吉原 泰介

Delphi/400 上級

株式会社ミガロ.

RAD事業部 技術支援課

[Delphi/400]
RAD Serverを使った新しい多層アプリケーション構築

略歴
1978 年 3 月 26 日生まれ
2001 年 3 月 龍谷大学 法学部卒業
2005 年 7 月 株式会社ミガロ. 入社
2005 年 7 月 システム事業部配属
2007 年 4 月 RAD 事業部配属

現在の仕事内容
Delphi/400 を中心に製品試験および月 100 件に及ぶ問い合せサポートやセミナー講師などを担当している。

1.はじめに

ここ数年で Delphi/400 によるモバイルアプリケーションの業務開発も多くなってきた。そうしたモバイルアプリケーション開発において、IBM i を利用するために重要となるのが中間サーバーを使った多層構成の仕組みである。【図1】

Delphi/400 でこうした多層アプリケーションを開発する場合、サーバーもクライアントも Delphi/400 だけですべて開発することができる強みがある。この中間サーバーのアプリケーション開発では、DataSnap Server という技術が使われているが、Delphi/400 の最新版である 10.2 Tokyo では、新しく RAD Server というサーバー構築技術が利用できるようになっている。

本稿ではこの新しい RAD Server についての特徴や実装方法、そして RAD Server を活用した他言語アプリケーションとの連携テクニックなどの応用について検証した内容を説明する。

なお本稿では RAD Server が利用できる Delphi/400　10.2Tokyo の環境を前提としている。

2.RAD Serverの特徴

新しく利用できるようになった RAD Server について、主な特徴を確認する。

まず RAD Server は、DataSnap Server と同様に、多層アプリケーションの中間サーバー部分を実装する技術である。多層アプリケーションとは、複数層で構成されたアプリケーションのことで、たとえば Web アプリケーションやモバイルアプリケーションで IBM i にアクセスする場合には、中間層のサーバーを経由する 3 層構成となる。この中間サーバーのアプリケーションを実装できるのが DataSnap Server や RAD Server である。

ではこの 2 つの技術にどのような特徴の違いがあるかをまとめてみる。

まず、従来のバージョンでも使用可能な DataSnap Server は、次のような特徴を持つ。

● DataSnap Server
- 多層アプリケーションの開発を可能にする SDK
- サーバー機能はプログラムで開発する必要がある
- 開発での実装となるため、プログラムの自由度が高い
- TCP/IP、HTTP（S）、REST、JSON、COM などの標準技術をサポート

次に RAD Server は次のような特徴を持つ。

● RAD Server
- 多層アプリケーションの REST API を公開するサーバー
- サーバーで必要となる高度な機能がいくつも提供されている
- ユーザー管理機能、認証機能、分析機能などの標準機能を豊富に搭載

図1　中間サーバーを使った多層構成例

図2　DataSnap ServerとRAD Serverの違い

	DataSnap Server	RAD Server
機能開発	全て開発で実装が必要	必要な部分のみ開発
標準通信	TCPIP/HTTP(S)	HTTP(S) （REST形式固定）
DBエンジン	FireDAC、 dbExpress	FireDAC
モバイル対応機能	開発が必要	Push通知、デバイス認証等が標準機能
管理ツール	開発が必要	標準で付属（分析も可能）
ライセンス	開発ライセンスに含まれる（Enterprise以上）	開発ライセンスに1サイトライセンス付属（10. 2 Tokyo Enterprise以降）

図3　RAD Serverを使った多層アプリケーションの実装例

・HTTP（S）、REST、JSON などの標準技術をサポート

細かい比較については【図 2】にまとめている。

比較するとわかるが、最も大きな違いは DataSnap Server はサーバーアプリケーションを細かく自由に開発することを目的とした技術で、RAD Server は大枠が完成されたサーバーアプリケーションを、カスタマイズをして使う技術ということである。

RAD Server は標準の機能がいろいろ揃っている分、DataSnap Server と違ってアプリケーションの仕組みは REST 形式で固定となっている。REST とは Representational State Transfer の略で、そのサービスの URL が持つメソッドにアクセスすることでデータの送受信をステートレスで行う技術である。汎用性が高く、Web やモバイルのアプリケーションで広く使用されている。そのため、Delphi/400 のアプリケーションだけに限定されずさまざまなアプリケーションから活用することもできる（これについては後述する）。

どちらが優れているかは開発するアプリケーションの要件によっても異なるが、シンプルな機能の中間サーバーであれば RAD Server の標準機能が強みを発揮できる。

この 2 つの技術は、中間サーバーで担っている役割は似ているが、実際の実装手順は異なる部分も多い。そのため、次章では新しい RAD Server の基本的な実装手順について確認していく。

3.RAD Server の実装手順

RAD Server で構築するアプリケーションは、DataSnap Server と同様に、サーバーアプリケーションとそれを利用するクライアントアプリケーションの 2 種類で構成される。それぞれの実装手順を 2 つのステップに分けて説明していく。

・3-1. サーバーアプリケーションの構築
・3-2. クライアントアプリケーションの構築

なお本稿では、【図 3】に示すように PC やモバイルのクライアントアプリケーションから IBM i のデータを取得する基本的な多層アプリケーションを題材とする。

3-1.サーバーアプリケーション構築手順

初めに中間サーバーに実装するアプリケーションを構築する。構築は RAD Server にウィザードが用意されているため、それほど難しいものではない。

【手順①プロジェクトの作成】

新規作成より、RAD Server（EMS）パッケージを選択してウィザードを起動する。【図 4】

EMS という名称は、RAD Server 関連の機能を意味する。

RAD Server が当初 EMS Server という製品名であったため、機能やコンポーネント名で使われていることが多い。

【手順②リソースの指定】

次にウィザードに従って設定を進めることになるが、【図 5】のように「リソースを含むパッケージを作成する」を選択して、そのリソース名を任意で命名する。

RAD Server は REST 形式になるため、URL でアクセスする際に利用する機能をリソースとして実装する。ここで設定しておくと、指定したリソース名のソースが作成される。たとえば【図 5】のように「CUST」というリソース名を指定すると、実行時に下記のような REST サービスとして呼び出すことができる。

http（s）:// サーバー /CUST

【手順③機能の指定】

②で指定したリソースに対して、どういった機能のテンプレートを作成するかを【図 6】で設定する。デフォルトは基本機能としてデータを取得する Get、GetItem が選択されているが、Put、PutItem、DeleteItem といった更新系の機能も用意されている。なお名称に Item と付く機能は、パラメータを渡して処理できることを意味している。

