【A5】Delphi/C++テクニカルセッション

EMBARCADERO TECHNOLOGIES®

DEVELOPER CAMP

詳説！DataSnap 2010

エンバカデロ・テクノロジーズ
エヴァンジェリスト　高橋智宏

アジェンダ

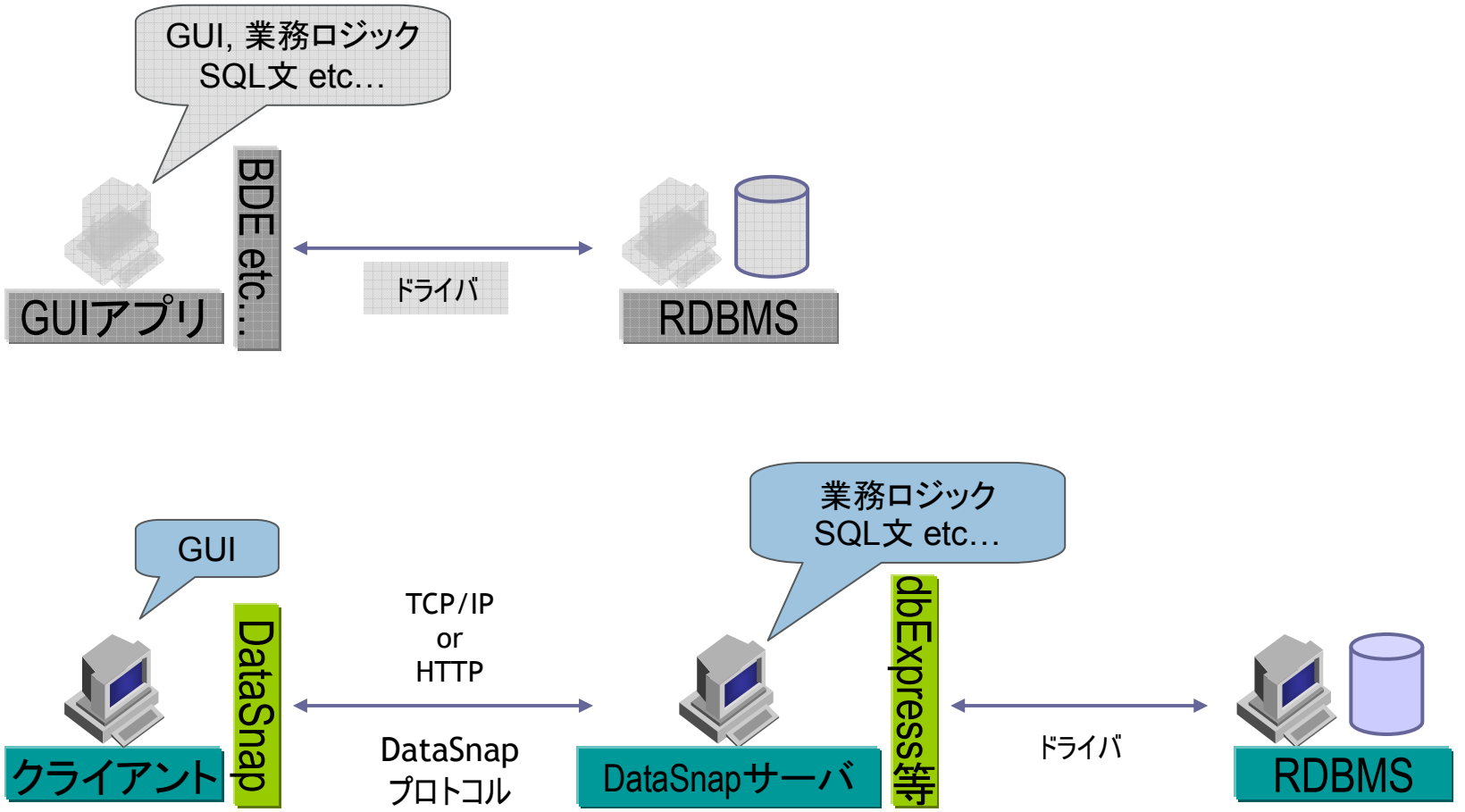
EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- DataSnap 2010(2009)とは?
  - プロトコル
  - データ型
- フィルターの使い方 (Demo)
  - ZLibCompressionフィルター + カスタムフィルター
  - サーバー <----> クライアント
- C++Builder 2010 での開発手順 (Demo)
  - サーバーメソッド
  - プロキシ
- コネクションプーリング (Advanced !)
  - ライフサイクル
  - コンポーネント化

DataSnap 2010(2009)とは?

