尾崎 浩司

Delphi/400 中級

株式会社ミガロ.

RAD事業部 営業・営業推進課

[Delphi/400] RESTによるWebサービスを活用した機能拡張テクニック

略歴
1973年8月16日生まれ
1996年3月 三重大学 工学部卒業
1999年10月 株式会社ミガロ. 入社
1999年10月 システム事業部配属
2013年4月 RAD事業部配属

現在の仕事内容：
ミガロ. 製品の素晴らしさをアピールするためのセミナーやイベントの企画・運営等を主に担当している。

1.はじめに

近年アプリケーションの開発において、Webサービスの活用が盛んになっている。Webサービスとは、インターネット技術を応用し、他のWebサイト上のソフトウェアを呼び出して利用する仕組みのことである。【図1】

現在では、大量のデータを蓄積している業者等が、そのデータをWebサービスの形で一般のユーザーやプログラマーに提供する事例が多くなっている。たとえば、ネットショッピングサイト大手のアマゾンには「Product Advertising API」、楽天には「楽天市場商品検索API」といったWebサービスがあり、これらWebサービスを利用すると、サイト上の商品検索等を自分のプログラムに組み込むことができる。さらに近年では、従来の大量データをもつ業者だけでなく、IBM Watsonのような自然言語を理解し、機械学習により人間の意思決定を支援するシステムまでもが、Webサービスとして利用可能になっている。

Webサービスは、インターネット技術を使用するのが特徴だが、その手法にはいくつかあり、代表的なのがSOAP（Simple Object Access Protocol）およびREST（REpresentational State Transfer）である。

SOAPは、SOAPメッセージというXMLによってメッセージ交換を行う方法で事前にやり取りの定義が必要なため、難易度が高い。最近は、よりシンプルなRESTが主流である。本稿では、Delphi/400を使用したRESTによるWebサービスの使用方法や機能拡張方法について説明する。なお、本稿のプログラムはDelphi/400 10 Seattle以降の環境を前提としている。

2.RESTによるWebサービスとは?

RESTとは、Webサービスの設計モデルのことで、ネットワーク上のデータ（リソース）を一意なURLで表すものである。サービスのURLにHTTPメソッドでアクセスすることでデータの送受信が行える。パラメータを指定してURLにアクセスすると特定の形式でデータが返ってくるものだ。データ形式には、XMLあるいはJSONが利用可能であるが、近年は、よりシンプルなJSONが使われることが多い。

JSONとは、JavaScript Object Notationの略で、軽量のデータ交換フォーマットのことである。key（名前）とvalue（値）を「:」で対にして記述し、まとまりごとに｛｝で囲うといった表記法で、[ ]で配列を表現することもできる。たとえば、【図2】のようなJSONは、"result"というキーの配列の第一要素の中にある"score"というキーの値が80であると解釈できる。

では、ここでREST + JSONによるWebサービスを試してみる。livedoorが提供する「お天気Webサービス」を使ってみる。

http://weather.livedoor.com/weather_hacks/webservice

図1　Webサービス

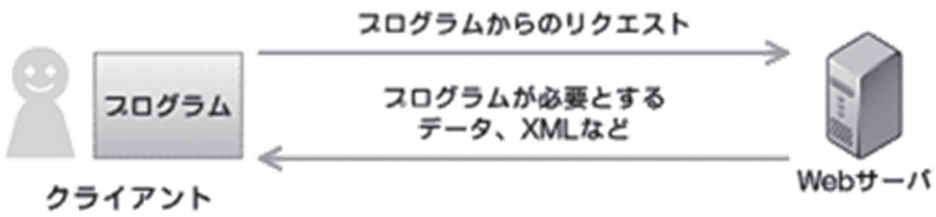
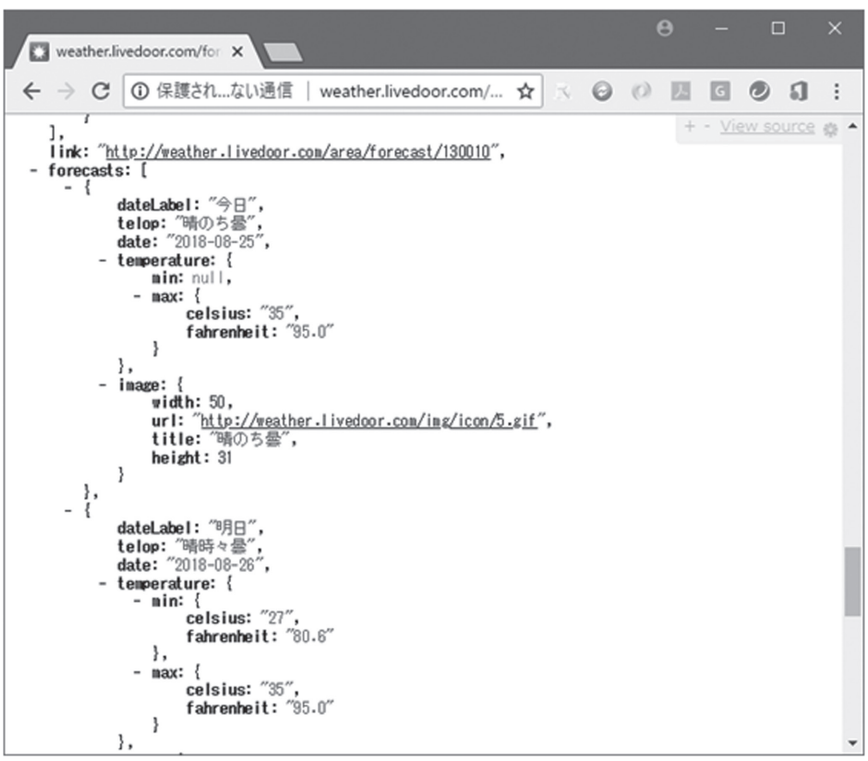


図2　JSONの例

```
{
"result":[

    {
     "subject":"english",
     "score": 80,
     "state":"good"
   },

    {
     "subject":"math",
     "score": 70,
     "state":"normal"
   }

]
}
```

