【A3】Delphi/C++Builder テクニカルセッション

「VCLネットワークプログラミング」

Indyの基礎

（株）シリアルゲームズ
取締役／シニアエンジニア
細川　淳

CODE GEAR FROM Borland
Where Developers Matter
Integrated Development Environments for Windows, Java, and Web Developers
DEVELOPER CAMP

アジェンダ

- Indy について
- TIdHTTP のプログラム
- TIdTCPClient のプログラム

本文書の一部または全部の転載を禁止します。本文書の著作権は、著作者に帰属します。

Indy について

Indy とは？

- Indy = Internet Direct
  - インターネットを使用するコンポーネント群
  - オープンソースで開発されている
  - 今では、Delphi 標準のネット用コンポーネントになっている
- 1993年
  - VB 用のプロジェクトとして始まる
- 1995年
  - Delphi 用の Indy プロジェクトが始まる

Indy の代表的なコンポーネント

- TIdTCPClient
  - TCP を使用するコンポーネント
- TIdHTTP
  - HTTP 通信を行うコンポーネント
- TIdPOP3
  - POP3 でメールを取得するコンポーネント
- TIdSMTP
  - SMTP でメールを送信するコンポーネント
- 今回は
  - TIdHTTP
  - TIdTCPClient
- について解説

CODE GEAR FROM Borland
Where Developers Matter
Integrated Development Environments for Windows, Java, and Web Developers
DEVELOPER CAMP

Indy のユニット

- コンポーネント毎に1つのユニット
  - たとえば TIdHTTP は IdHTTP.pas に定義されている
- 押さえておきたい IdGlobal ユニット
  - ネットワークプログラムに便利な関数や定数が用意されている
    - 各プラットフォーム用に VCL と同名の関数を置き換えた物も多数定義されている

IdGlobal 定数

- HoursPerDay = 24;
- MinsPerHour = 60;
- SecsPerMin = 60;
- MSecsPerSec = 1000;
- LF = #10;
- CR = #13;
- EOL = CR + LF; (sLineBreak)
- CHAR0 = #0;
- BACKSPACE = #8;
- TAB = #9;
- CHAR32 = #32;
- WhiteSpace = [0..12, 14..32];
- IdHexDigits: array [0..15] of AnsiChar = ('0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F');
- IdOctalDigits: array [0..7] of AnsiChar = ('0','1','2','3','4','5','6','7');

CODE GEAR FROM Borland
Where Developers Matter
Integrated Development Environments for Windows, Java, and Web Developers
DEVELOPER CAMP

IdGlobal のクラス

- 有用なストリームクラス
  - TFileCreateStream
    - ファイルを新規作成するストリーム
  - TReadFileNonExclusiveStream
    - 他のアプリケーションがファイルを開いていても例外を出さずに、ファイルを読めるストリーム

IdGlobal ユニット

- InMainThread 関数
  - メインスレッドを実行中なら True
- IPv4ToDWord 関数
  - IPv4 形式の文字列を Cardinal にして返す
    - 192.168.0.1 → C0 A8 00 01 (3232235521)
- Fetch
  - 指定された区切り文字まで文字列を出力する
    - Fetch('192.168.0.1', '.') → '192'

CODE GEAR FROM Borland
Where Developers Matter
Integrated Development Environments for Windows, Java, and Web Developers
DEVELOPER CAMP

TIdHTTP を使ってみよう

スレッドを使って、非同期に Indy を使う！

本文書の一部または全部の転載を禁止します。本文書の著作権は、著作者に帰属します。

TIdHTTP で HTML を取得する

```
procedure ReadFromURL(const iURL: String);
var
  IdHTTP: TIdHTTP;
begin
  IdHTTP := TIdHTTP.Create(nil);
  try
    Memo1.Lines.Text := IdHTTP.Get(iURL);
    ShowMessage('読み込み完了'); // Get完了まで実行されない
  finally
   IdHTTP.Free;
  end;
end;
```


Indy のプログラミングについて

- Indy は同期コンポーネント
  - 非同期で動作させるためには
    - スレッドを使用する
    - メッセージの処理だけなら TIdAntiFreeze でもよい

スレッドを使用する1

```
THTTPGetEvent = procedure (Sender: TObject; const iHTML: String) of object;

TSampleThread = class(TThread)
private
  FRunning: Boolean;
  FURL: String;
  FHTML: String;
  FOnGet: THTTPGetEvent;
private
  procedure CallOnGet;
protected
  procedure Execute; override;
public
  constructor Create; reintroduce;
  procedure SetURL(const iURL: String);
  property Running: Boolean read FRunning;
  property OnGet: THTTPGetEvent read FOnGet write FOnGet;
end;
```