DataSnapサーバ

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP



DataSnapプロトコル

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- TCP/IP
  - デフォルト211番ポート
  - SSL, IPv6 は未サポート
- HTTP
  - デフォルト80番ポート
  - SSLはIIS上のISAPI/CGIを利用
- RPC形式
  - サーバーメソッド
    - **要は、RDBMSのストアドプロシージャと同じ扱い!!**
    - パラメータ
      - in, var, out, inout, return
    - 組み込みの非同期呼び出しは無い
  - インスタンスのライフサイクル
    - Server, Session, Invocation
  - Callback, Filter, Event
- データフォーマット
  - JSON(JavaScript Object Notation)

クライアントからのリクエストのサンプル

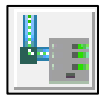
```
{"method":"connect","params":"drivername":"DATASNAP",
"port":"211","communicationprotocol":"tcp/ip","hostna
me":"127.0.0.1","DriverUnit":"DbxDatasnap",....]}
{"method":"execute","params":[{"fields":[-
1,false,"Dbx.MetaData","GetDatabase"]}]}
{"method":"reader_close","params":[1,0]}
{"method":"disconnect","params":[0]}
```

サーバーからのレスポンスのサンプル

```
{"result":[0,"DataSnap",2]}
{"result":[{"rows":[0]},{"data":[1,`]},{"table":[{"fi
elds":[0,false,1,0,true]},{"columns":[13,["QuoteChar"
,26,0,0,0,16,0,0],["ProcedureQuoteChar",26,0,0,0,16,0
,0],["MaxCommands",6,0,0,0,0,0,0],["SupportsTransacti
ons",4,0,0,0,0,0,0],["SupportsNestedTransactions",4,0
,0,0,0,0,0],["SupportsRowSetSize",4,0,0,0,0,0,0]...
...
...
```

DataSnapサーバーを構成するコンポーネント

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- TDSServer 
  - すべての他のDataSnapコンポーネントを結び付けるために必要なメインのサーバー設定コンポーネント
- TDSServerClass 
  - 公開するクラスごとに必要なコンポーネント。このコンポーネントは、実際に利用可能にするクラスでなく、リモートクライアントから呼び出したいクラスオブジェクトを作成するクラスファクトリとして動作します。
- TDSTCPServerTransport 
  - 転送プロトコルと、使用するTCP/IPポートなどの設定を定義するコンポーネント
- TDSHTTPService 
  - 転送プロトコルと、使用するHTTPポートなどの設定を定義するコンポーネント

サーバーメソッドで扱える型

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

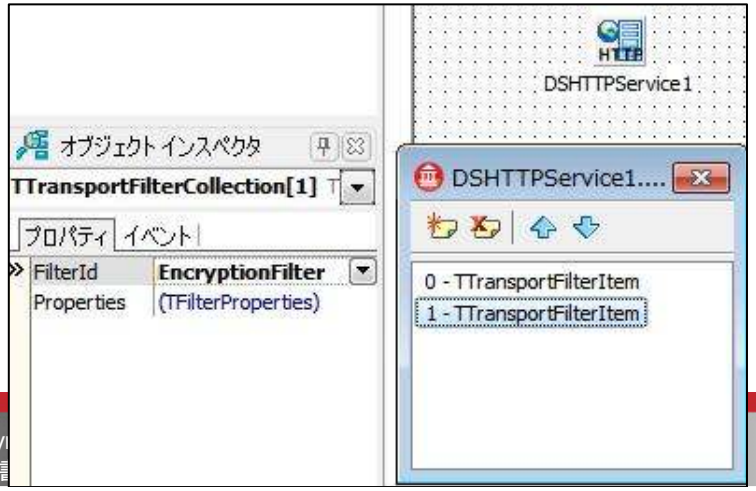
- 基本型
  - AnsiString
  - Boolean
  - Byte
  - Currency
  - TDateTime
  - TDBXDate
  - TDBXTime
  - Double
  - Int64
  - Integer
  - LongInt
  - OleVariant
  - Single
  - ShortInt
  - SmallInt
  - String(UnicodeString)
  - WideString
  - TStream
- DBXValue型
  - TDBXAnsiStringValue
  - TDBXAnsiCharsValue
  - TDBXBcdValue
  - TDBXBooleanValue
  - TDBXDateValue
  - TDBXDoubleValue
  - TDBXUInt8Value
  - TDBXInt8Value
  - TDBXInt16Value
  - TDBXInt32Value
  - TDBXInt64Value
  - TDBXJSonValue
  - TDBXReaderValue
  - TDBXSingleValue
  - TDBXStringValue
  - TDBXTimeStampValue
  - TDBXTimeValue
  - TDBXWideCharsValue
  - TDBXWideStringValue
  - TDBXStreamValue
- テーブル型
  - TDataSet
  - TParams
  - TDBXReader
- JSON型(REST用)
  - TJSONArray
  - TJSONNumber
  - TJSONObject
  - TJSONString
  - TJSONValue