“result”キーの値が配列となっており、配列要素1つ目の中にある、”score”キーの値が80である。

図3　お天気Webサービス実行例



（Google で“お天気 Web サービス”を検索すると上位に表示される「お天気 Web サービス仕様」。）

このサービスは、現在全国 142 カ所の今日・明日・明後日の天気予報・予想気温と都道府県の天気概況情報を提供するものである。

Chrome ブラウザを立ち上げてアドレス欄に

「http://weather.livedoor.com/forecast/webservice/json/v1?city=130010」

と入力してアクセスを行う。すると、【図 3】のような JSON が表示される。

これは、地域 ID=130010（東京都）の天気予報情報にアクセスした結果の JSON である。天気や最高気温などの情報が JSON の中に含まれていることがわかる。このように REST + JSON による Web サービスは、URL にパラメータを付けて呼び出すとレスポンスとして JSON データが返ってくることが確認できる。

3.REST機能を利用する方法

では、この「お天気 Web サービス」を Delphi/400 から使用する方法を検討する。Delphi/400 には、REST による Web サービスを使用するためのコンポーネントが用意されている。それが、「TRESTClient」、「TRESTRequest」そして「TRESTResponse」である。【図 4】

「TRESTClient」は、Web サービスへのリクエストを実行するコンポーネントで、サービスに対する HTTP 接続を管理し、HTTP ヘッダーおよびプロキシ サーバーを処理し、応答データを受け取るものである。「TRESTRequest」は、HTTP リクエストを形成するパラメータや設定をすべて保持する。「TRESTResponse」は、Web サービスからのすべての戻りデータを保持する。実際の設定は次のようになる。【図 5】

ポイントは、RESTClient1 コンポーネントの BaseURL プロパティに WEB サービスの基底 URL を指定すること、RESTRequest1 コンポーネントの Method プロパティに HTTP メソッドの種類を、Resource プロパティに実行パラメータを指定することである。

このプログラムの [検索] ボタンクリック時の処理は、【ソース 1】となる。

1 行目は、画面上で指定した地域 ID を Resource プロパティに記した "{CITY}" にセットする処理である。Params プロパティの AddItem メソッドがリクエストのパラメータを定義するメソッドである。2 行目は、Web サービスへのリクエスト実行になり、レスポンスの JSON 文字列を取得して Memo1 にセットするのが、3 行目である。実際に実行した結果が【図 6】となる。

REST による Web サービスによって JSON データが取得できることを確認したが、実際にはこの JSON データをパース（解釈）し必要な情報を抜き出す必要がある。Delphi/400 にはこの JSON を取り扱うためのユニットが用意されている。それが、"System.JSON" ユニットである。この中に、JSON オブジェクトを実装したクラス TJSONObject や、文字列、数値、オブジェクト、配列、true/false の型を持つすべての JSON クラスの上位クラス TJSONValue が用意されているので、これらを使用することでパースすることが可能である。

たとえば、【図 7】が JSON をパースして特定のキーの値を取得するロジック例である。

お天気情報の中から、今日と明日と明後日の天気を取得して表示する処理を実装してみる。

「お天気 Web サービス」の説明 Web ページを確認すると、レスポンスの仕様が記載されている。[forecasts] プロパティの中にある [date] プロパティが予報日、そして [telop] プロパティが天気である。なお、3 日分のデータは配列として定義されている。この情報をもとに JSON をパースして、天気予報を画面に表示するように改良したのが、【ソース 2】である。完成したプログラムを実行すると、【図 8】のように指定した地域の 3 日分の天気予測を取得することができる。

もう 1 つ Web サービスの例を紹介する。「HeartRails Geo API」（http://geoapi.heartrails.com/）である。これは、郵便番号／住所／緯度経度データ等の地理情報を提供する Web サービスである。この中に、「最寄駅情報取得 API」という機能があり、これは郵便番号を指定すると、その地区の一番近い最寄駅がわかる機能である。これを活用すると、たとえば社内の取引先マスタにある郵便番号を使用して、同じ最寄駅から歩いて訪問できる取引先をピックアップする使い方ができる。

サンプルプログラムを紹介する。REST コンポーネントの設定は、【図 9】、プログラムは【ソース 3】となる。この API は、パラメータ postal に郵便番号をセットし呼び出すと、レスポンスとして、response プロパティの station 配列の中にある prefecture プロパティには都道府県が、line プロパティに路線が、そして name プロパティには最寄駅がセットされる。作成したプログラムを実行して、郵便番号を入力し、[検索] ボタンをクリックすると、最寄駅が表示されることがわかる。【図 10】

この最寄駅情報取得 API も、先ほどの天気情報の Web サービスと全く同じやり方で処理ができることがわかる。このように、REST + JSON の Web サービスは、とても簡単に使用できるので、いろいろなサービスを試してみていただきたい。本稿執筆にあたり、Web サービスを調査したが、「API List 100+」（http://smsurf.app-rox.com/api/） というサイトが役立った。ここにはいろいろな Web サービスが一覧掲載されているので、便利な機能を見つけてみてほしい。【図 11】

4.IBM Watson API 活用方法

「コグニティブ」という言葉を聞いたことがないだろうか。日本語では「認知」のことで、ある事象についてコンピュータが自ら考え、学習し、自らの答えを導き出すシステムのことをいう。身近なところでは、iPhone の Siri やスマートスピーカー等が有名である。従来システムとの本質的な違いは、音声・画像・文章等の非定型データも処理できることである。従来システムがもつ定型データと組み合わせることで、人の作業を補助し、より便利なシステム構築が可能になる。この「コグニティブ」分野で IBM が提供するのが、Watson である。この