【手順④自動生成されるソース】

ここまでのウィザード操作が完了すると、【図 7】のように中間サーバーアプリケーションのソース一式が自動生成される。

次の手順からはこのソースに必要なプログラムを実装していく。

【手順⑤コンポーネントの配置と設定】

自動生成されたリソースに機能を実装するにあたって、まずは必要なコンポーネントを配置する。RAD Server で使えるデータベースエンジンは最新の FireDAC のみに限定されている。そのため、ここではデータ取得（Get）の仕組みを作ることを前提に FireDAC の主要なコンポーネントを配置する。【図 8】

・TFDConnection
・TFDPhysCO400DriverLink
・TFDTable
・TFDSchemaAdapter
・TFDStanStorageJSONLink
・TFDGUIxWaitCursor

TFDConnection 〜 TFDTable および TFDGUIxWaitCursor については、一般的な FireDAC のアプリケーションで実装する内容である。本稿では FireDAC の詳しい使い方は割愛するが、【図 9】のように TFDConnection や TFDTable を設定する。

RAD Server の構築でポイントになるのは TFDSchemaAdapter と TFDStanStorageJSONLink コンポーネントである。これは RAD Server が REST 形式として動作することが前提となるため、TFDTable で取得したデータを JSON 形式に変換する機能の実装に使用する。【図 9】で TFDTable の SchemaAdapter プロパティに TFDSchemaAdapter を設定しているのは、そのためである。

【手順⑥データ取得の機能を実装】

次に RAD Server がリクエストに応じてデータを返す機能をプログラムで実装する。ウィザードでリソースの Get 機能を指定しておけば、自動的に Get メソッドが作成されているので、ここで必要な処理だけを数行コーディングする。【ソース 1】

図4　プロジェクトの作成

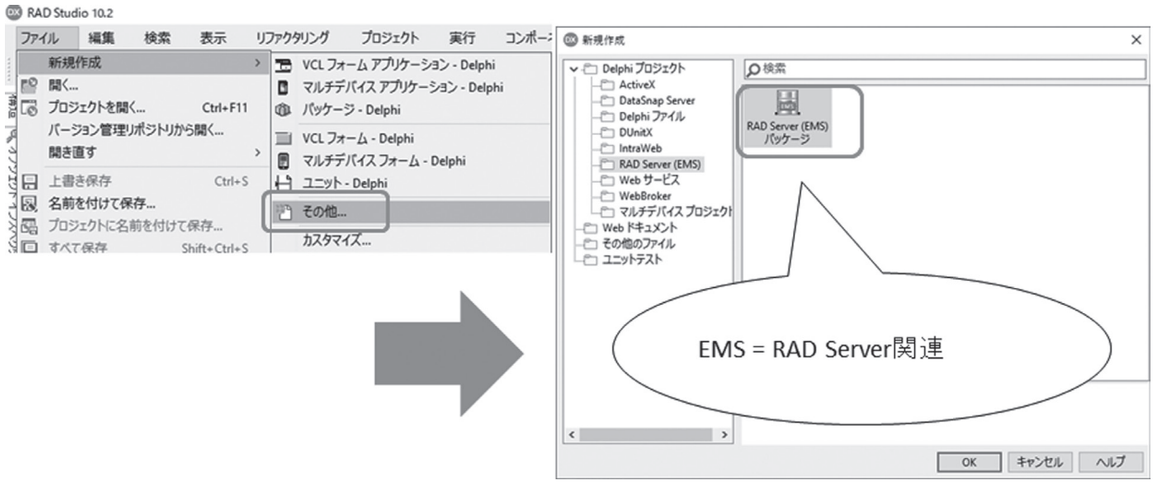


図5　リソースの指定

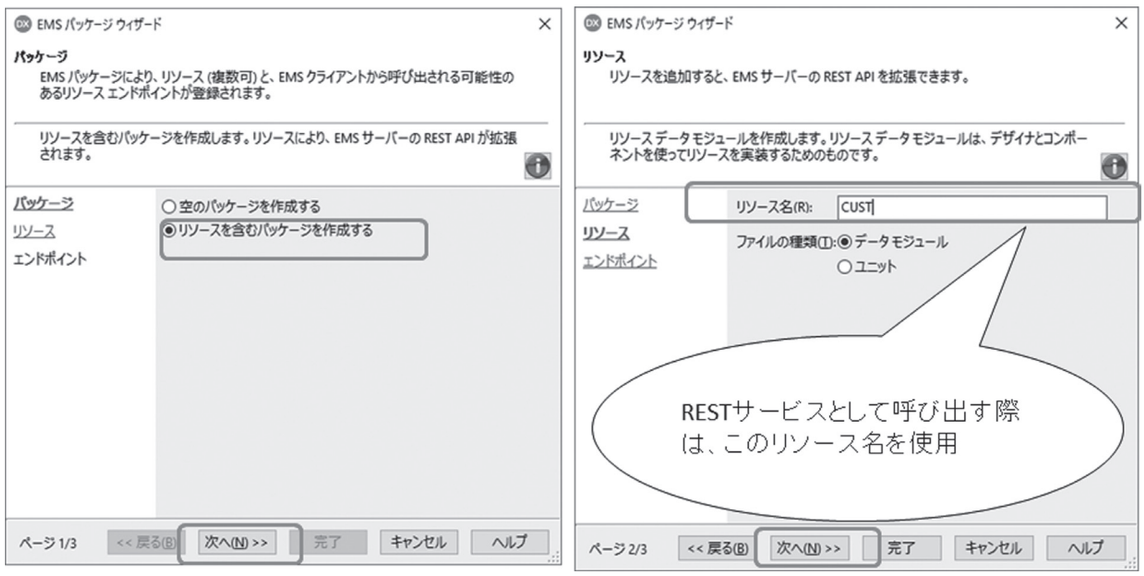
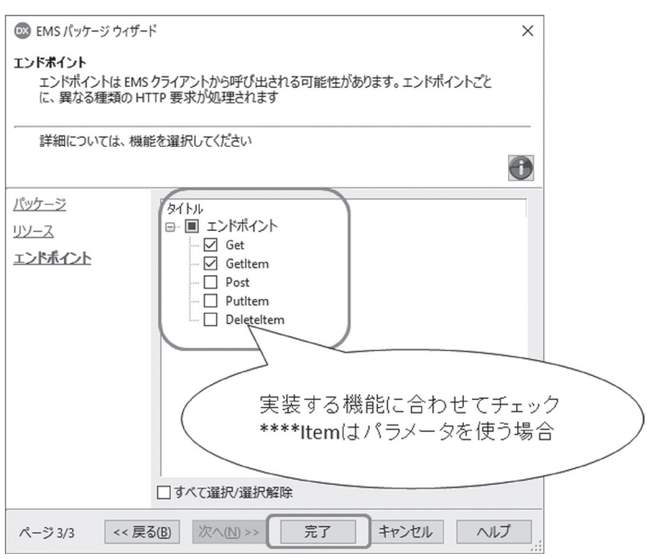