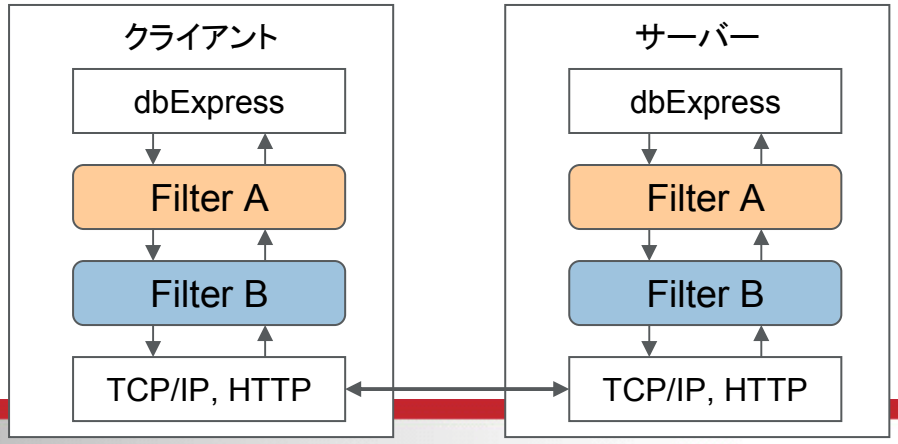
基本型	NULLサポート無し。 varや戻り値もOK
DBXValue型	inout型。NULLサポートあり。 varや戻り値はNG
テーブル型	inout型。 varや戻り値もOK

フィルターの使い方

フィルターとは?

- DataSnapで送受信されるメッセージ本体を、可変長のバイト配列(TBytes)としてトラップし、変換できる
  - フィルターの設定はサーバーでのみ行う(クライアントはサーバーに従う)
    - TDSTCPServerTransportコンポーネントのFiltersプロパティ
    - TDSHTTPServiceコンポーネントのFiltersプロパティ
- Filtersプロパティに、FilterIdを登録する
  - フィルターはチェイン可能
  - 予めフィルターにプロパティを設定可能(サーバーからクライアントに伝達)
  - 予め用意されているフィルター: 'ZLibCompression'

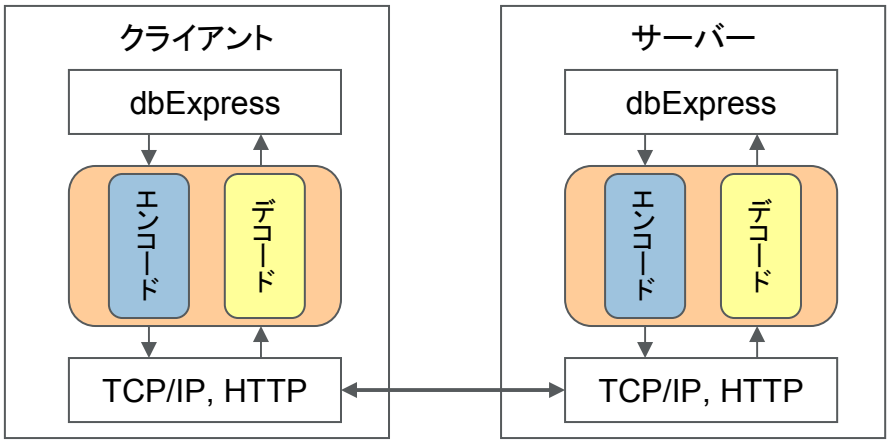




カスタムフィルターの設計

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- DBXTransportユニットのTTransportFilterクラスから派生
  - 7個のメソッドをオーバーライドして実装
    - ProcessInputメソッドでエンコードし、ProcessOutputメソッドでデコードする
      - TBytes(=array of Byte)をパラメータとして受け取り、新しいTBytesを返す
- カスタムフィルターのクラス(.pas)は、カスタムパッケージ(.bpl)内に格納してIDEにインストールする便利
  - IDE内で、カスタムフィルターが認識されるようになる



