asStr['message'] := 'fail';
            res.asStr['exception'] := 'bin.func.select()' + e.Message
        end;
    end;
  finally
    pool.Unlock(db);//连接归还池中
  end;
end;
```

网络通讯

delphi 免费开源的网络通讯库有很多种，比较常用的有：cross socket、mormot、diocp、brook、rtc。

cross socket 支持 windows\linux\macos。在 windows 下面使用 iocp；在 linux 下面 epoll；在 macos 下面使用 equeue。支持 TCP\HTTP\WEBSOCKET。

mormot 封装了 windows 的 http.sys。支持 HTTP\WEBSOCKET。

diocp 封装了 windows 的 iocp。支持 TCP\HTTP\WEBSOCKET。

Brook 是 c 的开源项目，delphi 通过动态库来使用它，支持 windows\linux。

rtc 是异步通讯库，支持数千连接，支持 windows\linux。

可以通过开关来区分使用哪一种。

```
  var ini: TIniFile := TIniFile.Create(SvrCfg);
  SOCKET := ini.ReadString('config', 'socket', 'crosssocket');
  ini.free;
  TDBPool.start;
  if SameText('crosssocket', SOCKET) then      //cross socket http
    TcrossHttpSvr.Create
  else if SameText('mormot', SOCKET) then      //mormot http
    THttpsys.Create
  else if SameText('diocp', SOCKET) then      //diocp http
    TiocpHttpServer.Create
  else if SameText('brook', SOCKET) then    //brook http
    TbrookServer.Create
  else if SameText('rtc', SOCKET) then //rtc http
    TrtcServer.Create;
```

http 路由

Sock.router.pas

可以分别路由 REST API（包括 json 和 protobuf 二种数据序列）和传统 delphi 二进制 API。

```
type      //remote function
   TFuncBin = procedure(req, res: tserialize);      //binary api, only surport delphi

   TFuncJson = function(url: string; body: tbytes): string;    //rest api, surport cross language and platform

   TFuncPB = function(url: string; body: tbytes): tbytes;      //protobuf api, surport cross language and platform

procedure router(const Ctxt: TDiocpHttpRequest); overload;      //diocp router

procedure router(const Ctxt: THttpServerRequest); overload; //mormot router

procedure router(const Ctxt: ICrossHttpRequest; const Ctxt2: ICrossHttpResponse); overload; //cross socket router

procedure router(const Ctxt: TBrookHTTPRequest; const Ctxt2: TBrookHTTPResponse); overload; //brook router

procedure router(const Ctxt: TRtcConnection); overload; //rtc router

procedure FuncBin(const Ctxt: THttpServerRequest; func: TFuncBin); overload;      //mormot binary remote function

procedure FuncBin(const Ctxt: ICrossHttpRequest; const Ctxt2: ICrossHttpResponse; func: TFuncBin); overload; //cross socket binary remote function

procedure funcBin(const Ctxt: TDiocpHttpRequest; func: TFuncBin); overload;    //diocp binary remote function

procedure funcBin(const Ctxt: TBrookHTTPRequest; const Ctxt2: TBrookHTTPResponse; func: TFuncBin); overload; //brook binary remote function

procedure funcBin(const Ctxt: TRtcConnection; func: TFuncBin); overload; //rtc binary remote function

procedure funcJson(const Ctxt: THttpServerRequest; func: TFuncJson); overload; //mormot Json remote function

procedure funcJson(const Ctxt: ICrossHttpRequest; const Ctxt2: ICrossHttpResponse; func: TFuncJson); overload;    //cross socket Json remote function
```

```
procedure funcJson(const Ctxt: TDiocpHttpRequest; func: TFuncJson); overload;    //diocp Json remote function

procedure funcJson(const Ctxt: TBrookHTTPRequest; const Ctxt2: TBrookHTTPResponse; func: TFuncJson); overload; //brook Json remote function

procedure funcJson(const Ctxt: TRtcConnection; func: TFuncJson); overload; //rtc Json remote function

procedure funcPB(const Ctxt: ICrossHttpRequest; const Ctxt2: ICrossHttpResponse; func: TFuncPB); overload; //cross socket protobuf remote function

procedure funcPB(const Ctxt: THttpServerRequest; func: TFuncPB); overload; //mormot protobuf remote function

procedure funcPB(const Ctxt: TBrookHTTPRequest; const Ctxt2: TBrookHTTPResponse; func: TFuncPB); overload; //brook protobuf remote function

procedure funcPB(const Ctxt: TDiocpHttpRequest; func: TFuncPB); overload; //diocp protobuf remote function

procedure funcPB(const Ctxt: TRtcConnection; func: TFuncPB); overload; //rtc protobuf remote function

procedure setBINHeader(const Ctxt: TDiocpHttpRequest); overload;       //diocp binary http header

procedure setBINHeader(const Ctxt: ICrossHttpResponse); overload; //cross socket binary http header

procedure setBINHeader(const Ctxt: THttpServerRequest); overload; //mormot binary http header

procedure setBINHeader(const Ctxt: TBrookHTTPResponse); overload; //brook binary http header

procedure setBINHeader(const Ctxt: TRtcConnection); overload; //rtc binary http header

procedure setJsonHeader(const Ctxt: TDiocpHttpRequest); overload; //diocp rest http header

procedure setJsonHeader(const Ctxt: THttpServerRequest); overload;    //mormot rest http header
```

```
procedure setJsonHeader(const Ctxt: ICrossHttpResponse); overload; //cross socket rest http header

procedure setJsonHeader(const Ctxt: TBrookHTTPResponse); overload; //brook rest http header

procedure setJsonHeader(const Ctxt: TRtcConnection); overload; //rtc rest http header
```

rtc 路由

```
procedure router(const Ctxt: TRtcConnection); overload; //rtc router
//Ctxt.Request.FileName
  procedure call(method, url: string; func: TFuncJson); overload;
  begin
    if SameText(Ctxt.Request.Method, method) and (Pos(url, Ctxt.Request.FileName) > 0) then
      funcJson(Ctxt, func);
  end;

  procedure call(method, url: string; func: TFuncPB); overload;
  begin
    if SameText(method, Ctxt.Request.Method) and (Pos(url, Ctxt.Request.FileName) > 0) then
      funcPB(Ctxt, func);
  end;

  procedure call(url: string; func: TFuncBin); overload;
  begin
    if   (Pos(url, Ctxt.Request.FileName) > 0) then
      FuncBin(Ctxt, func);
  end;

begin
  call('/bin/select', bin.func.select);
  call('/bin/save', bin.func.save);
  call('/bin/spopen', bin.func.spopen);
  call('/bin/download', bin.func.download);
  call('/bin/upload', bin.func.upload);
  call('/bin/execsql', bin.func.execsql);
  call('/bin/verifycode', bin.func.verifycode);
  call('/bin/snowflakeid', bin.func.snowflakeid);
  call('get', '/rest/tunit', rest.tunit.select);
  call('put', '/rest/tunit', rest.tunit.update);
  call('post', '/rest/tunit', rest.tunit.insert);
  call('delete', '/rest/tunit', rest.tunit.delete);
  call('get', '/protobuf/tunit', proto.tunit.select);
  call('put', '/protobuf/tunit', proto.tunit.update);
  call('post', '/protobuf/tunit', proto.tunit.insert);
```

```
  call('delete', '/protobuf/tunit', proto.tunit.delete);
end;
```

brook 路由

```
procedure router(const Ctxt: TBrookHTTPRequest; const Ctxt2: TBrookHTTPResponse); overload;
//brook router
//Ctxt.path
  procedure call(method, url: string; func: TFuncJson); overload;
  begin
    if (Pos(url, Ctxt.path) > 0) and SameText(method, Ctxt.Method) then
      funcJson(Ctxt, Ctxt2, func);
  end;

  procedure call(method, url: string; func: TFuncPB); overload;
  begin
    if (Pos(url, Ctxt.path) > 0) and SameText(method, Ctxt.Method) then
      funcPB(Ctxt, Ctxt2, func);
  end;

  procedure call(url: string; func: TFuncBin); overload;
  begin
    if Pos(url, Ctxt.path) > 0 then
      FuncBin(Ctxt, Ctxt2, func);
  end;
begin
  call('/bin/select', bin.func.select);
  call('/bin/save', bin.func.save);
  call('/bin/spopen', bin.func.spopen);
  call('/bin/download', bin.func.download);
  call('/bin/upload', bin.func.upload);
  call('/bin/execsql', bin.func.execsql);
  call('/bin/verifycode', bin.func.verifycode);
  call('/bin/snowflakeid', bin.func.snowflakeid);
  call('get', '/rest/tunit', rest.tunit.select);
  call('put', '/rest/tunit', rest.tunit.update);
  call('post', '/rest/tunit', rest.tunit.insert);
  call('delete', '/rest/tunit', rest.tunit.delete);
  call('get', '/protobuf/tunit', proto.tunit.select);
  call('put', '/protobuf/tunit', proto.tunit.update);
  call('post', '/protobuf/tunit', proto.tunit.insert);
  call('delete', '/protobuf/tunit', proto.tunit.delete);
end;
```

diocp 路由

```
procedure router(const Ctxt: TDiocpHttpRequest);
//Ctxt.RequestURL
  procedure call(method, url: string; func: TFuncJson); overload;
  begin
    if (Pos(url, Ctxt.RequestURL) > 0) and SameText(method, Ctxt.RequestMethod) then
      funcJson(Ctxt, func);
  end;

  procedure call(method, url: string; func: TFuncPB); overload;
  begin
    if (Pos(url, Ctxt.RequestURL) > 0) and SameText(method, Ctxt.RequestMethod) then
      funcPB(Ctxt, func);
  end;

  procedure call(url: string; func: TFuncBin); overload;
  begin
    if Pos(url, Ctxt.RequestURL) > 0 then
      FuncBin(Ctxt, func);
  end;
begin
  call('/bin/select', bin.func.select);
  call('/bin/save', bin.func.save);
  call('/bin/spopen', bin.func.spopen);
  call('/bin/download', bin.func.download);
  call('/bin/upload', bin.func.upload);
  call('/bin/execsql', bin.func.execsql);
  call('/bin/verifycode', bin.func.verifycode);
  call('/bin/snowflakeid', bin.func.snowflakeid);
  call('get', '/rest/tunit', rest.tunit.select);
  call('put', '/rest/tunit', rest.tunit.update);
  call('post', '/rest/tunit', rest.tunit.insert);
  call('delete', '/rest/tunit', rest.tunit.delete);
  call('get', '/protobuf/tunit', proto.tunit.select);
  call('put', '/protobuf/tunit', proto.tunit.update);
  call('post', '/protobuf/tunit', proto.tunit.insert);
  call('delete', '/protobuf/tunit', proto.tunit.delete);
end;
```

mormot 路由

```
procedure router(const Ctxt: THttpServerRequest);
//Ctxt.URL
  procedure call(method, url: string; func: TFuncJson); overload;
  begin
    if (Pos(url, Ctxt.URL) > 0) and SameText(method, Ctxt.Method) then
```

