embarcadero
Developer Camp

【G6】ライトニングトーク

StateパターンをDelphiで実装する

東洋テクニカルシステム株式会社
システム開発部
福士　光

embarcadero
Developer Camp

基本知識

状態とは？

- 状態(State)とは？
  - 同じ入力に対しても状態が異なれば振る舞いも異なる。
    - 同じ話を同じ人にしても、そのときの気分(=状態)で反応が異なる、みたいなこと。

状態の遷移とは？

- 状態の遷移とは
  - 外部からの入力や内部的な動作によって状態が変化してゆくこと。
  - 状態とその遷移を考えるときは、一定の間そこに留まるものだけを『状態』として考える。
    - 一瞬だけその状態に属し、何かをし終わったら別の状態に遷移する、といったものは基本的に状態として扱わない。

デザインパターンとは？

- パターン(パターンランゲージ)とは？
  - もともとは建築家のクリストファー アレグザンダー (Christopher Alexander) が建築物をデザインするときの手法として考案した。
  - ソフトウェアに対してパターンという考え方を適用した。
  - 問題とその解決方法、そして適用した結果やトレードオフ、そしてそれらの組み合わせに名前をつけたもの。
- デザインパターンとは？
  - デザイン(設計)における問題を解決して適切な構造をコードに与えるための手法。

GoFのデザインパターンとは？

- GoF(Gang of Four)のデザインパターンとは？
  - オブジェクト指向の設計に対してデザインパターンを適用。
  - 設計上の課題とその典型的な解決方法(コーディング)に名前をつけたもの。
  - 3つのカテゴリー、23のパターンに分類している。
  - あくまで一つの考え方です(絶対的なものとして捉えないほうがいいでしょう)。
  - GoF本(参考文献(1))の著者であるErich Gamma、Richard Helm、Ralph Johnson、John M. Vlissidesの4人をGang of Fourと呼んでいます。

Stateパターンとは？

- Stateパターンとは？
  - GoFの『振る舞いに関する』パターンの一つ。
  - オブジェクトが内部状態に従って振る舞いを変えるような状況に適用する。
  - 詳細はGoF本(と付属のCD-ROM)をお読みください。
  - Delphiではクラス参照型の変数とその仮想関数が有効に機能するので、C++のように状態毎のインスタンスを生成しない実装が可能(状態の持つ内部情報はコンテキスト側に保持する)。

Stateパターンを使わない場合

- if文やcase文で実装(発生事象と状態で二重になる？)。
- あるいは関数テーブルで実装。
- Stateパターンの実装は関数テーブルをDelphiのVMT(仮想メソッドテーブル)に任せたもの、と考えることができるかも。

embarcadero
Developer Camp

サンプルについて

サンプルについて

- 今回のサンプルは…
  TCP/IPで接続して郵便番号を問い合わせると該当する住所を返してくれるサーバがあり、このサーバとの通信を行う、という要求を想定する。

  応答には時間が掛かるかもしれないし、エラー(応答が不正だったりタイムアウトしたり)が発生するかもしれない。

サンプルについて

- ポート番号は32100(適当)。
- 手抜きです。電文のデリミタはCR(0x0D)。
- 要求電文

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10
0	0	0	0	SP	G	E	T	SP	#	#	#	#	#	#	#	CR
シーケンス番号					要求				郵便番号(7桁)							

- 回答電文

00	01	02	03	04	05	06	07	08	09	…	…	…	…	…	…	…
0	0	0	0	SP	O	K	SP	SP	X	…	…	…	…	…	…	CR
シーケンス番号					応答				住所(Shift_JIS)							

00	01	02	03	04	05	06	07	08	09	…	…	…	…	…	…	…
0	0	0	0	SP	N	G	SP	SP	X	…	…	…	…	…	…	CR
シーケンス番号					応答				メッセージ							