カスタムフィルターの定義

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

DSFEncryptClasses.pas

```
type
  TTransportEncryptFilter = class(TTransportFilter)
  private
    FParameters: TDictionary<String, String>;
  protected
    function GetParameters: TDBXStringArray; override;
    function GetUserParameters: TDBXStringArray; override;
  public
    constructor Create; override;
    destructor Destroy; override;
    function GetParameterValue(const ParamName: UnicodeString): UnicodeString; override;
    function SetParameterValue(const ParamName: UnicodeString; const ParamValue: UnicodeString): Boolean; override;
    function ProcessInput(const Data: TBytes): TBytes; override;
    function ProcessOutput(const Data: TBytes): TBytes; override;
    function Id: UnicodeString; override;
  end;

const
  EncryptFilterName = 'EncryptionFilter';
  EncryptKey = 'Key';
```

```
initialization
  TTransportFilterFactory.RegisterFilter(EncryptFilterName, TTransportEncryptFilter);
finalization
  TTransportFilterFactory.UnregisterFilter(EncryptFilterName);
end.
```

カスタムフィルターの実装

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

DSFEncryptClasses.pas

```
constructor TTransportEncryptFilter.Create;
begin
  inherited;
  FParameters := TDictionary<String, String>.Create;
end;

destructor TTransportEncryptFilter.Destroy;
begin
  FreeAndNil(FParameters);
  inherited;
end;

function TTransportEncryptFilter.Id: UnicodeString;
begin
  Result := EncryptFilterName;
end;

function TTransportEncryptFilter.SetParameterValue(
      const ParamName, ParamValue: UnicodeString): Boolean;
begin
  FParameters.AddOrSetValue(ParamName, ParamValue);
  Result := True;
end;

function TTransportEncryptFilter.GetParameterValue(
       const ParamName: UnicodeString): UnicodeString;
begin
  FParameters.TryGetValue(ParamName, Result);
  if (ParamName = EncryptKey) and (Result = '') then
    Result := '0'; // フィルターのパラメータのデフォルト値
end;
```

```
function TTransportEncryptFilter.GetUserParameters: TDBXStringArray;
begin
  SetLength(Result, 1);
  Result[0] := EncryptKey;
end;

function TTransportEncryptFilter.GetParameters: TDBXStringArray;
begin
  SetLength(Result, 1);
  Result[0] := EncryptKey;
end;

function TTransportEncryptFilter.ProcessInput(const Data: TBytes): TBytes;
var
  param: UnicodeStrting
begin
  param := GetParameterValue(EncryptKey);
  Result := ...; // エンコード処理
end;

function TTransportEncryptFilter.ProcessOutput(const Data: TBytes):
TBytes;
var
  param: UnicodeStrting
begin
  param := GetParameterValue(EncryptKey);
  Result := ...; // デコード処理
end;
```

カスタムフィルターをセットする

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- フィルターは、サーバー側でセットアップする
  - デザイン時でも実行時でもOK
- uses節にフィルターのユニットを追加
  - サーバー、クライアントともに必須

サーバー側のコード(実行時の場合)

```
uses
  ..., DSFEncryptClasses, DbxCompressionFilter, ...
...
procedure TServerForm.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  // フィルターを実行時にセット
  i := DSTCPServerTransport1.Filters.AddFilter('EncryptionFilter');
  DSTCPServerTransport1.Filters.GetFilter(i).SetParameterValue('Key', '100');

  // セットされているフィルターのチェインを確認
  for i := 0 to DSTCPServerTransport1.Filters.Count - 1 do
    ListBox1.Items.Add(DSTCPServerTransport1.Filters.GetFilter(i).Id);
end;
```