図6　機能の指定



処理内容としてはTFDTableでデータを開き、そのデータをTFDSchemaAdapterでJSON形式のストリームに変換するだけである。この時、JSON形式を指定するためにTFDStanStorageJSONLinkが必要となっている。また図6でItemを使った機能を実装する場合は、パラメータであるARequestを使ってARequest.Params.Values['XXXX']という形で簡単に利用することができる。

プログラムの実装はこれで一通り完了である。

【手順⑦コンパイルと実行】

最後に作成したRAD Serverの中間サーバーアプリケーションをコンパイルして実行する。ただし初回のコンパイル時にはコンポーネントのパッケージ開発時と同様（RAD Serverもパッケージ方式）にパッケージの参照確認やInterBaseの設定確認のダイアログが表示されるので応答が必要となる。【図10】

InterBaseは意識して使用するわけではないが、RAD Serverが標準で備える管理分析機能等で内部的に使用している。なおInterBaseのライセンスはRAD Serverに含まれているので別途購入の必要はない。

コンパイルが完了してアプリケーションを実行すると、【図11】のような画面が起動して完成である。この画面では、自動的にアクセスログが出力されるようになっており、クライアントからリクエストがあるとこの画面にログが表示される。

この起動画面はあくまでRAD Serverに用意された標準のexeアプリケーションである。DataSnap Serverと違い、作成したプログラム自体がサーバーとして起動しているわけではない。

コンパイルしたリソースはbplとして作成されてRAD Serverに読み込まれている。運用環境の構築やリソースの配布をする際には、エンバカデロ社の下記オンラインヘルプに詳細情報が公開されているため、参考いただきたい。

http://docwiki.embarcadero.com/RADStudio/Tokyo/ja/

3-2.クライアントアプリケーション構築手順

次に、前節で作成したRAD Serverアプリケーションに接続してデータを取得するクライアントアプリケーションを構築する。

【手順①プロジェクトの作成】

新規作成よりマルチデバイスアプリケーションを選択して、FireMonkeyの新規プロジェクトを作成する。【図12】

今回はモバイルでの動作を見るためにFireMonkeyで作成しているが、デスクトップアプリケーションとしてVCLで作成する場合も実装内容は同じである。

【手順②コンポーネントの配置・設定】

新規作成したアプリケーションに必要なコンポーネントを配置する。【図13】

・TEMSProvider
・TEMSFireDACClient
・TFDSchemaAdapter
・TFDTableAdapter
・TFDMemTable
・TFDGUIxWaitCursor

このアプリケーションでもFireDACコンポーネントを使用するが、直接IBM iに接続するわけではなく、前節で作成したRAD Serverアプリケーションに接続してデータを取得する仕組みとなる。そのため、TFDConnectionではなく、TEMSProviderおよびTEMSFireDACClientの専用コンポーネントを使用する（EMSはRAD Server関連を意味する）。

またRAD Serverより取得するデータはJSON形式で送られてくるため、それをDelphi/400のデータセット形式に変換して戻すためにTFDSchemaAdapterを使用する。それぞれのプロパティ設定は【図14】に示す。

ポイントとしてはTEMSProviderで接続するRAD ServerのIPやポートを指定し、TEMSFireDACClientで呼び出すリソース情報を設定する点である。リソース名はRAD Serverのウィザードで指定したものを設定する。

なおデータの画面表示はFireMonkeyなので、【図15】のようにライブバインディングを使って実装できる。VCLの場合はTDBGridを使うこともできる。

【手順③RAD Serverの呼び出し機能の実装】

最後にRAD Serverからデータ取得のGet機能をRESTで呼び出す処理をプログラムで実装する。RAD Serverはほとんどのアクセス制御を前手順のコンポーネントで処理してくれるため、コードは1行で済む。【ソース2】

これでクライアントアプリケーションのデータアクセスのプログラムが完成したことになる。RAD ServerとDataSnap Serverの開発は似ているが、RAD Serverでは標準機能がコンポーネントなどで備わっている分、簡単に実装ができる部分も多い。

【手順④コンパイル・実行】

完成したプログラムを目的のプラットフォームを指定してコンパイルして実行すると、WindowsやiOS、あるいはAndroid向けにアプリケーションを利用することができる。【図16】