図4　お天気情報取得画面

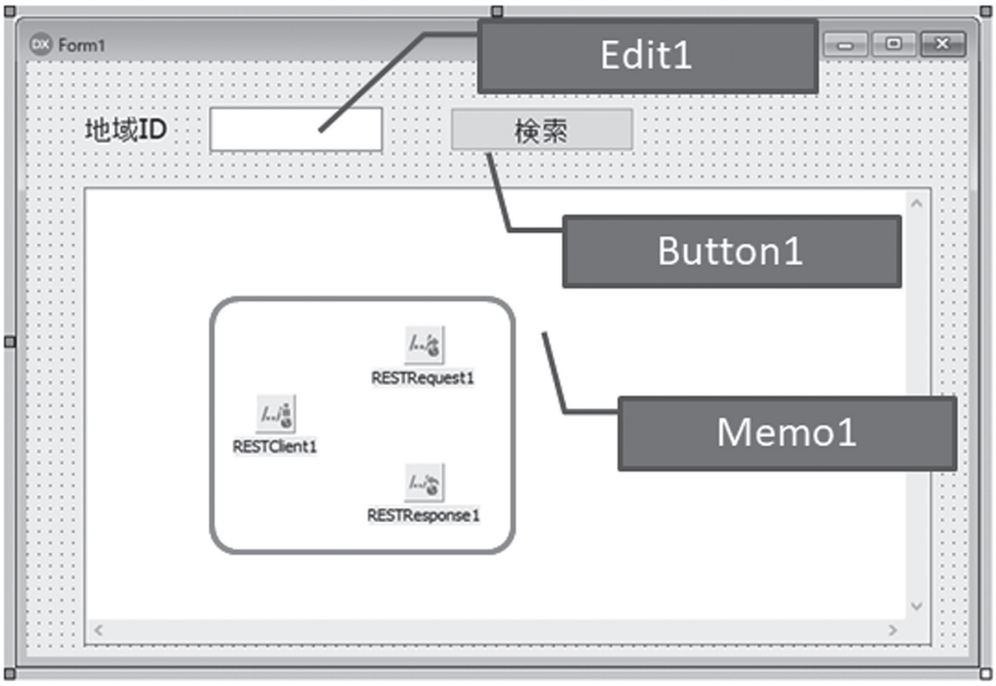
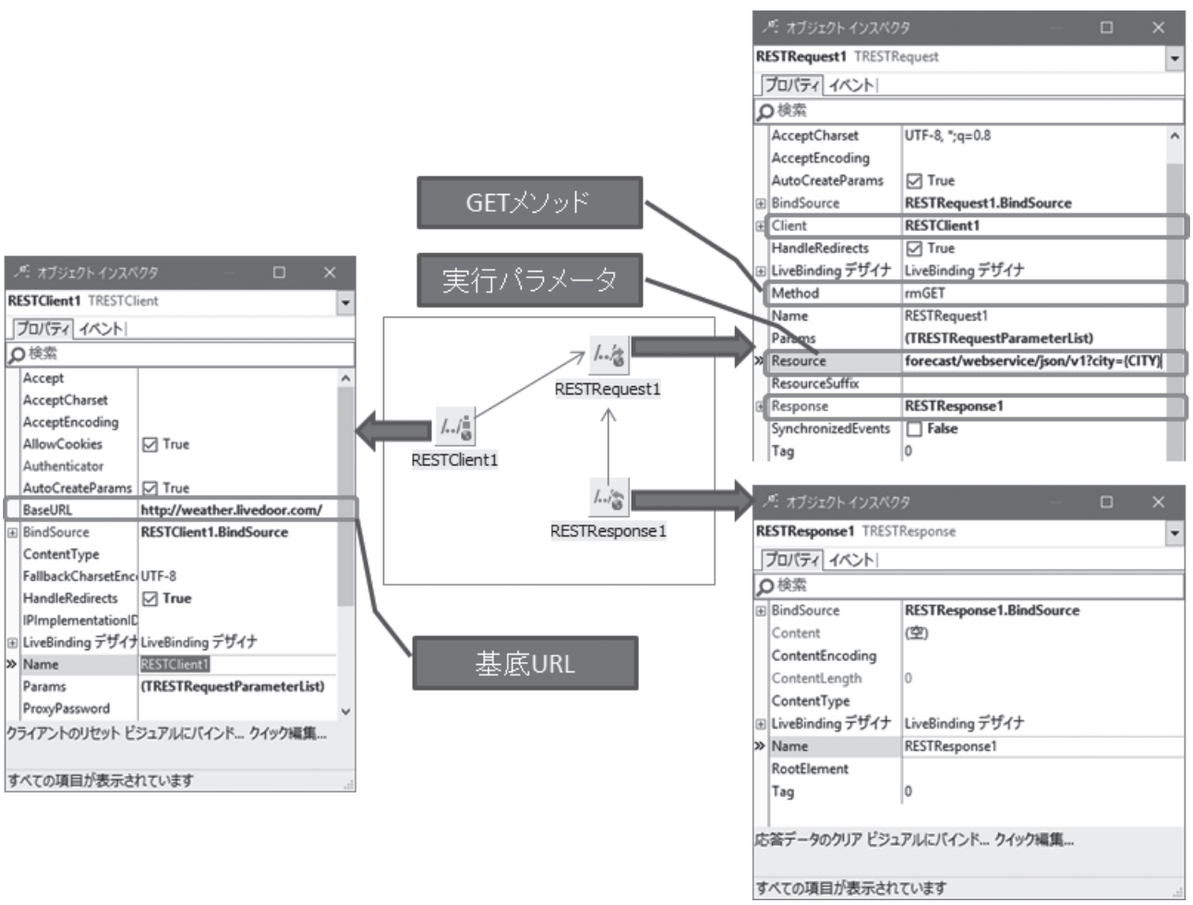


図5　TRESTコンポーネント設定



Watson も Web サービスとして利用することができるので本章で説明する。

Watson は、IBM のクラウドサービス（IBM Cloud）上で API として提供されており、クラウドサービスに登録すれば誰でも使用できる。本格的な利用には有償プランが必要だが、IBM Cloud の各種サービスを無料で使用できる「ライト・アカウント」があるので、こちらを使用するとよい。「ライト・アカウント」には、サービス使用量や機能の制約はあるが、クレジットカード登録不要で簡単に登録できる。

https://www.ibm.com/cloud-computing/jp/ja/bluemix/lite-account/

（Google で“IBM Cloud ライト・アカウント”を検索すると上位に表示される「IBM Cloud ライト・アカウント -Japan」。）

2018 年 8 月時点で確認した Watson の API が【図 12】である。調べたすべてのサービスが「ライト・アカウント」で使用可能だ。

今回は、Language Translator サービスを例に説明する。これは言語変換（翻訳）サービスで、特長としては、一般的な WEB 翻訳サービスと違い、専門用語等を個別に登録することや、機械学習によるカスタム翻訳モデルを作成することが可能で、高精度の翻訳が行えることである。このサービスを活用すると、海外担当者とのやり取り時の自動翻訳や、システムの多言語対応で、DB 上に日本語でしか保持していない情報を翻訳して、画面に出力することができる。

Watson は、サービスごとにインスタンスの作成が必要である。まず IBM Cloud（https://console.bluemix.net/）からサインインを行い、表示されたダッシュボード画面より、[リソースの作成] ボタンをクリックする。Watson をはじめとするサービス選択画面が表示されるので、[Language Translator] を選択する。サービス概要画面が表示され、プランの選択ができるようになるため、「ライト」が選択されていることを確認し、[作成] をクリックすれば、完了である。インスタンスの作成が完了すると、作成したサービスの管理画面が表示される。【図 13】

