【セッションNo. 1】

IBM Watson AI と連携！ Delphi/400からのAPI呼び出しと実装テクニック

株式会社ミガロ.
RAD事業部　営業・営業推進課
尾崎　浩司

はじめに

■ コグニティブ技術の発達

- 「コグニティブ」とは？
  - 日本語で「認知」のこと。
  - 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

- 近年　コグニティブやAI技術の発達が凄まじい
  - 身近な機器も進化
    - iPhone　Siri
    - スマートスピーカー
  - 検索サイトも進化
    - Google画像検索

音声を使用して
色々な操作が可能

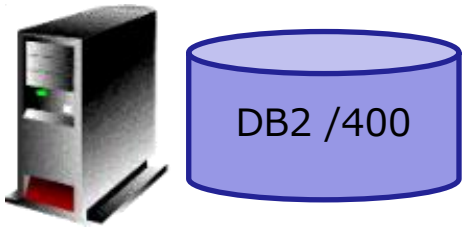
撮影した写真

写真をアップロード

画像を分析し、
「姫路城」を特定

■ コグニティブ技術の発達

- 従来のシステムとの違いは？
  - 本質的な違いとして、従来システムの定型データに加え、音声・画像・文章などの非定型データも処理できること。



定型データ

非定型データ

技術をミックスすれば、人の作業を補助し、より便利なシステム構築が可能に！

【アジェンダ】

1. Watson とは？
2. Watson APIを使った開発方法
3. 音声を使ったWatson APIの活用
4. Watson APIを活用したチャットボット
5. まとめ

・補足資料
- 自動翻訳機アプリ　ソースコード
- 物件紹介チャットアプリ　ソースコード

1．Watson とは？

■ Watson 概要

IBM Watson

- IBMが開発した質問応答システム・意思決定支援システム
- 人工知能ではなく、自然言語を理解・学習し人間の意思決定を支援する『コグニティブ・コンピューティング・システム（Cognitive Computing System）』と定義

- IBMのクラウドサービス（IBM Cloud）上のAPIとして提供されており、誰でも利用可能！

IBM Cloud ライト・アカウント

- IBM Cloudの各種サービスを無料で使用できるアカウント
  - 利用期間の制限もなく、商用に使用してもOK
  - クレジットカード登録も不要

- ただし下記制限あり
  - 上位版でないと提供されないサービスあり
  - サービス毎に無料で使用できる上限あり
  - 各サービスの中でも一部使えない機能あり
  - 一定期間未使用で、サービスが自動削除

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

■ IBM Cloud上のWatson API

- 日本環境で使用できる主なWatson API
  - 一部APIを除き、ライト・アカウントでも使用可能

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 Classifier(自然言語分類)	自然言語テキストの背後にある意図を解釈し、関連度合いを信頼度レベル付けして分類	未対応
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

Watson API　活用例

- 色々な分野での活用が始まっている
  - チャットボットを使用した
    カスタマーサービス　（Assistant）
  - 迷惑メールの分類
    （Natural Language Classifier）
  - 製造ラインでの画像検査による
    不良品判定（Visual Recognition）
  - コールセンターでの自動テキスト
    作成（Speech To Text）
  - 目の不自由な方への音声案内
    （Text To Speech）

- IBM HP内に活用例ページあり

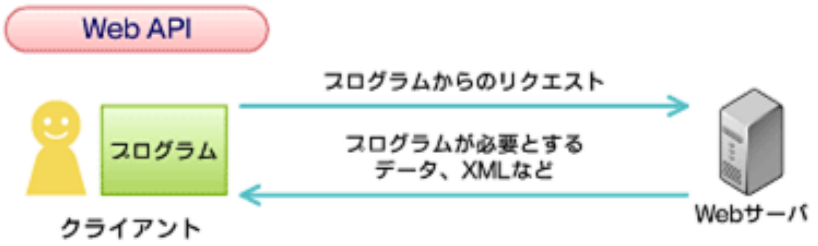
  https://www.ibm.com/watson/jp-ja/use-cases/

2．Watson APIを使った開発方法

■ Watson API　アクセス方式

- Watsonの各サービスは、Web APIとして提供

- Web API
  - コンピュータプログラムの提供する機能を外部の別のプログラムから呼び出して、利用するための手順・規約。HTTPなどWebの技術を使用して構築されたもの。



- Watson APIで使用されるアクセス方式
  - REST　＋　JSON方式
  - Web APIで現在最も利用されている方式を採用。

REST + JSON　はDelphi/400のコンポーネントで使用できる為、
Watson APIとの連携が容易に行える！

Watson API　アクセス方式

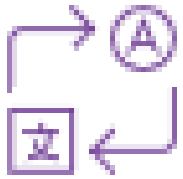
- RESTサービス（REpresentational State Transfer）
  - Webサービスの設計モデル
  - ネットワーク上のデータ（リソース）を一意なURIで表す
  - サービスのURIにHTTPメソッドでアクセスすることでデータの送受信を行う
  - パラメータを指定して特定のURIにアクセスするとJSON(またはXML)で応答される

- JSON (JavaScript Object Notation)
  - 軽量のデータ交換フォーマット
  - キーと値のペアでデータを管理
    ({“key”:”Value”})
  - 配列の使用もできる。（[ ]で表現）

JSON例

```
{
“result":[

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

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

]
}
```

■ Watson API　使用方法



- Language Translatorサービス
  - 言語変換サービス　（日本語→英語、英語→日本語など）
  - 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、高精度の翻訳が可能。

＜活用例＞
- 海外担当者とのメールのやりとりに活用
- 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

（デモサイト）

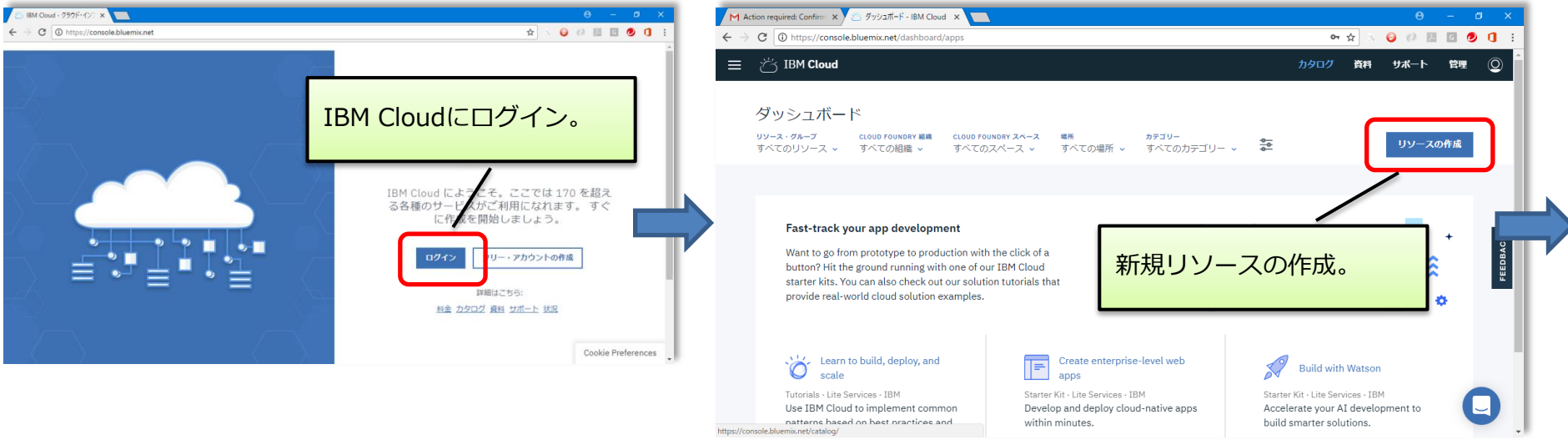
https://language-translator-demo.ng.bluemix.net/

Watson API　使用方法

- Watson APIを使用するために
  - Watsonの各サービスは、事前にインスタンスの作成が必要

- インスタンス作成手順

  IBMクラウドにログインして行う。　https://console.bluemix.net/



■ Watson API　使用方法

- インスタンス作成手順

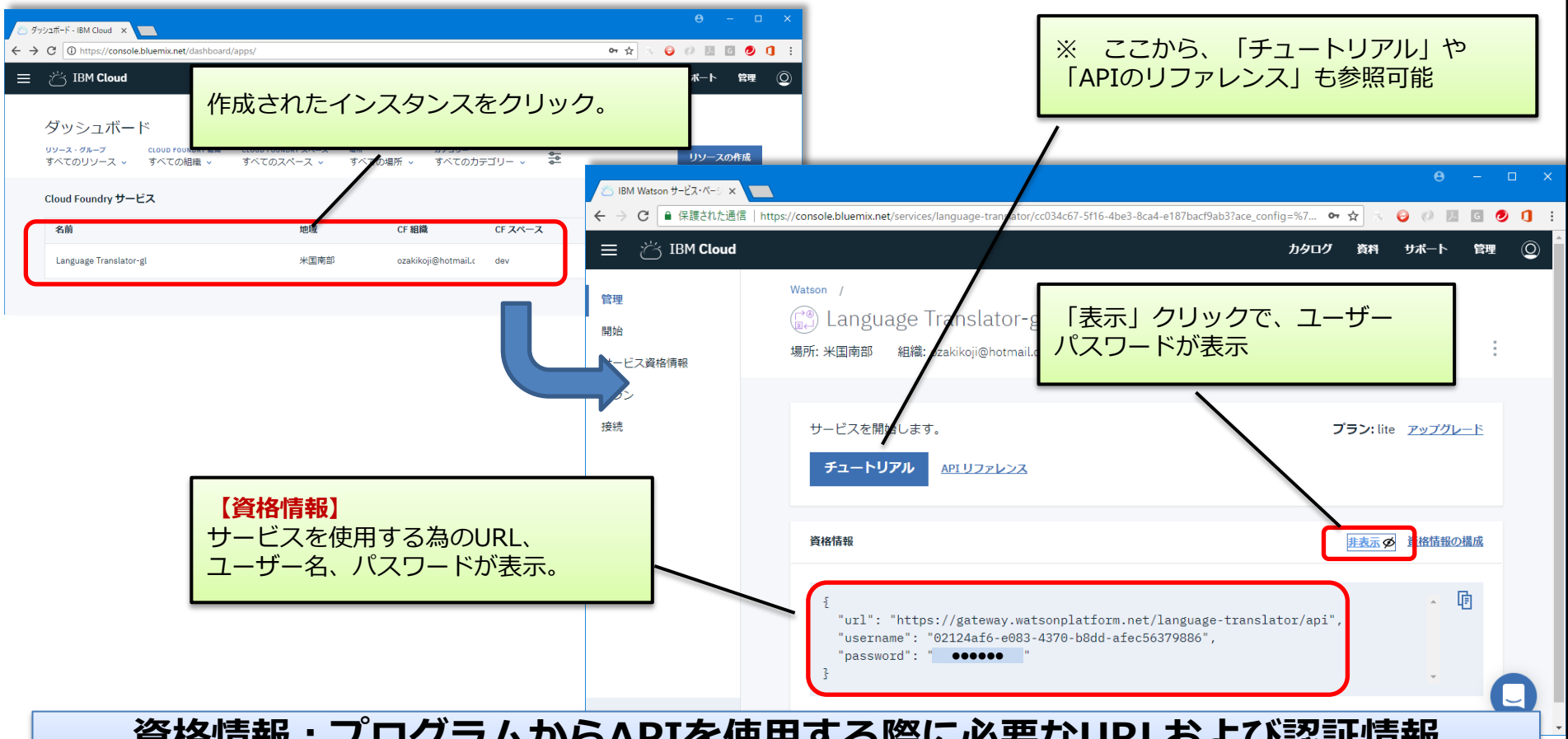
作成したいサービスを選択。

価格プランが表示。ライトの場合は、
月当たりの使用可能量が確認できる。

実行すると、インスタンスが作成される。

■ Watson API　使用方法

- インスタンス作成手順
  - インスタンス作成後、APIを使用するのに必要な資格情報を取得



資格情報：プログラムからAPIを使用する際に必要なURLおよび認証情報（使用するWatson API インスタンス単位に必要）

Watson API　使用方法

- Language Translatorサービスの API仕様

リクエスト
GET /v2/translate

※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
text	URL パラメータ	翻訳したい元のテキストを指定
source	URL パラメータ	元テキストの言語を指定（ja,en...）
target	URL パラメータ	翻訳後の言語を指定（ja,en...）

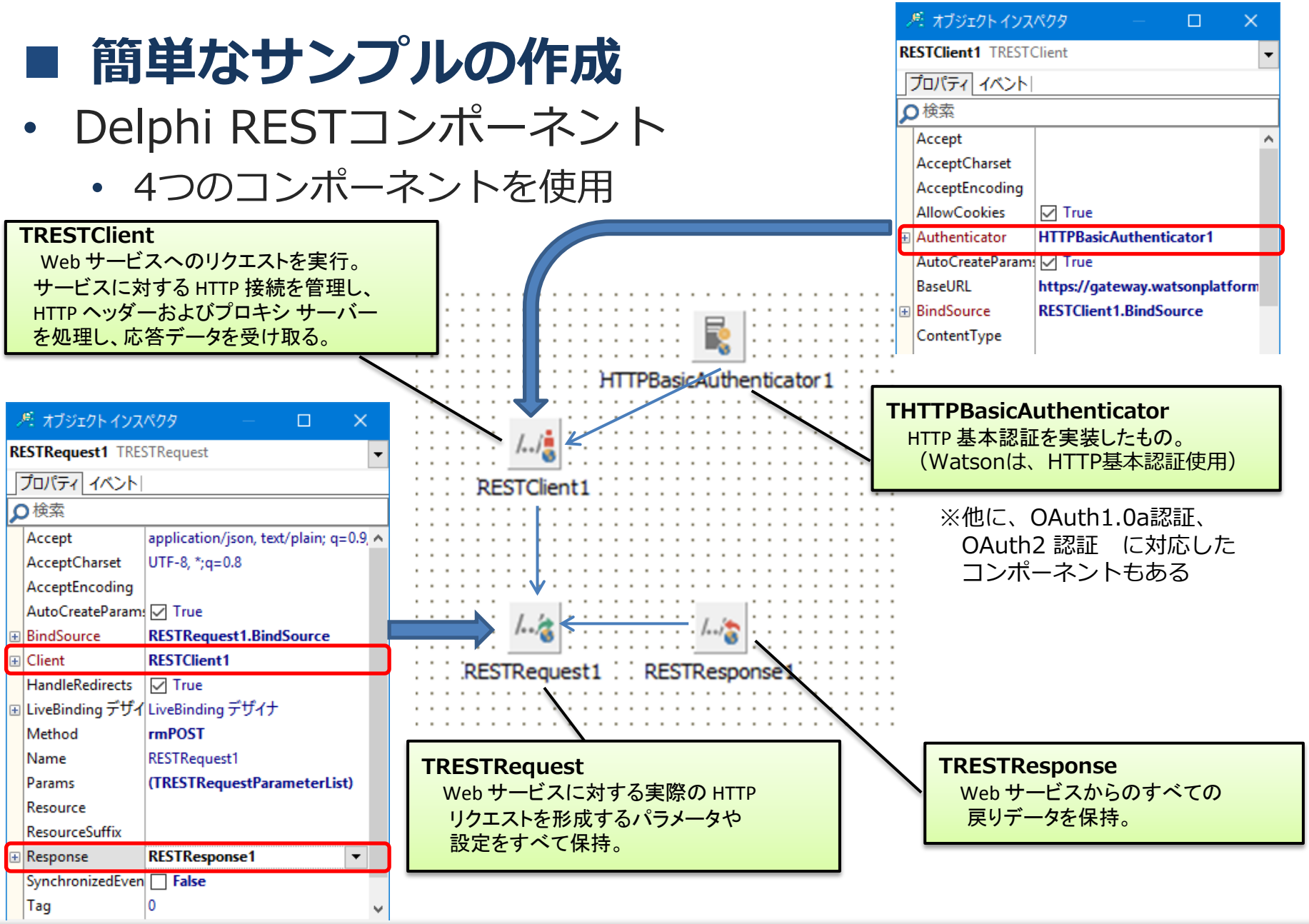
レスポンス例

```
{
  "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

■ 簡単なサンプルの作成

- Delphi RESTコンポーネント
  - 4つのコンポーネントを使用

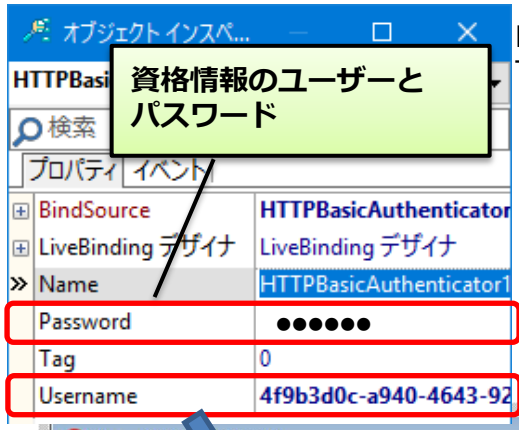


■ 簡単なサンプルの作成

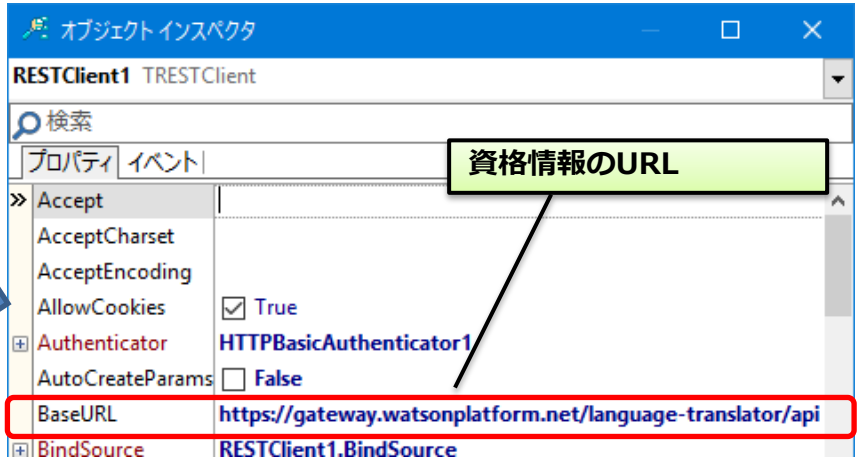
- 日本語↔英語翻訳ツール

HTTPBasicAuthenticator1:
THTTPBasicAuthenticator

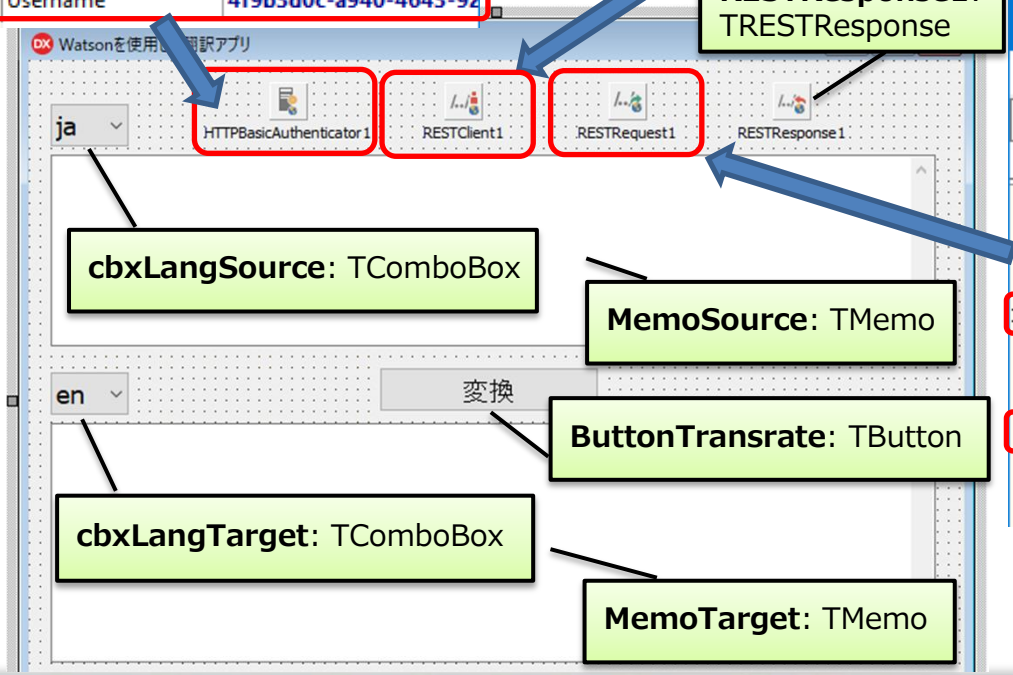
資格情報のユーザーとパスワード



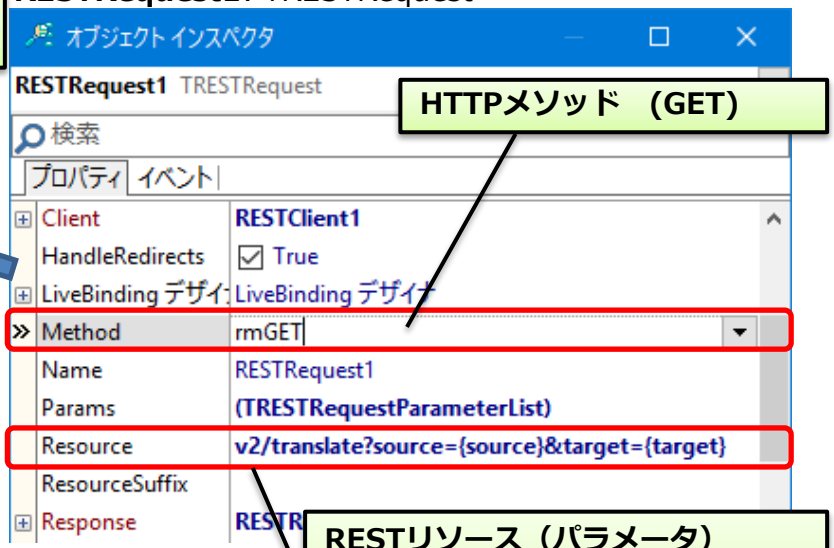
RESTClient1: TRESTClient



RESTResponse1:
TRESTResponse



RESTRequest1: TRESTRequest



■ 簡単なサンプルの作成

- コード例