全体的な仕組みは冒頭で説明した【図3】のとおりである。

4.RAD Serverの管理分析機能

前章でRAD Serverの基本的な実装手順を説明したが、RAD Serverが持つ機能について少しだけ補足を加える。

3-1.で実装したRAD Serverのアプリケーションは、起動画面でアクセスログが確認できることを説明したが、利用できる機能はそれだけではない。

RAD Serverは管理機能を標準で備えているため、たとえばサーバーに対するアクセス分析などの機能は開発しなくとも自動で実装されている。使い方としては起動画面メニューの「コンソールを開く」ボタンをクリックする。【図17】

そうすると【図18】のようなコンソールのログイン画面がブラウザで起動する。このログインはユーザー名がconsoleuser、パスワードがconsolepassがデフォルトになっている。

図7　自動生成されるソース

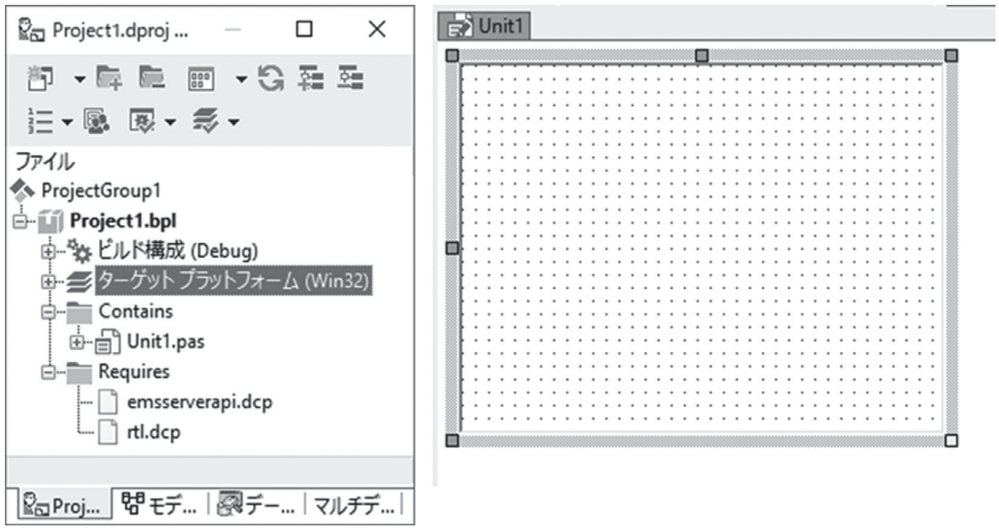


図8　サーバー機能のコンポーネント配置

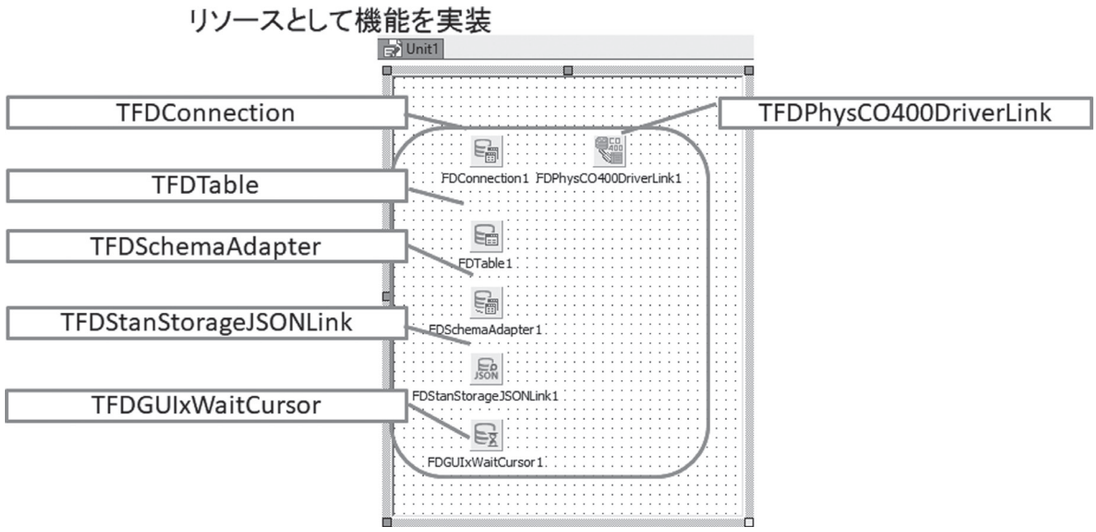


図9　コンポーネントの設定

このアカウント情報はRAD Server上の下記iniファイルに設定されており、テキスト編集で変更が可能である。

C:¥Users¥Public¥Documents¥Embarcadero¥EMS¥emsserver.ini

ログインすると、【図19】のようにRAD Serverにアクセスするクライアントの状況や作成したRESTリソースの使用頻度について、グラフを使ったビジュアル分析が行える。これによってどの機能がよく使われているか、どんな時間帯に処理が集中しているか、などの把握も容易である。

5.補足:他言語からの利用

ここまでRAD Serverを使った基本的なアプリケーションの実装内容や機能についてまとめてきたが、最後にRAD Serverをさらに活用するための使い方を考察していく。

RAD ServerがREST形式になることは冒頭で説明したが、このREST形式はここまでの実装でも説明したとおり、JSONを利用できる。このREST/JSONの組み合わせは、非常によく使用される通信方式であるため、実はDelphi/400以外の他言語アプリケーションでも利用可能である。

本稿では他言語の例として、Delphi言語と同じ開発元であるエンバカデロ・テクノロジーズ社が提供するSencha Architectを題材としてRAD Serverの利用を説明する。Sencha Architectとは、表、グラフ、ツリーなどの多彩なコンポーネントをドラッグ&ドロップして簡単にHTML5対応のWebアプリケーションを構築できる開発ツールである。【図20】

この Sencha Architect は REST/JSONデータと連携してデータを扱うこともできるため、RAD Serverとも容易に連携ができる。

●RAD Serverの実装調整

3章で実装したGET機能のJSONはDelphi/400のコンポーネントに特化した作り方となっているため、もっと一般的なJSON形式に調整する。【ソース3】

●Sencha Architectからの利用

本稿ではSencha Architectの詳しい開発手順については割愛し、ポイントになる部分を説明する。

Sencha Architectにも Delphi/400と同じようなコンポーネント（部品）が用意されているため、RAD Serverから取得したデータを表示するためにGrid Panelというコンポーネントをプログラムに配置する。【図21】