【図 13】の管理画面の中にある「資格情報」が重要である。2018 年 8 月現在サービスによって「資格情報」には 2 種類あり、ユーザーとパスワードが表示されるサービスと、API Key が表示されるサービスとがある。Language Translator サービスの場合、API Key が表示されるため、この API Key と URL を控えておけばよい。

また、管理画面には「API リファレンス」画面へのリンクがあり、そこにアクセスすれば API の仕様が記載されている。2018 年 8 月 現 在 Language Translator サービスは、V3 というバージョンになっており、API の仕様は、【図 14】のとおりである。

Delphi/400 から利用するポイントを説明する。今回は、画面上に日本語で入力したテキストを英語に翻訳するアプリを作成する。（英語から日本語への翻訳も可能にする。）

WatsonAPI の場合、資格情報が必要なため、認証が必要である。WatsonAPI では、基本認証を使用することができ、これは「THTTPBasic Authenticator」コンポーネントが使用できる。

REST コンポーネントの設定は【図 15】のとおりである。

サービスの資格情報がユーザーとパスワードの場合は、そのまま「THTTPBasicAuthenticator」コンポーネントの Username プロパティと Password プロパティにセットすればよく、APIKey の場合は、Username プロパティに "apikey"、Password プロパティに資格情報の APIKey を入力すればよい。

今回のアプリの画面レイアウトは、【図 16】、プログラムは【ソース 4】のようになる。この Web サービスは、POST メソッドとなり、リクエスト本体に JSON 形式でパラメータを渡すところがポイントである。

実際に実行したアプリケーションが、【図 17】である。Watson API を利用するアプリケーションも REST + JSON で簡単に構築できることがわかる。

5.REST機能をもつコンポーネントの作成

今回、REST + JSON を使用した Web サービスの活用方法について具体例を挙げながら説明したが、Web サービスの課題点は、サービス提供者の都合により、サービスの仕様が変更されたり、サービス自体が終了してしまう可能性があることだ。

ある Web サービスを活用したアプリケーションを使用していた場合に、このような事態が発生すると、新しい仕様にあわせてプログラムを変更したり、あるいは代替サービスに置換したりといった作業が必要になる。こういったことを想定した場合、Web サービスの機能を個々のプログラムに都度記述する方法だと、修正ボリュームが多大になることが想像できる。

また、プログラムの中から Web サービスの部分を抜き出して修正しなければいけないため、煩雑な作業になることが予想される。

この問題を解決するには、どうすればよいか？　1 つの方法が各 Web サービスごとにコンポーネント化してしまうことである。そうすれば、Web サービスの仕様変更時にも、コンポーネントソースのみ修正し、各プログラムは、リコンパイルだけすれば済むはずである。

今回は、コンポーネント化の例として、前章で使用した Language Translator サービスのコンポーネント化を検討する。（TComponent を継承した TTranslator コンポーネント（非ビジュアルコンポーネント）を作成する。）

本稿では、コンポーネントそのものの基本的な作成手順については割愛するが、作成手順が分からない場合は、2012 年度版『ミガロ. テクニカルレポート』の SE 論文「カスタマイズコンポーネント入門」を参照していただきたい。

宣言部は、【ソース 5】となる。変換元の言語（SourceLanguage プロパティ）と変換後の言語（TargetLanguage プロパティ）、そして変換対象の文字列（Source プロパティ）を設定した後、Translate メソッドを実行すると翻訳が行われ、その結果は、Destination プロパティにセットされるという仕様を想定している。

ソース1　検索ボタンのOnClickイベント

```
uses REST.Types;

procedure TForm1.Button1Click(Sender: TObject);
begin
  //URLパラメータの指定
  RESTRequest1.Params.AddItem('CITY', Edit1.Text, pkURLSEGMENT);

  //リクエスト実行
  RESTRequest1.Execute;

  //レスポンスJSONを表示
  Memo1.Text := RESTResponse1.JSONText;
end;
```

図6　お天気情報取得実行例

図7　JSON パース例

変数sRet の値

```
{
"result":[
    {
    "subject":"english",
    "score": 80,
    "state":"good"
  },
   {
    "subject":"math",
    "score": 70,
    "state":"normal"
  }
]
}
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
  JSONValue: TJSONValue;
begin
  //JSONデータのパース
  JSONValue := TJSONObject.ParseJSONValue(sRet);
  //結果の取得
  ShowMessage(JSONValue.GetValue<string>('result[0].subject'));
end;
```

実行結果

実装部は、【ソース 6】および【ソース 7】となる。コンポーネントの生成時（Create メソッド）において、内部的に REST コンポーネントを生成し、Language Translator サービスの仕様に基づいたパラメータの設定を行っている。あとは、Translate メソッドにて、【ソース 4】と同様の変換処理を行っている。

このコンポーネントを使用したサンプルプログラムは、とてもシンプルである。画面レイアウトは、【図 18】、プログラムは【ソース 8】である。Web サービス自体をコンポーネント化しているため、API の仕様部分はこのプログラムには含まれていないことがわかる。これによって、将来 Web サービスが終了しても、コンポーネントの内容を別の Web サービスに変更すれば、個々のプログラムを変更する必要がなくなり、耐性の強いプログラムであることがわかるだろう。

6.さいごに

本稿では、REST による Web サービスを活用した Delphi/400 の機能拡張として、いくつかの Web サービスを使用した具体例を紹介してきた。REST + JSON 方式が簡単に Delphi/400 から活用できることがわかる。単純に REST コンポーネントを組み込むだけでも十分活用できるが、コンポーネント化まで検討することにより、より耐性の強い仕組みが作れるのである。ぜひ本稿を参考にいろいろな Web サービスの活用をご検討いただきたい。

M

ソース2　検索ボタンのOnClickイベント

```
uses REST.Types, System.JSON;

procedure TForm1.Button1Click(Sender: TObject);
var
  JSONValue: TJSONValue;
begin
  //URLパラメータの指定
  RESTRequest1.Params.AddItem('CITY', Edit1.Text, pkURLSEGMENT);

  //リクエスト実行
  RESTRequest1.Execute;

  //レスポンスJSONを表示
  JSONValue := RESTResponse1.JSONValue;

  //天気予報の取得
  EditDate1.Text := JSONValue.GetValue<string>('forecasts[0].date');
  EditDate2.Text := JSONValue.GetValue<string>('forecasts[1].date');
  EditDate3.Text := JSONValue.GetValue<string>('forecasts[2].date');
  EditWeather1.Text := JSONValue.GetValue<string>('forecasts[0].telop');
  EditWeather2.Text := JSONValue.GetValue<string>('forecasts[1].telop');
  EditWeather3.Text := JSONValue.GetValue<string>('forecasts[2].telop');
end;
```