```
procedure TForm1.ButtonTransrateClick(Sender: TObject);
begin
  //翻訳パラメータの指定
  RESTRequest1.Params.AddItem('source', cbxLangSource.Text
    , pkURLSEGMENT);        //翻訳元言語
  RESTRequest1.Params.AddItem('target', cbxLangTarget.Text
    , pkURLSEGMENT);        //翻訳先言語
  RESTRequest1.Params.AddItem('text', MemoSource.Text
    , pkGETorPOST);         //翻訳元テキスト

  //リクエストの実行
  RESTRequest1.Execute;

  //処理結果の表示
  MemoTarget.Lines.Clear;
  MemoTarget.Lines.Add(RESTResponse1.Content);
end;
```

AddItem パラメータをセット

pkURLSEGMENT：パラメータを URL セグメントの値として使用。リソース中の { } 部分を置換。

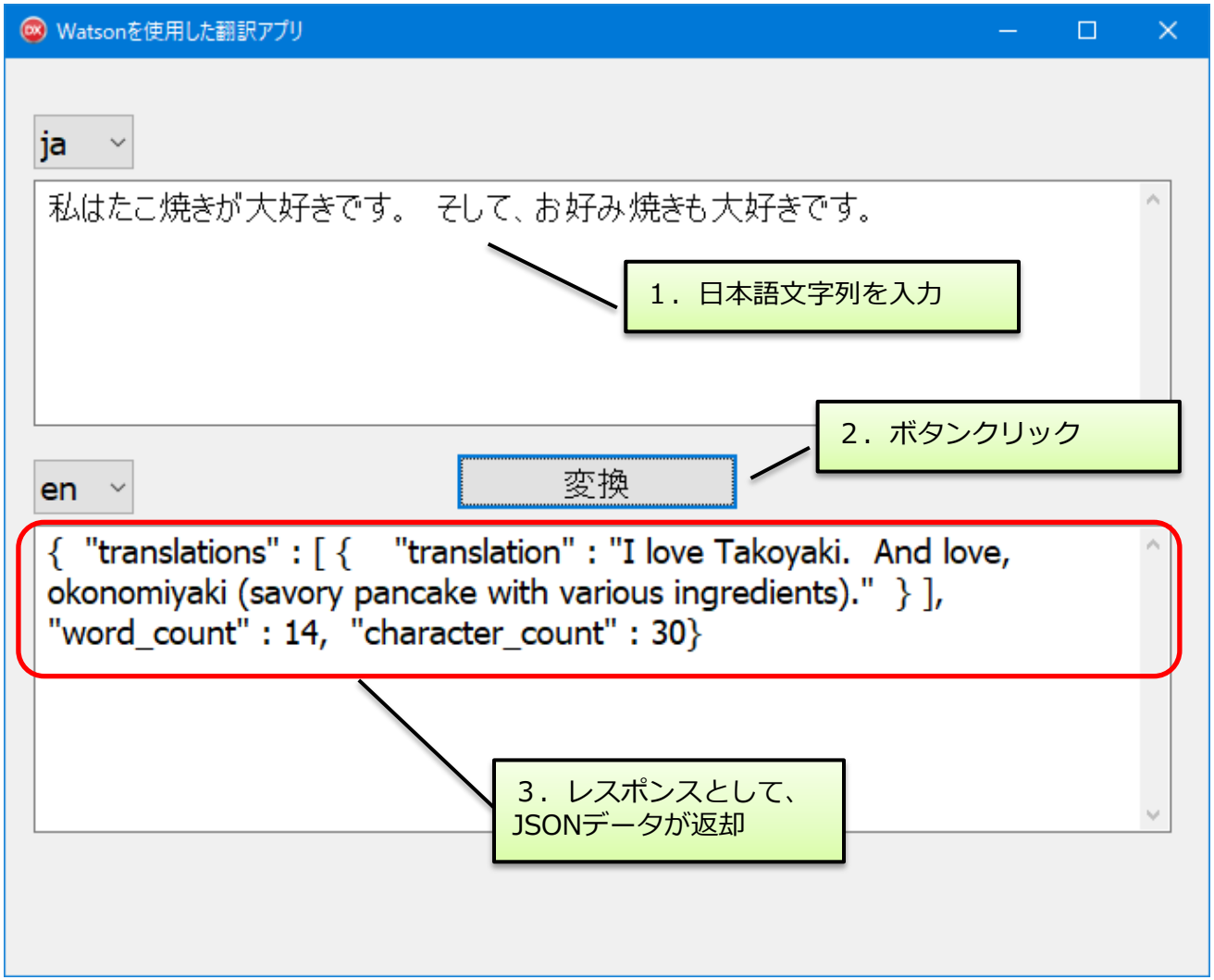
pkGETorPOST：パラメータを、URL パラメータ（GET リクエストの場合）または本体のパラメータ（POST リクエストの場合）として、送信。

HTTP 要求を実行

応答内容を表す文字列

■ 簡単なサンプルの作成

- 実行結果



■ 簡単なサンプルの作成

- JSONデータの取り扱い
  - Delphi/400には、JSONデータを取り扱うためのRTLとして、[**System.JSON**]ユニットが用意されている。

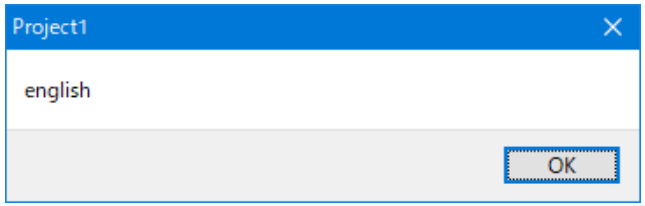
  - **TJSONObject**： JSON オブジェクトを実装したクラス
    （メソッド）**ParseJSONValue**： 解析したデータに対応する JSON 値を返す。
  - **TJSONValue** ： 文字列、数値、オブジェクト、配列、true/falseの型を持つ
    すべての JSON クラスの上位クラス
    （メソッド）**GetValue<T>**： 指定したパスの値をT型として取得

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



■ 簡単なサンプルの作成

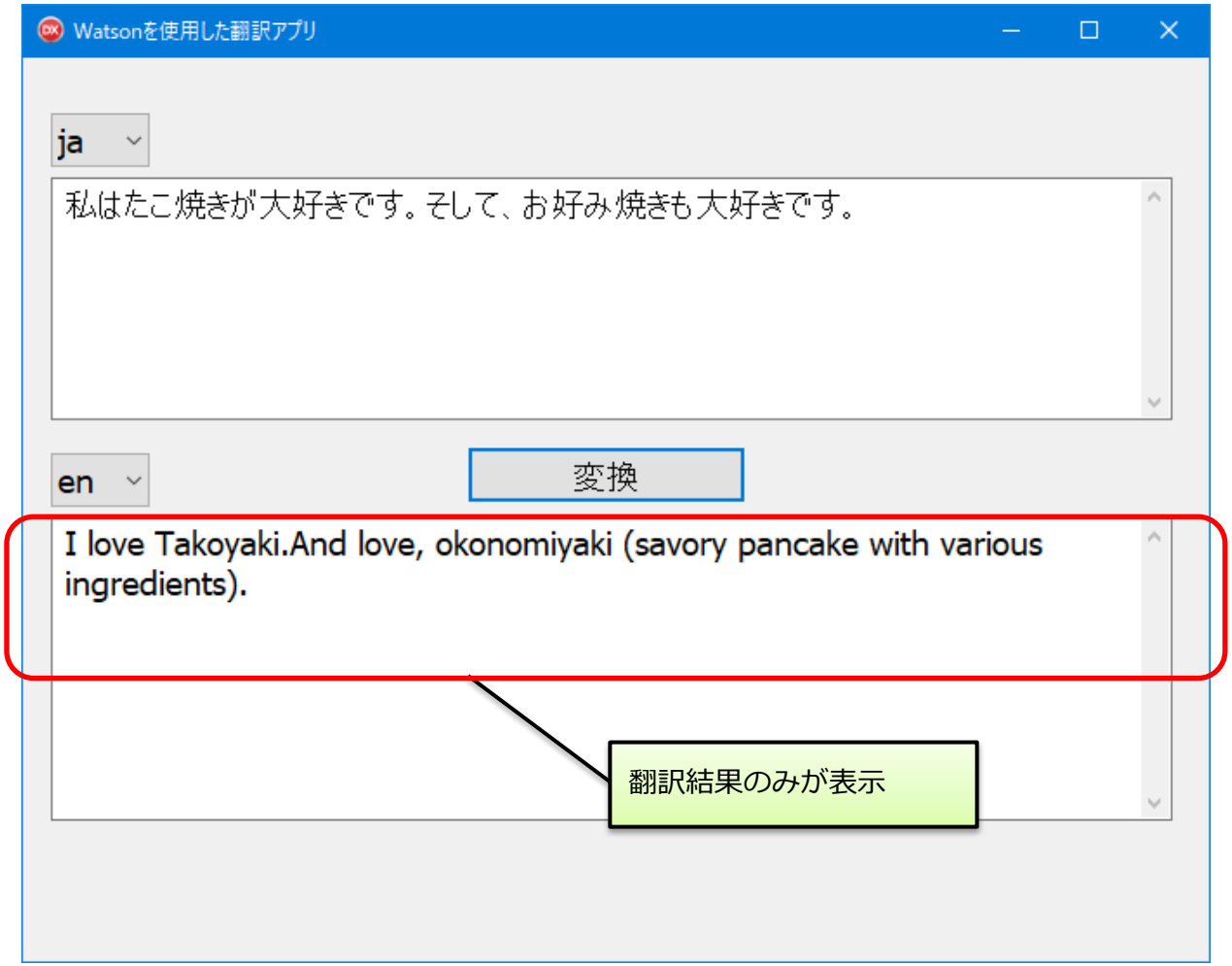
- 修正コード例

```
procedure TForm1.ButtonTransrateClick(Sender: TObject);
var
  JSONValue: TJSONValue;
  sRet: String;
begin
  //翻訳パラメータの指定
  RESTRequest1.Params.AddItem('source', cbxLangSource.Text
    , pkURLSEGMENT);        //翻訳元言語
  RESTRequest1.Params.AddItem('target', cbxLangTarget.Text
    , pkURLSEGMENT);        //翻訳先言語
  RESTRequest1.Params.AddItem('text', MemoSource.Text
    , pkGETorPOST);         //翻訳元テキスト

  //リクエストの実行
  RESTRequest1.Execute;

  //翻訳テキストの取得
  JSONValue := TJSONObject.ParseJSONValue(RESTResponse1.Content);
  sRet := JSONValue.GetValue<string>('translations[0].translation');

  //処理結果の表示
  MemoTarget.Lines.Clear;
  MemoTarget.Lines.Add(sRet);
end;
```

パースした結果を文字列変数にセット

■ 簡単なサンプルの作成

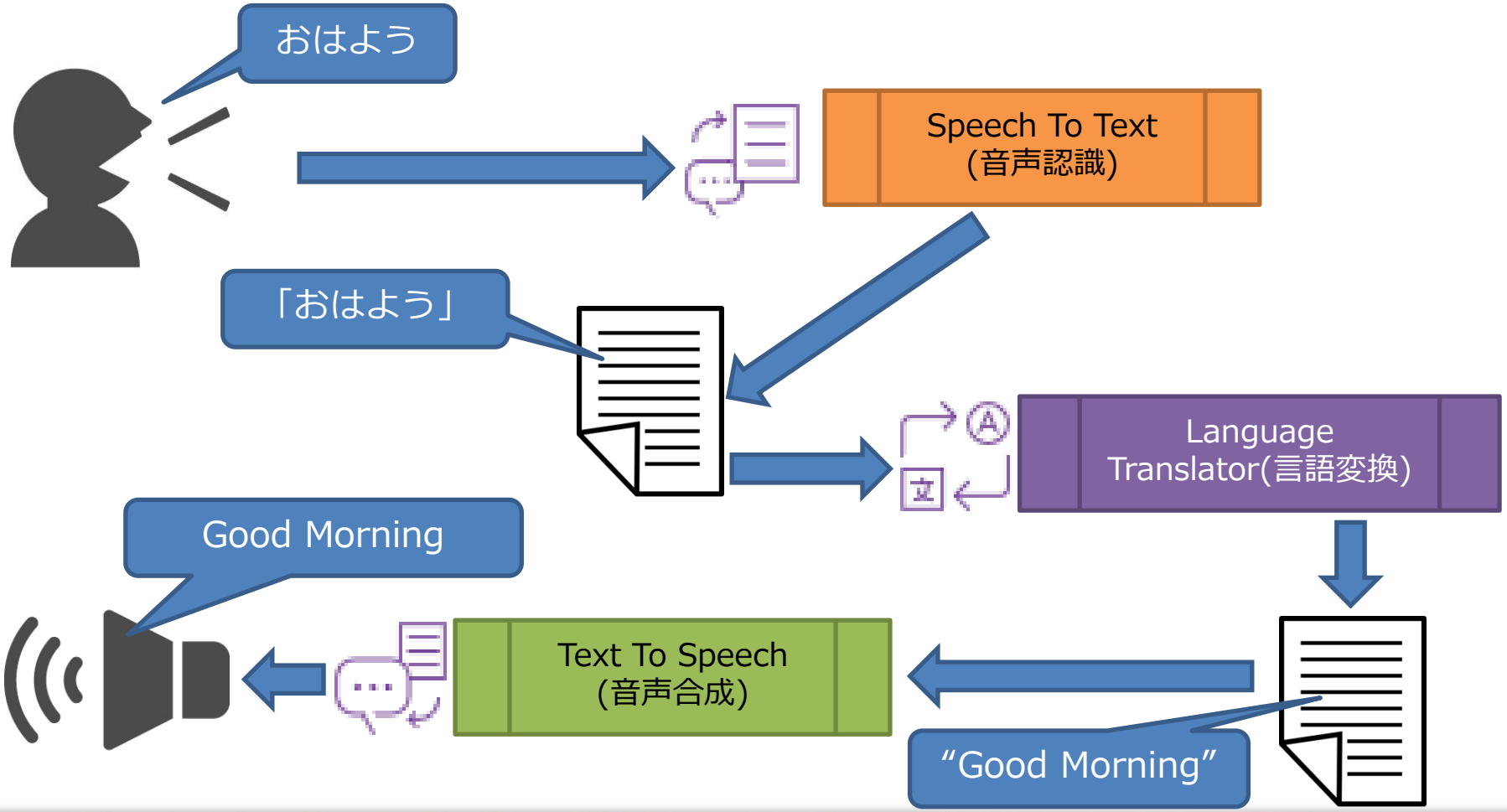
- 実行結果



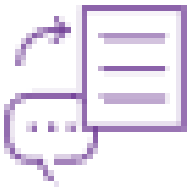
３．音声を使ったWatson APIの活用

■ 検討するアプリ

- 自動翻訳機
  - 日本語でしゃべった内容を英語に変換し、音声出力する。



■ 音声認識の作成



- Speech To Textサービス
  - 音声をテキスト文字列に変換するサービス
  - カスタマイズ辞書を登録することで、専門用語等も正確に変換することが可能
  - サポート言語：ブラジル・ポルトガル語、フランス語、日本語、中国語（標準）、アラビア語、スペイン語、イギリス英語、アメリカ英語

＜活用例＞
- アプリケーションの音声操作
- コールセンターにおける通話内容の自動テキスト化
- 会議のリアルタイム議事録作成

（デモサイト）

https://speech-to-text-demo.ng.bluemix.net/

■ 音声認識の作成

- Speech To TextサービスのAPI仕様

リクエスト
POST /v1/recognize

※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
Content-Type	リクエストの HTTP ヘッダー	音声データのフォーマット（MIME type)
model	URL パラメータ	言語と帯域のモデルを指定 （日本語狭帯域：ja-JP_NarrowbandModel)
audio	リクエストの本体	音声データ（ストリーム）

レスポンス例