この Grid Panel は Delphi/400 のTDBGridと扱いが似ている。

次にGrid Panelのデータ参照元を指定する。

これはGrid PanelにGrid Builderというウィザード機能が用意されており、【図22】のようにJSON WebService形式でRAD ServerのURLが参照元となるように設定する。これによってフィールド情報なども自動で取り込んで定義できる。

このデータをGrid Panelの参照元にセットすれば、Delphi/400と同じようにSencha Architect上でRAD Serverからの情報を表示することができる。【図23】

このプログラムをSencha Architectでコンパイルして実行するとブラウザでWebアプリケーションとして実行され、RAD Serverの情報を表示することが可能である。【図24】

Sencha Architectの細かい設定手順は省略しているが、RAD ServerのREST/JSON形式は非常に汎用性が高く、別の言語でも簡単に扱えることがわかる。

もちろんSencha Architectに限らず、他の言語でもREST/JSONが扱えれば同じように使用が可能である。これを理解していればRAD Serverをより広い範囲で活用することができる。

6.おわりに

本稿では、新しい多層アプリケーションの構築技術としてRAD Serverの特徴や実装手順について説明した。従来のDataSnap Serverと違って、RAD Serverは開発形式がある程度限定されているものの、あらかじめ専用コンポーネントやサーバーの管理機能を備えているため、効率よくアプリケーションを実装するには効果的である。

また5章で述べたように、このRESTサービスは非常に汎用性が高く、Delphi/400以外の言語でも利用することができる。

今後、クラウドやソリューションを含めて多様化していくシステム連携の中心となるサーバーとして活用が期待できる。

M

図10　初回コンパイル時の応答

ソース1　データ取得の機能を実装

```
procedure TCUSTResource1.Get(const AContext: TEndpointContext;
const ARequest: TEndpointRequest; const AResponse: TEndpointResponse);
var
  oStr: TMemoryStream;
begin
  oStr := TMemoryStream.Create;

  // クエリの実行結果をスキーマアダプタから
  // メモリストリーム経由で返す
  FDTable1.Open;
  FDSchemaAdapter1.SaveToStream(oStr,TFDStorageFormat.sfJSON);
  AResponse.Body.SetStream(oStr,'application/json', True);
  FDTable1.Close;
end;
```