図8　お天気情報取得実行例2

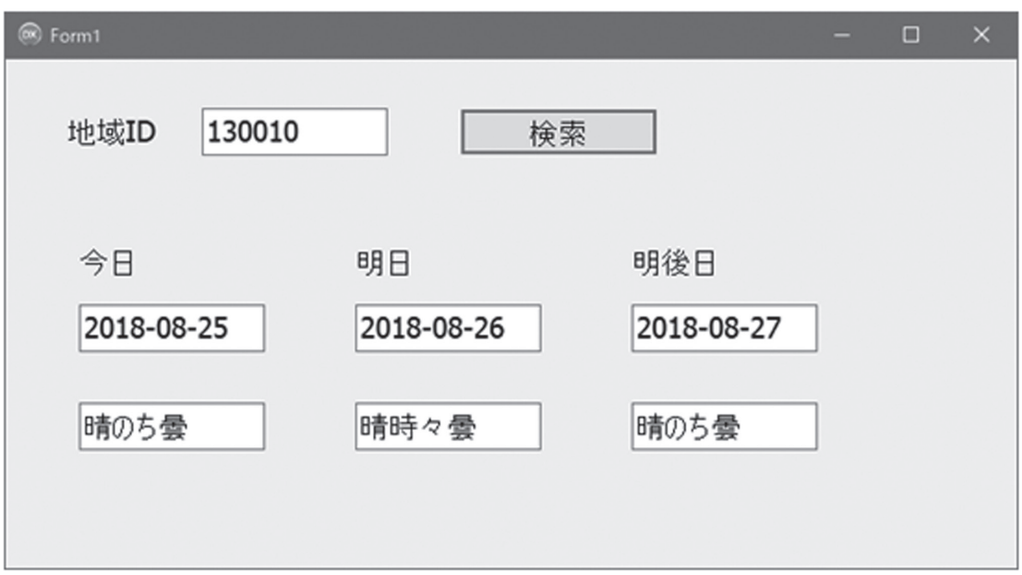
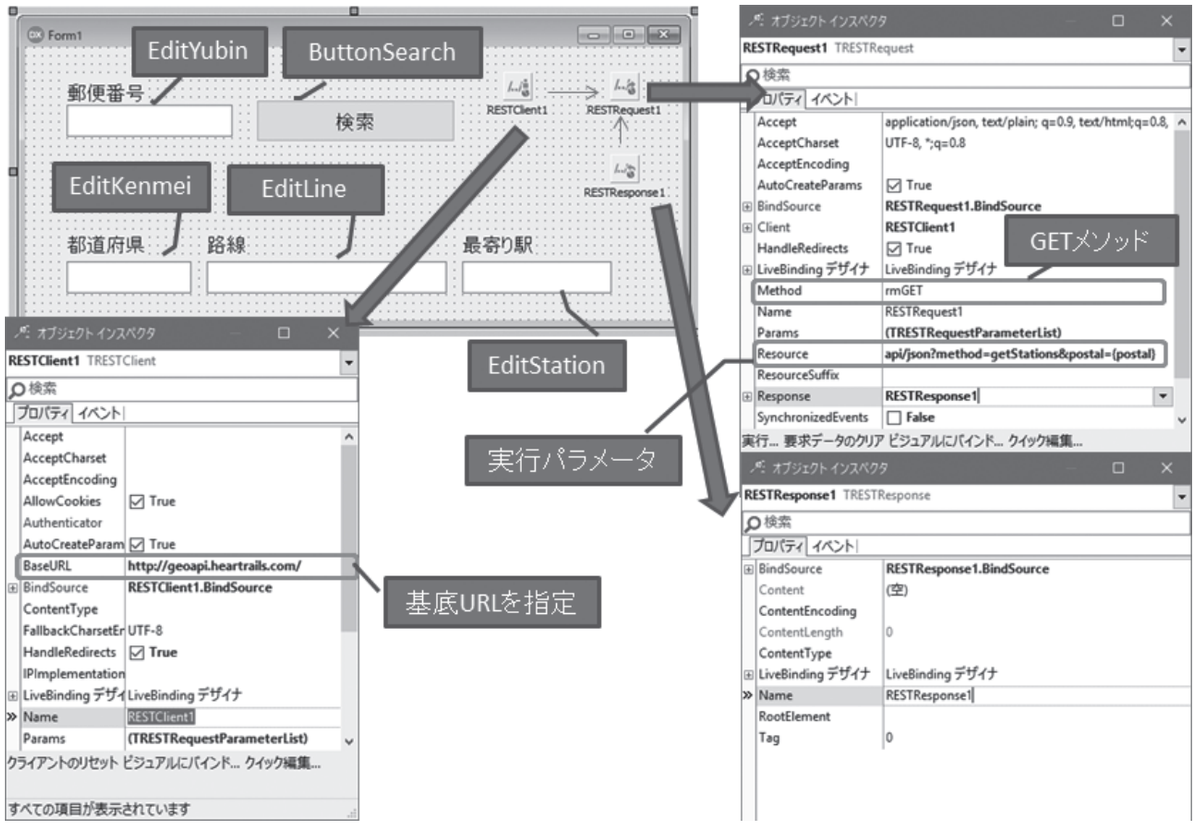


図9　最寄駅情報取得API



ソース3　検索ボタンのOnClickイベント

```
procedure TForm1.ButtonSearchClick(Sender: TObject);
var
  JSONValue: TJSONValue;
begin
  //URLパラメータの指定
  RESTRequest1.Params.AddItem('postal', EditYubin.Text, pkURLSEGMENT);

  //リクエスト実行
  RESTRequest1.Execute;

  //レスポンスJSON取得
  JSONValue := RESTResponse1.JSONValue;

  //JSONより値を取得
  EditKenmei.Text :=
    JSONValue.GetValue<string>('response.station[0].prefecture');
  EditLine.Text :=
    JSONValue.GetValue<string>('response.station[0].line');
  EditStation.Text :=
    JSONValue.GetValue<string>('response.station[0].name');
end;
```