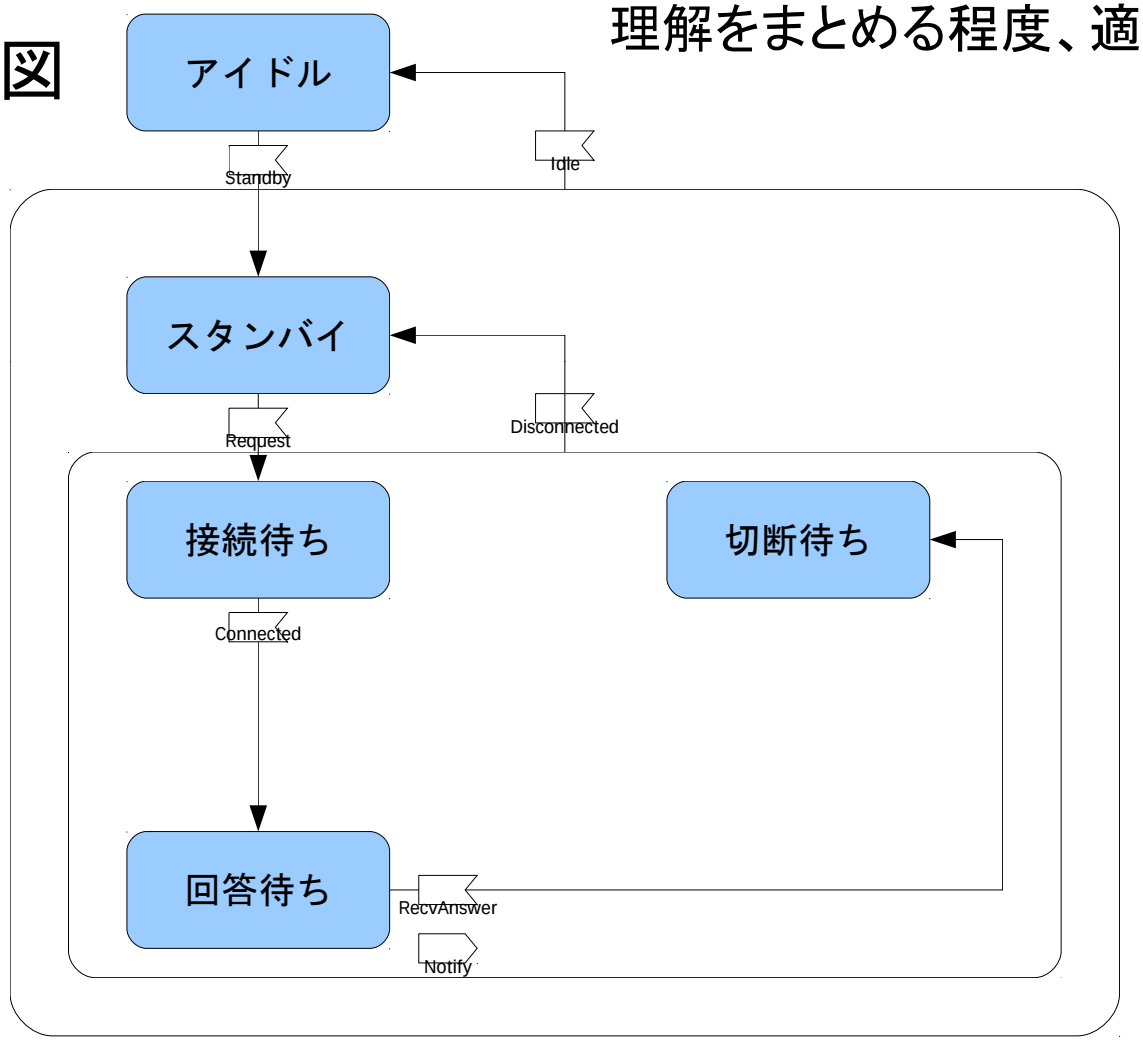
embarcadero
Developer Camp

分析

分析

- 状態遷移図

理解をまとめる程度、適当でよい



embarcadero
Developer Camp

設計

設計

- 状態マトリクス
  - 縦方向に状態(→状態クラスとして実装)を、横方向に事象(→メソッドとして実装)を配置
  - グループ化された状態も考慮しておくとよい