図11　RAD Serverの起動

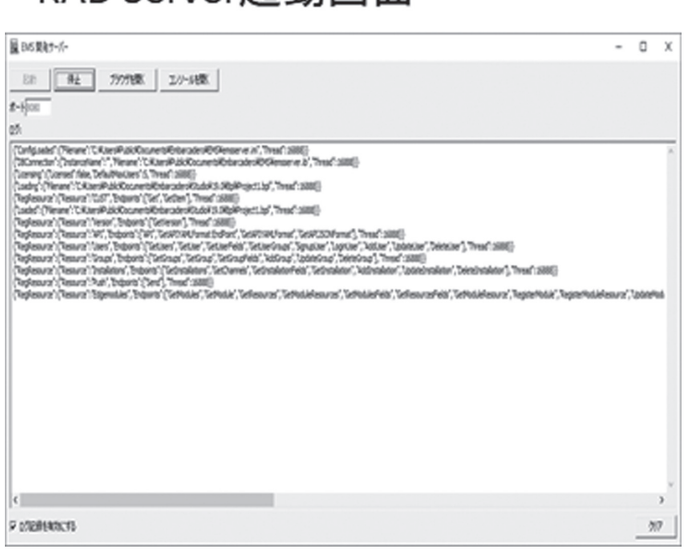


図12　クライアントアプリケーションの作成

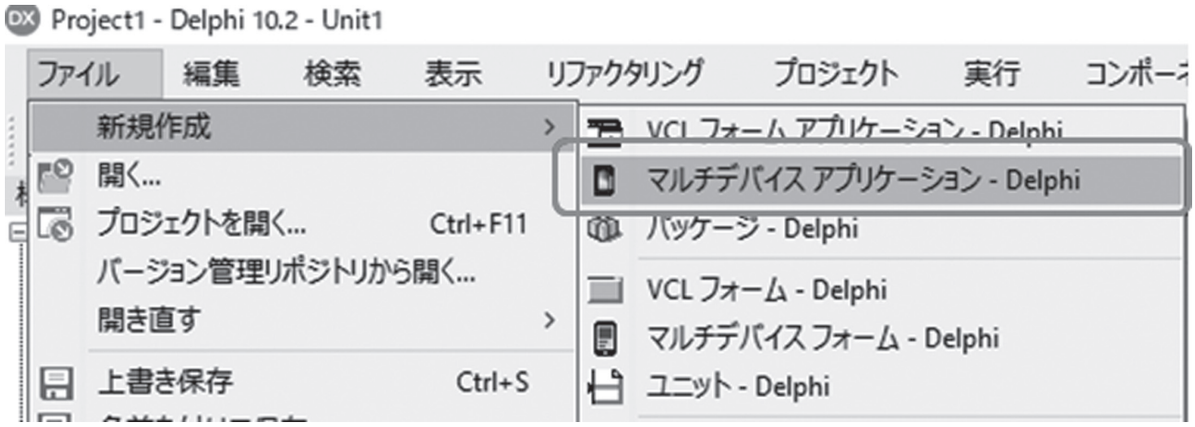


図13　クライアント機能のコンポーネント配置

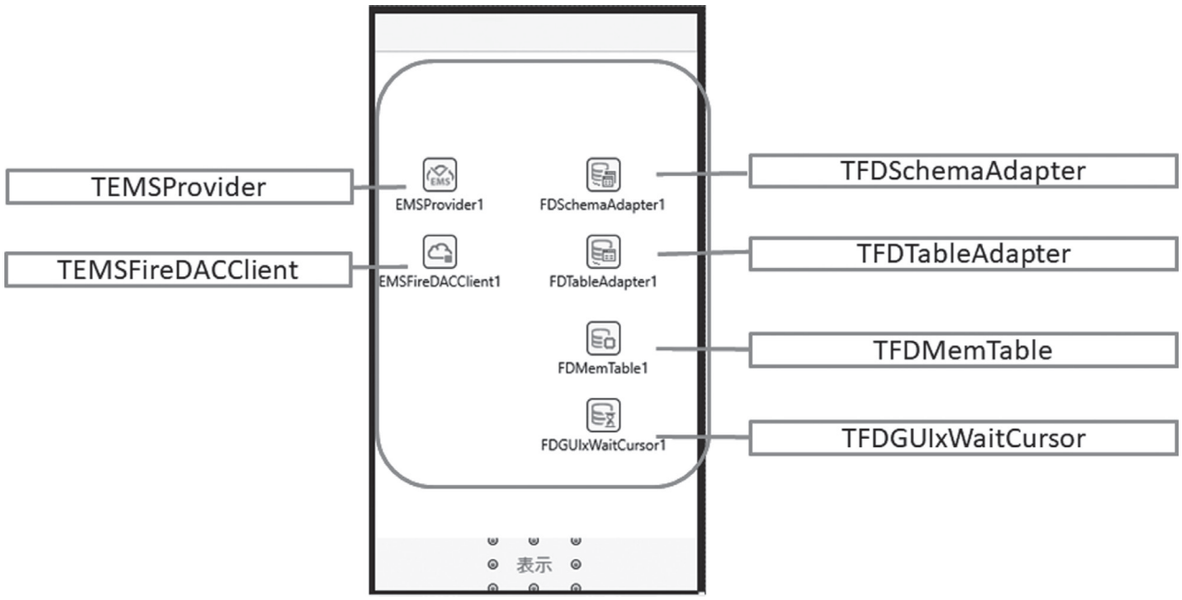


図14　コンポーネントの設定

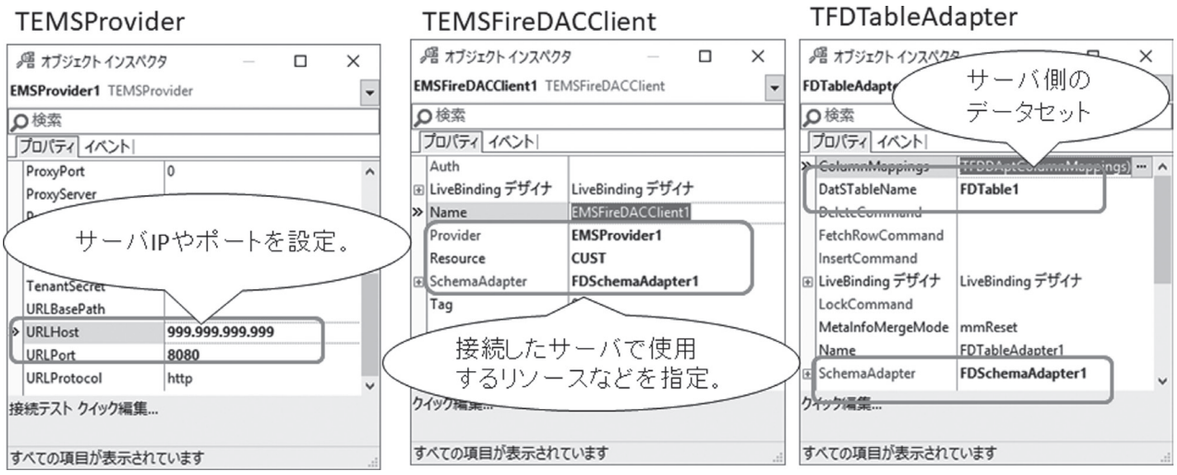
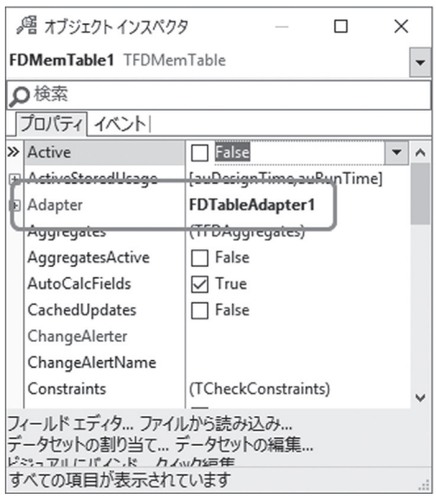
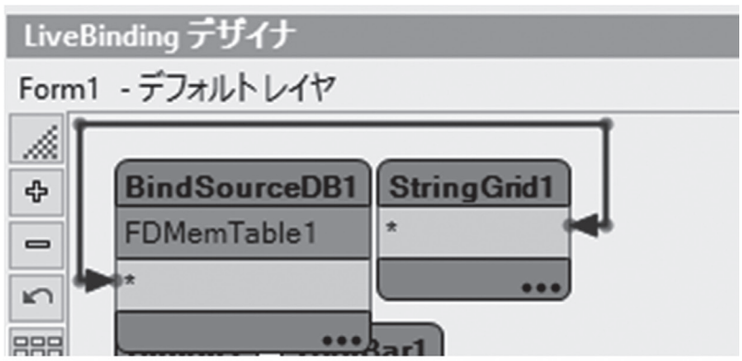