図10　最寄駅情報取得

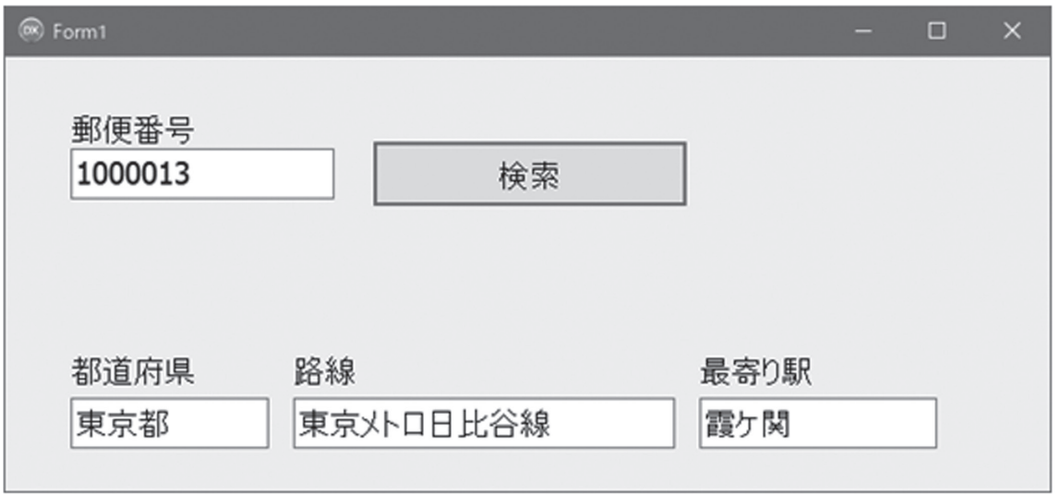


図11　Webサービス 一覧サイト

図12　Watson API一覧

分類	API種類	機能	ライト
照会応答系	Assistant（照会応答）	自然言語で対話可能なアプリケーションを、シンプルな開発ツールで迅速に構築	対応
言語系	Language Translator（言語変換）	コンテンツのテキストを、ある言語から別の言語にリアルタイムで翻訳	対応
	Natural Language Understanding(自然言語理解)	テキスト分析を行い、コンテンツから概念、エンティティー、キーワード、カテゴリー、感情、関係、意味役割などのメタデータを抽出	対応
心理系	Personality Insights（性格分析）	テキストから筆者のパーソナリティ（ビッグ・ファイブ、価値、ニーズ）の3つの特徴を推測	対応
音声系	Speech to Text(音声認識)	ディープ・ラーニングを活用して、音声からテキストを書き起こす	対応
	Text to Speech(音声合成)	テキストから自然な音声を合成	対応
知識探索系	Discovery(検索)	大量のデータを検索するとともに、適切な意思決定を支援	対応
	Knowledge Studio	業界や分野ごとの知識だけでなく、各分野の言葉の使われ方の微妙な違いをWatsonに教えることができるツール	対応
画像系	Visual Recognition(画像認識)	ディープ・ラーニングを使用して、画像に写った物体・情景・顔など様々なものを分析・認識	対応

※2018年8月現在

図13　Language Translatorサービス管理画面

図14　Language TranslatorサービスのAPI仕様

リクエスト
POST /v3/translate

※2018年8月現在
※API仕様の詳細は、APIリファレンスを参照

パラメータ	タイプ	内容
version	URLパラメータ	V3公開日（固定値：2018-05-01）
request	リクエストの本体	翻訳に必要な下記パラメータをJSON形式で指定
text	string []	翻訳したい元のテキストを指定
source	string	元テキストの言語を指定（ja,en...）
target	string	翻訳後の言語を指定（ja,en...）

レスポンス例

```
{
 "translations" : [ {
  "translation" : "Hi,"
 } ],
 "word_count" : 1,
 "character_count" : 5
}
```

パラメータ	内容
translations: [translation]	翻訳結果のテキスト
word_count	元テキストの単語数
character_count	元テキストの文字数