```
        funcJson(Ctxt, func);
    end;

    procedure call(method, url: string; func: TFuncPB); overload;
    begin
      if (Pos(url, Ctxt.URL) > 0) and SameText(method, Ctxt.Method) then
        funcPB(Ctxt, func);
    end;

    procedure call(url: string; func: TFuncBin); overload;
    begin
      if Pos(url, Ctxt.URL) > 0 then
        FuncBin(Ctxt, func);
    end;
begin
    call('/bin/select', bin.func.select);
    call('/bin/save', bin.func.save);
    call('/bin/spopen', bin.func.spopen);
    call('/bin/download', bin.func.download);
    call('/bin/upload', bin.func.upload);
    call('/bin/execsql', bin.func.execsql);
    call('/bin/verifycode', bin.func.verifycode);
    call('/bin/snowflakeid', bin.func.snowflakeid);
    call('get', '/rest/tunit', rest.tunit.select);
    call('put', '/rest/tunit', rest.tunit.update);
    call('post', '/rest/tunit', rest.tunit.insert);
    call('delete', '/rest/tunit', rest.tunit.delete);
    call('get', '/protobuf/tunit', proto.tunit.select);
    call('put', '/protobuf/tunit', proto.tunit.update);
    call('post', '/protobuf/tunit', proto.tunit.insert);
    call('delete', '/protobuf/tunit', proto.tunit.delete);
end;
```

Cross socket 路由

```
procedure router(const Ctxt: ICrossHttpRequest; const Ctxt2: ICrossHttpResponse);
//Ctxt.RawPathAndParams
    procedure call(method, url: string; func: TFuncJson); overload;
    begin
      if (Pos(url, Ctxt.RawPathAndParams) > 0) and SameText(method, Ctxt.Method) then
        funcJson(Ctxt, Ctxt2, func);
    end;

    procedure call(method, url: string; func: TFuncPB); overload;
    begin
```

```
      if (Pos(url, Ctxt.RawPathAndParams) > 0) and SameText(method, Ctxt.Method) then
         funcPB(Ctxt, Ctxt2, func);
   end;

   procedure call(url: string; func: TFuncBin); overload;
   begin
      if Pos(url, Ctxt.RawPathAndParams) > 0 then
         FuncBin(Ctxt, Ctxt2, func);
   end;
begin
   call('/bin/select', bin.func.select);
   call('/bin/save', bin.func.save);
   call('/bin/spopen', bin.func.spopen);
   call('/bin/download', bin.func.download);
   call('/bin/upload', bin.func.upload);
   call('/bin/execsql', bin.func.execsql);
   call('/bin/verifycode', bin.func.verifycode);
   call('/bin/snowflakeid', bin.func.snowflakeid);
   call('get', '/rest/tunit', rest.tunit.select);
   call('put', '/rest/tunit', rest.tunit.update);
   call('post', '/rest/tunit', rest.tunit.insert);
   call('delete', '/rest/tunit', rest.tunit.delete);
   call('get', '/protobuf/tunit', proto.tunit.select);
   call('put', '/protobuf/tunit', proto.tunit.update);
   call('post', '/protobuf/tunit', proto.tunit.insert);
   call('delete', '/protobuf/tunit', proto.tunit.delete);
end;
```

远程方法

传统二进制一

使用 olevariant 序列，TDataSetProvider+TClientDataSet 传统组合。

```
unit bin.func;
/// <author>cxg 2022-7-27</author>
/// This unit surport firedac and unidac
{$I def.inc}

interface

uses
   {$IFDEF firedac}    db.firedac, db.firedacPool, {$ENDIF}
   {$IFDEF unidac}db.unidac, db.unidacpool,    {$ENDIF}
   global,    verifyCode, snowflake,
   classes, keyValue.serialize, yn.log, SysUtils;
```

```
procedure select(req, res: TSerialize);

procedure save(req, res: TSerialize);

procedure execSQL(req, res: TSerialize);

procedure spOpen(req, res: TSerialize);

procedure download(req, res: TSerialize);

procedure upload(req, res: TSerialize);

procedure verifyCode(req, res: TSerialize);

procedure snowflakeID(req, res: TSerialize);

implementation

procedure snowflakeID(req, res: TSerialize);
var
  sf: tsnowflakeID;
begin
  sf := tsnowflakeID.Create;
  try
    try
      sf.DatacenterId := req.asByte['datacenterid'];
      sf.WorkerID := req.asByte['workerid'];
      res.asInt['status'] := 200;
      res.asStr['message'] := 'success';
      res.asInt64['snowflakeid'] := sf.NextId;
    except
      on E: Exception do
      begin
        res.asInt['status'] := 500;
        res.asStr['message'] := 'fail';
        res.asStr['exception'] := 'bin.func.snowflakeID()' + e.Message;
        WriteLog('bin.func.snowflakeID()' + e.Message);
      end;
    end;
  finally
    sf.Free;
  end;
end;
```

```
procedure verifyCode(req, res: TSerialize);
var vc: TGenerateVerifyCode;
  ms: tstream;
  code: string;
begin
  vc := TGenerateVerifyCode.Create;
  ms := TMemoryStream.Create;
  try
    try
      vc.GetVerifyCodeAndImage(ms, code);
      res.asInt['status'] := 200;
      res.asStr['message'] := 'success';
      res.asStr['code'] := code;
      res.asStream['image'] := ms;
    except
      on E: Exception do
      begin
        res.asInt['status'] := 500;
        res.asStr['message'] := 'fail';
        res.asStr['exception'] := 'bin.func.verifyCode()' + e.Message;
        WriteLog('bin.func.verifyCode()' + e.Message);
      end;
    end;
  finally
    vc.Free;
    ms.Free;
  end;
end;

procedure upload(req, res: TSerialize);
var
  i: integer;
  filename: string;
  ms: tstream;
begin
  try
    try
      for i := 1 to req.asByte['cnt'] do
      begin
        filename := req.asStr['filename' + i.ToString];
        if Assigned(ms) then
          TMemoryStream(ms).Clear;
        ms := req.asStream['file' + i.ToString];
```

```
            ForceDirectories(uploadpath);
            TMemoryStream(ms).SaveToFile(uploadpath + PathDelim + filename);
        end;
        res.asInt['status'] := 200;
        res.asStr['message'] := 'success';
    except
        on E: Exception do
        begin
            res.asInt['status'] := 500;
            res.asStr['message'] := 'fail';
            res.asStr['exception'] := 'bin.func.upload()' + e.Message;
            WriteLog('bin.func.upload()' + e.Message);
        end;
    end;
  finally
    ms.Free;
  end;
end;

procedure download(req, res: TSerialize);
var
  i: integer;
  filename: string;
  ms: tstream;
begin
  ms := TMemoryStream.Create;
  try
    try
        for i := 1 to req.asByte['cnt'] do
        begin
            filename := req.asStr['filename' + i.ToString];
            if Assigned(ms) then
                TMemoryStream(ms).Clear;
            TMemoryStream(ms).LoadFromFile(downloadPath + PathDelim + filename);
            res.asStr['filename' + i.ToString] := filename;
            res.asStream['file' + i.ToString] := ms;
        end;
        res.asInt['status'] := 200;
        res.asStr['message'] := 'success';
    except
        on E: Exception do
        begin
            res.asInt['status'] := 500;
            res.asStr['message'] := 'fail';
```

```
            res.asStr['exception'] := 'bin.func.download()' + e.Message;
            WriteLog('bin.func.download()' + e.Message);
        end;
    end;
  finally
    ms.Free;
  end;
end;

procedure spOpen(req, res: TSerialize);
var
  db: tdb;
  pool: tdbpool;
  params: string;
  list: TStringList;
  i: integer;
begin
  list := TStringList.Create;
  try
    try
        pool := getdbpool(req.asStr['dbid']);
        db := pool.Lock;
        db.sp.Close;
        db.dsp.DataSet := db.sp;
        {$IFDEF ado}
        db.sp.Prepared := False;
        db.sp.Parameters.Clear;
        db.sp.ProcedureName := req.asstr['spname'];
        db.sp.Prepared := True;
        {$ELSE}
        db.sp.UnPrepare;
        db.sp.Params.Clear;
        db.sp.StoredProcName := req.asstr['spname'];
        db.sp.Prepare;
        {$ENDIF}
        params := req.asstr['params'];
        if params > '' then //  不是所有的存储过程都有参数
        begin
            list.Delimiter := ';';
            list.DelimitedText := params;
            for i := 0 to list.Count - 1 do
                {$IFDEF ado}
                db.sp.Parameters.FindParam(list.Names[i]).Value := list.Values[list.Names[i]];
                {$ELSE}
```

```
                db.sp.FindParam(list.Names[i]).AsString := list.Values[list.Names[i]];
                {$ENDIF}
            end;
            res.AsVariant['ds'] := db.dsp.Data;
            res.asInt['status'] := 200;
            res.asStr['message'] := 'success';
        except
            on E: Exception do
            begin
                res.asInt['status'] := 500;
                res.asStr['message'] := 'fail';
                res.asStr['exception'] := 'bin.func.SPOPEN()' + E.Message;
                writelog('bin.func.SPOPEN()' + E.Message);
            end;
        end;
    finally
        pool.Unlock(db);
        list.Free;
    end;
end;

procedure execSQL(req, res: TSerialize);
var
    db: tdb;
    pool: tdbpool;
begin
    try
        try
            pool := GetDBPool(req.asStr['dbid']);
            db := pool.Lock;
            db.startTrans;
            db.qry.Close;
            db.qry.SQL.Clear;
            db.qry.SQL.Text := req.asStr['sql'];
            db.qry.ExecSQL;
            db.commitTrans;
            res.asInt['status'] := 200;
            res.asStr['message'] := 'success';
        except
            on E: Exception do
            begin
                db.rollbackTrans;
                res.asInt['status'] := 500;
                res.asStr['message'] := 'fail';
```

```
            res.asStr['exception'] :=    'bin.func.execSQL()' + e.Message;
            WriteLog('bin.func.execSQL()' + e.Message);
        end;
    end;
  finally
    pool.Unlock(db);
  end;
end;

procedure save(req, res: TSerialize);
var
  db: tdb;
  pool: tdbpool;
  i, errCnt: Integer;
begin
  try
    try
        pool := GetDBPool(req.asStr['dbid']);
        db := pool.Lock;
        db.startTrans;
        db.dsp.DataSet := db.qry;
        for i := 1 to req.asByte['cnt'] do
        begin
            db.qry.Close;
            db.qry.SQL.Clear;
            db.qry.SQL.Text := 'select ' + db.getSaveFieldNames(req.asStr['tablename' + i.ToString],
                req.asStr['nosavefields' + i.ToString]) + ' from ' + req.asStr['tablename' + i.ToString] + '
where 1=2';
            db.dsp.ApplyUpdates(req.AsVariant['delta' + i.ToString], 0, errCnt);
            if errCnt > 0 then
            begin
                res.asInt['status'] := 500;
                res.asStr['err'] := 'bin.func.save() fail';
                db.rollbackTrans;
                Exit;
            end;
        end;
        db.commitTrans;
        res.asInt['status'] := 200;
        res.asStr['message'] := 'success';
    except
        on E: Exception do
        begin
            db.rollbackTrans;
```