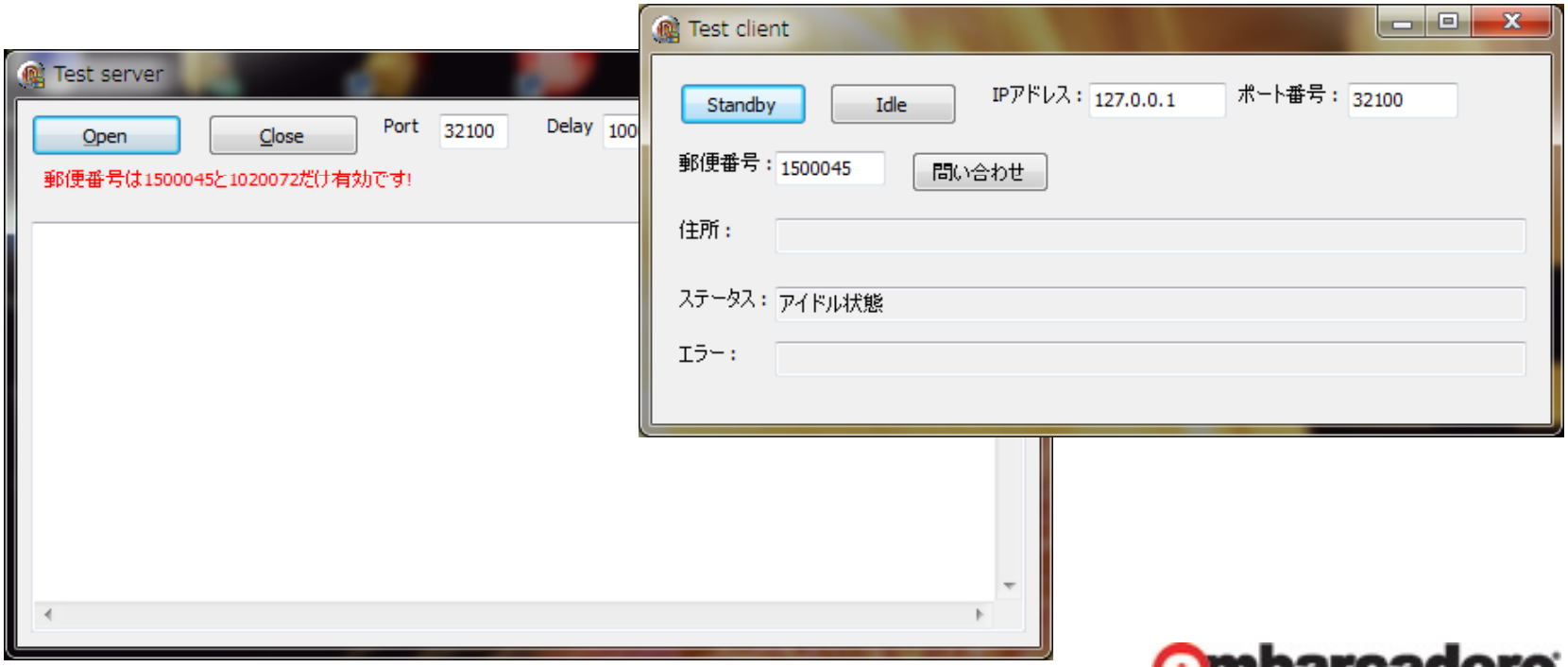
No	状態	Entry	Exit	Go Idle	Go StandBy	Request addr	Connected	Receive answer	Disconnected	Error
0	Idle	N/A	N/A	無視	→1	無視	無視	無視	無視	無視
1	Stand by	タイマ停止	N/A	→0	無視	→2	無視	無視	無視	無視
	(Connected)	タイマ開始	N/A	切断 →0	無視	無視	無視	無視	→1	エラー通知 →1
2	Wait to connect	タイマ開始 接続	N/A	切断 →0	無視	無視	要求送信 →3	無視	→1	エラー通知 →1
3	Wait answer	タイマ開始	N/A	切断 →0	無視	無視	無視	回答通知 →4	→1	エラー通知 →1
4	Wait to disconect	タイマ開始 切断	N/A	切断 →0	無視	無視	無視	無視	→1	エラー通知 →1

embarcadero
Developer Camp

実装

実装

- 画面
  - 都合によりDelphi 2007です
  - ソースコードはダウンロードできるようにする予定です(テスト用サーバも)