図15　Language Translatorサービス REST設定

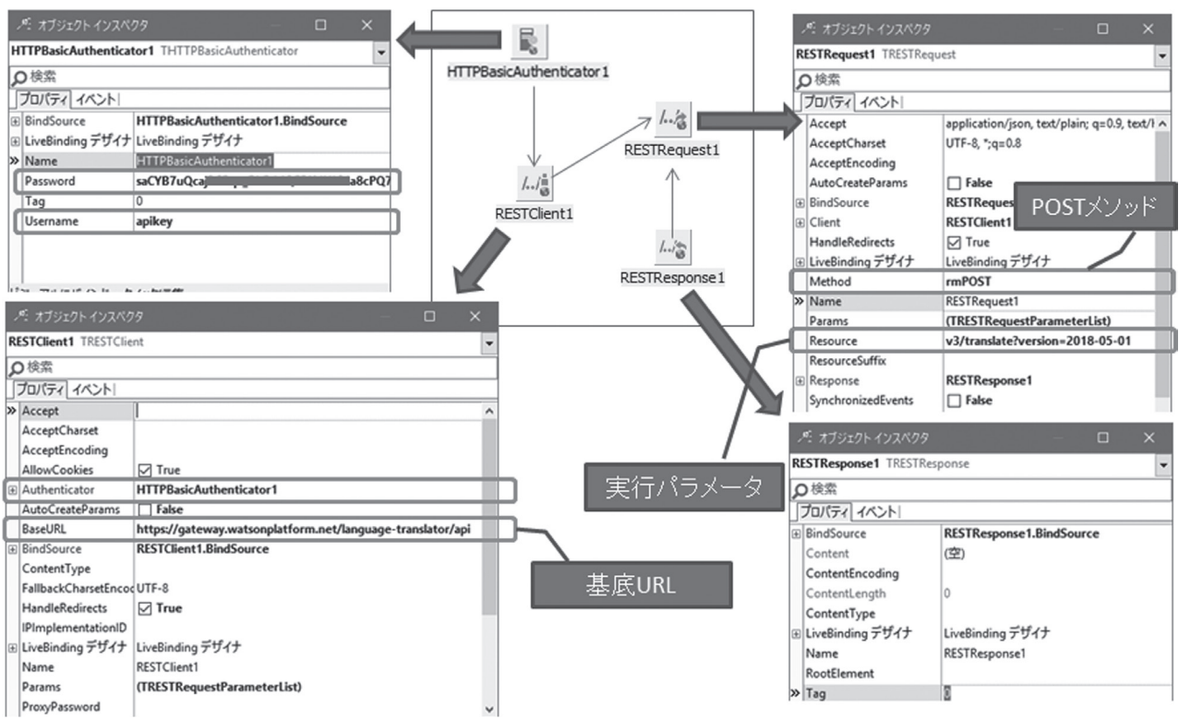
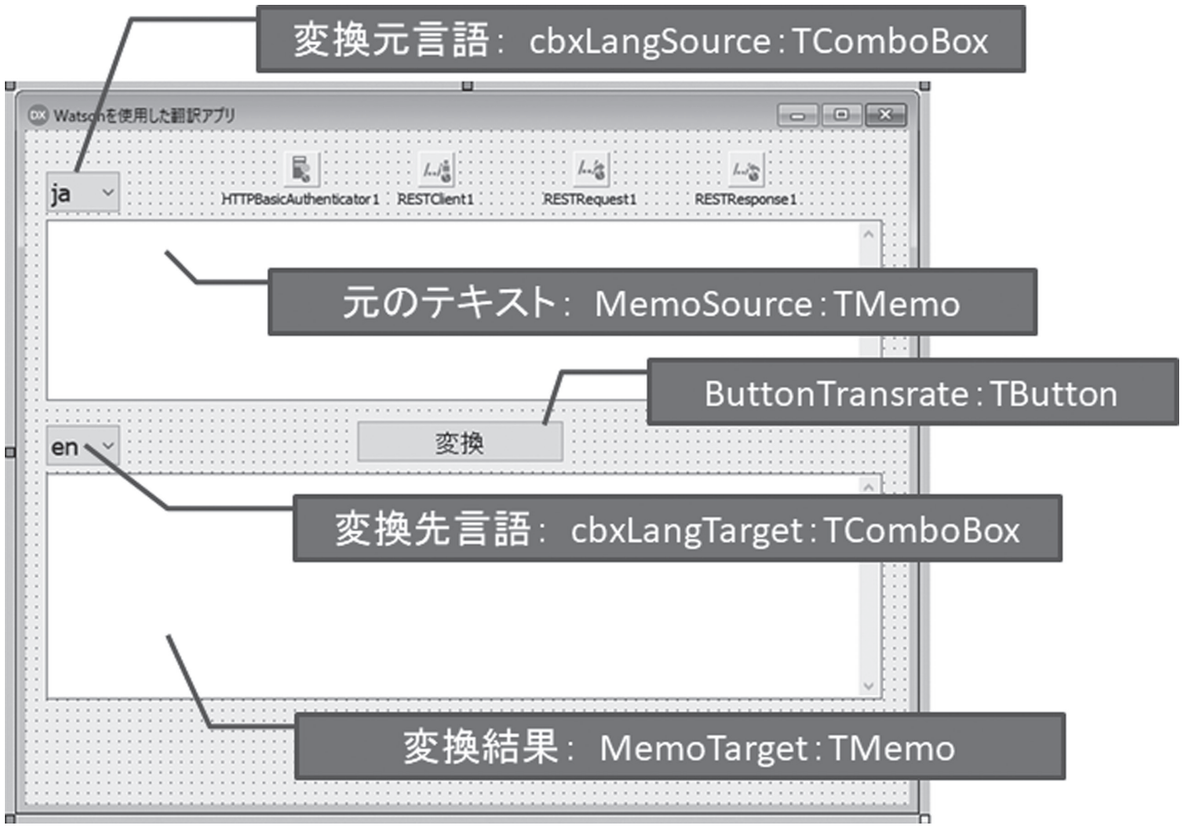


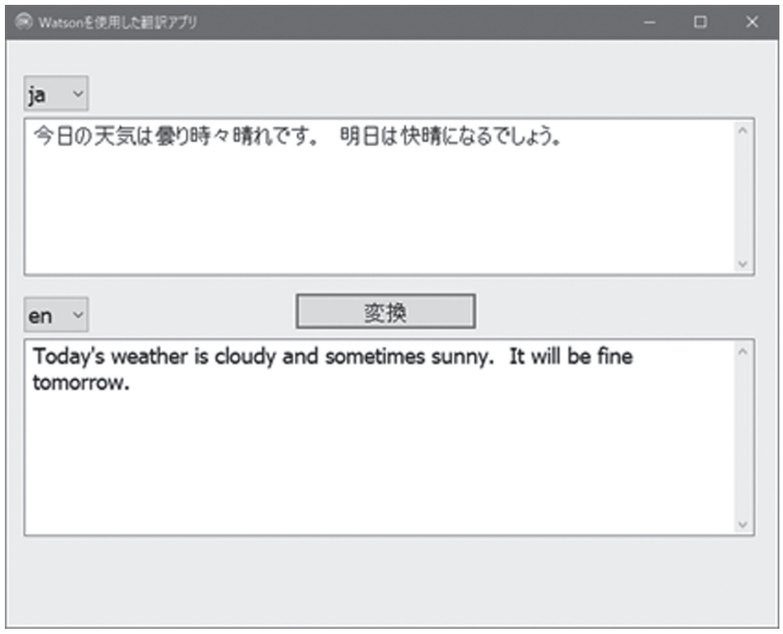
図16　翻訳アプリ画面レイアウト



ソース4　変換ボタンのOnClickイベント

```
procedure TForm1.ButtonTransrateClick(Sender: TObject);
var
  RequestJson: TJSONObject;
  JSONValue: TJSONValue;
begin
  //翻訳パラメータの指定
  RequestJson := TJSONObject.Create;
  RequestJson.AddPair('text', MemoSource.Text);       // 翻訳対象文字列
  RequestJson.AddPair('source', cbxLangSource.Text); // 変換元言語
  RequestJson.AddPair('target', cbxLangTarget.Text); // 変換先言語

  //条件の指定
  RESTRequest1.Params.AddItem('', RequestJson.ToString, pkGETorPOST
    , [], ctAPPLICATION_JSON);

  //リクエストの実行
  RESTRequest1.Execute;

  //レスポンスJSONの取得
  JSONValue := RESTResponse1.JSONValue;

  //翻訳結果の取得と表示
  MemoTarget.Text
    := JSONValue.GetValue<string>('translations[0].translation');
end;
```

図17　翻訳アプリ実行結果



ソース5　TTranslatorコンポーネントの宣言部

```
unit Translator;

interface

uses
  System.SysUtils, System.Classes, System.JSON, REST.Client,
  REST.Authenticator.Basic, Rest.Types, IPPeerClient;

type
  TLanguage = (Japanese, English);

  TTranslator = class(TComponent)
  private
    { Private 宣言 }
    FRestClienct: TRESTClient;
    FRestRequest: TRESTRequest;
    FRestResponse: TRESTResponse;
    FBasicAuthenticator: THTTPBasicAuthenticator;
    FSourceLanguage: TLanguage;
    FTargetLanguage: TLanguage;
    FSource: String;
    FDestination: String;
    function GetLanguageName(ALanguage: TLanguage): String;
  protected
    { Protected 宣言 }
  public
    { Public 宣言 }
    constructor Create(AOwner: TComponent); override;
    procedure Translate;
    property Destination: String read FDestination;
  published
    { Published 宣言 }
    property SourceLanguage: TLanguage read FSourceLanguage write FSourceLanguage;
    property TargetLanguage: TLanguage read FTargetLanguage write FTargetLanguage;
    property Source: String read FSource write FSource;
  end;

procedure Register;
```