```
            res.asInt['status'] := 500;
            res.asStr['message'] := 'fail';
            res.asStr['exception'] := 'bin.func.save()' + e.Message
        end;
    end;
  finally
    pool.Unlock(db);
  end;
end;

procedure select(req, res: TSerialize);
var
  db: tdb;
  pool: tdbpool;
  i: Integer;
begin
  try
    try
        pool := GetDBPool(req.asStr['dbid']);
        db := pool.Lock;
        db.dsp.DataSet := db.qry;
        for i := 1 to req.asByte['cnt'] do
        begin
            db.qry.Close;
            db.qry.SQL.Clear;
            db.qry.SQL.Text := req.asStr['sql' + i.tostring];
            res.AsVariant['ds' + i.ToString] := db.dsp.Data;
        end;
        res.asInt['status'] := 200;
        res.asStr['message'] := 'success';
    except
        on E: Exception do
        begin
            res.asInt['status'] := 500;
            res.asStr['message'] := 'fail';
            res.asStr['exception'] := 'bin.func.select()' + e.Message
        end;
    end;
  finally
    pool.Unlock(db);
  end;
end;

end.
```

传统二进制二

使用 firedac 的 TFDQuery+TFDMemTable 组合。

```
unit bin.func2;
/// <author>cxg 2022-7-27</author>
/// This unit only surport firedac
{$I def.inc}

interface

uses
  {$IFDEF firedac}       db.firedac, db.firedacPool, FireDAC.Stan.StorageBin,
FireDAC.Stan.Intf, {$ENDIF}
  global, verifyCode, snowflake, classes, keyValue.serialize, yn.log, SysUtils;

procedure select(req, res: TSerialize);

procedure save(req, res: TSerialize);

procedure execSQL(req, res: TSerialize);

procedure spOpen(req, res: TSerialize);

procedure download(req, res: TSerialize);

procedure upload(req, res: TSerialize);

procedure verifyCode(req, res: TSerialize);

procedure snowflakeID(req, res: TSerialize);

implementation

procedure snowflakeID(req, res: TSerialize);
var
  sf: tsnowflakeID;
begin
  sf := tsnowflakeID.Create;
  try
    try
      sf.DatacenterId := req.asByte['datacenterid'];
      sf.WorkerID := req.asByte['workerid'];
      res.AsInt['status'] := 200;
      res.AsStr['message'] := 'success';
```

```
        res.asInt64['snowflakeid'] := sf.NextId;
      except
        on E: Exception do
        begin
          res.AsInt['status'] := 500;
          res.AsStr['message'] := 'fail';
          res.asStr['exception'] := 'bin.func.snowflakeID()' + e.Message;
          WriteLog('bin.func.snowflakeID()' + e.Message);
        end;
      end;
    finally
      sf.Free;
    end;
end;

procedure verifyCode(req, res: TSerialize);
var
  vc: TGenerateVerifyCode;
  ms: tstream;
  code: string;
begin
  vc := TGenerateVerifyCode.Create;
  ms := TMemoryStream.Create;
  try
    try
      vc.GetVerifyCodeAndImage(ms, code);
      res.AsInt['status'] := 200;
      res.AsStr['message'] := 'success';
      res.asStr['code'] := code;
      res.asStream['image'] := ms;
    except
      on E: Exception do
      begin
        res.AsInt['status'] := 500;
        res.AsStr['message'] := 'fail';
        res.asStr['exception'] := 'bin.func.verifyCode()' + e.Message;
        WriteLog('bin.func.verifyCode()' + e.Message);
      end;
    end;
  finally
    vc.Free;
    ms.Free;
  end;
end;
```

```
procedure upload(req, res: TSerialize);
var
  i: integer;
  filename: string;
  ms: tstream;
begin
  try
    try
      for i := 1 to req.asByte['cnt'] do
      begin
        filename := req.asStr['filename' + i.ToString];
        if Assigned(ms) then
          TMemoryStream(ms).Clear;
        ms := req.asStream['file' + i.ToString];
        ForceDirectories(uploadpath);
        TMemoryStream(ms).SaveToFile(uploadpath + PathDelim + filename);
      end;
      res.AsInt['status'] := 200;
      res.AsStr['message'] := 'success';
    except
      on E: Exception do
      begin
        res.AsInt['status'] := 500;
        res.AsStr['message'] := 'fail';
        res.asStr['exception'] := 'bin.func.upload()' + e.Message;
        WriteLog('bin.func.upload()' + e.Message);
      end;
    end;
  finally
    ms.Free;
  end;
end;

procedure download(req, res: TSerialize);
var
  i: integer;
  filename: string;
  ms: tstream;
begin
  ms := TMemoryStream.Create;
  try
    try
      for i := 1 to req.asByte['cnt'] do
```

```
        begin
          filename := req.asStr['filename' + i.ToString];
          if Assigned(ms) then
            TMemoryStream(ms).Clear;
          TMemoryStream(ms).LoadFromFile(downloadPath + PathDelim + filename);
          res.asStr['filename' + i.ToString] := filename;
          res.asStream['file' + i.ToString] := ms;
        end;
        res.AsInt['status'] := 200;
        res.AsStr['message'] := 'success';
      except
        on E: Exception do
        begin
          res.AsInt['status'] := 500;
          res.AsStr['message'] := 'fail';
          res.asStr['exception'] := 'bin.func.download()' + e.Message;
          WriteLog('bin.func.download()' + e.Message);
        end;
      end;
    finally
      ms.Free;
    end;
end;

procedure spOpen(req, res: TSerialize);
var
  db: tdb;
  pool: tdbpool;
  params: string;
  list: TStringList;
  i: integer;
begin
  list := TStringList.Create;
  try
    try
      pool := getdbpool(req.asStr['dbid']);
      db := pool.Lock;
      db.sp.Close;
      db.dsp.DataSet := db.sp;
      db.sp.UnPrepare;
      db.sp.Params.Clear;
      db.sp.StoredProcName := req.asstr['spname'];
      db.sp.Prepare;
      params := req.asstr['params'];
```

```
        if params > '' then //  不是所有的存储过程都有参数
        begin
          list.Delimiter := ';';
          list.DelimitedText := params;
          for i := 0 to list.Count - 1 do
            db.sp.FindParam(list.Names[i]).AsString := list.Values[list.Names[i]];
        end;
        res.AsVariant['ds'] := db.dsp.Data;
        res.AsInt['status'] := 200;
        res.AsStr['message'] := 'success';
      except
        on E: Exception do
        begin
          res.AsInt['status'] := 500;
          res.AsStr['message'] := 'fail';
          res.asStr['exception'] := 'bin.func.SPOPEN()' + E.Message;
          writelog('bin.func.SPOPEN()' + E.Message);
        end;
      end;
    finally
      pool.Unlock(db);
      list.Free;
    end;
end;

procedure execSQL(req, res: TSerialize);
var
  db: tdb;
  pool: tdbpool;
begin
  try
    try
      pool := GetDBPool(req.asStr['dbid']);
      db := pool.Lock;
      db.startTrans;
      db.qry.Close;
      db.qry.SQL.Clear;
      db.qry.SQL.Text := req.asStr['sql'];
      db.qry.ExecSQL;
      db.commitTrans;
      res.AsInt['status'] := 200;
      res.AsStr['message'] := 'success';
    except
      on E: Exception do
```

```
        begin
          db.rollbackTrans;
          res.AsInt['status'] := 500;
          res.AsStr['message'] := 'fail';
          res.asStr['exception'] := 'bin.func.execSQL()' + e.Message;
          WriteLog('bin.func.execSQL()' + e.Message);
        end;
      end;
    finally
      pool.Unlock(db);
    end;
end;

procedure save(req, res: TSerialize);
var
  db: tdb;
  pool: tdbpool;
  i, errCnt: Integer;
begin
  try
    try
      pool := GetDBPool(req.asStr['dbid']);
      db := pool.Lock;
      db.startTrans;
      for i := 1 to req.asByte['cnt'] do
      begin
        db.qry.Close;
        db.qry.CachedUpdates := True;
       // db.qry.UpdateOptions.UpdateTableName := req.asStr['tablename' + i.ToString];
       // db.qry.UpdateOptions.KeyFields := '';
        db.qry.SQL.Clear;
        db.qry.SQL.Text := 'select ' + db.getSaveFieldNames(req.asStr['tablename' + i.ToString],
req.asStr[req.asStr['nosavefields' + i.ToString]]) + ' from ' + req.asStr['tablename' + i.ToString] + '
where 1=2';
        db.qry.LoadFromStream(req.asStream['delta' + i.ToString], sfBinary);
        errCnt := db.qry.ApplyUpdates;
        if errCnt > 0 then
        begin
          res.AsInt['status'] := 500;
          res.AsStr['message'] := 'fail';
          res.asStr['exception'] := 'bin.func.save() fail';
          db.rollbackTrans;
          Exit;
        end;
```

```
        end;
        db.commitTrans;
        res.AsInt['status'] := 200;
        res.AsStr['message'] := 'success';
    except
        on E: Exception do
        begin
            db.rollbackTrans;
            res.AsInt['status'] := 500;
            res.AsStr['message'] := 'fail';
            res.asStr['exception'] := 'bin.func2.save()' + e.Message
        end;
    end;
  finally
    db.qry.CachedUpdates := False;
    pool.Unlock(db);
  end;
end;

procedure select(req, res: TSerialize);
var
  db: tdb;
  pool: tdbpool;
  i: Integer;
  ms: tstream;
begin
  ms := TMemoryStream.Create;
  try
    try
        pool := GetDBPool(req.asStr['dbid']);
        db := pool.Lock;
        for i := 1 to req.asByte['cnt'] do
        begin
            db.qry.Close;
            db.qry.SQL.Clear;
            db.qry.SQL.Text := req.asStr['sql' + i.tostring];
            db.qry.Open;
            TMemoryStream(ms).Clear;
            db.qry.SaveToStream(ms, sfBinary);
            res.asStream['ds' + i.ToString] := ms;
        end;
        res.AsInt['status'] := 200;
        res.AsStr['message'] := 'success';
    except
```

```
        on E: Exception do
        begin
            res.AsInt['status'] := 500;
            res.AsStr['message'] := 'fail';
            res.asStr['exception'] := 'bin.func2.select()' + e.Message
        end;
    end;
  finally
    pool.Unlock(db);
    ms.Free;
  end;
end;

end.
```

Restful json

```
unit rest.tunit;
//代码由代码工厂自动生成
//2022-09-01 07:17:31
{$I def.inc}
interface

uses
  {$IFDEF firedac}    db.firedac, db.firedacPool, {$ENDIF}
  {$IFDEF unidac}db.unidac, db.unidacpool,    {$ENDIF}
  classes, db, System.NetEncoding, serialize,
  system.JSON.Serializers, yn.log, SysUtils;

type
  Ttunit = record
    [Serialize(1)] unitid: string;
    [Serialize(2)] unitname: string;
  end;