```
{  "results": [{
     "alternatives": [{
        "confidence": 1.0,
        "transcript": "おはよう"
       }],
     "final": true
    } ],
  "result_index": 0
}
```

パラメータ	内容
confidence	変換の信頼度（0-1）
transcript	変換結果
final	変換の終了を示す
result_index	結果のインデックス

■ 音声認識の作成

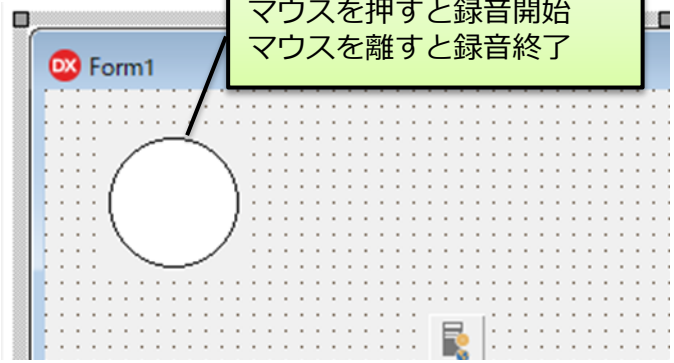
- マイクから録音する方法
  - MediaPlayerコンポーネント
    - 動画の再生や、音声の録音/再生が可能

.wav(音声ファイル）形式で
録音を開始

```
procedure TForm1.Shape1MouseDown(Sender: TObject; Bu
  Shift: TShiftState; X, Y: Integer);
begin
  MediaPlayer1.DeviceType := dtWaveAudio;
  MediaPlayer1.FileName := #0;
  MediaPlayer1.Open;
  MediaPlayer1.StartRecording;
  Shape1.Brush.Color := clRed;
end;
```

Shape1: TShape

マウスを押すと録音開始
マウスを離すと録音終了



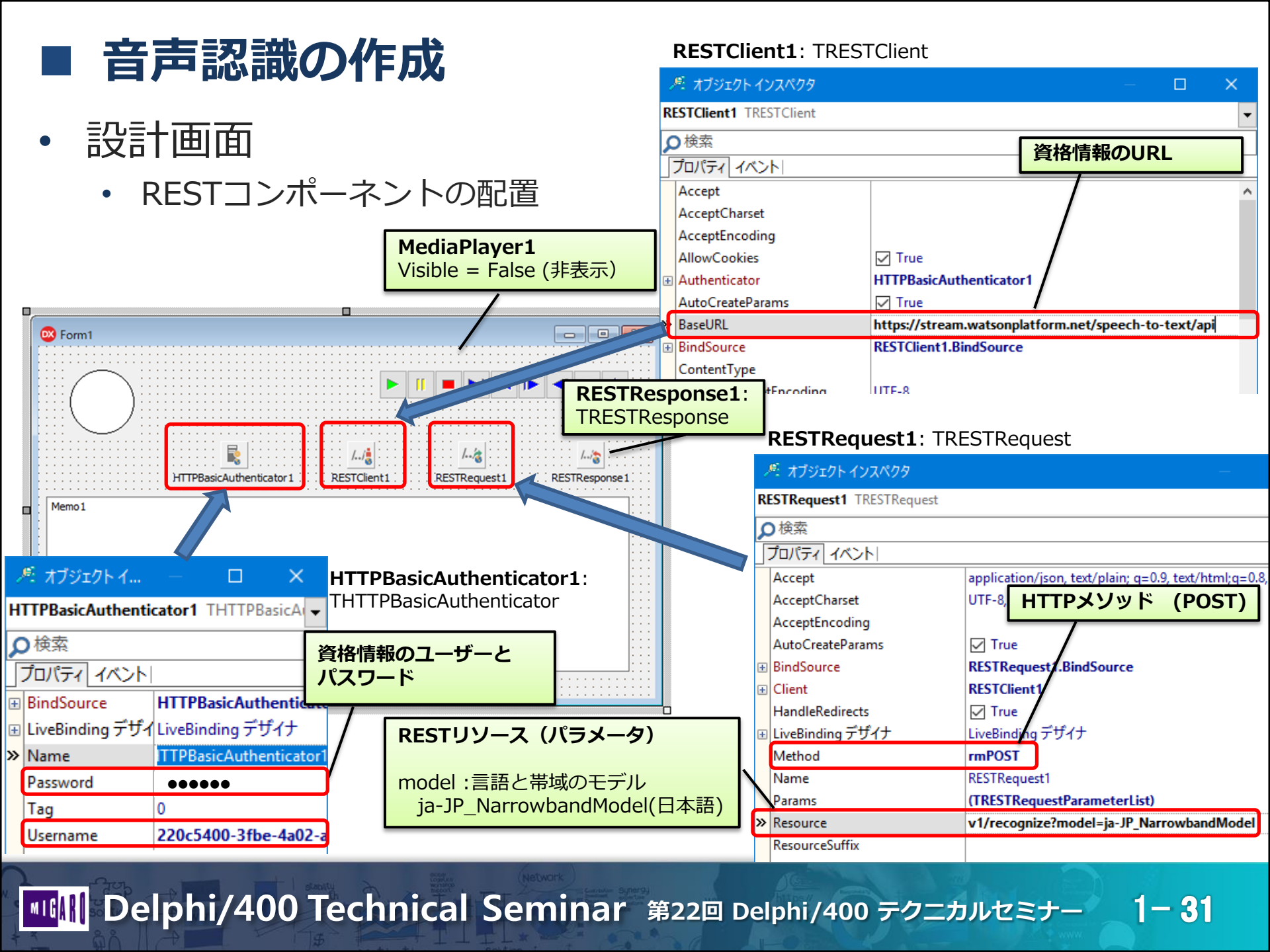

```
procedure TForm1.Shape1MouseUp(Sender: TObject; Butt
  Shift: TShiftState; X, Y: Integer);
var
  sFileName: String;
begin
  sFileName := ChangeFileExt(ParamStr(0), '.wav');

  Shape1.Brush.Color := clWindow;
  Application.ProcessMessages;
  MediaPlayer1.FileName:= sFileName;
  MediaPlayer1.Stop;
  MediaPlayer1.Save;
  MediaPlayer1.Close;
end;
```

ファイル名の指定
[プロジェクト名].wav

録音した内容を
.wavファイル
に保存

■ 音声認識の作成

- 設計画面
  - RESTコンポーネントの配置



■ 音声認識の作成

- 音声ファイルを元に変換を実施

ソース（1/2）

```
procedure TForm1.Shape1MouseUp(Sender: TObject; Button:
  Shift: TShiftState; X, Y: Integer);
var
  memStream: TMemoryStream;
  JSONValue: TJSONValue;
  sFileName: String;
  sStr: String;
begin
  【録音結果のファイル作成部分は割愛】
  memStream := TMemoryStream.Create;
  try
    memStream.LoadFromFile(sFileName);

    RESTRequest1.Params.AddItem('Content-Type', 'audio/wav'
      , pkHTTPHEADER, [poDoNotEncode]);     //フォーマット指定
    RESTRequest1.Params.AddItem('audio', memStream
      , pkREQUESTBODY, [], ctAUDIO_BASIC);  //音声データ

    //リクエストの実行
    RESTRequest1.Execute;
```

メモリ上にストリームデータを保持するオブジェクト変数

音声ファイルの取得

AddItem パラメータをセット

pkHTTPHEADER：パラメータをリクエストの HTTP ヘッダーとして使用

pkREQUESTBODY：パラメータを、リクエストの本体として使用

ストリームデータ

パラメータのコンテンツ タイプ

■ 音声認識の作成

- 音声ファイルを元に変換を実施

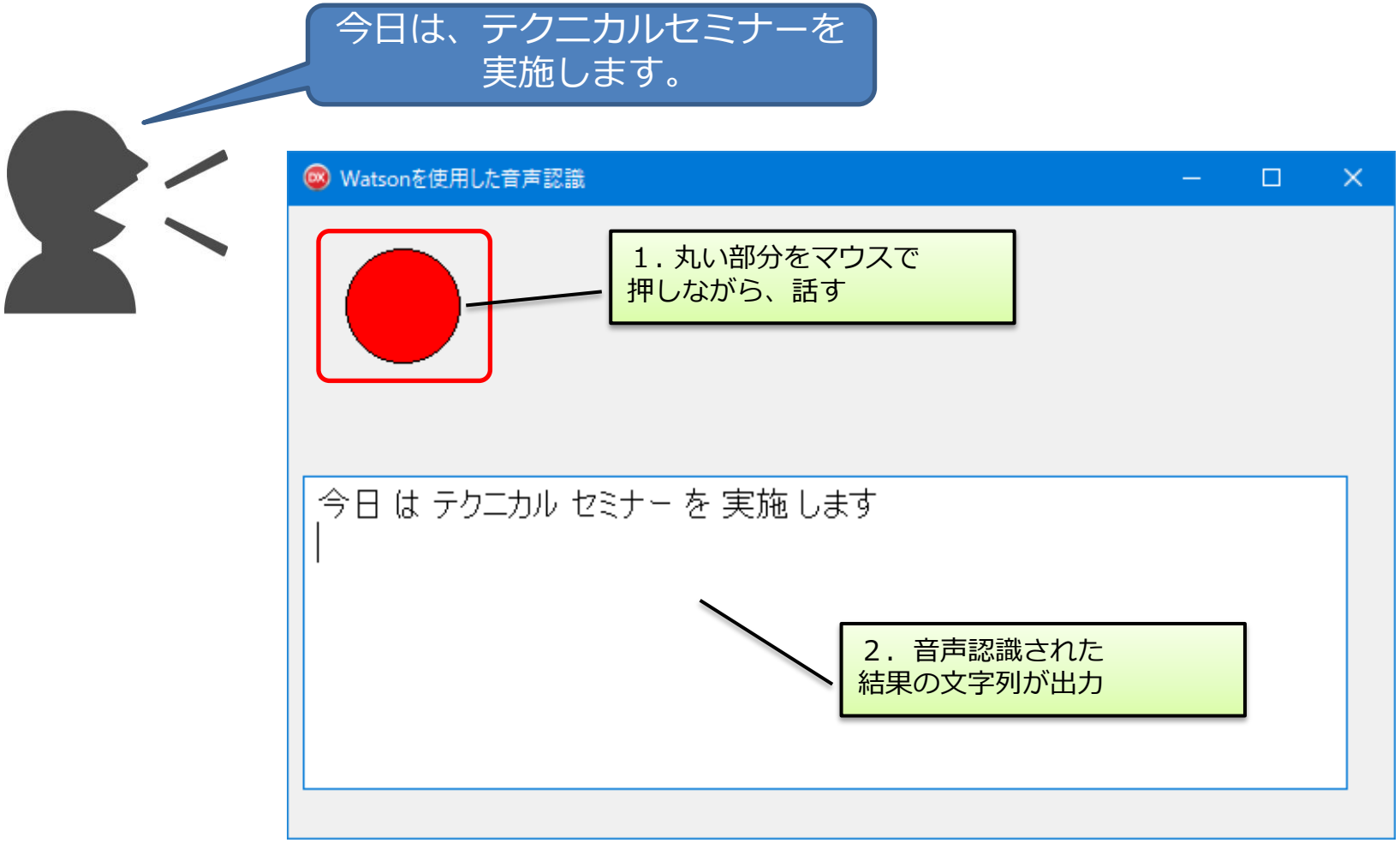
ソース（2/2）

```
    //処理結果JSONのパース
    JSONValue := TJSONObject.ParseJSONValue(RESTResponse1.Content);

    //結果の取得
    try
      sStr := JSONValue.GetValue<String>
        ('results[0].alternatives[0].transcript');
      Memo1.Lines.Add(sStr);
    except
      //対象データが無い場合何もしない
    end;
  finally
    memstream.Free;
  end;
end;
```

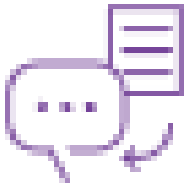
JSONより変換結果の文字列を取得

■ 音声認識の作成

- 実行結果



■ 音声合成の作成



- Text To Speechサービス
  - テキスト文字列を元に音声読み上げを行うサービス
  - 話速・声の高さ・ポーズの位置などを自由に設定可能
  - カスタマイズ機能により独自の単語も登録可能
  - サポート言語：ドイツ語、英語、スペイン語、フランス語、イタリア語、日本語、ブラジル・ポルトガル語

＜活用例＞
- 電話の自動応答システムで、音声案内を実施
- 業務システムにおける文字データの音声読み上げ
- 外国語文章の読み上げによる英語学習

（デモサイト）

https://text-to-speech-demo.ng.bluemix.net/

■ 音声合成の作成

- Text To SpeechサービスのAPI仕様

リクエスト
GET /v1/synthesize

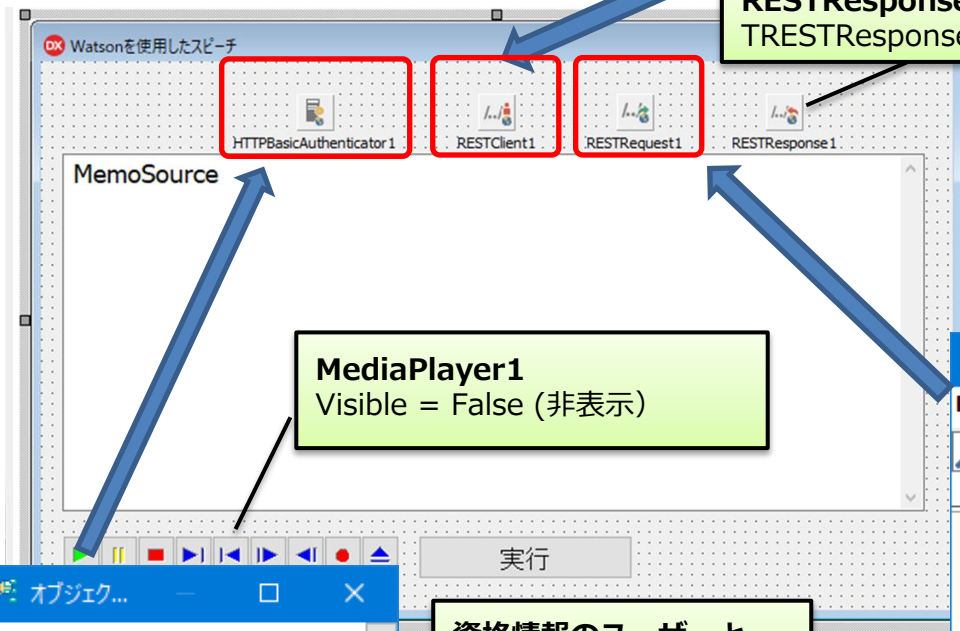
※API仕様の詳細は、APIリファレンスを参照

パラメータ	タイプ	内容
Accept	URL パラメータ	要求する音声データのフォーマット（MIME type)
voice	URL パラメータ	音声合成に使用される声 （日本語：ja-JP_EmiVoice)
text	URL パラメータ	変換対象テキスト

レスポンス
要求したフォーマットの 音声データ

■ 音声合成の作成

- 設計画面
  - RESTコンポーネント

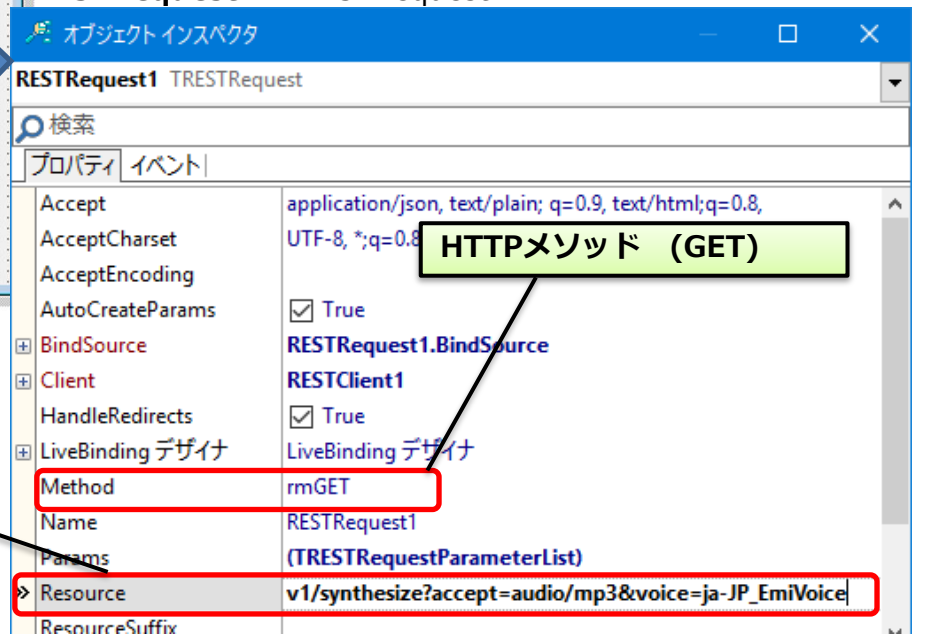


RESTResponse1: TRESTResponse

RESTClient1: TRESTClient

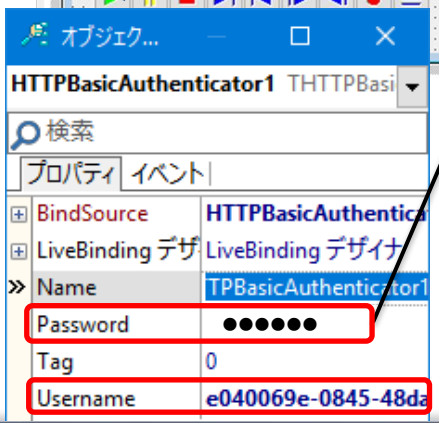
資格情報のURL

RESTRequest1: TRESTRequest



HTTPメソッド　(GET)

MediaPlayer1
Visible = False (非表示)



資格情報のユーザーと
パスワード

RESTリソース（パラメータ）

accept：音声データの形式
voice：音声合成で使用される声
（日本語：ja-JP_EmiVoice）

■ 音声合成の作成

- テキストから音声ファイルを作成

ソース