ソース6　TTranslatorコンポーネントの実装部①

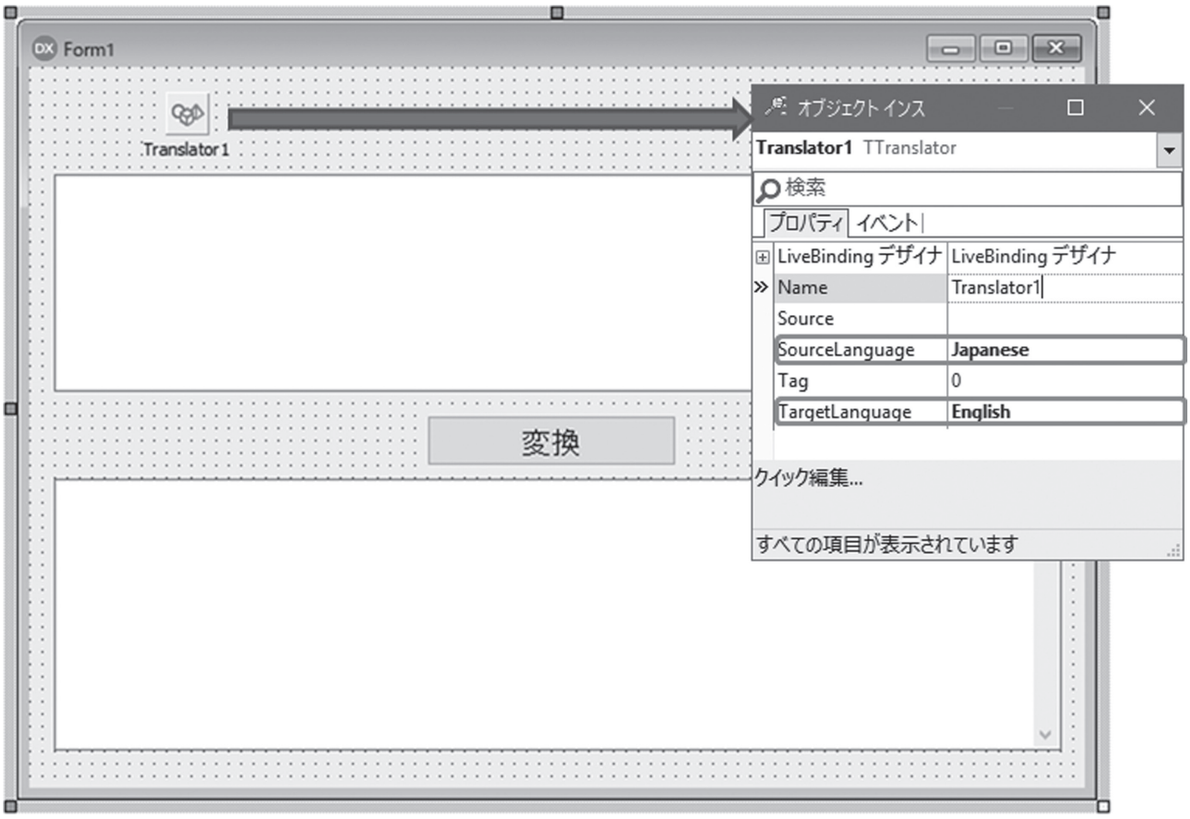
```
implementation

procedure Register;
begin
  RegisterComponents('Samples', [TTranslator]);
end;

{ TTranslator }

constructor TTranslator.Create(AOwner: TComponent);
begin
  inherited;
  //RESTコンポーネントの生成
  FRestClienct := TRESTClient.Create('https://gateway.watsonplatform.net'
    + '/language-translator/api');
  FRestRequest := TRESTRequest.Create(FRestClienct);
  FRestResponse := TRESTResponse.Create(Self);
  FBasicAuthenticator := THTTPBasicAuthenticator.Create('apikey',
    'saCYB7uC[redacted]69MxzH');

  //RESTコンポーネント関連付け
  FRestClienct.Authenticator := FBasicAuthenticator;
  FRestRequest.Response := FRestResponse;
  //実行メソッドおよびパラメータの初期設定
  FRestRequest.Method := rmPost;
  FRestRequest.Resource := 'v3/translate?version=2018-05-01';

  //プロパティ値の初期設定
  FSourceLanguage := Japanese;
  FTargetLanguage := English;
end;

function TTranslator.GetLanguageName(ALanguage: TLanguage): String;
begin
  case ALanguage of
    Japanese: Result := 'ja';  //日本語="ja"
    English: Result := 'en';   //英語="en"
  end;
end;
```

Language Translatorサービスの
RESTコンポーネントを生成

ソース7　TTranslatorコンポーネントの実装部②

```
procedure TTranslator.Translate;
var
  RequestJson: TJSONObject;
  JSONValue: TJSONValue;
begin
  //翻訳パラメータの指定
  RequestJson := TJSONObject.Create;
  RequestJson.AddPair('text', FSource);      // 翻訳対象文字列
  RequestJson.AddPair('source', GetLanguageName(FSourceLanguage)); // 変換元言語
  RequestJson.AddPair('target', GetLanguageName(FTargetLanguage)); // 変換先言語

  //条件の指定
  FRestRequest.Params.AddItem('', RequestJson.ToString, pkGETorPOST
    , [], ctAPPLICATION_JSON);

  //リクエストの実行
  FRESTRequest.Execute;

  //レスポンスJSONデータの取得
  JSONValue := FRESTResponse.JSONValue;

  //翻訳結果を変数に格納
  FDestination := JSONValue.GetValue<string>('translations[0].translation');
end;
```

図18　コンポーネント化した翻訳アプリ



ソース8　変換ボタンのOnClickイベント

```
procedure TForm1.ButtonTransrateClick(Sender: TObject);
begin
  //変換対象文字列のセット
  Translator1.Source := MemoSource.Text;

  //変換実行
  Translator1.Translate;

  //結果の表示
  MemoTarget.Text := Translator1.Destination;
end;
```