  TtunitArray = record
    [Serialize(1)] status: integer;
    [Serialize(2)] exception: string;
    [Serialize(3)] message: string;
    [Serialize(4)] tunits: TArray<Ttunit>;
  end;

  TRes = record
    [Serialize(1)] status: integer;
    [Serialize(2)] exception: string;
```

```
      [Serialize(3)] message: string;
   end;

function select(url: string; body: TBytes): string;

function insert(url: string; body: TBytes): string;

function update(url: string; body: TBytes): string;

function delete(url: string; body: TBytes): string;

implementation

function select(url: string; body: TBytes): string;
var
   db: tdb;
   pool: tdbpool;
   rows: TtunitArray;
   i: integer;
   res: TRes;
begin
   try
      try
         pool := GetDBPool('1');
         db := pool.Lock;
         db.qry.Close;
         db.qry.SQL.Clear;
         db.qry.SQL.Text := 'select * from tunit';
         db.qry.Open;
         SetLength(rows.tunits, db.qry.RecordCount);
         db.qry.First;
         i := 0;
         while not db.qry.Eof do
         begin
            rows.tunits[i].unitid := db.qry.fieldbyname('unitid').asstring;
            rows.tunits[i].unitname := db.qry.fieldbyname('unitname').asstring;
            inc(i);
            db.qry.Next;
         end;
         rows.status := 200;
         rows.message := 'success';
         result := Tserial.marshal<TtunitArray>(rows);
      except
         on E: Exception do
```

```
        begin
            res.status := 500;
            res.exception := E.message;
            result := Tserial.marshal<TRes>(res);
        end;
    end;
  finally
    pool.Unlock(db);
  end;
end;

function insert(url: string; body: TBytes): string;
var
  db: tdb;
  pool: tdbpool;
  arr: tarray<string>;
  res: TRes;
begin
  try
    try
        var rows: TtunitArray;
        rows := Tserial.unmarshal<TtunitArray>(TEncoding.UTF8.GetString(body));
        arr := url.Split(['/']);
        pool := GetDBPool('1');
        db := pool.Lock;
        db.startTrans;
        for var row: Ttunit in rows.tunits do
        begin
            db.qry.Close;
            db.qry.SQL.Clear;
            db.qry.SQL.Text := 'insert into tunit (unitid,unitname) values (:unitid,:unitname)';
            db.qry.ParamByName('unitid').AsString := row.unitid;
            db.qry.ParamByName('unitname').AsString := row.unitname;
            db.qry.ExecSQL;
        end;
        db.commitTrans;
        res.status := 200;
        res.message := 'success';
        Result := Tserial.marshal<TRes>(res);
    except
        on E: Exception do
        begin
            db.rollbackTrans;
            res.status := 500;
```

```
            res.exception := E.Message;
            Result := Tserial.marshal<TRes>(res);
        end;
    end;
  finally
    pool.Unlock(db);
  end;
end;

function update(url: string; body: TBytes): string;
var
  db: tdb;
  pool: tdbpool;
  arr: tarray<string>;
  res: TRes;
begin
  try
    try
        var rows: TtunitArray;
        rows := Tserial.unmarshal<TtunitArray>(TEncoding.UTF8.GetString(body));
        arr := url.Split(['/']);
        pool := GetDBPool('1');
        db := pool.Lock;
        db.startTrans;
        for var row: Ttunit in rows.tunits do
        begin
            db.qry.Close;
            db.qry.SQL.Clear;
            db.qry.SQL.Text := 'update tunit set unitid=:unitid,unitname=:unitname where
unitid=:key0';
            db.qry.ParamByName('unitid').AsString := row.unitid;
            db.qry.ParamByName('key0').value := row.unitid;
            db.qry.ParamByName('unitname').AsString := row.unitname;
            db.qry.ExecSQL;
        end;
        db.commitTrans;
        res.status := 200;
        res.message := 'success';
        Result := Tserial.marshal<TRes>(res);
    except
        on E: Exception do
        begin
            db.rollbackTrans;
            res.status := 500;
```

```
            res.exception := E.Message;
            Result := Tserial.marshal<TRes>(res);
        end;
    end;
  finally
    pool.Unlock(db);
  end;
end;

function delete(url: string; body: TBytes): string;
var
  db: tdb;
  pool: tdbpool;
  arr: tarray<string>;
  res: TRes;
begin
  try
    try
      arr := url.Split(['/']);
      pool := GetDBPool('1');
      db := pool.Lock;
      db.qry.Close;
      db.qry.SQL.Clear;
      var where: string := ' where ' + TNetEncoding.URL.Decode(arr[3]);
      db.qry.SQL.Text := 'delete from tunit' + where;
      db.qry.ExecSQL;
      res.status := 200;
      res.message := 'success';
      Result := Tserial.marshal<TRes>(res);
    except
      on E: Exception do
      begin
        res.status := 500;
        res.exception := E.Message;
        Result := Tserial.marshal<TRes>(res);
      end;
    end;
  finally
    pool.Unlock(db);
  end;
end;

end.
```

Restful protobuf

```
unit proto.tunit;
//代码由代码工厂自动生成
//2022-09-01 07:17:37
{$I def.inc}
interface

uses
  {$IFDEF firedac}    db.firedac, db.firedacPool, {$ENDIF}
  {$IFDEF unidac}db.unidac, db.unidacpool,    {$ENDIF}
  Grijjy.ProtocolBuffers, SysUtils, classes, System.NetEncoding, db, serialize;

type
  Ttunit = record
    [Serialize(1)] unitid: string;
    [Serialize(2)] unitname: string;
  end;

  TtunitArray = record
    [Serialize(1)] status: integer;
    [Serialize(2)] exception: string;
    [Serialize(3)] message: string;
    [Serialize(4)] tunits: TArray<Ttunit>;
  end;

  TRes = record
    [Serialize(1)] status: integer;
    [Serialize(2)] exception: string;
    [Serialize(3)] message: string;
  end;

function select(url: string; body: TBytes): TBytes;

function insert(url: string; body: TBytes): TBytes;

function update(url: string; body: TBytes): TBytes;

function delete(url: string; body: TBytes): TBytes;

implementation

function select(url: string; body: TBytes): TBytes;
var
  db: tdb;
```

```
  pool: tdbpool;
  rows: TtunitArray;
  i: integer;
  res: TRes;
begin
  try
    try
      pool := GetDBPool('1');
      db := pool.Lock;
      db.qry.Close;
      db.qry.SQL.Clear;
      db.qry.SQL.Text := 'select * from tunit';
      db.qry.Open;
      SetLength(rows.tunits, db.qry.RecordCount);
      db.qry.First;
      i := 0;
      while not db.qry.Eof do
      begin
        rows.tunits[i].unitid := db.qry.fieldbyname('unitid').asstring;
        rows.tunits[i].unitname := db.qry.fieldbyname('unitname').asstring;
        inc(i);
        db.qry.Next;
      end;
      rows.status := 200;
      rows.message := 'success';
      result := Tserial.marshal2<TtunitArray>(rows);
    except
      on E: Exception do
      begin
        res.status := 500;
        res.exception := E.message;
        result := Tserial.marshal2<TRes>(res);
      end;
    end;
  finally
    pool.Unlock(db);
  end;
end;

function insert(url: string; body: TBytes): TBytes;
var
  db: tdb;
  pool: tdbpool;
  arr: tarray<string>;
```

```
  res: TRes;
begin
  try
    try
      var rows: TtunitArray := TSerial.unmarshal<TtunitArray>(body);
      arr := url.Split(['/']);
      pool := GetDBPool('1');
      db := pool.Lock;
      db.startTrans;
      for var row: Ttunit in rows.tunits do
      begin
        db.qry.Close;
        db.qry.SQL.Clear;
        db.qry.SQL.Text := 'insert into tunit (unitid,unitname) values (:unitid,:unitname)';
        db.qry.ParamByName('unitid').AsString := row.unitid;
        db.qry.ParamByName('unitname').AsString := row.unitname;
        db.qry.ExecSQL;
      end;
      db.commitTrans;
      res.status := 200;
      res.message := 'success';
      Result := Tserial.marshal2<TRes>(res);
    except
      on E: Exception do
      begin
        db.rollbackTrans;
        res.status := 500;
        res.exception := E.Message;
        Result := Tserial.marshal2<TRes>(res);
      end;
    end;
  finally
    pool.Unlock(db);
  end;
end;

function update(url: string; body: TBytes): TBytes;
var
  db: tdb;
  pool: tdbpool;
  arr: tarray<string>;
  res: TRes;
begin
  try
```

```
      try
        var rows: TtunitArray := Tserial.unmarshal<TtunitArray>(body);
        arr := url.Split(['/']);
        pool := GetDBPool('1');
        db := pool.Lock;
        db.startTrans;
        for var row: Ttunit in rows.tunits do
        begin
          db.qry.Close;
          db.qry.SQL.Clear;
          db.qry.SQL.Text := 'update tunit set unitid=:unitid,unitname=:unitname where
unitid=:key0';
          db.qry.ParamByName('unitid').AsString := row.unitid;
          db.qry.ParamByName('key0').value := row.unitid;
          db.qry.ParamByName('unitname').AsString := row.unitname;
          db.qry.ExecSQL;
        end;
        db.commitTrans;
        res.status := 200;
        res.message := 'success';
        Result := Tserial.marshal2<TRes>(res);
      except
        on E: Exception do
        begin
          db.rollbackTrans;
          res.status := 500;
          res.exception := E.Message;
          Result := Tserial.marshal2<TRes>(res);
        end;
      end;
    finally
      pool.Unlock(db);
    end;
end;

function delete(url: string; body: TBytes): TBytes;
var
    db: tdb;
    pool: tdbpool;
    arr: tarray<string>;
    res: TRes;
begin
    try
      try
```

```
        arr := url.Split(['/']);
        pool := GetDBPool('1');
        db := pool.Lock;
        db.qry.Close;
        db.qry.SQL.Clear;
        var where: string := ' where ' + TNetEncoding.URL.Decode(arr[3]);
        db.qry.SQL.Text := 'delete from tunit' + where;
        db.qry.ExecSQL;
        res.status := 200;
        res.message := 'success';
        Result := Tserial.marshal2<TRes>(res);
      except
        on E: Exception do
        begin
          res.status := 500;
          res.exception := E.Message;
          Result := Tserial.marshal2<TRes>(res);
        end;
      end;
    finally
      pool.Unlock(db);
    end;
end;

end.
```

跨平台和语言

接口描述语言（IDL）

IDL 是 Interface description language 的缩写，指接口描述语言，是跨平台开发的基础。目前流行的 IDL 有：WEBSERVICE、RESTFUL、gRPC、THRIFT。。。

面向 model 编程

用 google protobuf 生成*.proto 模板，用工具将模板翻译为目标语言的代码。

delphi 面向 model 编程