図15　データの表示設定

TFDMemTable



ライブバインディング設定



ソース2　データ取得の機能を実装

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  //設定しているリソースのGetDataを呼び出す
  EMSFireDACClient1.GetData;
end;
```

図16　プラットフォーム別にコンパイル・実行

Windowsでコンパイル

iOSでコンパイル

図17　コンソールの起動

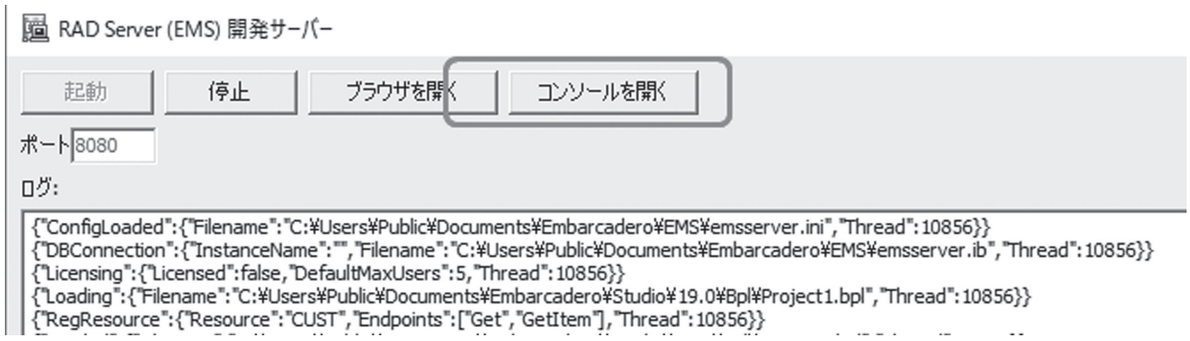


図18　コンソールのログイン

図19　RAD Serverによる分析機能

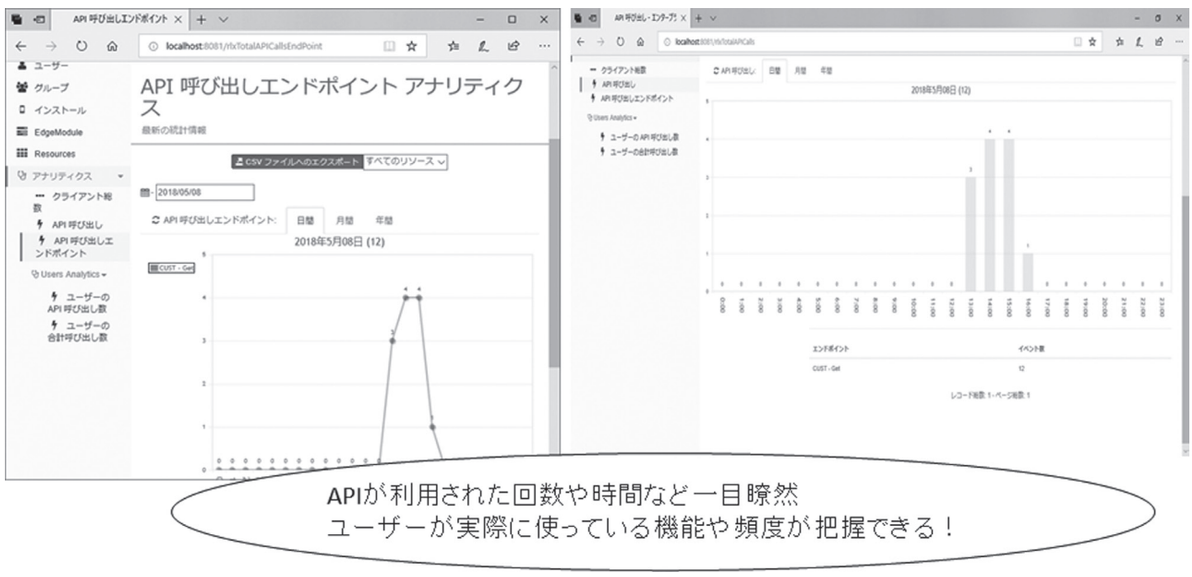


図20　WebアプリケーションにSencha Architect

ソース3　一般的なJSON形式でRESTの結果を戻す

```
procedure TCUSTResource1.Get(const AContext: TEndpointContext;
const ARequest: TEndpointRequest; const AResponse: TEndpointResponse);
var
  JSONResponse: TJSONObject;
  JSONArray: TJSONArray;
  JSONRecord: TJSONObject;
  aField: TField;
  count: Integer;
begin
  JSONResponse := TJSONObject.Create;
  JSONArray    := TJSONArray.Create;
  count := 0;
 // データセットの結果を1件づつ取得してJSONArrayに追加する
  FDTable1.Open;
  while (not(FDTable1.Eof)) do
  begin
    JSONRecord := TJSONObject.Create;

    for aField in FDTable1.Fields do
    begin
      JSONRecord.AddPair(LowerCase(aField.FieldName),aField.AsString);
    end;

    JSONArray.Add(JSONRecord);
    FDTable1.Next;
    inc(count);
  end;