```
procedure TForm1.ButtonTransrateClick(Sender: TObj
var
  memStream: TMemoryStream;
begin
  //パラメータの指定
  RESTRequest1.Params.AddItem('text', MemoSource.Lines.Text
    , pkGETorPOST);

  //リクエストの実行
  RESTRequest1.Execute;

  //レスポンスのMP3データをファイルに保管する
  memStream := TMemoryStream.Create;
  try
    memStream.WriteData(RESTResponse1.RawBytes
      , Length(RESTResponse1.RawBytes));
    memStream.SaveToFile(FFileName);
  finally
    memStream.Free;
  end;

  //MP3ファイルをメディアプレーヤーで再生
  MediaPlayer1.FileName := FFileName;
  MediaPlayer1.Open;
  MediaPlayer1.Notify := True;
  MediaPlayer1.Play;
end;
```

AddItem パラメータをセット

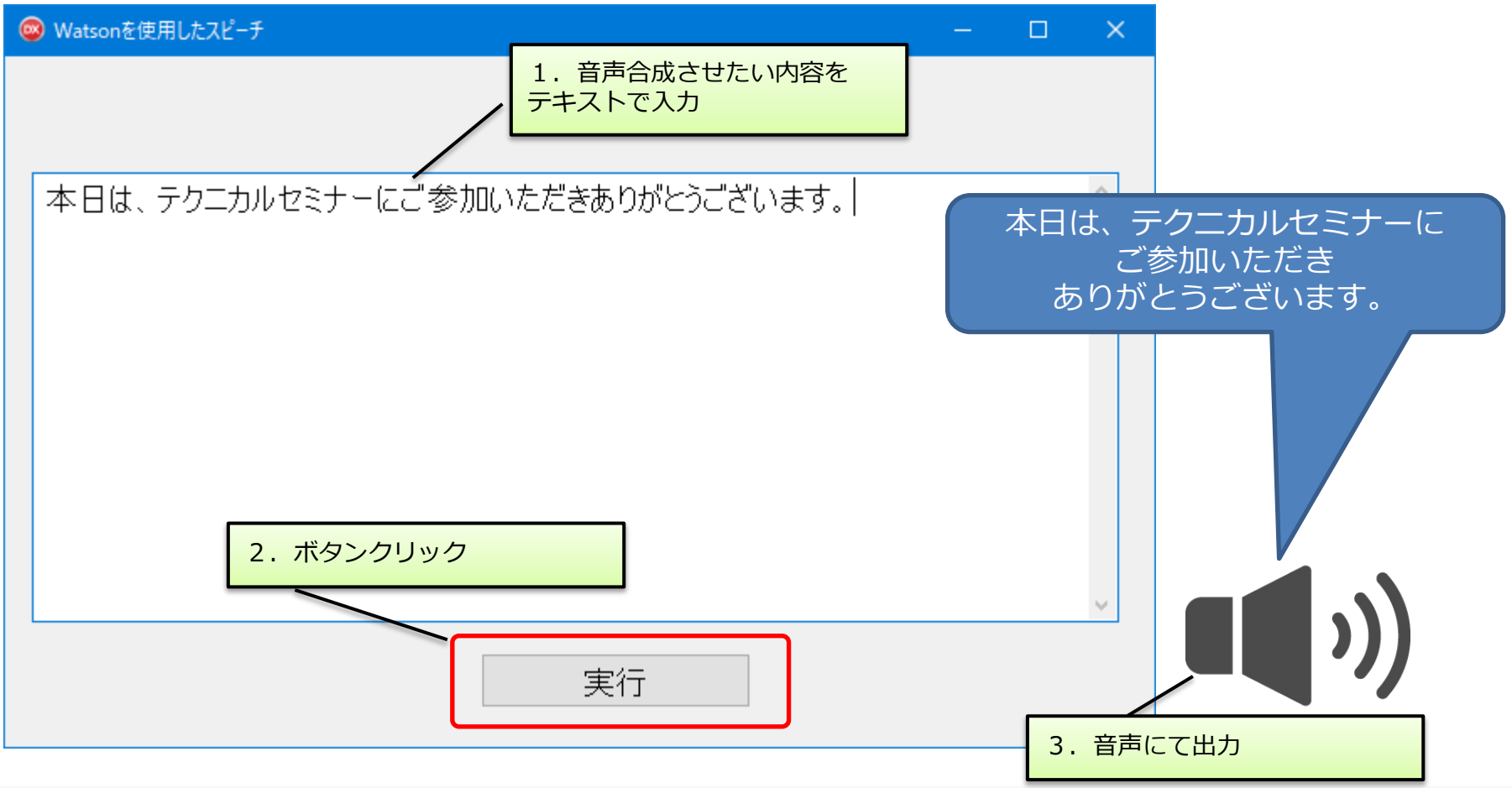
pkGETorPOST：パラメータを、URL パラメータとして、送信。

レスポンスの音声データを
ストリームに書き込み、
ファイルとして保存

MP3音声ファイルを
メディアプレイヤーにて再生

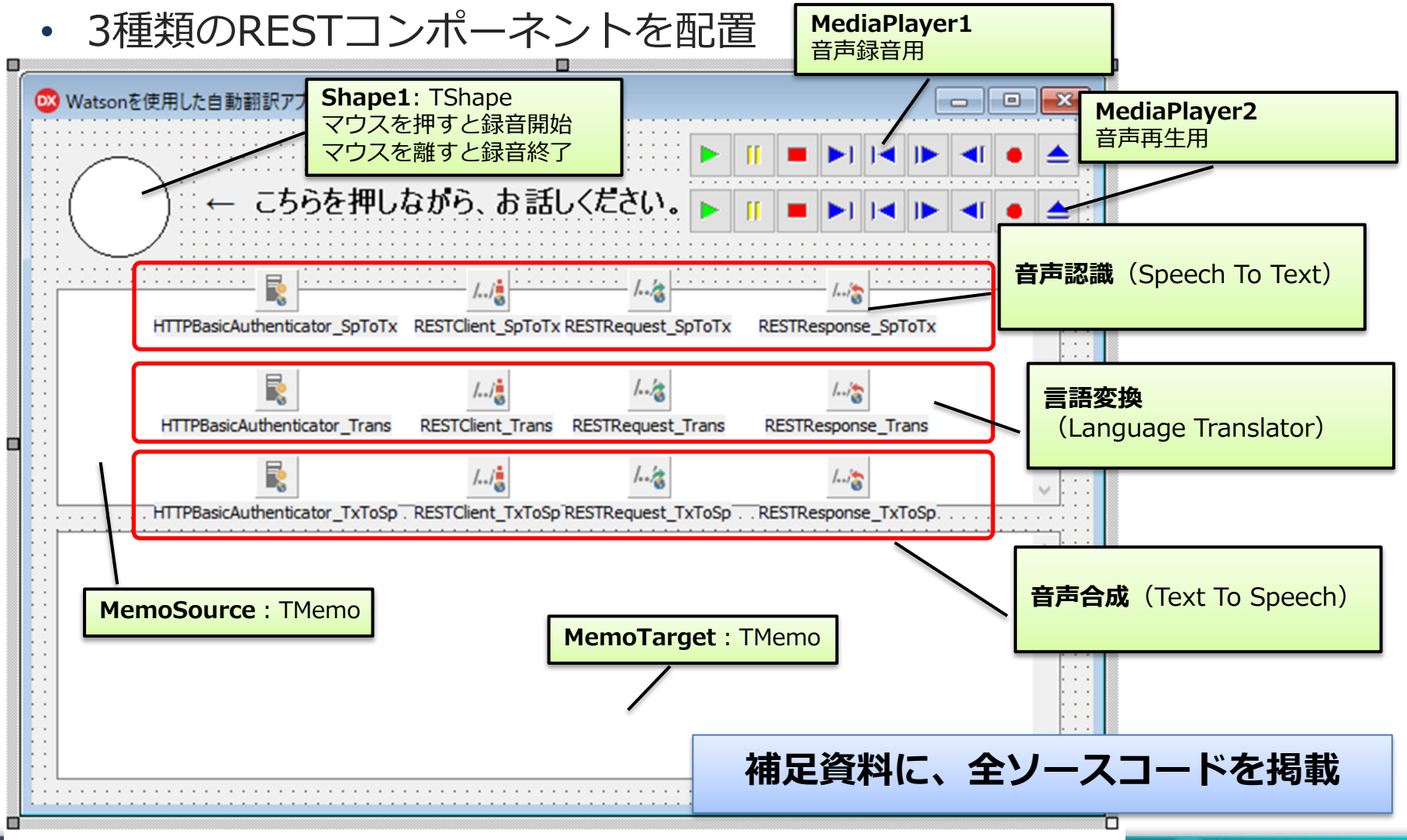
■ 音声合成の作成

- 実行結果



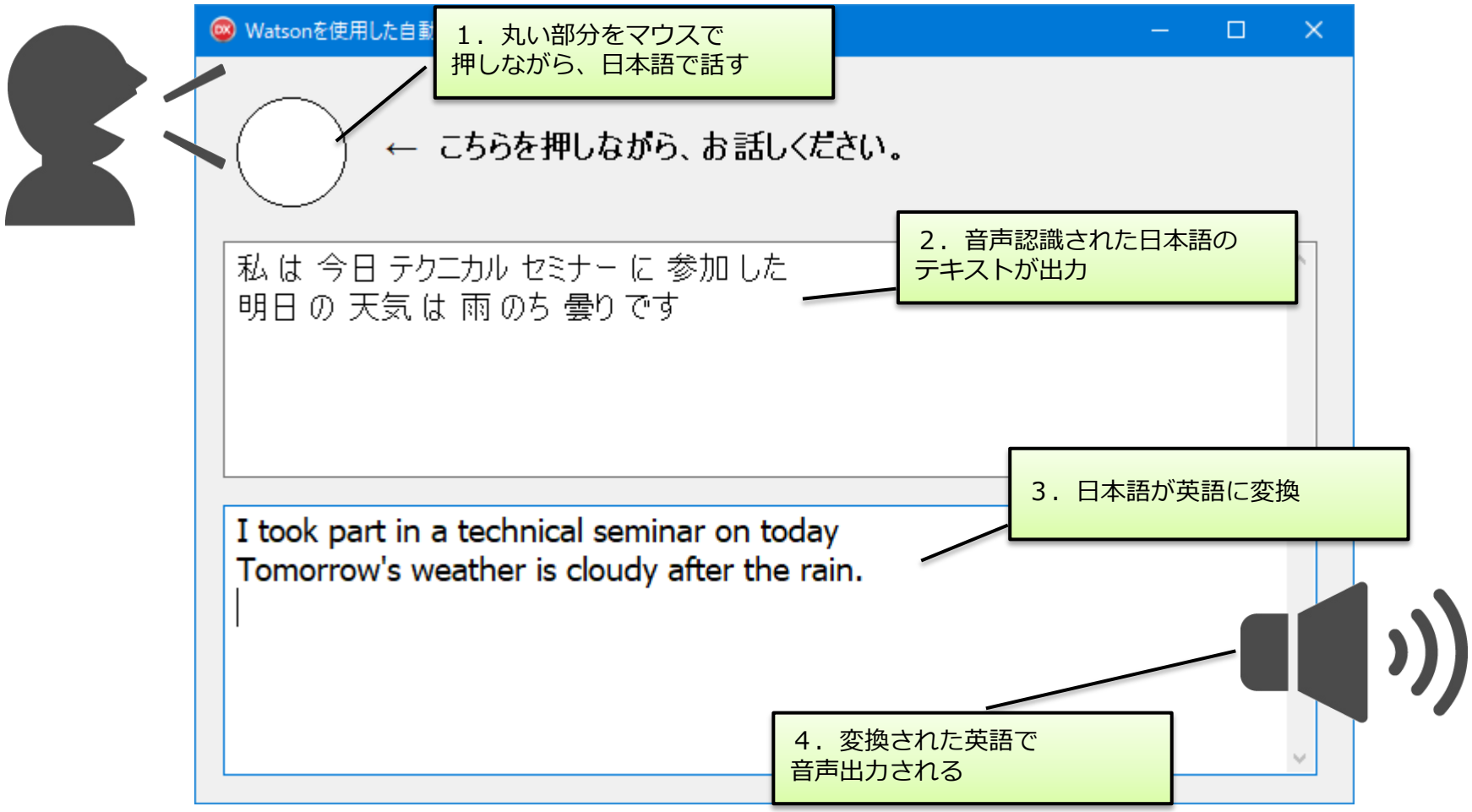
■ 自動翻訳機アプリ

- 設計画面
  - 3種類のRESTコンポーネントを配置



■ 自動翻訳機アプリ

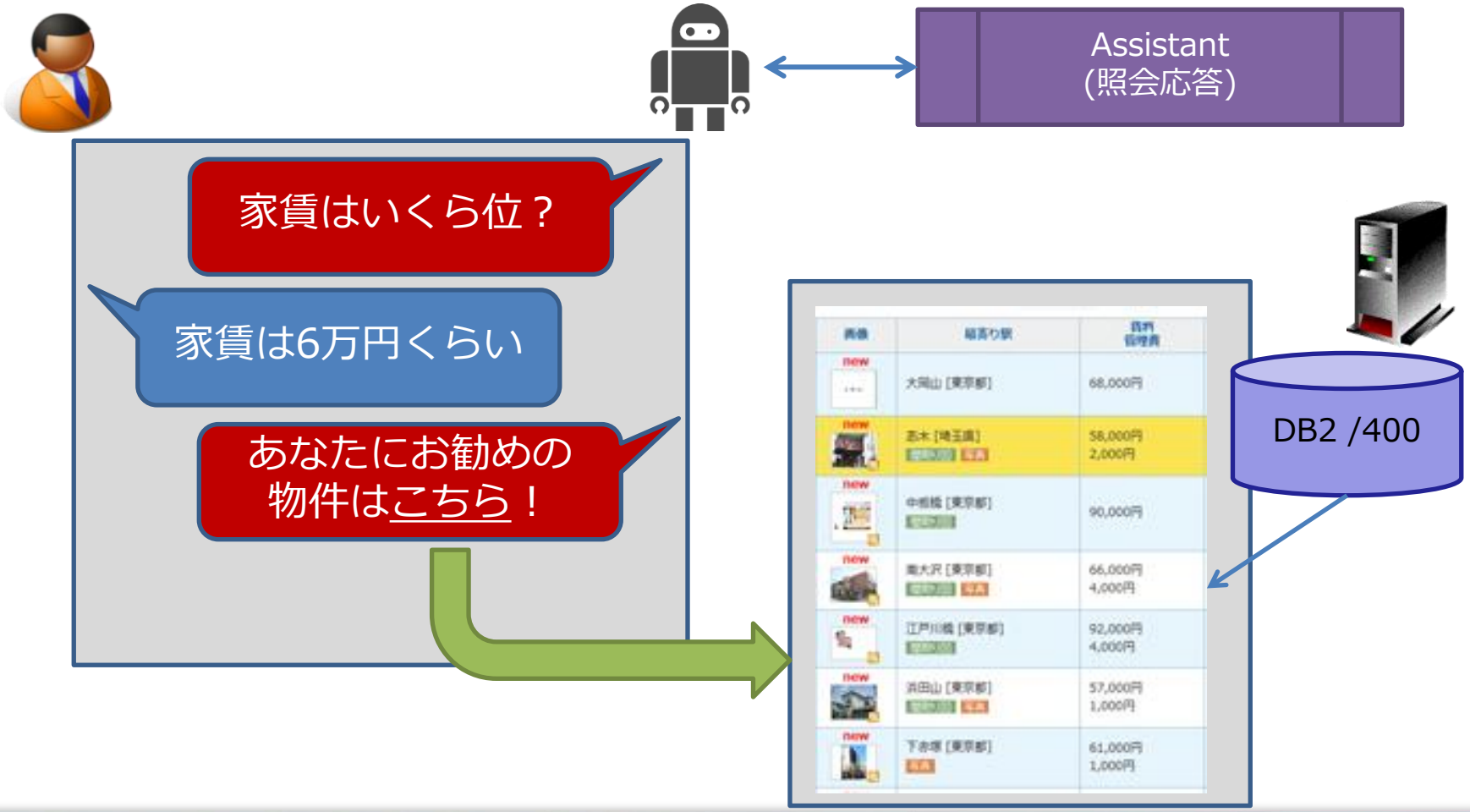
- 実行（デモ）



４．Watson APIを活用したチャットボット

■ 検討するアプリ

- チャットボットによる　物件紹介
  - 質問に対し、自然言語で回答していくことで、お勧め物件を紹介する。



■ チャットボットについて