```
unit serialize;
/// <author>cxg 2022-8-21</author>
interface

uses
  System.SysUtils, Grijjy.ProtocolBuffers,
  System.JSON.Serializers;

type
```

```
  TSerial = class
  public
    //还原
    class function unmarshal<T: record>(const value: string): T; overload; //json
    class function unmarshal<T: record>(const value: tbytes): T; overload; //protobuf
    //序列
    class function marshal<T: record>(const aRecord: T): string;      //json
    class function marshal2<T: record>(const aRecord: T): TBytes;    //protobuf
  end;

implementation

class function TSerial.marshal2<T>(const aRecord: T): TBytes;
begin
  var s: TgoProtocolBuffer := TgoProtocolBuffer.Create;
  Result := s.Serialize<T>(aRecord);
  s.Free;
end;

class function TSerial.marshal<T>(const aRecord: T): string;
begin
  var s: TJsonSerializer := TJsonSerializer.Create;
  Result := s.Serialize<T>(aRecord);
  s.Free;
end;

class function TSerial.unmarshal<T>(const value: string): T;
begin
  var s: TJsonSerializer := TJsonSerializer.Create;
  Result := s.Deserialize<T>(value);
  s.free;
end;

class function TSerial.unmarshal<T>(const value: tbytes): T;
begin
  var s: TgoProtocolBuffer := TgoProtocolBuffer.Create;
  s.Deserialize<T>(result, value);
  s.Free;
end;



end.
```

```
unit server.rest.api;
```

```
/// <author>cxg 2022-8-23</author>
{.$DEFINE indy}
interface

uses
  {$IFDEF indy}
  IdHTTP,
  {$ELSE}
  System.Net.HttpClientComponent, System.Net.HttpClient, System.Net.URLClient,
  {$ENDIF}
  serialize,
    System.Classes,    System.NetEncoding,
    System.SysUtils;

type
  TRest = class
  public
     /// <summary>
     ///  查询
     /// </summary>
     /// <param name="resource">资源</param>
     /// <param name="dbid">数据库 id</param>
     /// <param name="where">查询条件，例如：unitname like '%'</param>
     /// <returns>记录数组</returns>
     class function select<T: record>(const resource: string; where: string = ''): T;
     class function insert<T: record>(const resource: string; const aRecord: T): string;
     class function update<T: record>(const resource: string; const aRecord: T): string;
     /// <summary>
     ///  删除
     /// </summary>
     /// <param name="resource">资源</param>
     /// <param name="where">删除条件，例如：unitid='10'</param>
     /// <param name="dbid">数据库 id</param>
     /// <returns>json</returns>
     class function delete(const resource: string; where: string): string;
     //protobuf CRUD
     class function select2<T: record>(const resource: string; where: string = ''): T;
     class function delete2(const resource: string; where: string): string;
     class function insert2<T: record>(const resource: string; const aRecord: T): tbytes;
     class function update2<T: record>(const resource: string; const aRecord: T): tbytes;
  end;

var
  ipport: string = 'http://localhost:1234';
```

```
  {$IFDEF indy}
  http: TIdHTTP;
  {$ELSE}
  http: TNetHTTPClient;
  {$ENDIF}

implementation

{ TRest }

class function TRest.update<T>(const resource: string; const aRecord: T): string;
begin
  var bs: TBytes := TEncoding.UTF8.GetBytes(TSerial.marshal<T>(aRecord));
  var req: TBytesStream := TBytesStream.Create(bs);
  req.Position := 0;
  {$IFDEF indy}
  Result := http.Put(ipport + '/rest/' + resource, req);
  {$ELSE}
  Result := http.Put(ipport + '/rest/' + resource, req).ContentAsString;
  {$ENDIF}
  req.Free;
end;

class function TRest.update2<T>(const resource: string; const aRecord: T): tbytes;
begin
  var bs: TBytes := TSerial.marshal2<T>(aRecord);
  var req: TBytesStream := TBytesStream.Create(bs);
  req.Position := 0;
  var res: TMemoryStream := TMemoryStream.Create;
  var s: string := ipport + '/protobuf/' + resource;
  http.Put(s, req, res);
  {$IFDEF indy}
  var at: Int64 := res.Position;
  {$ELSE}
  var at: Int64 := res.Size;
  {$ENDIF}
  SetLength(result, at);
  Move(res.Memory^, result[0], at);
  req.Free;
  res.Free;
end;

class function TRest.select<T>(const resource: string; where: string = ''): T;
begin
```

```
  var s: string;
  if where = '' then
     s := ipport + '/rest/' + resource
  else
     s := ipport + '/rest/' + resource + TNetEncoding.URL.Encode(where);
  {$IFDEF indy}
  Result := TSerial.unmarshal<T>(http.Get(s));
  {$ELSE}
  Result := TSerial.unmarshal<T>(http.Get(s).ContentAsString);
  {$ENDIF}
end;

class function TRest.select2<T>(const resource: string; where: string = ''): T;
begin
  var s: string;
  if where = '' then
     s := ipport + '/protobuf/' + resource
  else
     s := ipport + '/protobuf/' + resource + '/' + TNetEncoding.URL.Encode(where);
  var res: TMemoryStream := TMemoryStream.Create;
  http.Get(s, res);
  var bs: TBytes;
  {$IFDEF indy}
  var at: Int64 := res.Position;
  {$ELSE}
  var at: Int64 := res.Size;
  {$ENDIF}
  SetLength(bs, at);
  Move(res.Memory^, bs[0], at);
  res.Free;
  Result := TSerial.unmarshal<T>(bs);
end;

class function TRest.delete(const resource: string; where: string): string;
begin
  var s: string := ipport + '/rest/' + resource + '/' + TNetEncoding.URL.Encode(where);
  {$IFDEF indy}
  Result := http.Delete(s);
  {$ELSE}
  Result := http.Delete(s).ContentAsString;
  {$ENDIF}
end;

class function TRest.delete2(const resource: string; where: string): string;
```

```
begin
  var s: string := ipport + '/protobuf/' + resource + '/' + TNetEncoding.URL.Encode(where);
  {$IFDEF indy}
  Result := http.Delete(s);
  {$ELSE}
  var res: TStringStream := TStringStream.Create;
  http.Delete(s, res);
  Result := res.DataString;
  res.Free;
  {$ENDIF}
end;

class function TRest.insert<T>(const resource: string; const aRecord: T): string;
begin
  var bs: TBytes := TEncoding.UTF8.GetBytes(TSerial.marshal<T>(aRecord));
  var req: TBytesStream := TBytesStream.Create(bs);
  req.Position := 0;
  {$IFDEF indy}
  Result := http.post(ipport + '/rest/' + resource, req);
  {$ELSE}
  Result := http.post(ipport + '/rest/' + resource, req).ContentAsString;
  {$ENDIF}
  req.Free;
end;

class function TRest.insert2<T>(const resource: string; const aRecord: T): tbytes;
begin
  var bs: TBytes := TSerial.marshal2<T>(aRecord);
  var req: TBytesStream := TBytesStream.Create(bs);
  var res: TMemoryStream := TMemoryStream.Create;
  req.Position := 0;
  var s: string := ipport + '/protobuf/' + resource;
  http.post(s, req, res);
  {$IFDEF indy}
  var at: Int64 := res.Position;
  {$ELSE}
  var at: Int64 := res.Size;
  {$ENDIF}
  SetLength(result, at);
  Move(res.Memory^, result[0], at);
  req.Free;
  res.Free;
end;
```

```
initialization
  {$IFDEF indy}
  http := TIdHTTP.Create(nil);
  http.HandleRedirects := True;
  {$ELSE}
  http := TNetHTTPClient.Create(nil)
  {$ENDIF}

finalization
  FreeAndNil(http);

end.
unit Unit1;
/// <author>cxg 2022-7-13</author>
interface

uses     Grijjy.ProtocolBuffers, serialize,
  Data.DB, FireDAC.Stan.Intf, FireDAC.Stan.Option, server.rest.api,
  FireDAC.Stan.Param, FireDAC.Stan.Error, FireDAC.DatS, FireDAC.Phys.Intf,
  FireDAC.DApt.Intf, Datasnap.DBClient, FireDAC.Comp.DataSet,
  FireDAC.Comp.Client, Vcl.StdCtrls, Vcl.Controls, Vcl.Grids, Vcl.DBGrids,
  System.Classes ,vcl.forms ,    vcl.dialogs
  , system.SysUtils
  ;

type
  {$REGION '定义 model'}
  Ttunit = record
     [Serialize(1)] unitid: string;
     [Serialize(2)] unitname: string;
  end;

  TtunitArray = record
     [Serialize(1)] status: integer;
     [Serialize(2)] exception: string;
     [Serialize(3)] message: string;
     [Serialize(4)] tunits: array of Ttunit; //TArray<Ttunit>;
  end;

  TRes = record
     [Serialize(1)] status: integer;
     [Serialize(2)] exception: string;
     [Serialize(3)] message: string;
  end;
```

```
  {$ENDREGION}

  TForm1 = class(TForm)
    DBGrid1: TDBGrid;
    DataSource1: TDataSource;
    FDMemTable1: TFDMemTable;
    Button3: TButton;
    Button4: TButton;
    Button5: TButton;
    Button6: TButton;
    Button1: TButton;
    Button2: TButton;
    Button7: TButton;
    Button8: TButton;
    procedure FormCreate(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button6Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button7Click(Sender: TObject);
    procedure Button8Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
//新增 json
begin
  var t: TtunitArray;
  SetLength(t.tunits, 1);
  t.tunits[0].Unitid := '1';
  t.tunits[0].Unitname := '新增';
  var res: string := TRest.insert<TtunitArray>('tunit', t);
```

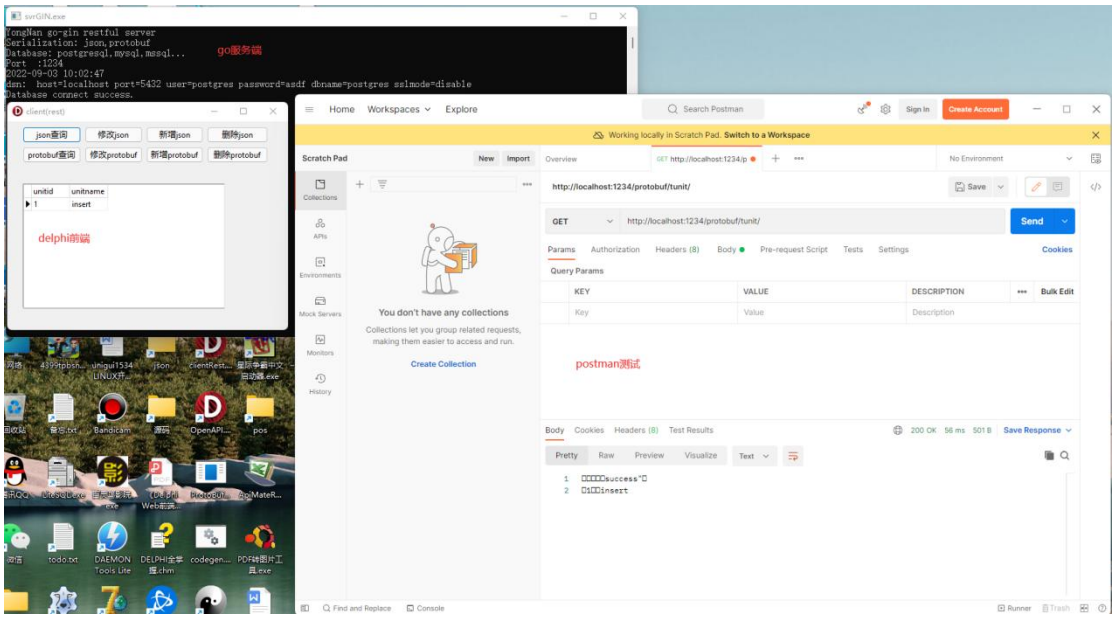
```
  var r: TRes := TSerial.unmarshal<TRes>(res);
  if r.Status = 500 then
    ShowMessage('err: ' + r.Exception)
  else
    ShowMessage('ok');
end;