  AResponse.Body.SetValue(JSONArray,True);
end;
```

図21　Sencha Architectの開発（GridPanel）

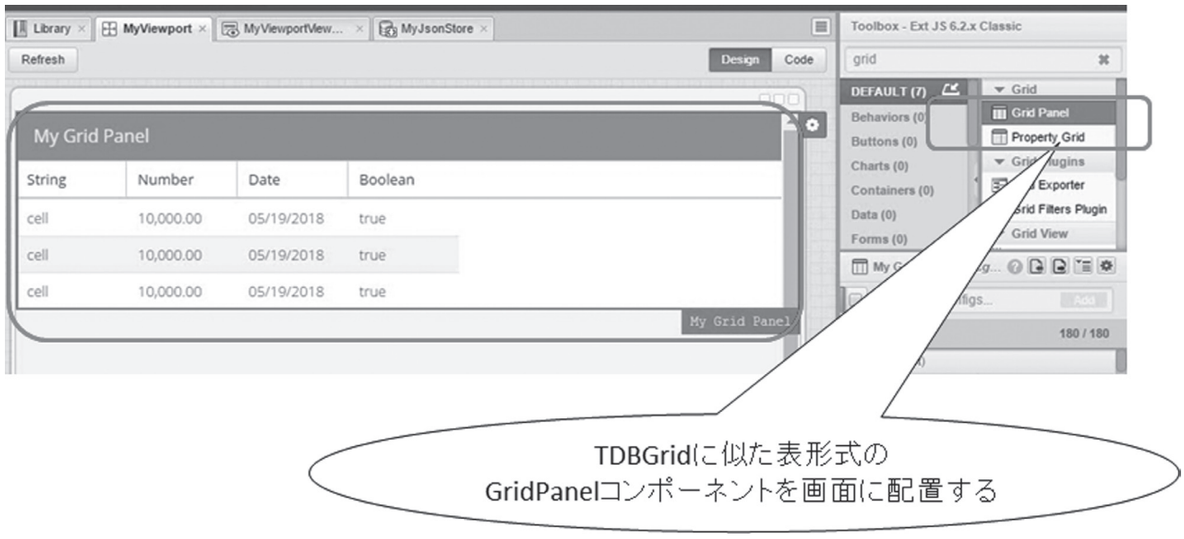


図22　Grid Panelのデータ参照設定

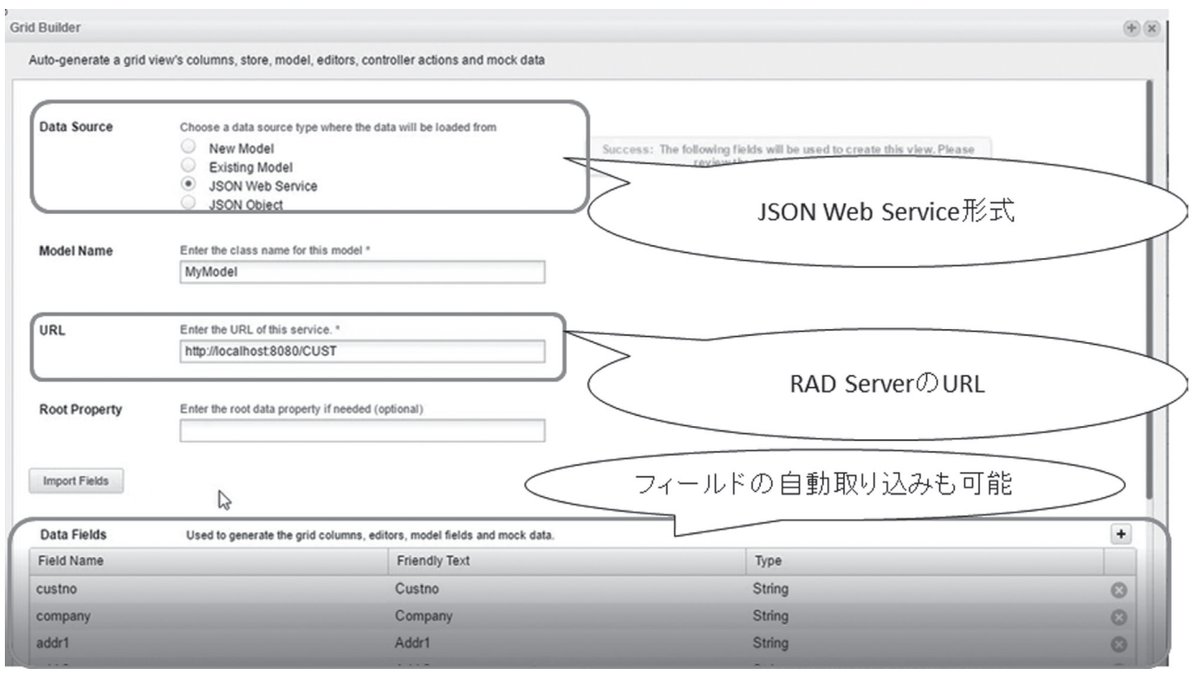


図23　Grid Panelへの表示

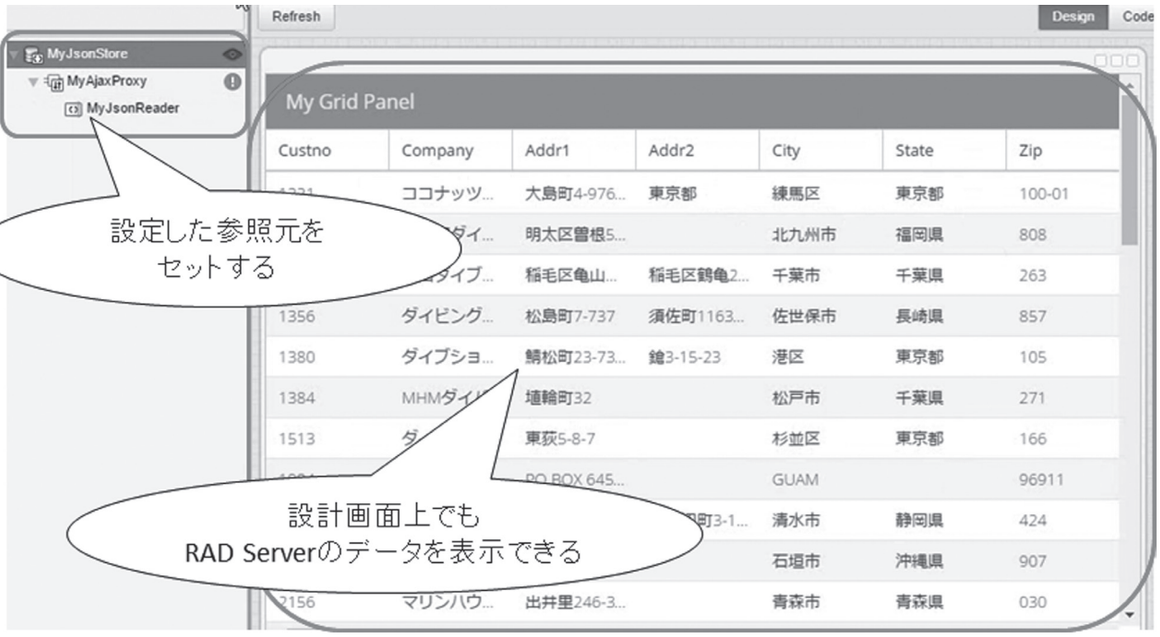


図24　Sencha ArchitectアプリからRAD Serverを利用