クライアント側のコード(フィルターのセットアップは自動で行われる)

```
uses
  ..., DSFEncryptClasses, DbxCompressionFilter, ...
```

C++Builder 2010 での開発手順

C++版DataSnapサーバーの設計

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- サーバーメソッドを公開するクラスをC++のクラスでは直接記述できない、という制限あり
  - C++Builderでは「.pasファイル」をコンパイル可能なので、この仕組みを活用する
  - 「.pasファイル」で通常のサーバークラス&サーバーメソッドを定義する
  - サーバークラスから処理を委譲する抽象クラスを、同じ「.pasファイル」内に定義する
  - C++では、上の抽象クラスを実装したクラスを作成する
  - サーバークラスのインスタンスの生成と破棄および、C++で実装したクラスのインスタンスの生成と破棄は、自前で行う
    - TDSServerClassクラスのOnCreateInstanceイベント、OnDestroyInstanceイベントを利用する
- その他のサーバー機能は、VCLを利用して構築可能

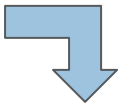
C++版DataSnapサーバーの実装

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

ServerMethodsUnit1.pas

```
unit ServerMethodsUnit1;

interface

uses
  SysUtils, Classes, DSServer;

type
  TAbstractServerMethods = class
  public
    function EchoString(Value: string): string; virtual; abstract;
  end;

{$METHODINFO ON}
  TServerMethods1 = class(TPersistent)
  public
    impl: TAbstractServerMethods;
    function EchoString(Value: string): string;
  end;
{$METHODINFO OFF}

implementation

function TServerMethods1.EchoString(Value: string
begin
  Result := impl.EchoString(Value);
end;

end.
```

コンパイルすると、C++用のヘッダファイル(.hpp)が生成される



ServerMethodsUnit1.hpp を抜粋したもの

```
namespace Servermethodsunit1 {
class PASCALIMPLEMENTATION TAbstractServerMethods : public System::TObject {
public:
  virtual System::UnicodeString __fastcall EchoString(System::UnicodeString Value) = 0 ;
public:
  __fastcall TAbstractServerMethods(void) : System::TObject() { }
  __fastcall virtual ~TAbstractServerMethods(void) { }
};

class PASCALIMPLEMENTATION TServerMethods1 : public Classes::TPersistent {
public:
  TAbstractServerMethods* impl;
  System::UnicodeString __fastcall EchoString(System::UnicodeString Value);
public:
  __fastcall TServerMethods1(void) : Classes::TPersistent() { }
  __fastcall virtual ~TServerMethods1(void) { }
};
}
```

C++版DataSnapサーバーの実装(続き)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

サーバーメソッドの実装クラス

```
#include "ServerMethodsUnit1.hpp"

class TServerMethodsImpl : public TAbstractServerMethods {
public:
  __fastcall TServerMethodsImpl() : TAbstractServerMethods() { }
  __fastcall virtual ~TServerMethodsImpl() { }
public:
  System::UnicodeString __fastcall EchoString(System::UnicodeString Value) {
    return Value;
  }
};
```

TDSServerClassクラスのOnCreateInstanceイベント、OnDestroyInstanceイベントを利用する

```
void __fastcall TForm1::DSServerClass1GetClass(TDSServerClass *DSServerClass,
                                               TPersistentClass &PersistentClass)
{
  PersistentClass = __classid(TServerMethods1);
}

void __fastcall TForm1::DSServerClass1CreateInstance(TDSCreateInstanceEventObject *DSCreateInstanceEventObject)
{
  TServerMethods1* inst = new TServerMethods1();
  inst->impl = new TServerMethodsImpl();
  DSCreateInstanceEventObject->ServerClassInstance = inst;
}

void __fastcall TForm1::DSServerClass1DestroyInstance(TDSDestroyInstanceEventObject *DSDestroyInstanceEventObject)
{
  TServerMethods1* inst = (TServerMethods1 *)DSDestroyInstanceEventObject->ServerClassInstance;
  delete inst->impl;
  delete inst;
}
```

C++版DataSnapクライアント

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- Delphiと同様に、TSQLConnectionでDataSnapサーバーに接続する
- サーバーメソッドの呼び出し
  - Delphi同様、TSQLServerMethodコンポーネントを利用する
  - クライアント用に自動生成されるプロキシ(.h, .cpp)を利用する