- チャットボットの採用例　（Watson事例より）
  - 日本航空
    - ハワイに関する現地情報を紹介
  - ケイ・オプティコム
    - お問合せ用アシスタントとして使用

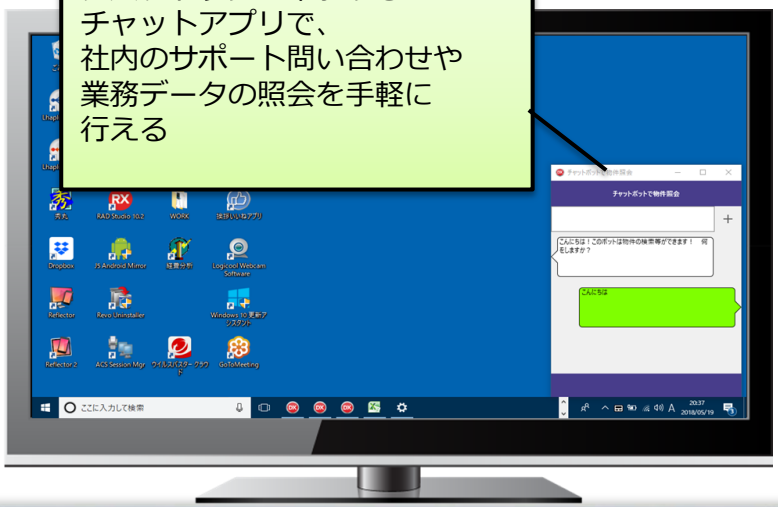
インターフェースとしては、Webやモバイルでの使用が多い

■ チャットボットについて

- チャットボットのメリット
  - システム操作に詳しくない人でも会話形式で目的が達成できる。
  - 対話内容がログとして残るため、履歴管理が可能。
  - 一つのインターフェースで、色々な作業ができる。
    （商品検索する、在庫を確認する、納期確認する、発注する　といった操作が、全て一つのチャット上で行える）

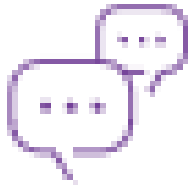
業務システムでも、十分活用可能！

デスクトップに常駐する
チャットアプリで、
社内のサポート問い合わせや
業務データの照会を手軽に
行える



以前は、出先で調べ物がある
場合、電話連絡していたが、
現在は、
チャットボットにより簡単に
問合せが行える

■ 照会応答サービス



- Assistantサービス
  - ユーザーとコンピューターが自然言語で対話可能なアプリケーションを簡単に開発するためのサービス
  - 開発者はシンプルで洗練された開発ツールを用いることで、対話の流れを直感的な操作で作ることが可能。

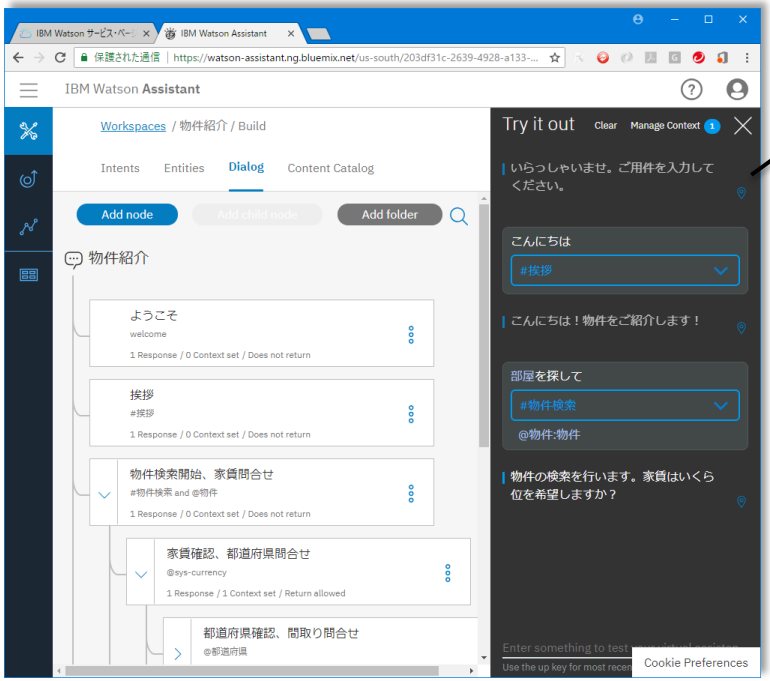
＜活用例＞
- オペレータが対応していたお客様や社内ユーザーからの問合せや手続きをチャットボットが代行
- ロボット等の機械と人間の音声対話の実現

（デモサイト）

https://conversation-demo.ng.bluemix.net/　（英語）

■ 照会応答サービス

- Assistantサービスの概要



Watson Assistantサービス内にある
開発ツールを使用して、会話フローを作成

登録内容がWatsonによって
学習される

Watson Assistant

こんにちは。

画面から取得した
入力文字列をAPIでPOST

【入力】“こんにちは。”

「挨拶」が入力された場合、
フローにある「挨拶」の応答を返す。

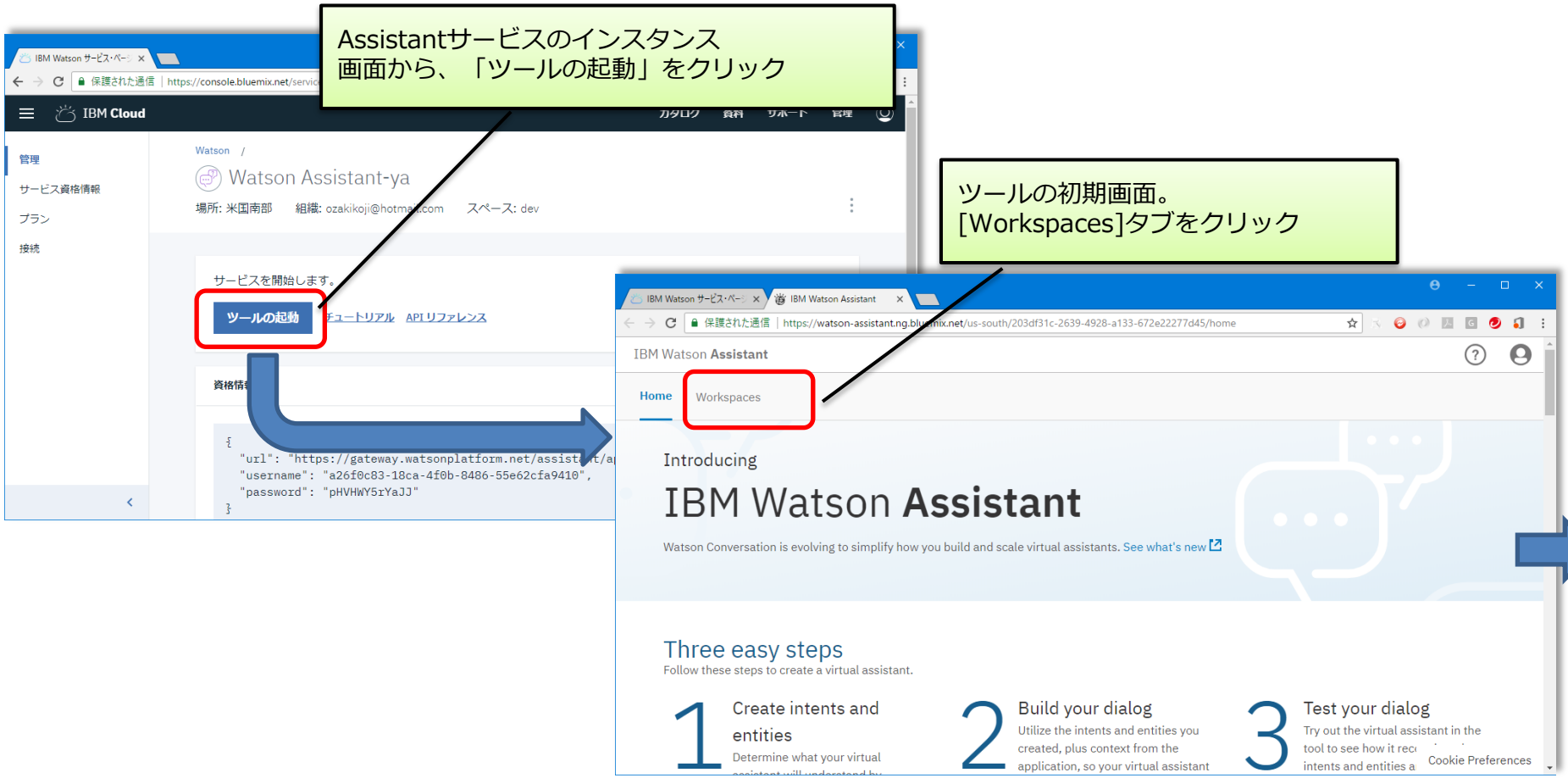
【出力】“こんにちは。物件を
ご紹介します！“