実装 (状態クラスの宣言/1)

```
{ TCommStatus : Status class (abstract) }
 TCommStatus = class(TObject)
 public
   class function GetDescription: String; virtual;
   class procedure EntryState(DM: TDataModuleComm); virtual;
   class procedure ExitState(DM: TDataModuleComm); virtual;
   class procedure GoIdle(DM: TDataModuleComm); virtual;
   class procedure GoStandby(DM: TDataModuleComm); virtual;
   class procedure RequestAddr(DM: TDataModuleComm; const APostalCode: String); virtual;
   class procedure Connected(DM: TDataModuleComm; Socket: TCustomWinSocket); virtual;
   class procedure ReceiveAnswer(DM: TDataModuleComm; Socket: TCustomWinSocket;
                                 const RecvData: String); virtual;
   class procedure Disconnected(DM: TDataModuleComm; Socket: TCustomWinSocket); virtual;
   class procedure SocketError(DM: TDataModuleComm; Socket: TCustomWinSocket;
                               ErrorEvent: TErrorEvent; var ErrorCode: Integer); virtual;
   class procedure ReceiveError(DM: TDataModuleComm; Socket: TCustomWinSocket;
                                const Data: String); virtual;
   class procedure TimeoutError(DM: TDataModuleComm); virtual;
 end;
```

実装 (状態クラスの宣言/2)

```
type
  { TCommStatusIdle : Idle }
  TCommStatusIdle = class(TCommStatus)
  public
    class function GetDescription: String; override;
    class procedure GoStandby(DM: TDataModuleComm); override;
  end;

  { TCommStatusStandby : Standby }
  TCommStatusStandby = class(TCommStatus)
  public
    class function GetDescription: String; override;
    class procedure EntryState(DM: TDataModuleComm); override;
    class procedure GoIdle(DM: TDataModuleComm); override;
    class procedure RequestAddr(DM: TDataModuleComm; const APostalCode: String); override;
  end;

  { TCommStatusConnected : Connencted (abstract) }
  TCommStatusConnected = class(TCommStatus)
  public
    class procedure EntryState(DM: TDataModuleComm); override;
    class procedure GoIdle(DM: TDataModuleComm); override;
    class procedure Disconnected(DM: TDataModuleComm; Socket: TCustomWinSocket); override;
    class procedure SocketError(DM: TDataModuleComm; Socket: TCustomWinSocket;
                                ErrorEvent: TErrorEvent; var ErrorCode: Integer); override;
    class procedure ReceiveError(DM: TDataModuleComm; Socket: TCustomWinSocket;
                                 const Data: String); override;
    class procedure TimeoutError(DM: TDataModuleComm); override;
  end;
```

実装 (状態クラスの宣言/3)

```
type
  { TCommStatusWaitToConnect : Wait to connenct }
  TCommStatusWaitToConnect = class(TCommStatusConnected)
  public
    class function GetDescription: String; override;
    class procedure EntryState(DM: TDataModuleComm); override;
    class procedure Connected(DM: TDataModuleComm; Socket: TCustomWinSocket); override;
  end;

  { TCommStatusWaitToAnswer : Wait answer }
  TCommStatusWaitAnswer = class(TCommStatusConnected)
  public
    class function GetDescription: String; override;
    class procedure ReceiveAnswer(DM: TDataModuleComm; Socket: TCustomWinSocket;
                                  const RecvData: String); override;
  end;

  { TCommStatusWaitToDisconnect :Wait to disconnect }
  TCommStatusWaitToDisconnect = class(TCommStatusConnected)
  public
    class function GetDescription: String; override;
    class procedure EntryState(DM: TDataModuleComm); override;
  end;
```

実装 (状態クラスの定義/1)

```
{ TCommStatusIdle }
class function TCommStatusIdle.GetDescription: String;
begin
  Result := 'アイドル状態';
end;

class procedure TCommStatusIdle.GoStandby(DM: TDataModuleComm);
begin
  DM.CommStatus := TCommStatusStandby;
end;

{ TCommStatusStandby }
class function TCommStatusStandby.GetDescription: String;
begin
  Result := 'スタンバイ状態';
end;

class procedure TCommStatusStandby.EntryState(DM: TDataModuleComm);
begin
  DM.DoStopTimer;
end;

class procedure TCommStatusStandby.GoIdle(DM: TDataModuleComm);
begin
  DM.CommStatus := TCommStatusIdle;
end;

class procedure TCommStatusStandby.RequestAddr(DM: TDataModuleComm; const APostalCode: String);
begin
  DM.FPostalCode := APostalCode;
  DM.CommStatus := TCommStatusWaitToConnect;
end;
```