自動生成されたプロキシのヘッダ

```
#ifndef Unit2H
#define Unit2H
#include "DBXCommon.hpp"
#include "Classes.hpp"
#include "SysUtils.hpp"
#include "DB.hpp"
#include "SqlExpr.hpp"
#include "DBXDBReaders.hpp"
  class TServerMethods1Client : public TObject {
  private:
    TDBXConnection *FDBXConnection;
    bool FInstanceOwner;
    TDBXCommand *FEchoStringCommand;
  public:
    __fastcall TServerMethods1Client::TServerMethods1Client(TDBXConnection *ADBXConnection);
    __fastcall TServerMethods1Client::TServerMethods1Client(TDBXConnection *ADBXConnection, bool AInstanceOwner);
    __fastcall TServerMethods1Client::~TServerMethods1Client();
    System::UnicodeString __fastcall EchoString(System::UnicodeString Value);
  };
#endif
```

プロキシを利用するコード

```
#include "Unit2.h"
...
static TServerMethods1Client* proxy = NULL;
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (proxy == NULL) {
   proxy = new TServerMethods1Client(localcpp->DBXConnection);
  }
  UnicodeString test = L"日本語";
  UnicodeString ret = proxy->EchoString(test);
  ShowMessage(ret);
}
```

Tips: C++向けプロキシコードの不具合

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- 必ずしも必要ではないのですが…… プロキシのインスタンスを破棄する場合、そのままdeleteするとエラーが発生します
- 自動生成されたプロキシのデストラクタを一部修正してください

```
__fastcall  TServerMethods1Client::~TServerMethods1Client()
{
  FreeAndNil(FEchoStringCommand);
}
```

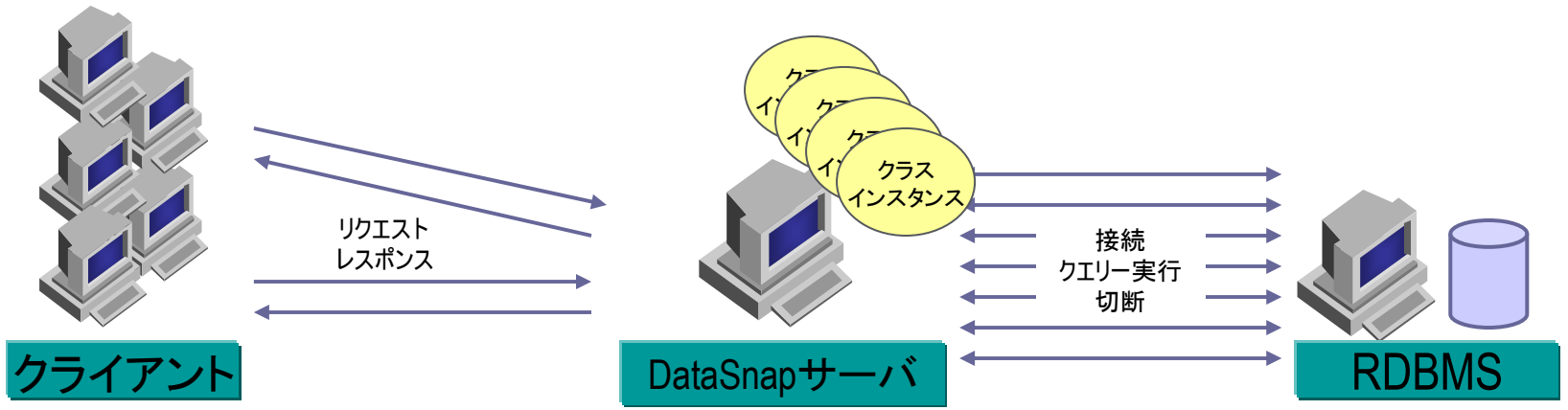
```
__fastcall  TServerMethods1Client::~TServerMethods1Client()
{
  delete FEchoStringCommand;
  FEchoStringCommand = NULL;
}
```

コネクションプーリング

サーバークラスのインスタンスのライフサイクル

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- Server
  - シングルトン
- Session
  - クライアントアプリケーションからの接続数
- Invocation
  - サーバーメソッドの呼び出しごとに生成&破棄を繰り返す