CODE GEAR FROM Borland
Where Developers Matter
Integrated Development Environments for Windows, Java, and Web Developers
DEVELOPER CAMP

スレッドを使用する2

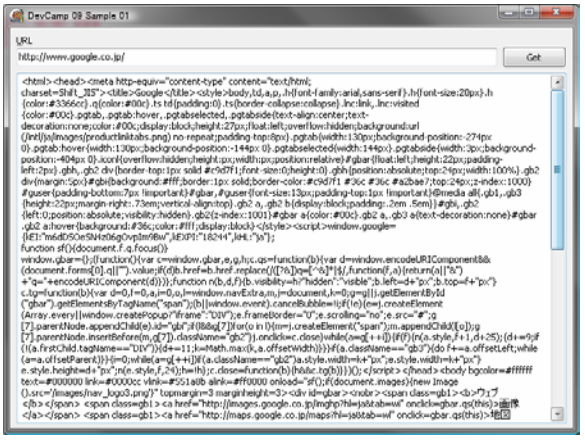
```
procedure TSampleThread.Execute;
var
  IdHTTP: TIdHTTP;
begin
  FRunning := True;
  try
    IdHTTP := TIdHTTP.Create(nil);
    try
      IdHTTP.HandleRedirects := True; // リダイレクトの設定

      while (not Terminated) and (not Application.Terminated) do begin
        if (FURL <> '') then begin
          FHTML := IdHTTP.Get(FURL);

          FURL := '';

          if (FHTML <> '') then
            Synchronize(CallOnGet);
        end;
        Sleep(100);
      end;
    finally
      IdHTTP.Free;
    end;
  finally
    FRunning := False;
  end;
end;
```

スレッドを利用する3

```
procedure TForm1.btnGetClick(Sender: TObject);
begin
  SampleThread.SetURL(edtURL.Text);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  SampleThread := TSampleThread.Create;
  SampleThread.OnGet := SampleThreadGet;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  SampleThread.Terminate;

  while (SampleThread.Running) do
    Sleep(100);

  SampleThread.Free;
end;
```

```
procedure TForm1.SampleThreadGet(Sender: TObject; const iHTML: String);
begin
  memoHTML.Lines.Text := iHTML;
end;
```


実行結果



HTTP で画像を取得

- 画像も Get で取得可能
  - Get の TIdStream を引数に取るバージョンを使う
    - **procedure** Get(AURL: **string**; AResponseContent: TIdStream); **overload**;
    - いままでの Get を使う場合、戻り値が String なため、String からバイナリを起こす必要がある

画像を取得する1

```
procedure TSampleThread.Execute;
var
  IdHTTP: TIdHTTP;
begin
  FRunning := True;
  try
    FStream := nil;
    IdHTTP := nil;
    try
      // 結果受け取り用ストリーム
      FStream := TMemoryStream.Create;

      IdHTTP := TIdHTTP.Create(nil);
      IdHTTP.HandleRedirects := True;
```

```
      while (not Terminated) and (not Application.Terminated) do begin
        if (FURL <> '') then begin
          FStream.Clear;              // クリアして

          IdHTTP.Get(FURL, FStream); // 取得

          FURL := '';

          if (FStream.Size > 0) then begin // 取得していたら
            FStream.Position := 0;
            Synchronize(CallOnGet);
          end;
        end;

        Sleep(100);
      end;
    finally
      IdHTTP.Free;
      FStream.Free;
    end; // 以下略
```

画像を取得する2

```
procedure TForm1.SampleThreadGet(
  Sender: TObject;
  const iStream: TMemoryStream);
var
  Magic: String;
  IsImage: Boolean;
  HTML: String;
```

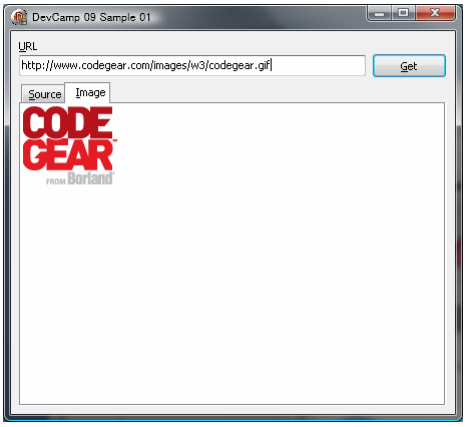
```
procedure AssignImage(
  const iMagic: String;
  const iGraphicClass: TGraphicClass);
var
  Graphic: TGraphic;
begin
  if (Copy(Magic, 1, Length(iMagic)) = iMagic) then begin
    Graphic := iGraphicClass.Create;
    try
      Graphic.LoadFromStream(iStream);
      IsImage := True;

      imgImage.Picture.Assign(Graphic);
    finally
      Graphic.Free;
    end;
  end;
end;
```