実装 (状態クラスの定義/2)

```
{ TCommStatusConnected }
class procedure TCommStatusConnected.EntryState(DM: TDataModuleComm);
begin
  DM.DoStartTimer;
end;

class procedure TCommStatusConnected.GoIdle(DM: TDataModuleComm);
begin
  DM.DoDisconnect;
  DM.CommStatus := TCommStatusIdle;
end;

class procedure TCommStatusConnected.Disconnected(DM: TDataModuleComm; Socket: TCustomWinSocket);
begin
  DM.CommStatus := TCommStatusStandby;
end;

class procedure TCommStatusConnected.SocketError(DM: TDataModuleComm; Socket: TCustomWinSocket;
                                                 ErrorEvent: TErrorEvent; var ErrorCode: Integer);
begin
  DM.NotifySocketError(Socket,ErrorEvent,ErrorCode);
  ErrorCode := 0;
  DM.DoDisconnect;
  DM.CommStatus := TCommStatusStandby;
end;

class procedure TCommStatusConnected.ReceiveError(DM: TDataModuleComm; Socket: TCustomWinSocket;
                                                  const Data: String);
begin
  DM.NotifyReceiveError(Socket,Data);
  DM.DoDisconnect;
  DM.CommStatus := TCommStatusStandby;
end;
```

実装 (状態クラスの定義/3)

```
class procedure TCommStatusConnected.TimeoutError(DM: TDataModuleComm);
begin
  DM.NotifyTimeoutError;
  DM.DoDisconnect;
  DM.CommStatus := TCommStatusStandby;
end;

{ TCommStatusWaitToConnect }
class function TCommStatusWaitToConnect.GetDescription: String;
begin
  Result := '接続待機状態';
end;

class procedure TCommStatusWaitToConnect.EntryState(DM: TDataModuleComm);
begin
  inherited;
  DM.DoConnect;
end;

class procedure TCommStatusWaitToConnect.Connected(DM: TDataModuleComm; Socket: TCustomWinSocket);
begin
  DM.DoSendRequest(Socket);
  DM.CommStatus := TCommStatusWaitAnswer;
end;
```

実装 (状態クラスの定義/4)

```
{ TCommStatusWaitAnswer }
class function TCommStatusWaitAnswer.GetDescription: String;
begin
  Result := '回答電文受信待ち状態';
end;

class procedure TCommStatusWaitAnswer.ReceiveAnswer(DM: TDataModuleComm; Socket: TCustomWinSocket; const
RecvData: String);
begin
  DM.NotifyReceiveData(Socket,RecvData);
  DM.CommStatus := TCommStatusWaitToDisconnect;
end;

{ TCommStatusWaitToDisconnect }
class function TCommStatusWaitToDisconnect.GetDescription: String;
begin
  Result := '切断待機状態';
end;

class procedure TCommStatusWaitToDisconnect.EntryState(DM: TDataModuleComm);
begin
  inherited;
  DM.DoDisconnect;
end;
```

実装 (データモジュールの宣言/1)

```
type
TCommStatus = class;
  TCommStatusClass = class of TCommStatus;

  TNotifyReceiveEvent = procedure (Sender: TObject; const RecvData: String) of object;
  TNotifyMessageEvent = procedure (Sender: TObject; const Msg: String) of object;

  TDataModuleComm = class(TDataModule)
    ClientSocket: TClientSocket;
    TimerTimeout: TTimer;
    procedure DataModuleCreate(Sender: TObject);
    procedure DataModuleDestroy(Sender: TObject);
    procedure ClientSocketConnect(Sender: TObject; Socket: TCustomWinSocket);
    procedure ClientSocketDisconnect(Sender: TObject; Socket: TCustomWinSocket);
    procedure ClientSocketRead(Sender: TObject; Socket: TCustomWinSocket);
    procedure ClientSocketError(Sender: TObject; Socket: TCustomWinSocket; ErrorEvent: TErrorEvent;
                                var ErrorCode: Integer);
    procedure TimerTimeoutTimer(Sender: TObject);
  private
    FCommStatus: TCommStatusClass;
    FRecvBuf: String;
    FSeqNo: String;
    FPostalCode: String;
    FOnStatusChanged: TNotifyEvent;
    FOnReceive: TNotifyReceiveEvent;
    FOnError: TNotifyMessageEvent;
    procedure SetCommStatus(const Value: TCommStatusClass);

(to be continued...)
```