APIのレスポンスで取得した
応答文字列を画面へ渡す

こんにちは。物件
をご紹介します！

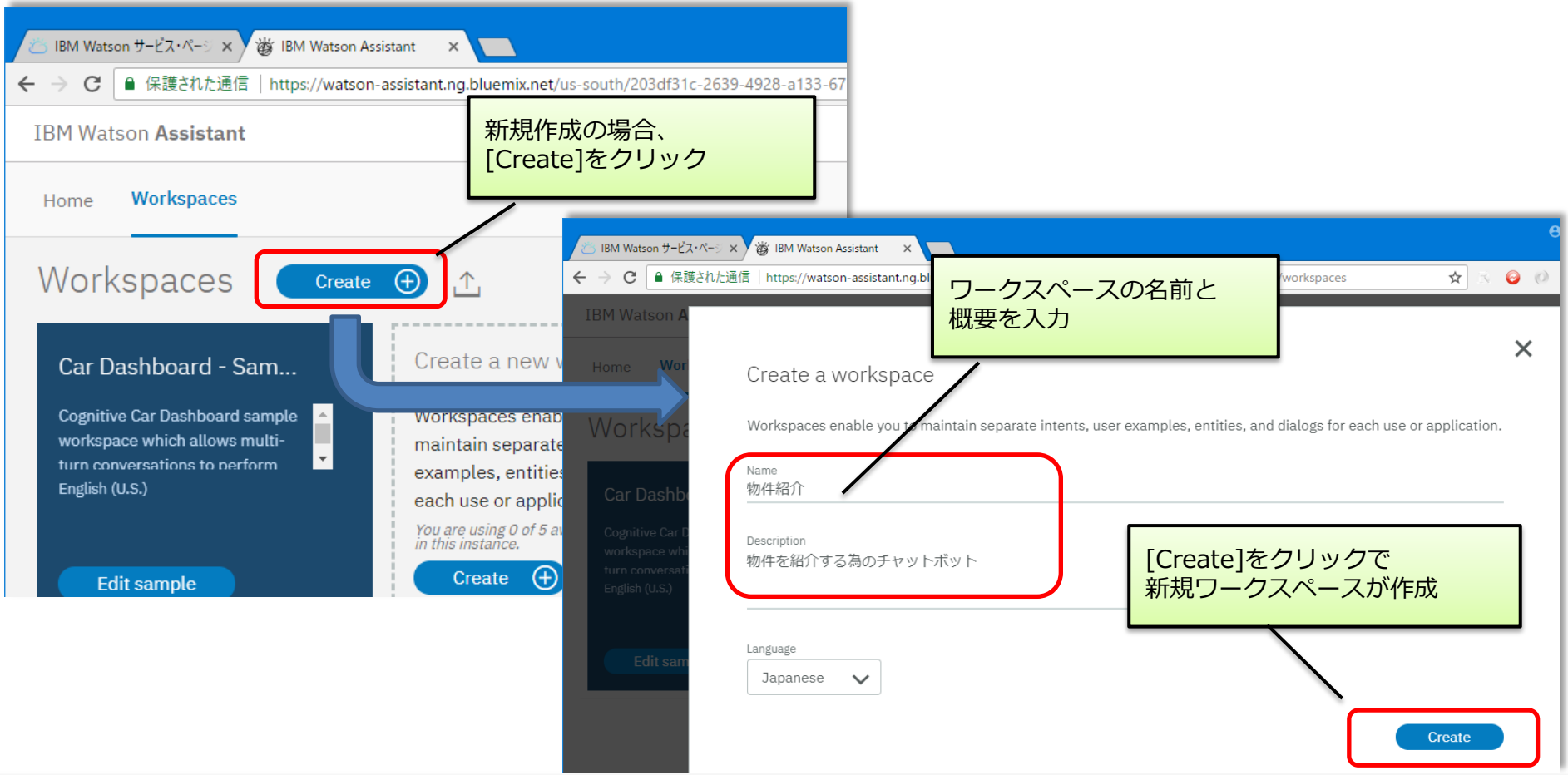
■ 照会応答サービス

- 会話フローの作成
  - 開発ツール起動方法



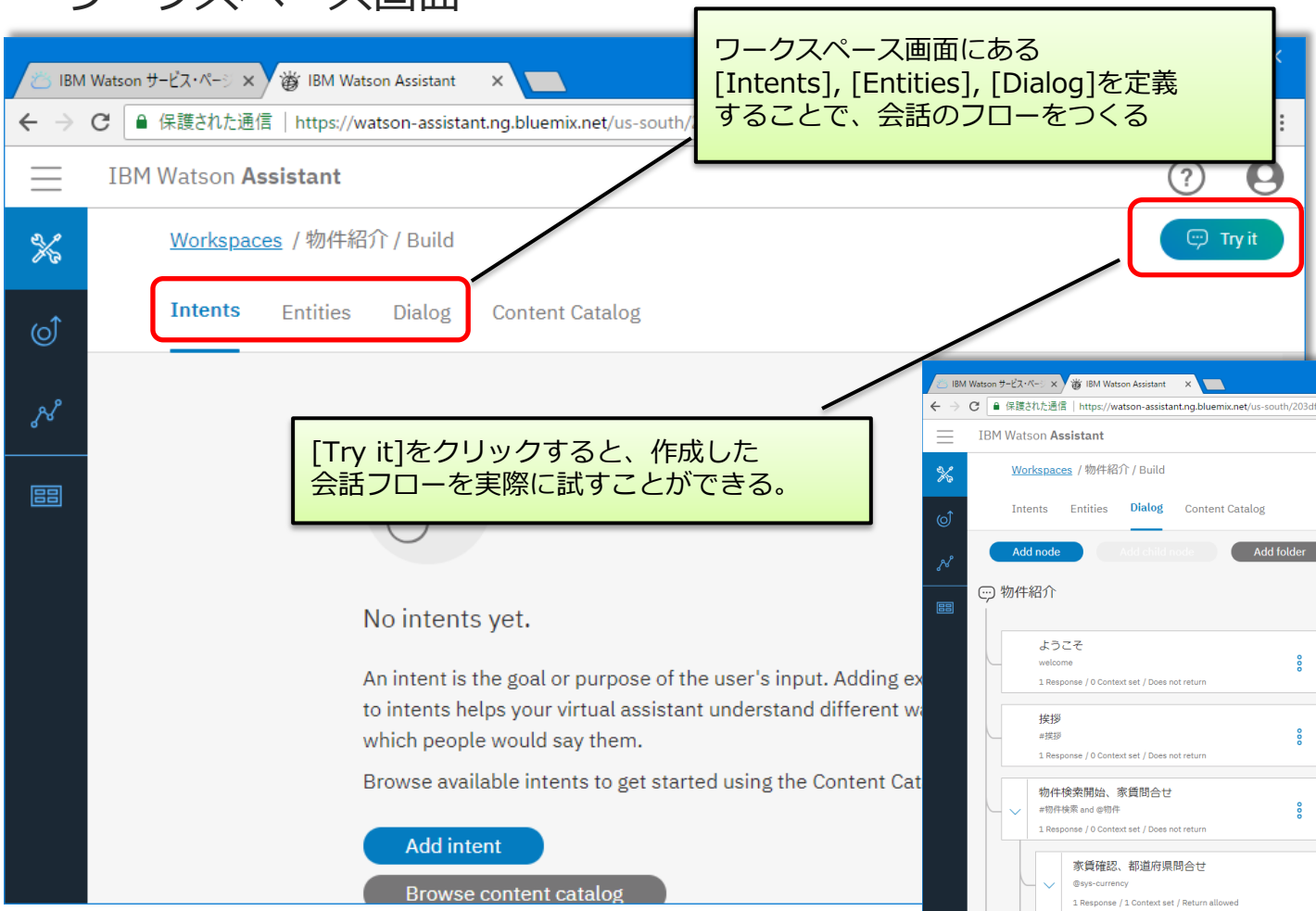
■ 照会応答サービス

- 会話フローの作成
  - ワークスペースの新規作成



■ 照会応答サービス

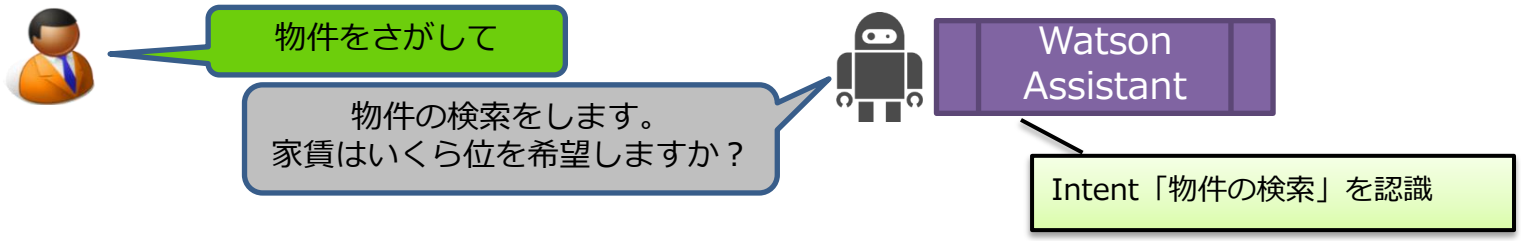
- 会話フローの作成
  - ワークスペース画面



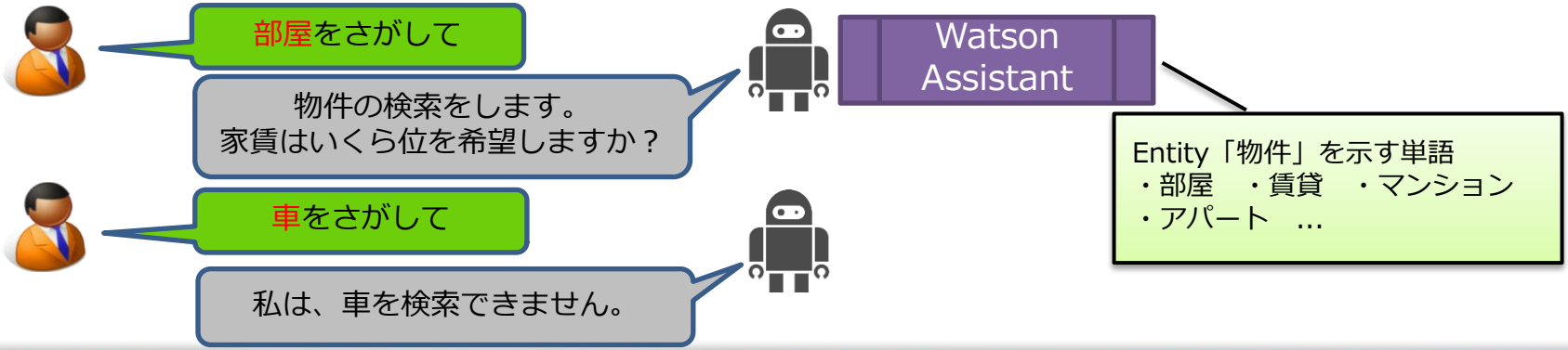
Try it Out 画面例

■ 照会応答サービス

- 会話フローに必要な4つの概念
  - Intent（発言の意図）
    - ユーザーが発言した入力テキストに含まれる「意図」。入力内容をAssistantサービスが分類し、発言でユーザーが求めている意図を理解。

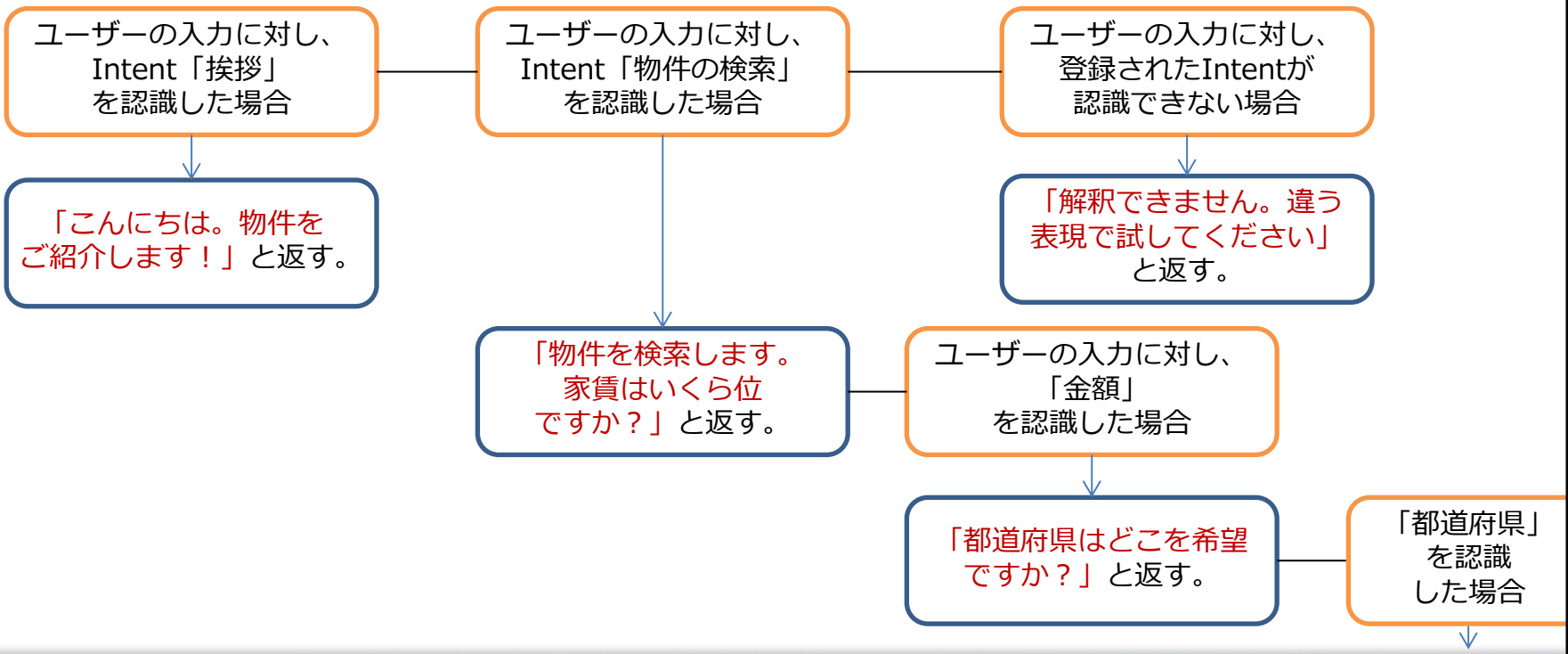


  - Entity（意味づけされた単語）
    - ユーザーが発言した入力テキストから抽出された意味付けされたキーワード。入力テキストから抽出されたEntityは、ユーザーの「目的」の対象物を示す。



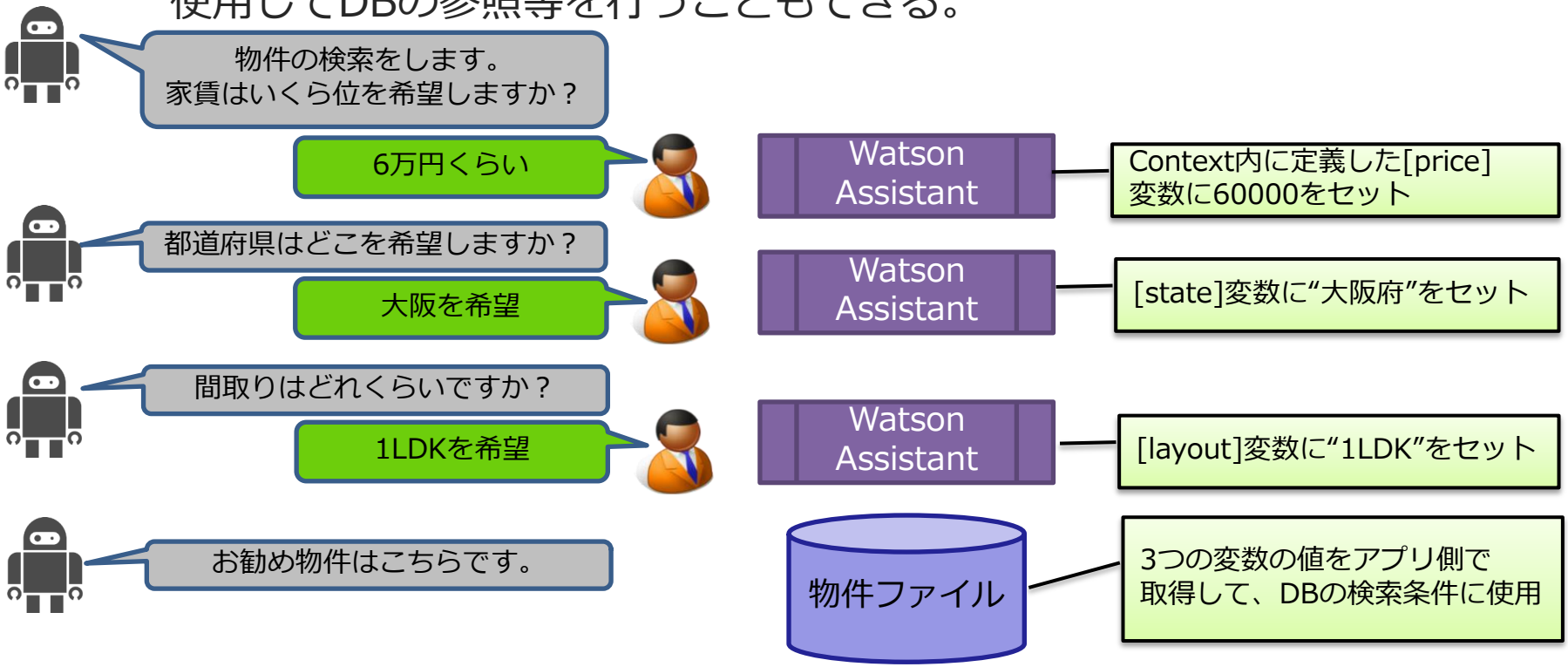
■ 照会応答サービス

- 会話フローに必要な4つの概念
  - Dialog（対話の流れ）
    - ユーザーの発言に対する対話の流れを制御するロジックをDialog（ノード）の組み合わせで実装。ユーザーの要求（IntentとEntity）に応じて一つの返事を返すだけのシンプルなノードだけでなく、ユーザーが求めるものを掘り下げていく木構造となったノードも作成可能。



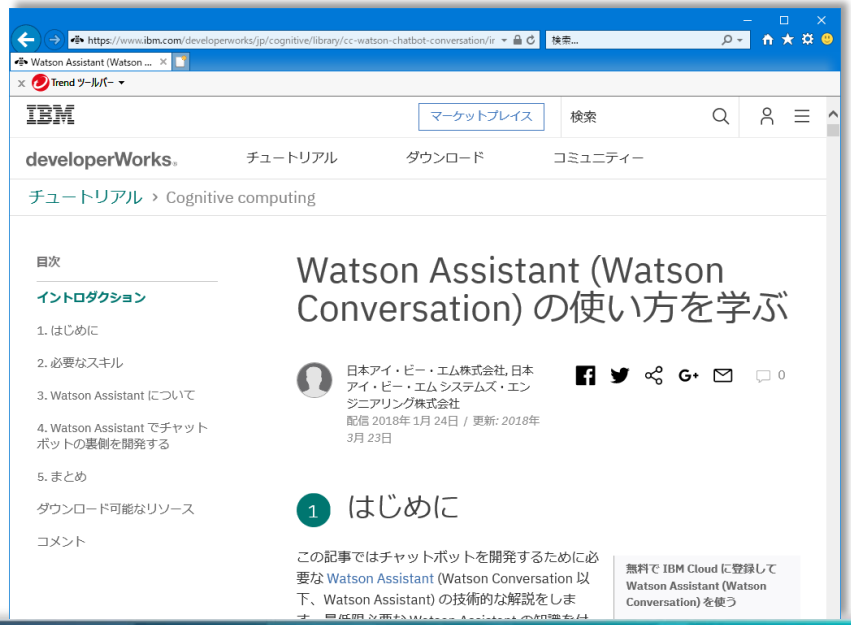
■ 照会応答サービス

- 会話フローに必要な4つの概念
  - Context（会話の背景）
    - ユーザーとWatson Assistantが対話を行うにあたって情報を保持できる領域。対話の回数の保持や、必要情報が全て入力されたかのチェック等に使用可能。Context上に任意の変数も設定でき、アプリ側でその値を使用してDBの参照等を行うこともできる。



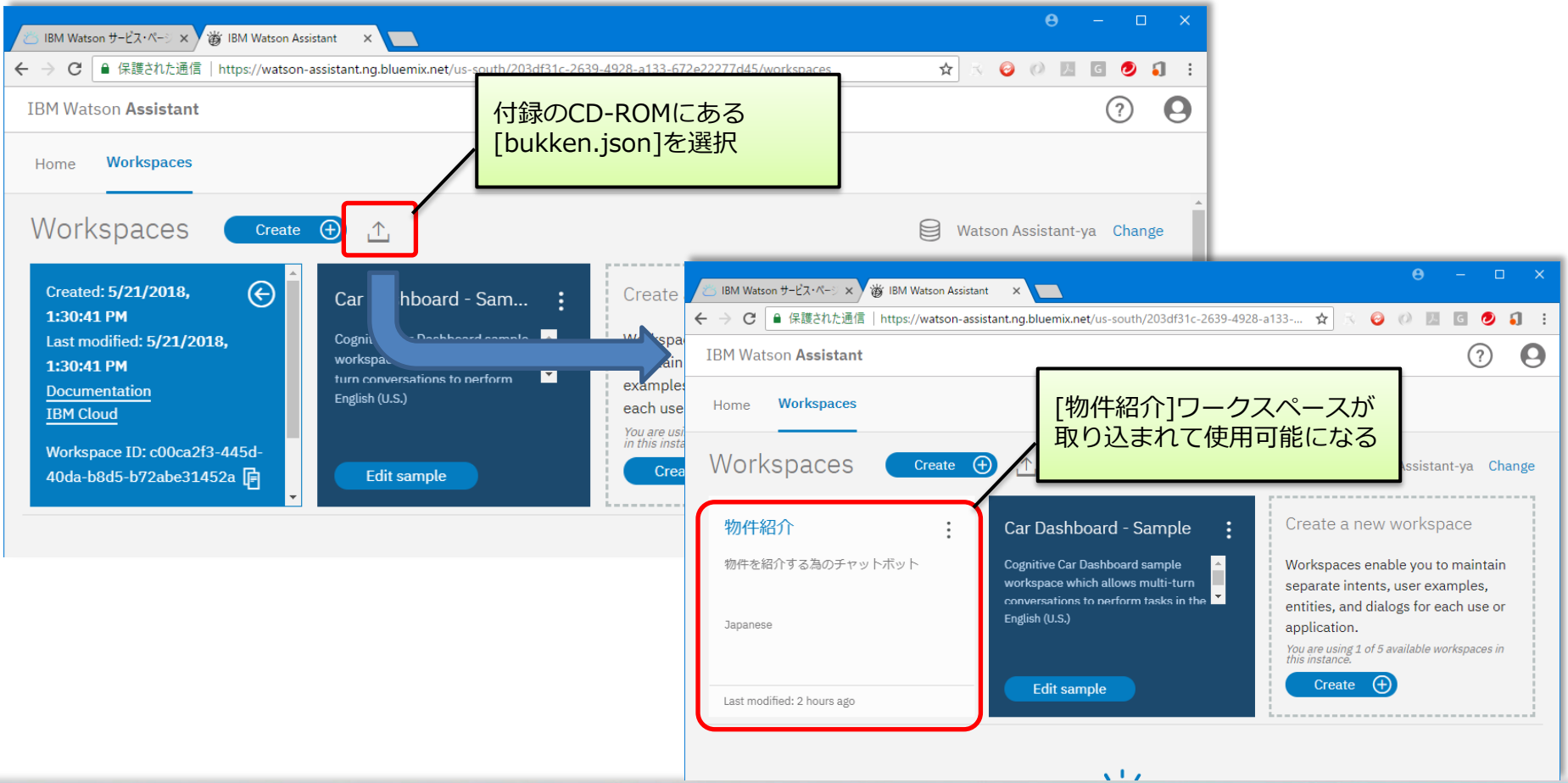