画像を取得する3

```
begin
  Magic := StringOfChar(#0, 4);
  Move(iStream.Memory^, PChar(Magic)^, Length(Magic));

  AssignImage('GIF', TGIFImage);
  AssignImage(#$ff#$d8#$ff#$e0, TJpegImage);
  AssignImage(#$89'PNG', TPngObject);
```

```
  if (IsImage) then begin
    // 画像だったら
    memoHTML.Lines.Text := '';
    pageHTML.ActivePage := tabImage;
  end
  else begin
    // 画像じゃなかったら
    imgImage.Picture.Assign(nil);

    SetLength(HTML, iStream.Size);
    Move(iStream.Memory^, PChar(HTML)^, Length(HTML));

    memoHTML.Lines.Text := HTML;

    pageHTML.ActivePage := tabSource;
  end;
end;
```

実行結果



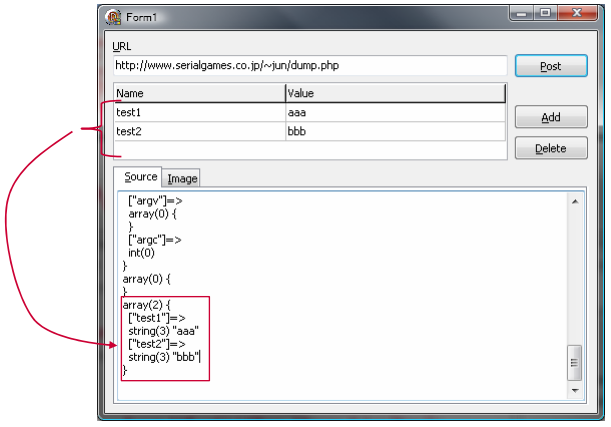
HTTPの引数

- Get
  - Get の場合は URL の後ろに自分で引数を付ける
    - IdHTTP.Get(URL + **'?aaa=bbb&cc=dd'**);
- Post
  - Post の場合は TIdMultiPartFormDataStream を使用する

Post を使う

```
while (not Terminated) and (not Application.Terminated) do begin
  if (FURL <> '') and (FParams.Count > 0) then begin
    FStream.Clear;

    Param := TIdMultiPartFormDataStream.Create;
    try
      for i := 0 to FParams.Count - 1 do begin
        Name := FParams[i];
        Param.AddFormField(Name, FParams.Values[Name]);
      end;

      IdHTTP.Post(FURL, FParams, FStream);
    finally
      Param.Free;
    end;
    FURL := '';
    FParams.Clear;

    if (FStream.Size > 0) then begin
      FStream.Position := 0;
      Synchronize(CallOnPost);
    end;
  end;

  Sleep(100);
end;
```


実行結果



Basic認証

Basic 認証のかかっているページにアクセスするには

- TIdHTTP.Reuqest.UserName
- TIdHTTP.Reuqest.Password
- TIdHTTP.BasicAuthentication

を使用する

```
FIdHTTP.BasicAuthentication := True;
FIdHTTP.Request.UserName := 'ユーザー名';
FIdHTTP.Request.Password := 'パスワード';
```


Proxy を使う

- Proxy は TIdHTTP.ProxyParams に設定する

```
ProxyParams.ProxyServer := プロクシサーバのＵＲＬ;
ProxyParams.ProxyPort := プロクシサーバのポート番号;
ProxyParams.ProxyUsername := プロクシのユーザー名;
ProxyParams.ProxyPassword := プロクシのパスワード;

// パスワードが必要な場合 BasicAuthentication を True にする
ProxyParams.BasicAuthentication := (ProxyParams.ProxyUsername <> '');
```

カスタムヘッダ

HTTP ヘッダに ヘッダを追加するには

TIdHTTP.Request.CustomHeaders

を使用する

```
CustomHeaders.Values['X-PoweredBy'] := 'CodeGear Delphi';
```


タイムアウトの制御

- タイムアウト値の制御は TIdHTTP.IOHandler を使用する

```
FHTTP := TIdHTTP.Create;

with FHTTP do begin
  if (IOHandler = nil) then
    IOHandler := TIdIOHandler.MakeDefaultIOHandler(FHTTP);

  IOHandler.Open;  // 簡便な初期化方法
  IOHandler.Close;

  IOHandler.InputBuffer.Capacity := 1024 * 1024;
  IOHandler.ReadTimeout := 60; // タイムアウト秒
end;
```

TIdTCPClient を使ってみよう

CODE GEAR FROM Borland
Where Developers Matter
Integrated Development Environments for Windows, Java, and Web Developers
DEVELOPER CAMP

TIdTCPClient とは

- TCP (Transmission Control Protocol) のクライアント
  - TCP (トランスポート層) の上に HTTP などのプロトコルが乗っている
- TIdTCPClient を使うと TCP を使ったサーバと通信できる