実装 (データモジュールの宣言/2)

```
protected
  procedure DoConnect;
  procedure DoDisconnect;
  procedure DoSendRequest(Socket: TCustomWinSocket);
  procedure DoStartTimer;
  procedure DoStopTimer;
  procedure NotifyReceiveData(Socket: TCustomWinSocket; const RecvData: String);
  procedure NotifySocketError(Socket: TCustomWinSocket; ErrorEvent: TErrorEvent; ErrorCode: Integer);
  procedure NotifyReceiveError(Socket: TCustomWinSocket; const Data: String);
  procedure NotifyTimeoutError;
  procedure DoStatusChanged; virtual;
  procedure DoReceive(const RecvData: String); virtual;
  procedure DoError(const Msg: String); virtual;
public
  procedure SetServerParams(const IPAddress: String; const Port: String);
  procedure GoIdle;
  procedure GoStandby;
  procedure SendRequest(const APostalCode: String);
  property CommStatus: TCommStatusClass read FCommStatus write SetCommStatus;
  property OnStatusChanged: TNotifyEvent read FOnStatusChanged write FOnStatusChanged;
  property OnReceive: TNotifyReceiveEvent read FOnReceive write FOnReceive;
  property OnError: TNotifyMessageEvent read FOnError write FOnError;
end;
```

実装 (データモジュールの定義/1)

```
procedure TDataModuleComm.ClientSocketRead(Sender: TObject; Socket: TCustomWinSocket);
var
  Recv: String;
  Position: Integer;
  RecvData: String;
  SeqNo: String;
  Response: String;
begin
  while True do
  begin
    Recv := Socket.ReceiveText;
    if Recv = '' then
    begin
      Exit;
    end;

    FRecvBuf := FRecvBuf + Recv;
    while FRecvBuf <> '' do
    begin
      Position := AnsiPos(#13,FRecvBuf);
      if Position <= 0 then
      begin
        Break;
      end;
      RecvData := Copy(FRecvBuf,1,Position - 1);
      Delete(FRecvBuf,1,Position);
      SeqNo := Copy(RecvData,1,4);
      Delete(RecvData,1,5);

(to be continued...)
```

実装 (データモジュールの定義/2)

```
      if SeqNo = FSeqNo then
      begin
        Response := Copy(RecvData,1,2);
        Delete(RecvData,1,5);
        if Response = 'OK' then
        begin
          CommStatus.ReceiveAnswer(Self,Socket,RecvData);
        end
        else
        begin
          CommStatus.ReceiveError(Self,Socket,RecvData);
        end;
      end;
    end;
  end;
end;
```

実装 (フォーム)

```
procedure TFormClient.Button1Click(Sender: TObject);
begin
  FDataModuleComm.GoStandby;
end;

procedure TFormClient.Button2Click(Sender: TObject);
begin
  FDataModuleComm.GoIdle;
end;

procedure TFormClient.Button3Click(Sender: TObject);
begin
  FDataModuleComm.SetServerParams(Edit1.Text,Edit2.Text);
  FDataModuleComm.SendRequest(Edit3.Text);
end;

procedure TFormClient.CommStatusCahnged(Sender: TObject);
begin
  Edit5.Text := FDataModuleComm.CommStatus.GetDescription;
end;

procedure TFormClient.CommReceive(Sender: TObject; const RecvData: String);
begin
  Edit4.Text := RecvData;
  Edit6.Text := '';
end;

procedure TFormClient.CommError(Sender: TObject; const Msg: String);
begin
  Edit6.Text := Msg;
  Edit4.Text := '';
end;
```

embarcadero
Developer Camp

デモンストレーション

embarcadero
Developer Camp

参考文献

参考文献 (1)

- オブジェクト指向における再利用のための
  デザインパターン(改訂版)
  - Erich Gamma、Richard Helm、Ralph Johnson、
    John M. Vlissides著
  - 本位田 真一、吉田 和樹監訳
  - ソフトバンククリエイティブ
  - ISBN4-7973-1112-6 (ISBN978-4797311129)
  - 5,040円
    - http://www.sbcr.jp/products/4797311126.html
    - http://www.amazon.co.jp/dp/4797311126

参考文献 (2)

- Head Firstデザインパターン
  - Eric Freeman、Elisabeth Freeman、Kathy Sierra、Bert Bates著
  - 木下 哲也、有限会社 福龍興業訳
  - 佐藤 直生監訳
  - ソフトバンククリエイティブ
  - ISBN4-87311-249-4
  - 4,830円
    - http://www.oreilly.co.jp/books/4873112494/
    - http://www.amazon.co.jp/dp/4873112494

embarcadero
Developer Camp

Q & A

想定される質問とその回答 (1)