procedure TForm1.Button2Click(Sender: TObject);
//新增 protobuf
begin
  var t: TtunitArray;
  SetLength(t.tunits, 1);
  t.tunits[0].Unitid := '1';
  t.tunits[0].Unitname := 'insert';
  var res: tbytes := TRest.insert2<TtunitArray>('tunit', t);
  var r: TRes := TSerial.unmarshal<TRes>(res);
  if r.Status = 500 then
    ShowMessage('err: ' + r.Exception)
  else
    ShowMessage('ok');
end;

procedure TForm1.Button3Click(Sender: TObject);
//修改 json
begin
  var t: TtunitArray;
  SetLength(t.tunits, 1);
  t.tunits[0].Unitid := '1';
  t.tunits[0].Unitname := '修改';
  var res: string := TRest.update<TtunitArray>('tunit', t);
  var r: TRes := TSerial.unmarshal<TRes>(res);
  if r.Status = 500 then
    ShowMessage('err: ' + r.Exception)
  else
    ShowMessage('ok');
end;

procedure TForm1.Button4Click(Sender: TObject);
//修改 protobuf
begin
  var t: TtunitArray;
  SetLength(t.tunits, 1);
  t.tunits[0].Unitid := '1';
  t.tunits[0].Unitname := 'update';
```

```
  var res: tbytes := TRest.update2<TtunitArray>('tunit', t);
  var r: TRes := TSerial.unmarshal<TRes>(res);
  if r.Status = 500 then
    ShowMessage('err: ' + r.Exception)
  else
    ShowMessage('ok');
end;

procedure TForm1.Button5Click(Sender: TObject);
//json 查询
begin
  var t: TtunitArray := TRest.select<TtunitArray>('tunit');
  if t.Status = 500 then
  begin
    ShowMessage(t.Exception);
    Exit;
  end;
  FDMemTable1.EmptyDataSet;
  FDMemTable1.DisableControls;
  for var dw: Ttunit in t.tunits do
    FDMemTable1.AppendRecord([dw.Unitid, dw.Unitname]);
  FDMemTable1.First;
  FDMemTable1.EnableControls;
end;

procedure TForm1.Button6Click(Sender: TObject);
//PROTOBUF 查询
begin
  var t: TtunitArray := TRest.select2<TtunitArray>('tunit');
  if t.Status = 500 then
  begin
    ShowMessage(t.Exception);
    Exit;
  end;
  FDMemTable1.EmptyDataSet;
  FDMemTable1.DisableControls;
  for var dw: Ttunit in t.tunits do
    FDMemTable1.AppendRecord([dw.Unitid, dw.Unitname]);
  FDMemTable1.First;
  FDMemTable1.EnableControls;
end;

procedure TForm1.Button7Click(Sender: TObject);
//删除 json
```

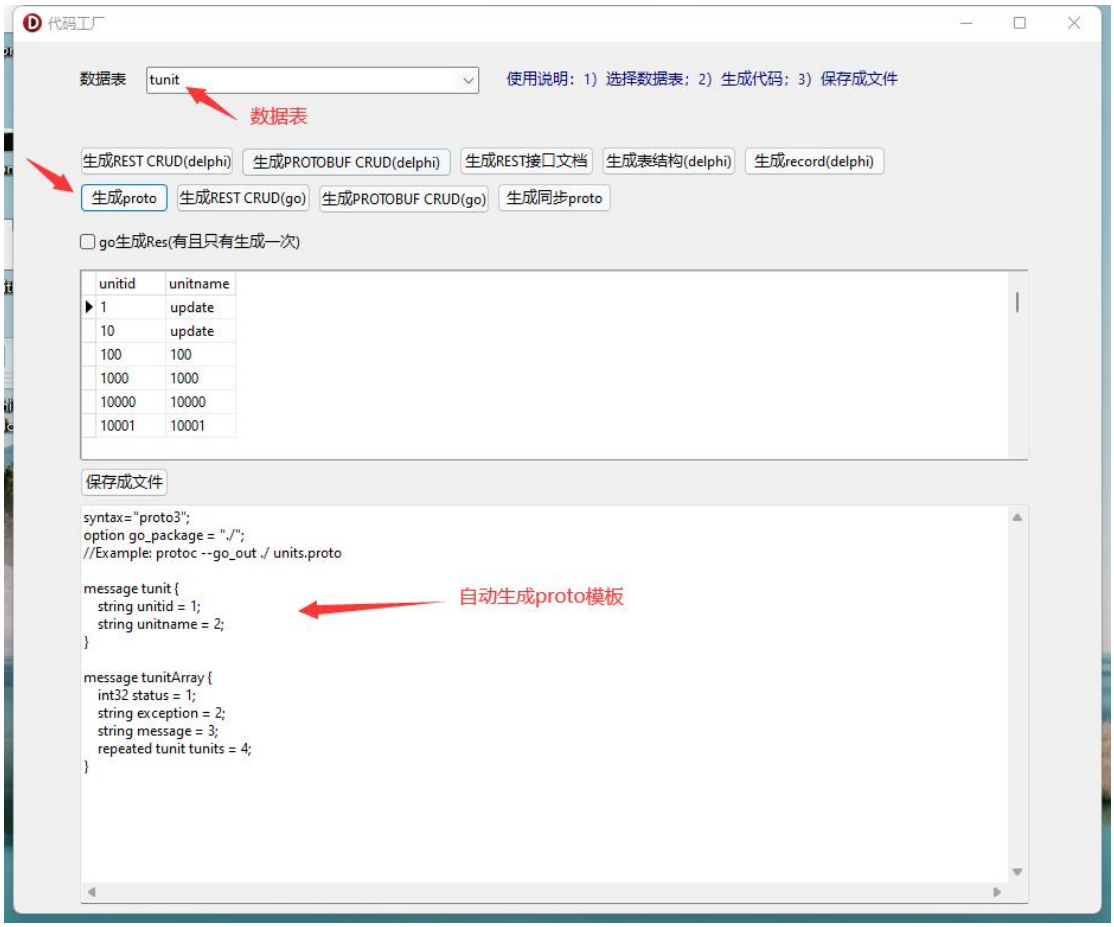
```
begin
   var res: string := TRest.delete('tunit', 'unitid=''1''');
   var r: TRes := TSerial.unmarshal<TRes>(res);
   if r.Status = 500 then
      ShowMessage('err: ' + r.Exception)
   else
      ShowMessage('ok');
end;

procedure TForm1.Button8Click(Sender: TObject);
//删除 protobuf
begin
   var res: TBytes := TEncoding.UTF8.GetBytes(TRest.delete2('tunit', 'unitid=''1'''));
   var r: TRes := TSerial.unmarshal<TRes>(res);
   if r.Status = 500 then
      ShowMessage('err: ' + r.Exception)
   else
      ShowMessage('ok');
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   FDMemTable1.FieldDefs.Add('unitid', ftString, 9);
   FDMemTable1.FieldDefs.Add('unitname', ftString, 9);
   FDMemTable1.CreateDataSet;
end;

end.
```

面向 model 的跨语言演示

咏南中间件 https://www.cnblogs.com/hnxxcxg/



数据表生成 proto 模板



Proto 模板翻译为 pascal

Unit.proto

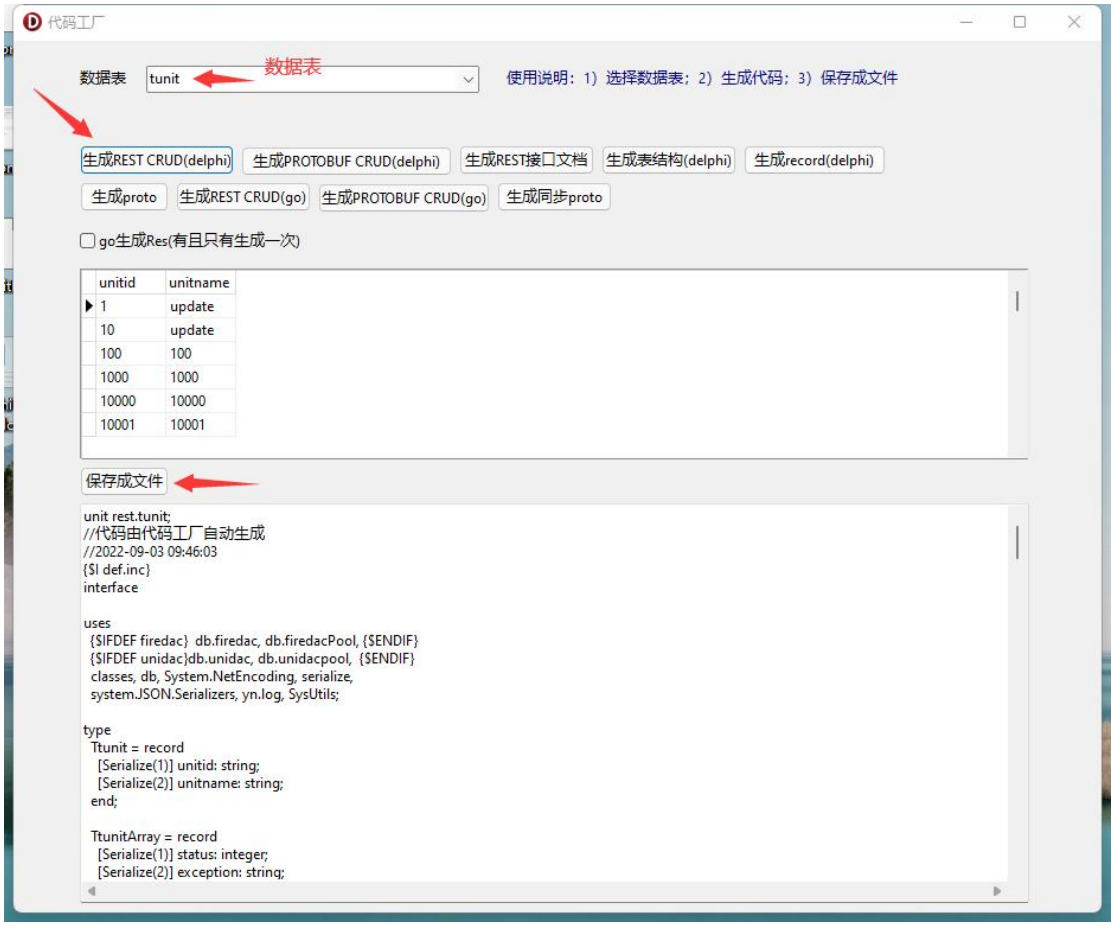
```
syntax="proto3";
option go_package = "./";
//Example: protoc --go_out ./ units.proto