■ 照会応答サービス

- Watson Assistant についての詳細な情報
  - チャットボット開発を手早く簡単に
    https://www.ibm.com/watson/jp-ja/how-to-build-a-chatbot/
  - Watson Assistantの使い方を学ぶ
    https://www.ibm.com/developerworks/jp/cognitive/library/cc-watson-chatbot-conversation/index.html



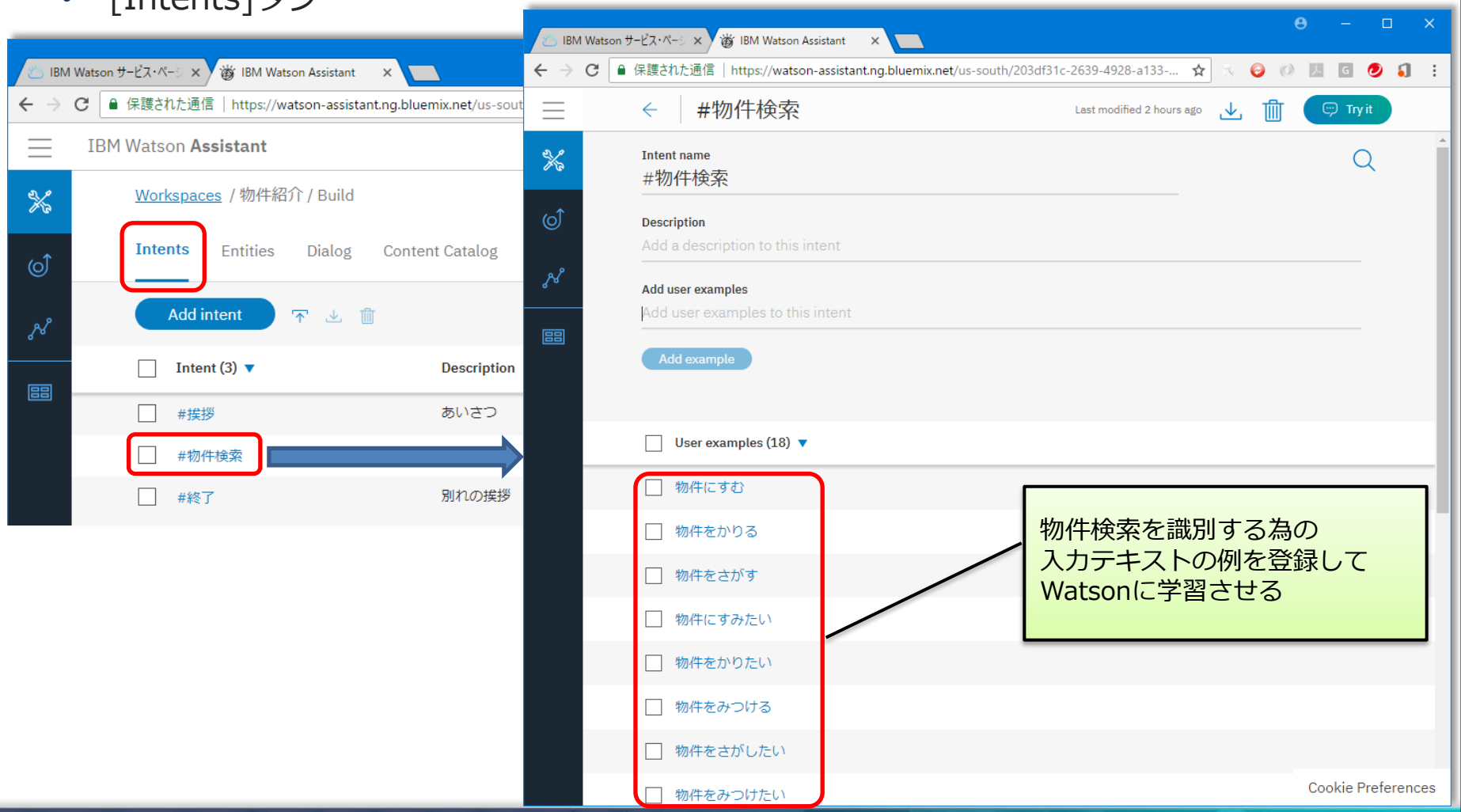
■ チャットボット機能の作成

- 物件紹介ワークスペース環境の準備
  - 付録CD-ROMに収録された[物件紹介]ワークスペースを取り込む



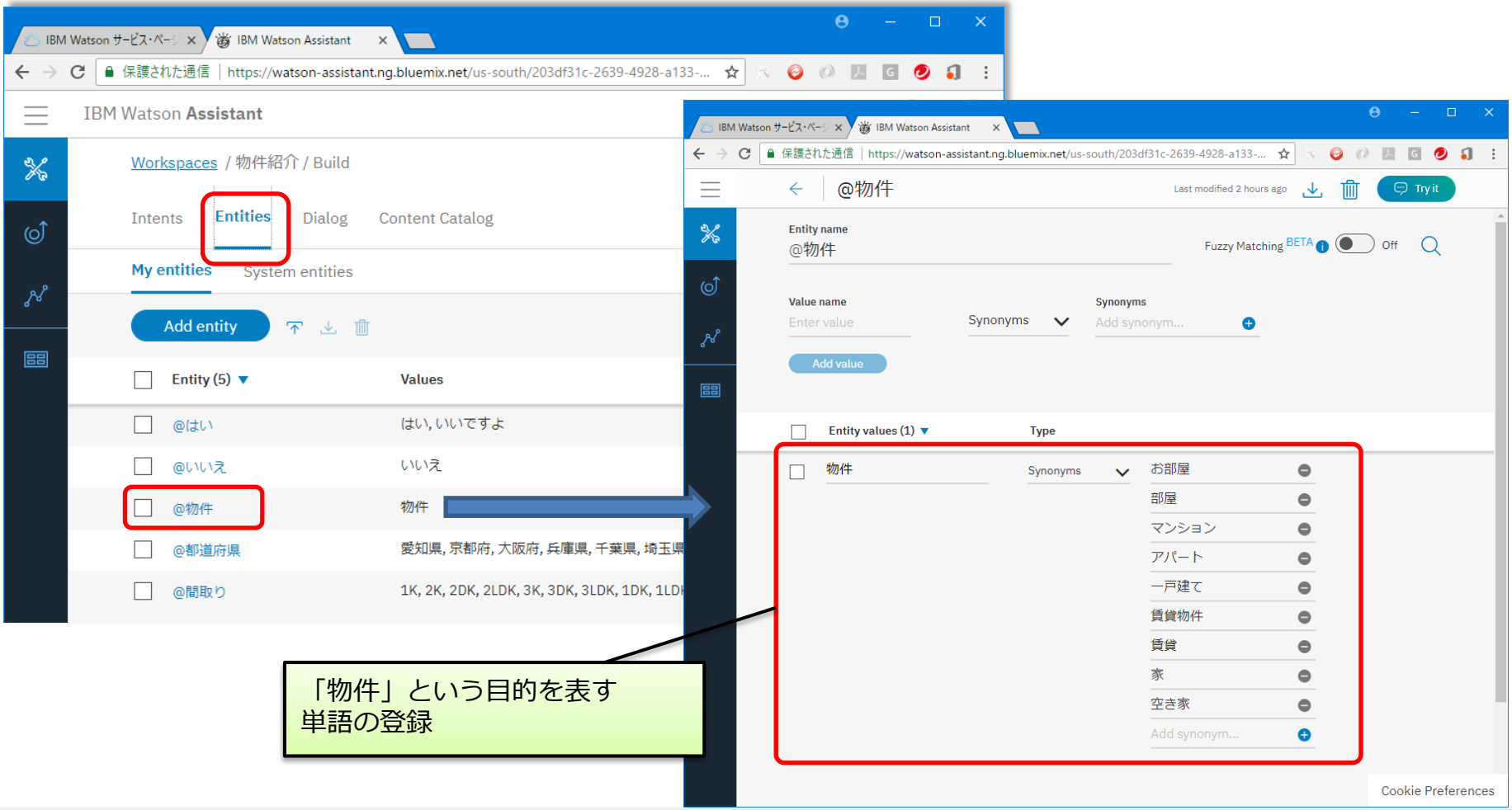
■ チャットボット機能の作成

- 物件紹介ワークスペース
  - [Intents]タブ



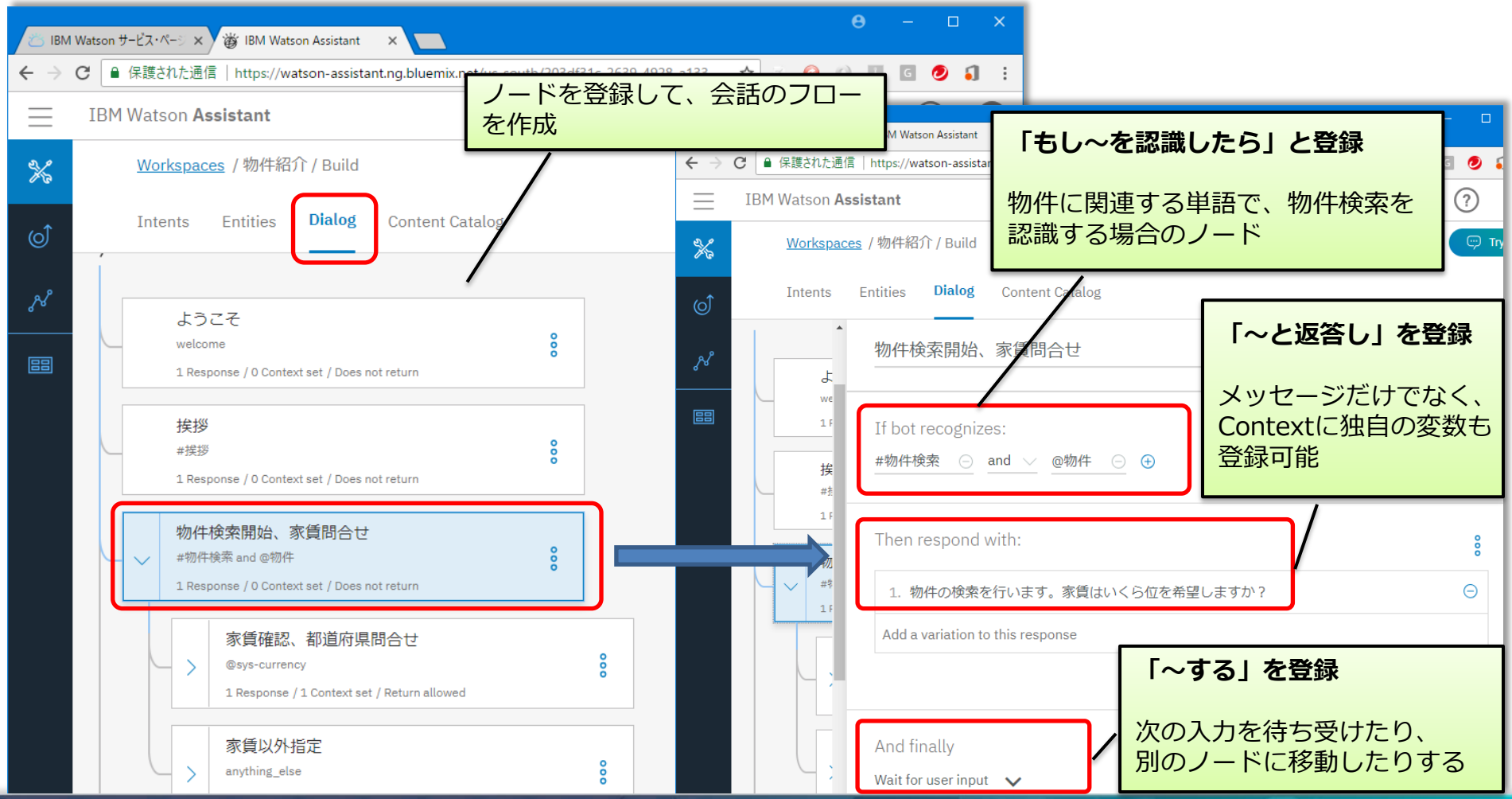
■ チャットボット機能の作成

- 物件紹介ワークスペース
  - [Entities]タブ



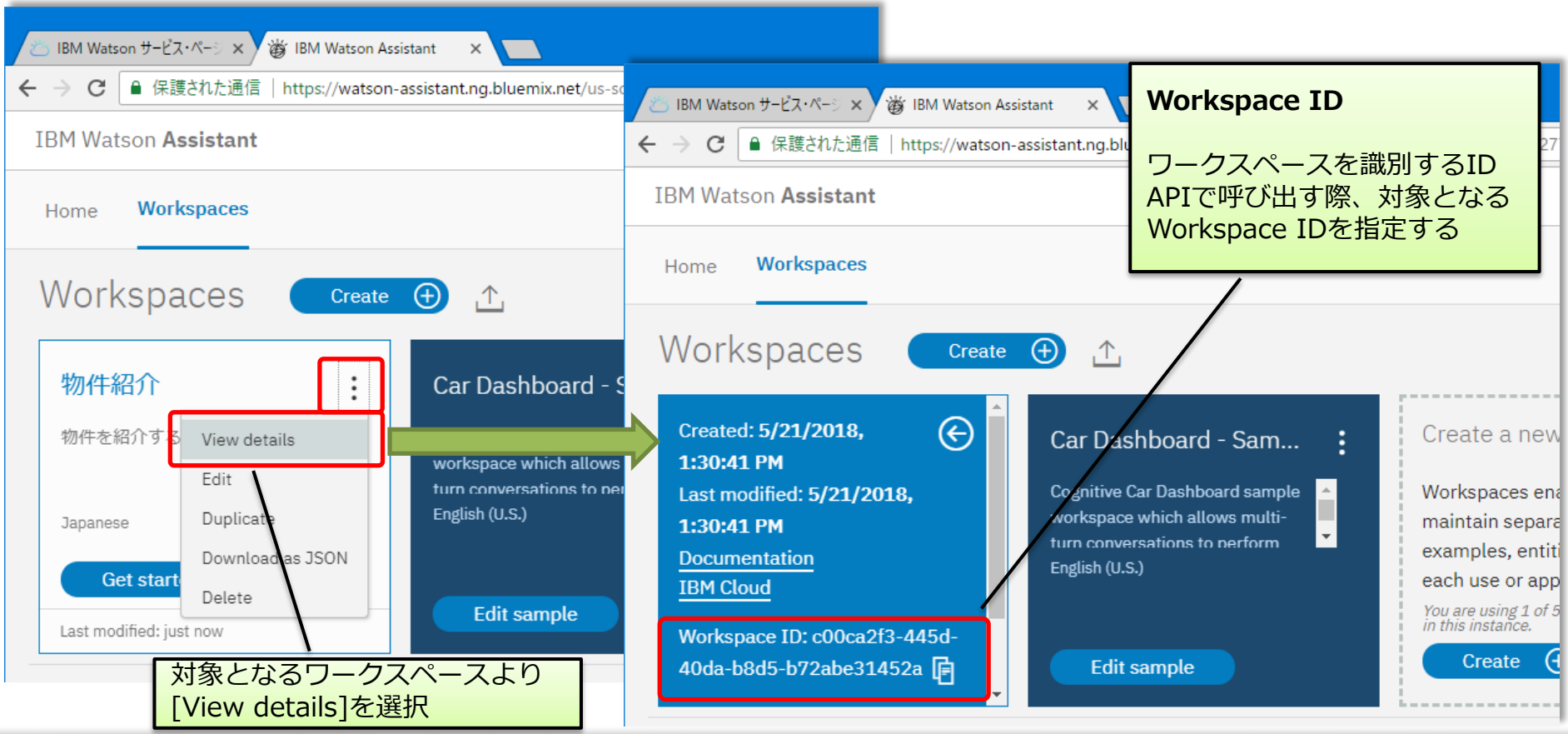
■ チャットボット機能の作成

- 物件紹介ワークスペース
  - [Dialog]タブ



■ チャットボット機能の作成

- ワークスペースIDの取得
  - プログラムからAPIを使用する場合、ワークスペースを識別する為にWorkspace IDを使用する



■ チャットボット機能の作成

- AssistantサービスのAPI仕様

リクエスト
POST /v1/workspaces/{**workspace_id**}/message

※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
workspace_id	URL パラメータ	使用するWorkspace ID
version	URL パラメータ	APIのバージョン日付　（現在は、2018-02-16を指定）
body	リクエストの本体	送信するメッセージおよび会話の状態（Context）をセット　（JSON形式）

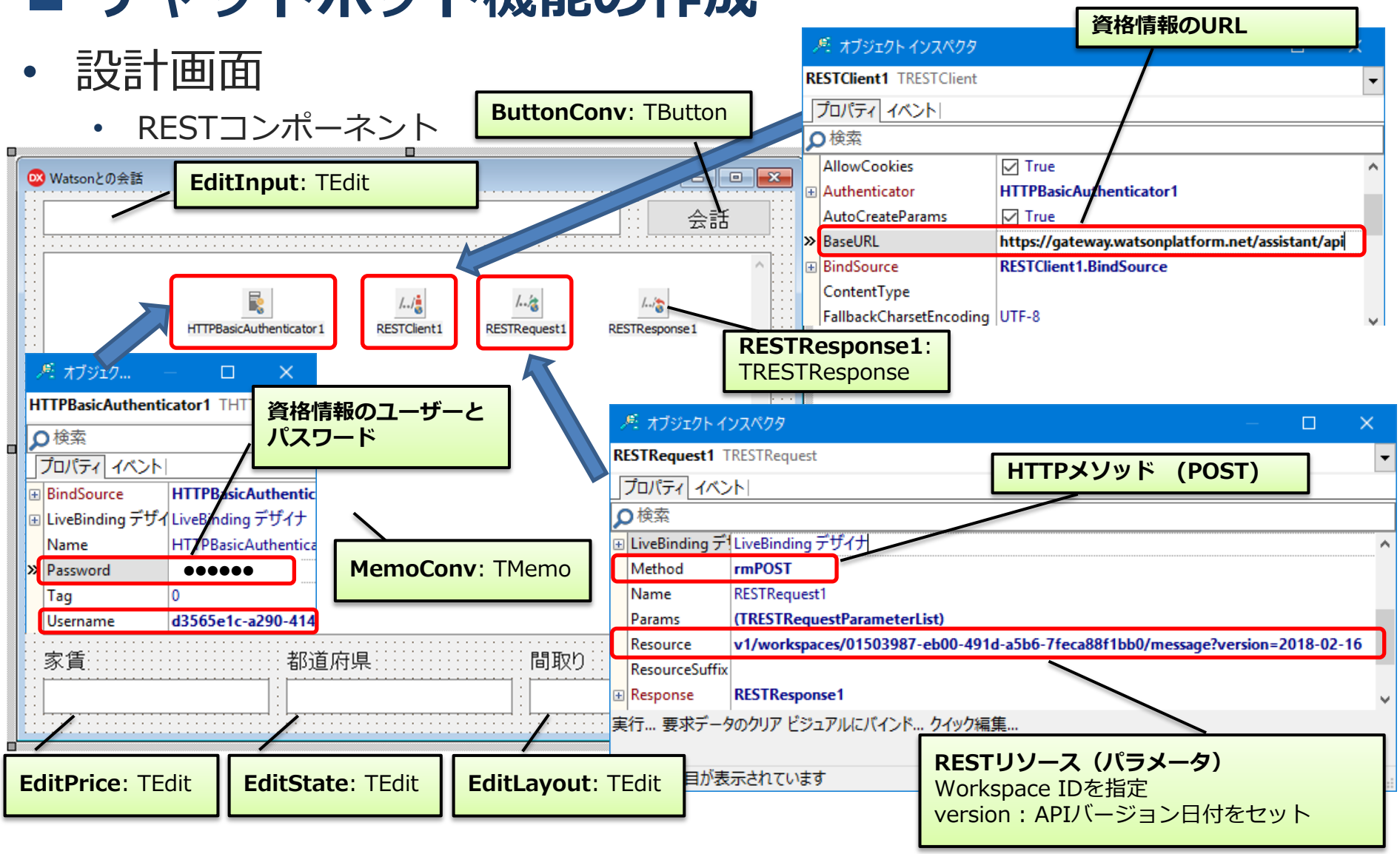
レスポンス例