- サンプルプロジェクトを開こうとするとエラーになるんですけど。

  →設計時パッケージ“CodeGear Socket Components”をIDEにインストールする必要があります。IDEのメインメニューからコンポーネント→パッケージのインストールでDelphiのインストール先のbinディレクトリにある“dclsocketsXXX.bpl”(Delphi 2007なら“dclsockets100.bpl”)を追加してから、サンプルプロジェクトを開きなおしてください。

- Delphi 2007以外ではサンプルプロジェクトは動作しないんですか？

  →Delphi 2007およびそれ以前のバージョンであれば基本的に動作するはずです。Delphi 2009以降ではString = UnicodeStringとなっている関係から、そのままではコンパイルすら通りません(申し訳ない)。TClientSocket/TServerSocketとインタフェースしている部分をAnsiStringとすることで動くのではないかと思いますが…。

想定される質問とその回答 (2)

- LTでの説明がはしょりすぎでよくわからなかったんですけど。

  →すいませんすいませんすいません

  とりあえずこの資料とサンプルコード、できれば参考文献のHead Firstデザインパターンをよくお読みください。その上での不明点は公式フォーラムかDelphi-mlで質問していただければと思います。

  - Embarcadero Discussion Forums: Delphi

    https://forums.embarcadero.com/forum.jspa?forumID=14

  - Delphi | freeml

    http://www.freeml.com/delphi-users

想定される質問とその回答 (3)

- サンプルプログラムの使い方を教えてください。

  →まずテスト用サーバ(TestSvr.exe)を起動します。待機ポートを確認してOpenボタンをクリックするとサーバソケットがlisten状態になります。Closeボタンでlisten状態が解除されます。

  - Delayには電文を受信してから回答を送信するまでの時間をmsec単位で指定します。また“シーケンス番号を置き換える”チェックボックスをチェックオンにすると回答電文に含まれるシーケンス番号が受信した要求電文と一致しなくなります。
  - 実装は手抜きされているので、“1500045”と“1020072”以外の郵便番号はエラーになります。

想定される質問とその回答 (4)

→次にクライアント(TestClient.exe)を起動します。起動直後はアイドル状態になっていますので、Standbyボタンをクリックしてスタンバイ状態に移行します。ここでサーバのIPアドレス、ポート番号を確認して問い合わせたい郵便番号(7桁)を入力し、問い合わせボタンをクリックするとサーバへの問い合わせのシーケンスが発動します。正常に回答を受信したときは住所が表示されます。またエラーのときはエラーメッセージが表示されます。なおクライアント側の通信タイムアウトは5000msec固定です。

想定される質問とその回答 (5)

- どのくらい複雑なケースならStateパターンを適用するべきでしょうか？
  →もちろんケースバイケースですが、Idleステート、Standbyステートを含め4状態以上ある場合は適用することを考慮していいのでは？というのが個人的な意見です。
- Idleステートの存在意義がわかりません。
  →プログラム起動直後やプログラムの環境設定を行っているとき、あるいはプログラムを終了しようとしているときなど、一連のシーケンスが発動してほしくないときに状態をIdleステートにしておく、という使い方を想定しています。
- エラー発生時にリトライしないとかずいぶん手抜きじゃないですか？
  →Stateパターンを考える上での本質ではないので外しました。もちろん現実のプロジェクトに適用するときはエラー発生時のリトライや、必要ならその際のウェイトなどもシーケンスに組み込む形で設計、 実装されるべきだと思います。

想定される質問とその回答 (6)

- ところでIDEに標準でインストールされていないTClientSocket/TServerSocketコンポーネントを使うのはいかがなものかと。

  →TClientSocket/ TServerSocketコンポーネントを使用するのは確かにあまりお勧めできません(SMP環境下で不具合があるとかいう話を聞いたことがあります)。本当はIndyにするべきなんでしょうけれども、Indyはブロッキング動作が基本なのでUIとはスレッドを分ける必要があるとか、非ブロッキング動作のWinSockをIndyでわざわざブロッキング動作にしているものをまた非ブロッキング動作にして使うのはどうなんだろう？とか、いろいろあってこういうサンプルになっています。有償でもいいので非ブロッキングで使用できるお勧めのSocket通信コンポーネントはありませんか？

embarcadero
Developer Camp

ありがとうございました