コネクションプーリングの設計

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

```
unit MyDBXPool;

uses
  SysUtils, Classes, SqlExpr, Generics.Collections;

type
  TMyDBXPool = class(TComponent)
  private
    FRootConnection: TSQLConnection;
    FMaxPoolSize: Integer;
    FLock: TObject;
    FPool: TList<TSQLConnection>;
    FBusy: TList<TSQLConnection>;
    procedure SetRootConnection(const Value: TSQLConnection);
  protected
    procedure Notification(AComponent: TComponent; Operation: TOperation); override;
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
    function GetPooledConnection: TSQLConnection;
    procedure ReturnPooledConnection(conn: TSQLConnection);
    function GetPooledCount: Integer;
  published
    property RootConnection: TSQLConnection read FRootConnection write SetRootConnection;
    property MaxPoolSize: Integer read FMaxPoolSize write FMaxPoolSize default 0;
  end;
...
```

```
...
initialization
  IsMultiThread := True;

end.
```

コネクションプーリングの実装

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

```
constructor TMyDBXPool.Create(AOwner: TComponent);
begin
  inherited;
  RootConnection := nil;
  FMaxPoolSize := 0;
  FLock := TObject.Create;
  FPool := TList<TSQLConnection>.Create;
  FBusy := TList<TSQLConnection>.Create;
end;

destructor TMyDBXPool.Destroy;
begin
  FreeAndNil(FBusy);
  while FPool.Count > 0 do
  begin
    try
      FPool.Items[0].Connected := False;
    except on E: Exception do
    end;
    FPool.Items[0].Free;
    FPool.Delete(0);
  end;
  FreeAndNil(FPool);
  FreeAndNil(FLock);
  inherited;
end;
```

```
procedure TMyDBXPool.Notification(AComponent: TComponent; Operation: TOperation);
begin
  inherited;
  if (Operation = opRemove) and (AComponent = FRootConnection) then
    FRootConnection := nil;
end;

procedure TMyDBXPool.SetRootConnection(const Value: TSQLConnection);
begin
  if FRootConnection <> Value then
  begin
    if Assigned(FRootConnection) then
      FRootConnection.RemoveFreeNotification(Self);
    if Assigned(Value) then
      Value.FreeNotification(Self);
    FRootConnection := Value;
  end;
end;
```

コネクションプーリングの実装(続き)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

```
function TMyDBXPool.GetPooledConnection: TSQLConnection;
begin
  Result := nil;
  if not Assigned(FRootConnection) then
    Exit;
  System.TMonitor.Enter(FLock);
  try
    while True do
    begin
      if FPool.Count <= 0 then
      begin
        if (FMaxPoolSize <= 0) or (FBusy.Count < FMaxPoolSize) then
          Break;
        while FPool.Count <= 0 do
        begin
          try
            System.TMonitor.Wait(FLock, 100);
          except on E: Exception do
          end;
        end;
      end;
      Result := FPool.Items[0];
      FPool.Delete(0);
      FBusy.Add(Result);
      Exit;
    end;
    Result := TSQLConnection.Create(nil);
    Result.ConnectionName := 'CONN' + IntToStr(Integer(Pointer(Result)));
    Result.DriverName     := FRootConnection.DriverName;
    Result.GetDriverFunc  := FRootConnection.GetDriverFunc;
    Result.LibraryName    := FRootConnection.LibraryName;
    Result.LoginPrompt    := False;
    Result.KeepConnection := True;
    Result.Params.Assign(FRootConnection.Params);
    try
      Result.Connected := True;
      FBusy.Add(Result);
    except on E: Exception do
      FreeAndNil(Result);
    end;
  finally
    System.TMonitor.Exit(FLock);
  end;
end;
```

```
procedure TMyDBXPool.ReturnPooledConnection(conn: TSQLConnection);
var
  i: Integer;
begin
  System.TMonitor.Enter(FLock);
  try
    i := FBusy.IndexOf(conn);
    if i >= 0 then
      FBusy.Delete(i);
    if Assigned(conn) then
      FPool.Add(conn);
    System.TMonitor.PulseAll(FLock);
  finally
    System.TMonitor.Exit(FLock);
  end;
end;

function TMyDBXPool.GetPooledCount: Integer;
begin
  System.TMonitor.Enter(FLock);
  try
    Exit(FPool.Count);
  finally
    System.TMonitor.Exit(FLock);
  end;
end;
```

Q&A