message tunit {
	string unitid = 1;
	string unitname = 2;
}

message tunitArray {
	int32 status = 1;
	string exception = 2;
	string message = 3;
	repeated tunit tunits = 4;
}
```

运行工具自动翻译

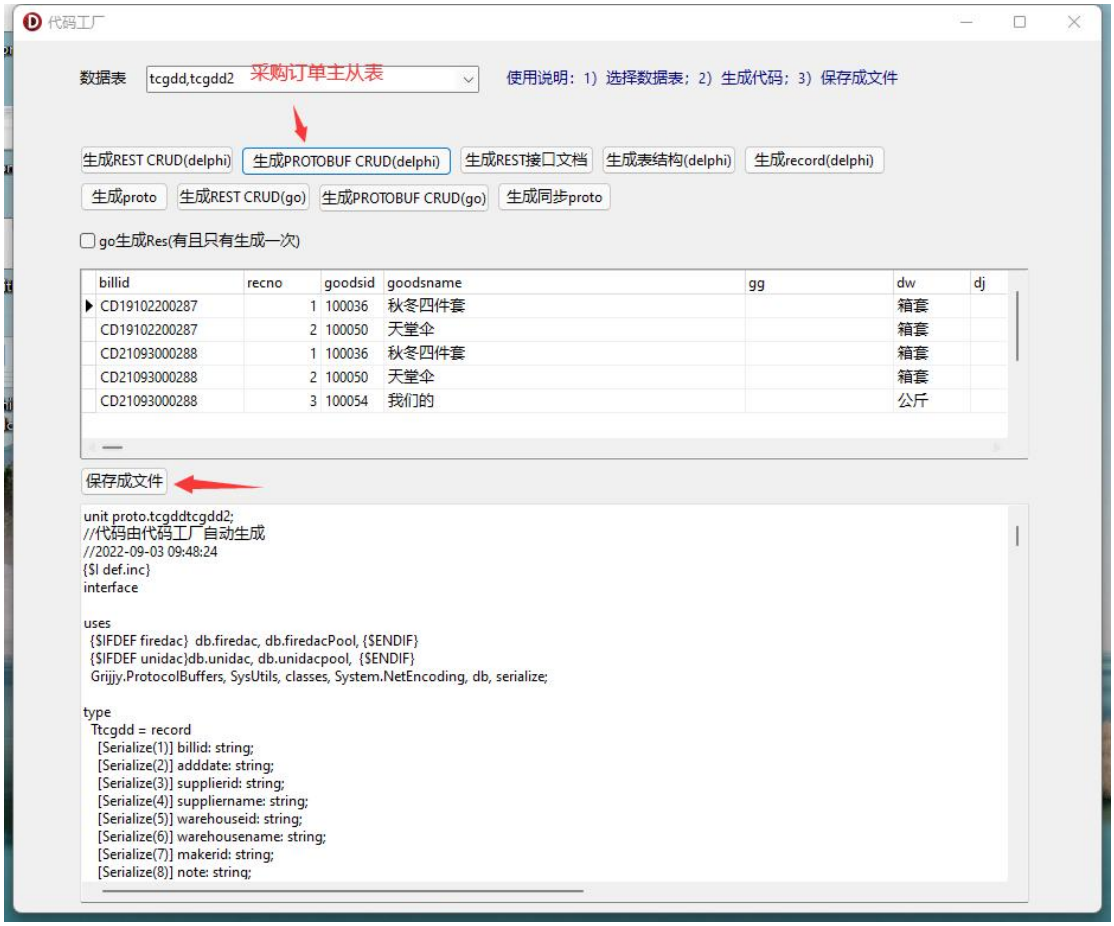

```
{ Unit pbUnitMessages.pas }
{ Generated from unit.proto }
{ Package Unit }

unit pbUnitMessages;

interface

uses Grijjy.ProtocolBuffers, SysUtils;

{ TTunit }

type
  TTunit = record
    [Serialize(1)] Unitid : String;
    [Serialize(2)] Unitname : String;
  end;

{ TTunitArray }
```

```
type
  TTunitArray = record
    [Serialize(1)] Status : Integer;
    [Serialize(2)] Exception : String;
    [Serialize(3)] Message : String;
    [Serialize(4)] Tunits : TArray<TTunit>;
  end;

implementation

end.
```

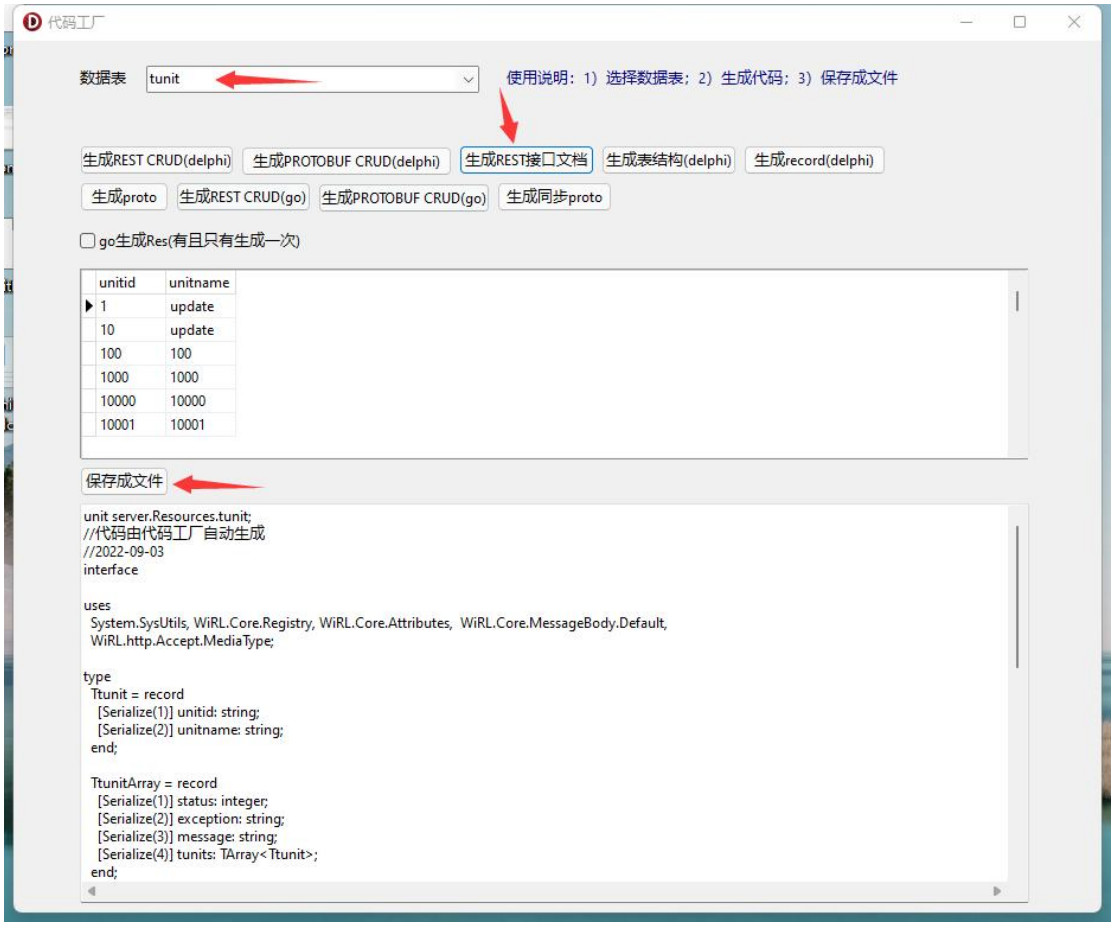
数据表自动生成 restful json CURD



数据表自动生成 restful protobuf CURD



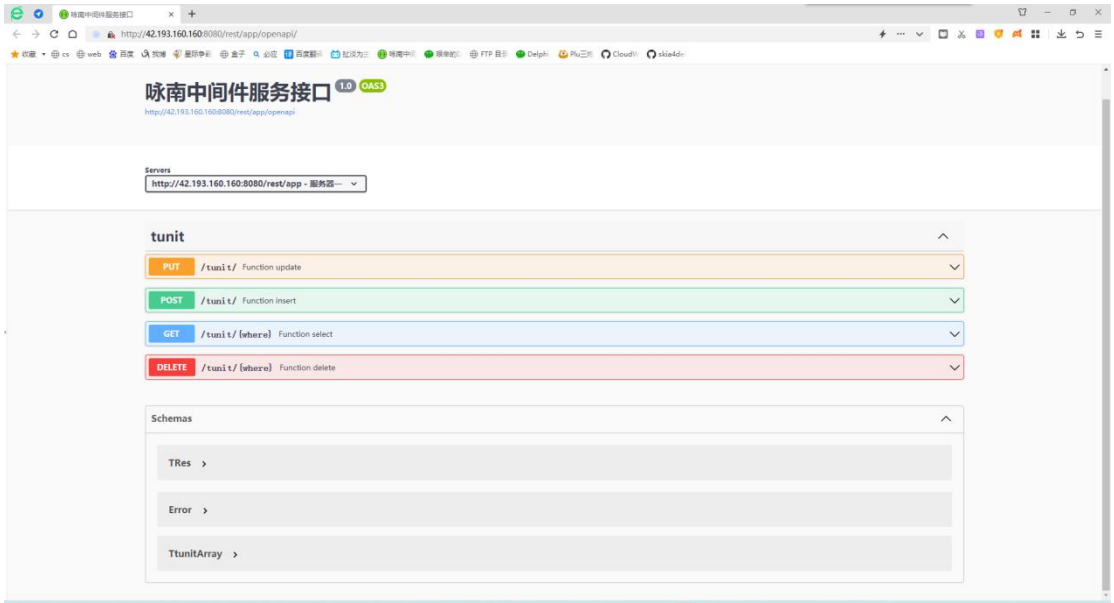
数据表自动生成 SWAGGER 接口文档




```
unit server.Resources.tunit;
//代码由代码工厂自动生成
//2022-09-03
interface

uses
  System.SysUtils, WiRL.Core.Registry, WiRL.Core.Attributes,   WiRL.Core.MessageBody.Default,
  WiRL.http.Accept.MediaType;

type
  Ttunit = record
    [Serialize(1)] unitid: string;
    [Serialize(2)] unitname: string;
  end;

  TtunitArray = record
    [Serialize(1)] status: integer;
    [Serialize(2)] exception: string;
    [Serialize(3)] message: string;
    [Serialize(4)] tunits: TArray<Ttunit>;
  end;
```