TIdTCPClient のプロパティ・メソッド

- Host プロパティ
  - サーバのアドレスを指定する
- Port プロパティ
  - 接続先ポートの番号を指定する
- Connect メソッド
  - 接続する
- IOHandler プロパティ
  - 接続後のデータの読み出しなどに使用する

TIdTCPClient を使ってみる

```
procedure ConnectServer(
  const iHost: String;
  const iPort: Integer);
var
  TCPClient: TIdTCPClient;
  Size: Integer;
  Read: String;
begin
  TCPClient := TIdTCPClient.Create;
  try
    TCPClient.Host := iHost;  // サーバー設定
    TCPClient.Port := iPort;  // ポート番号設定

    TCPClient.Connect;        // 接続
    try
      while (True) do begin
        // サーバからの戻り値のサイズ
        Size := TCPClient.ReadFromStack(False, 100, False);
        if (Size > 0) then // サイズが0じゃなかったら読み込む
          Read := TCPClient.IOHandler.InputBuffer.Extract(Size)
        else
          Break;
      end;
    finally
      TCPClient.Disconnect;  // 切断
    end;
  finally
    TCPClient.Free;
  end;

  ShowMessage(Read);  // 読み込んだ結果の表示
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  ConnectServer('127.0.0.1', 3786);
end;
```

TIdTCPClient でデータを送信

- 前述のようにサーバと接続後
  - TIdTCPClient.Write メソッドでデータを送信できる
    - Write メソッドには様々なバリエーションがある
      - 文字列
      - ストリーム
      - 整数型（SmallInt, Int64, Cardinal など）

```
TCPClient.Write('TEST'#0)
```


タイムアウトの制御

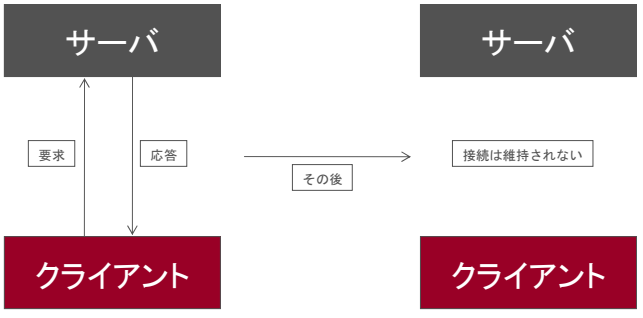
- TIdTCPClient のタイムアウトは2種類
- プロパティに設定するだけで良い
  - ConnectTimeout プロパティ
    - コネクト時のタイムアウト（ミリ秒）
  - ReadTimeout プロパティ
    - 読み出し時のタイムアウト（ミリ秒）

```
TCPClient.ReadTimeout := 60 * 1000; // ６０秒待つ
```

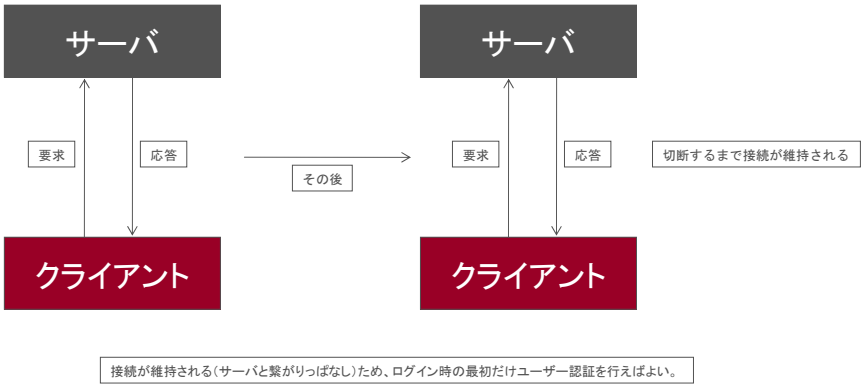
Indy を使って

TIdHTTP, TIdTCPClient を使ったアプリケーション

TIdHTTP で実現される接続形態



TIdTCPClient で実現される接続形態



接続が維持される（サーバと繋がりっぱなし）ため、ログイン時の最初だけユーザー認証を行えばよい。

TIdHTTP を使ったアプリケーション

- Twitter (ミニブログ）用アプリ
  - TIdHTTP を使って Twitter API を叩く
  - Post でこちらの欲しい情報を指定し、戻り値を使う

TIdTCPClient を使ったアプリケーション

- メッセンジャー
  - TIdTCPClient を使ってサーバと常に接続されている
  - サーバからはメンバーのログインや、ログアウト、チャットの情報が流れてくる
  - それを TIdTCPClient で取得し表示する
  - TIdTCPClient.Write を使ってチャットデータを送信する