```
  TRes = record
    [Serialize(1)] status: integer;
    [Serialize(2)] exception: string;
    [Serialize(3)] message: string;
  end;

  [Path('tunit')]
  TtunitAPI = class
    [get, path('/{where}'), Produces(TMediaType.APPLICATION_JSON)]
    function select([PathParam('where')] where: string): TtunitArray; virtual; abstract;
    [post,                                    Consumes(TMediaType.APPLICATION_JSON),
Produces(TMediaType.APPLICATION_JSON)]
    function insert([BodyParam] body: TtunitArray): TRes; virtual; abstract;
    [put,                                     Consumes(TMediaType.APPLICATION_JSON),
Produces(TMediaType.APPLICATION_JSON)]
    function update([BodyParam] body: TtunitArray): TRes; virtual; abstract;
    [delete, path('/{where}'), Produces(TMediaType.APPLICATION_JSON)]
    function delete([PathParam('where')] where: string): TRes; virtual; abstract;
  end;

implementation

initialization
  TWiRLResourceRegistry.Instance.RegisterResource<TtunitAPI>;

end.
```

浏览器打开：http://42.193.160.160:8080/rest/app/openapi/



客户端演示

restful 客户端

```
unit Unit1;
/// <author>cxg 2022-7-13</author>
interface

uses      Grijjy.ProtocolBuffers, serialize,
   Data.DB, FireDAC.Stan.Intf, FireDAC.Stan.Option, server.rest.api,
   FireDAC.Stan.Param, FireDAC.Stan.Error, FireDAC.DatS, FireDAC.Phys.Intf,
   FireDAC.DApt.Intf, Datasnap.DBClient, FireDAC.Comp.DataSet,
   FireDAC.Comp.Client, Vcl.StdCtrls, Vcl.Controls, Vcl.Grids, Vcl.DBGrids,
   System.Classes ,vcl.forms ,    vcl.dialogs
   , system.SysUtils
   ;

type
   {$REGION '定义 model'}
   Ttunit = record
      [Serialize(1)] unitid: string;
      [Serialize(2)] unitname: string;
   end;

   TtunitArray = record
      [Serialize(1)] status: integer;
      [Serialize(2)] exception: string;
      [Serialize(3)] message: string;
      [Serialize(4)] tunits: array of Ttunit; //TArray<Ttunit>;
   end;

   TRes = record
      [Serialize(1)] status: integer;
      [Serialize(2)] exception: string;
      [Serialize(3)] message: string;
   end;
   {$ENDREGION}

   TForm1 = class(TForm)
      DBGrid1: TDBGrid;
      DataSource1: TDataSource;
      FDMemTable1: TFDMemTable;
      Button3: TButton;
      Button4: TButton;
      Button5: TButton;
```

```
      Button6: TButton;
      Button1: TButton;
      Button2: TButton;
      Button7: TButton;
      Button8: TButton;
      procedure FormCreate(Sender: TObject);
      procedure Button3Click(Sender: TObject);
      procedure Button4Click(Sender: TObject);
      procedure Button5Click(Sender: TObject);
      procedure Button6Click(Sender: TObject);
      procedure Button1Click(Sender: TObject);
      procedure Button2Click(Sender: TObject);
      procedure Button7Click(Sender: TObject);
      procedure Button8Click(Sender: TObject);
   private
      { Private declarations }
   public
      { Public declarations }
   end;

var
   Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
//新增 json
begin
   var t: TtunitArray;
   SetLength(t.tunits, 1);
   t.tunits[0].Unitid := '1';
   t.tunits[0].Unitname := '新增';
   var res: string := TRest.insert<TtunitArray>('tunit', t);
   var r: TRes := TSerial.unmarshal<TRes>(res);
   if r.Status = 500 then
      ShowMessage('err: ' + r.Exception)
   else
      ShowMessage('ok');
end;

procedure TForm1.Button2Click(Sender: TObject);
//新增 protobuf
```

```
begin
   var t: TtunitArray;
   SetLength(t.tunits, 1);
   t.tunits[0].Unitid := '1';
   t.tunits[0].Unitname := 'insert';
   var res: tbytes := TRest.insert2<TtunitArray>('tunit', t);
   var r: TRes := TSerial.unmarshal<TRes>(res);
   if r.Status = 500 then
      ShowMessage('err: ' + r.Exception)
   else
      ShowMessage('ok');
end;

procedure TForm1.Button3Click(Sender: TObject);
//修改 json
begin
   var t: TtunitArray;
   SetLength(t.tunits, 1);
   t.tunits[0].Unitid := '1';
   t.tunits[0].Unitname := '修改';
   var res: string := TRest.update<TtunitArray>('tunit', t);
   var r: TRes := TSerial.unmarshal<TRes>(res);
   if r.Status = 500 then
      ShowMessage('err: ' + r.Exception)
   else
      ShowMessage('ok');
end;

procedure TForm1.Button4Click(Sender: TObject);
//修改 protobuf
begin
   var t: TtunitArray;
   SetLength(t.tunits, 1);
   t.tunits[0].Unitid := '1';
   t.tunits[0].Unitname := 'update';
   var res: tbytes := TRest.update2<TtunitArray>('tunit', t);
   var r: TRes := TSerial.unmarshal<TRes>(res);
   if r.Status = 500 then
      ShowMessage('err: ' + r.Exception)
   else
      ShowMessage('ok');
end;

procedure TForm1.Button5Click(Sender: TObject);
```

```
//json 查询
begin
  var t: TtunitArray := TRest.select<TtunitArray>('tunit');
  if t.Status = 500 then
  begin
    ShowMessage(t.Exception);
    Exit;
  end;
  FDMemTable1.EmptyDataSet;
  FDMemTable1.DisableControls;
  for var dw: Ttunit in t.tunits do
    FDMemTable1.AppendRecord([dw.Unitid, dw.Unitname]);
  FDMemTable1.First;
  FDMemTable1.EnableControls;
end;

procedure TForm1.Button6Click(Sender: TObject);
//PROTOBUF 查询
begin
  var t: TtunitArray := TRest.select2<TtunitArray>('tunit');
  if t.Status = 500 then
  begin
    ShowMessage(t.Exception);
    Exit;
  end;
  FDMemTable1.EmptyDataSet;
  FDMemTable1.DisableControls;
  for var dw: Ttunit in t.tunits do
    FDMemTable1.AppendRecord([dw.Unitid, dw.Unitname]);
  FDMemTable1.First;
  FDMemTable1.EnableControls;
end;

procedure TForm1.Button7Click(Sender: TObject);
//删除 json
begin
  var res: string := TRest.delete('tunit', 'unitid=''1''');
  var r: TRes := TSerial.unmarshal<TRes>(res);
  if r.Status = 500 then
    ShowMessage('err: ' + r.Exception)
  else
    ShowMessage('ok');
end;
```

```
procedure TForm1.Button8Click(Sender: TObject);
//删除 protobuf
begin
  var res: TBytes := TEncoding.UTF8.GetBytes(TRest.delete2('tunit', 'unitid=''1'''));
  var r: TRes := TSerial.unmarshal<TRes>(res);
  if r.Status = 500 then
    ShowMessage('err: ' + r.Exception)
  else
    ShowMessage('ok');
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  FDMemTable1.FieldDefs.Add('unitid', ftString, 9);
  FDMemTable1.FieldDefs.Add('unitname', ftString, 9);
  FDMemTable1.CreateDataSet;
end;

end.
```

传统二进制客户端

```
unit Unit1;
/// <author>cxg 2022-5-3</author>

interface

uses   dialogs,   keyValue.serialize, System.SysUtils,
  server.api, Data.DB, forms,   Vcl.ExtCtrls, Vcl.Controls,
  Vcl.Grids, Vcl.DBGrids, System.Classes, Vcl.StdCtrls, FireDAC.Stan.Intf,
  Datasnap.DBClient, System.Net.URLClient, System.Net.HttpClient,
  System.Net.HttpClientComponent;

type
  TForm1 = class(TForm)
    btn: TButton;
    dbgrd1: TDBGrid;
    dbgrd2: TDBGrid;
    ds1: TDataSource;
    ds2: TDataSource;
    Timer1: TTimer;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
```

```
      ClientDataSet1: TClientDataSet;
      ClientDataSet2: TClientDataSet;
      Button7: TButton;
      Button8: TButton;
      Image1: TImage;
      Memo1: TMemo;
      Button9: TButton;
      Button5: TButton;
      Button6: TButton;
      NetHTTPClient1: TNetHTTPClient;
      procedure btnClick(Sender: TObject);
      procedure Button1Click(Sender: TObject);
      procedure Button2Click(Sender: TObject);
      procedure Button3Click(Sender: TObject);
      procedure Button4Click(Sender: TObject);
      procedure Button7Click(Sender: TObject);
      procedure Timer1Timer(Sender: TObject);
      procedure Button8Click(Sender: TObject);
      procedure Button9Click(Sender: TObject);
      procedure Button5Click(Sender: TObject);
      procedure Button6Click(Sender: TObject);
  private        { Private declarations }
  public        { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.btnClick(Sender: TObject);
// /bin/qry  多表查询
begin
  qry2('1', 'select * from tgoods', 'select * from tunit', ClientDataSet1, ClientDataSet2);
end;

procedure TForm1.Button1Click(Sender: TObject);
// /bin/save  多表保存
begin
  save2('1', 'tgoods', 'tunit', ClientDataSet1, ClientDataSet2);
end;
```

```
procedure TForm1.Button2Click(Sender: TObject);
// /bin/spopen 存储过程
begin
   server.api.spopen('1', '存储过程名', 'param1=value1;param2=value2', ClientDataSet1);
end;

procedure TForm1.Button3Click(Sender: TObject);
// /bin/upload 上传
begin
   server.api.upload('c:\1.exe');
end;

procedure TForm1.Button4Click(Sender: TObject);
// /bin/download 下载
begin
  server.api.download('1.exe');
end;

procedure TForm1.Button5Click(Sender: TObject);
// /bin/qry 单表查询
begin
   server.api.qry1('1', 'select * from tgoods', ClientDataSet1);
end;

procedure TForm1.Button6Click(Sender: TObject);
// /bin/save 单表保存
begin
   server.api.save1('1', 'tgoods', ClientDataSet1);
end;

procedure TForm1.Button7Click(Sender: TObject);
// /bin/execsql
begin
   server.api.execsql('1', 'delete from tunit where unitid=''1''');
end;

procedure TForm1.Button8Click(Sender: TObject);
// /bin/verifycode 验证码
var code: string;
   ms: TStream;
begin
   ms := TMemoryStream.Create;
   server.api.verifyCode(code, ms);
   Memo1.Lines.Add(code);
```

```
  ms.Position := 0;
  Image1.Picture.LoadFromStream(ms);
  ms.Free;
end;

procedure TForm1.Button9Click(Sender: TObject);
// /bin/snowflakeid 雪花 ID
var id: Int64;
begin
  server.api.snowflakeID(id, 18, 18);
  Memo1.Lines.Add(id.ToString);
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Button5Click(nil);
end;

end.
```