```
{
  "context": { ... },
  "entities": [],
  "intents": [
    { ... }],
  "output": {
    "log_messages": [],
    "text": ["おはようございます！"],
    "nodes_visited": ["node_3_1480025899490"]},
  "input": {"text": "おはようございます"}
}
```

パラメータ	内容
output:text	会話の応答
input:text	送信した会話
context	会話の状態

■ チャットボット機能の作成

- 設計画面
  - RESTコンポーネント



■ チャットボット機能の作成

- ソース(1/2)

```
procedure TForm1.ButtonConvClick(Sender: TObject);
var
  JSONValue: TJSONValue;
  JSONPair: TJSONPair;
  sMsg: String;
  sRet: String;
begin

  MemoConv.Lines.Add('【あなた】 : ' +EditInput.Text);

  RESTRequest1.Params.Clear;
  //リクエストパラメータのセット
  if FContext = '' then  //----- 会話が初回の場合
    sMsg := '{"input": {"text": "' + EditInput.Text + '"}}'
  else                   //----- 会話が2回目以降の場合
    sMsg := '{"input": {"text": "' + EditInput.Text + '"}, '
      + FContext + '}';  //会話の状態(FContext)をセット

  RESTRequest1.Params.AddItem('body', sMsg, pkGETorPOST
    , [], ctAPPLICATION_JSON);

  //リクエストの実行
  RESTRequest1.Execute;

  //処理結果JSONのパース
  JSONValue := TJSONObject.ParseJSONValue(RESTResponse1.Content);
```

会話情報のJSONテキスト作成

Private変数 FContext
：コンテキスト（会話の状態）

会話を継続する為に、2回目以降
前回のコンテキスト値を再セット

AddItem パラメータをセット

会話のメッセージおよび状態をセット
コンテンツタイプには下記を指定
ctAPPLICATION_JSON

■ チャットボット機能の作成

- ソース(2/2)

```
  //結果（output->text）の取得
  sRet := JSONValue.GetValue<String>('output.text[0]');
  MemoConv.Lines.Add('【Watson】：' + sRet);

  //会話の状態（context）を取得
  JSONPair := TJSONObject(JSONValue).Get('context');
  FContext := JSONPair.ToJSON;

  //選択値の取得
  try
    EditPrice.Text := JSONValue.GetValue<String>('context.price');
  except
    EditPrice.Text := '';
  end;

  try
    EditState.Text := JSONValue.GetValue<String>('context.state');
  except
    EditState.Text := '';
  end;

  try
    EditLayout.Text := JSONValue.GetValue<String>('context.layout');
  except
    EditLayout.Text := '';
  end;
end;
```

会話の応答を取得

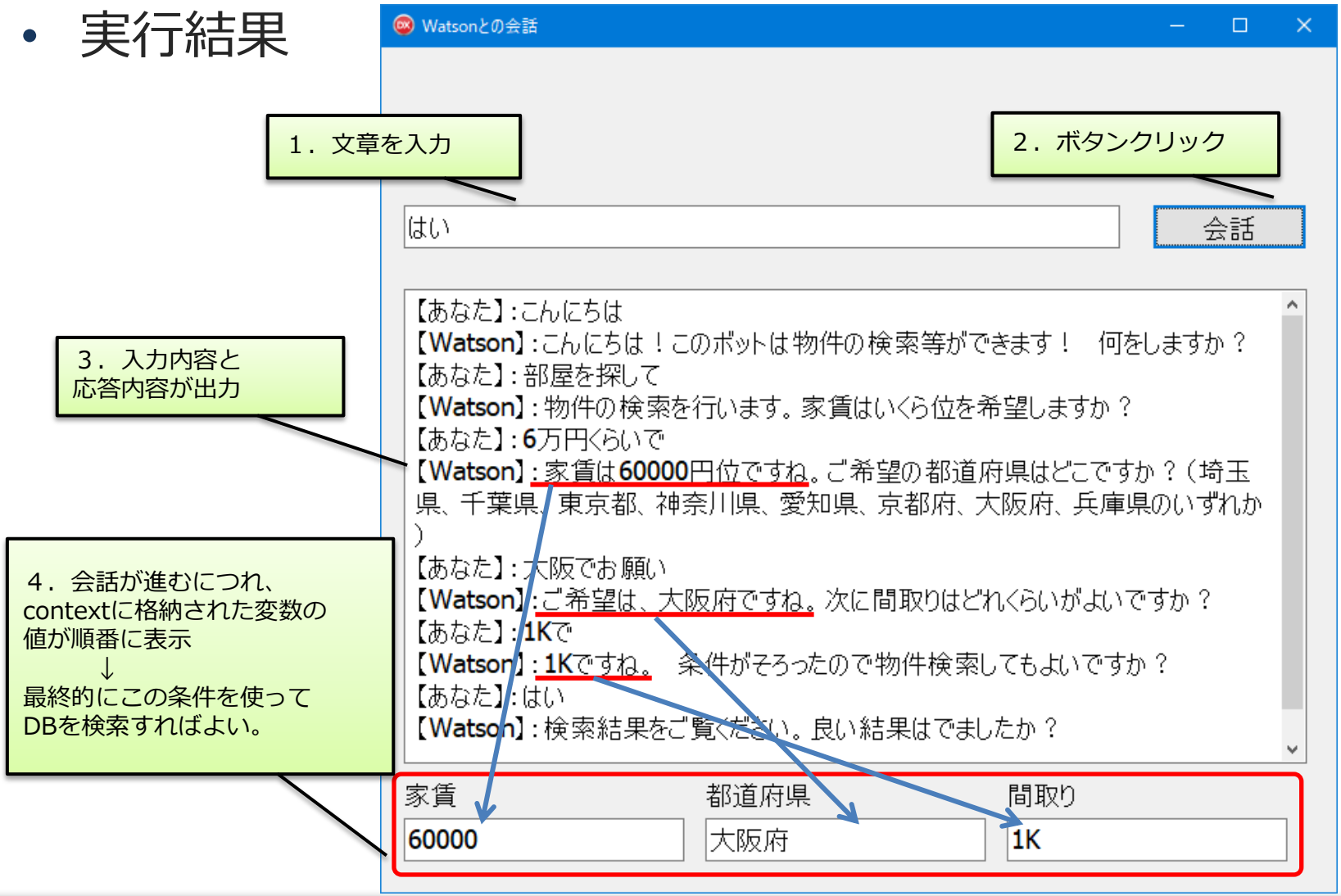
会話の状態を取得

※contextの中にある“conversation_id”が会話を特定するID

ワークスペース内でセットされた変数を取得

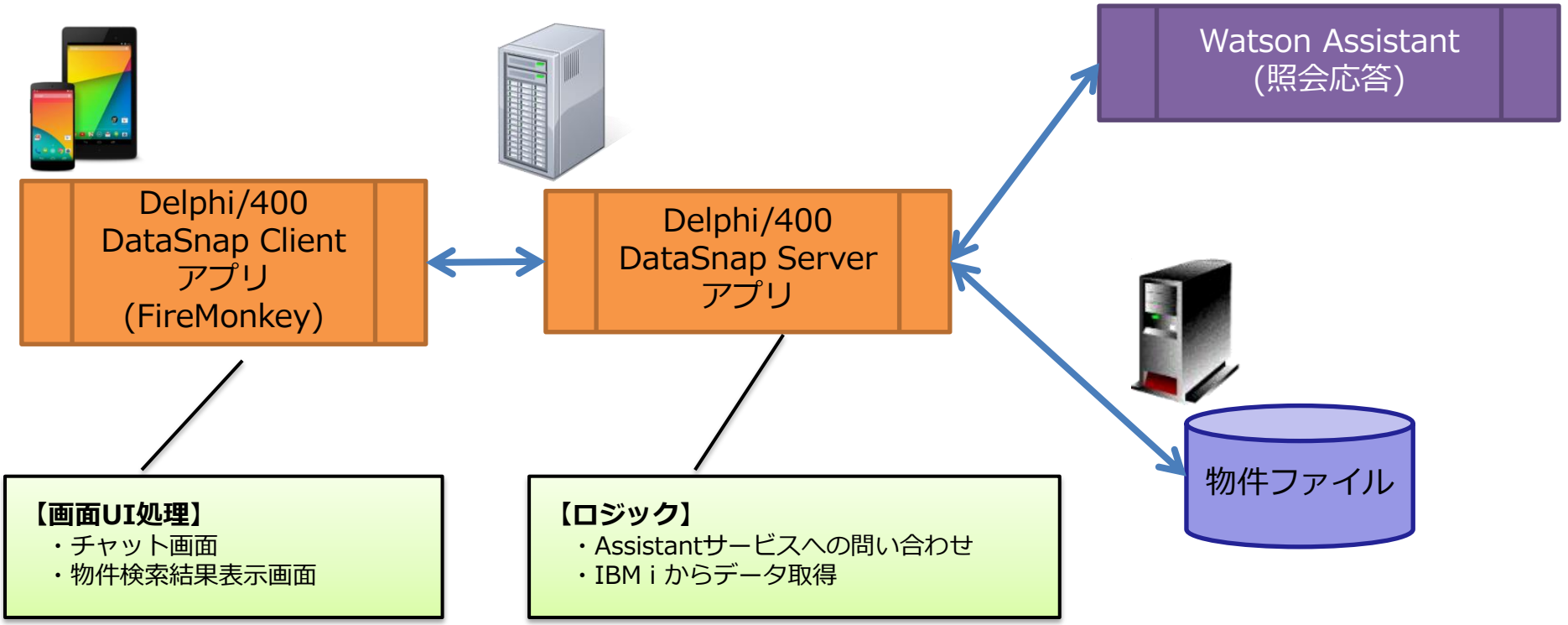
■ チャットボット機能の作成

- 実行結果



■ 物件紹介チャットアプリ

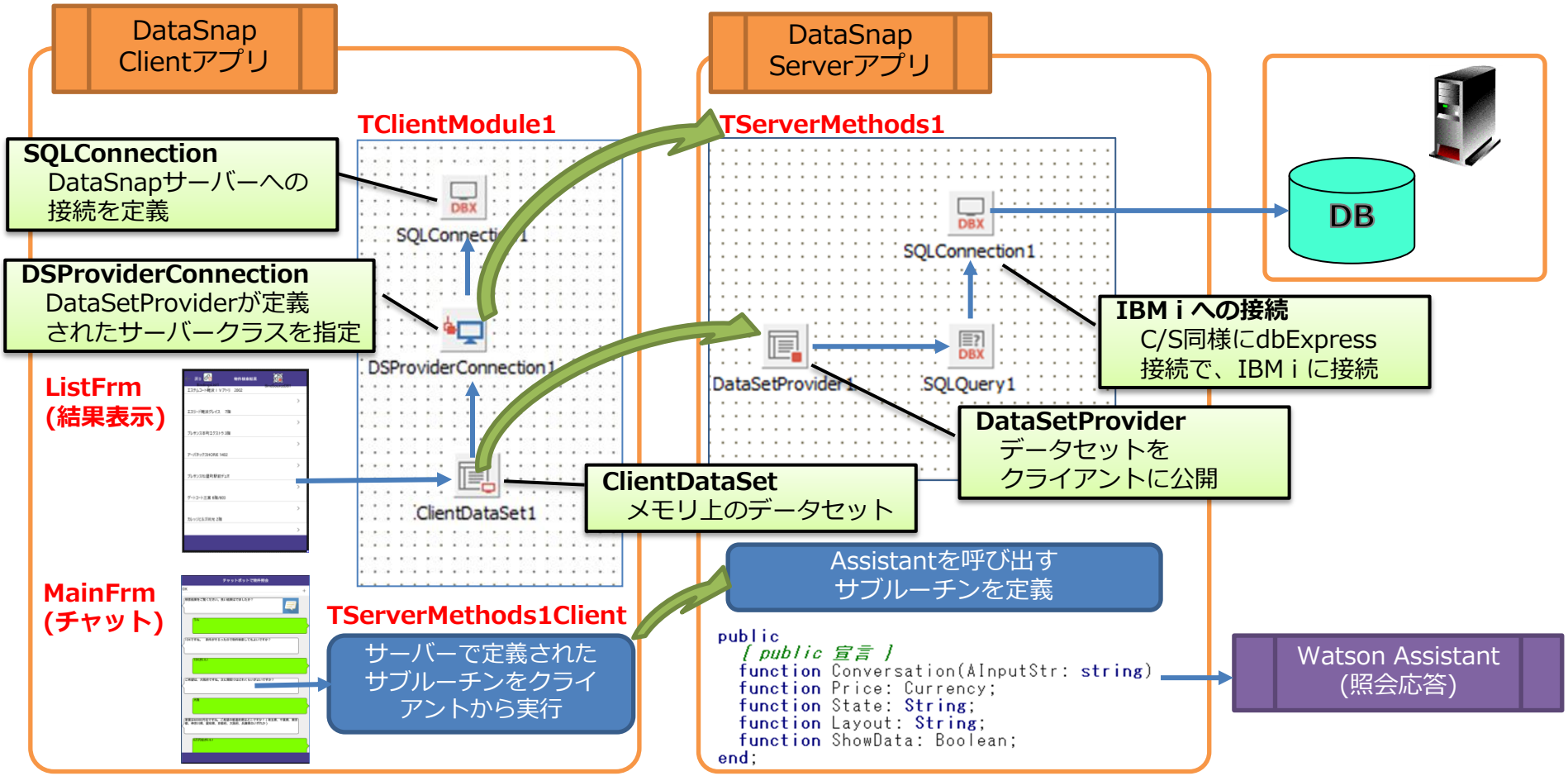
- モバイルから実行できるチャットボットアプリの作成
  - DataSnapサーバーを使用した3層アプリ



■ 物件紹介チャットアプリ

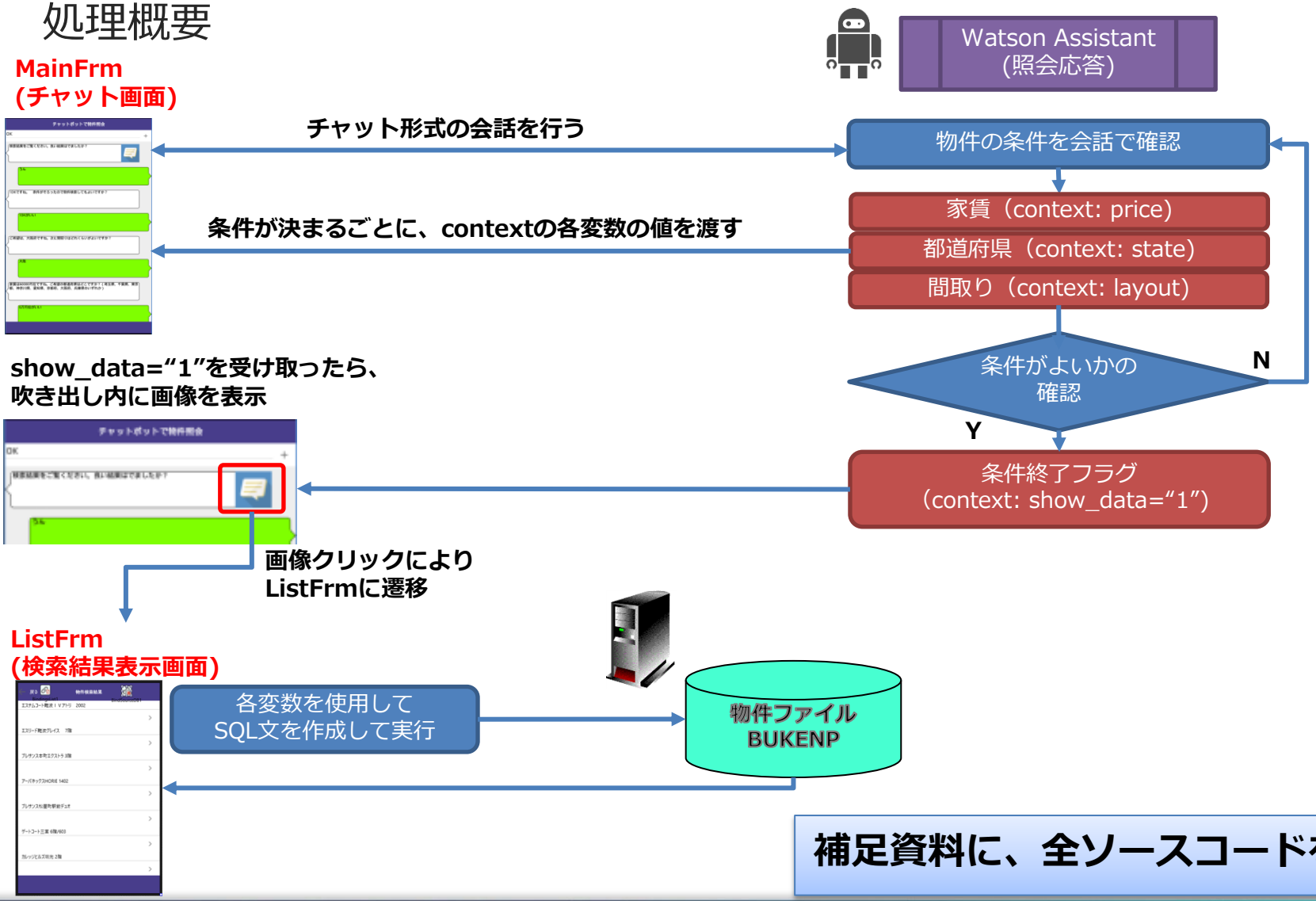
DataSnapアプリ開発の詳細については、下記を参照
第14回テクニカルセミナー
『Delphi/400 XE5
－こんなに簡単！IBM i スマートデバイスネイティブ開発－』

- DataSnapアプリ
  - dbExpress接続を使用したDataSnapアプリの構成



■ 物件紹介チャットアプリ

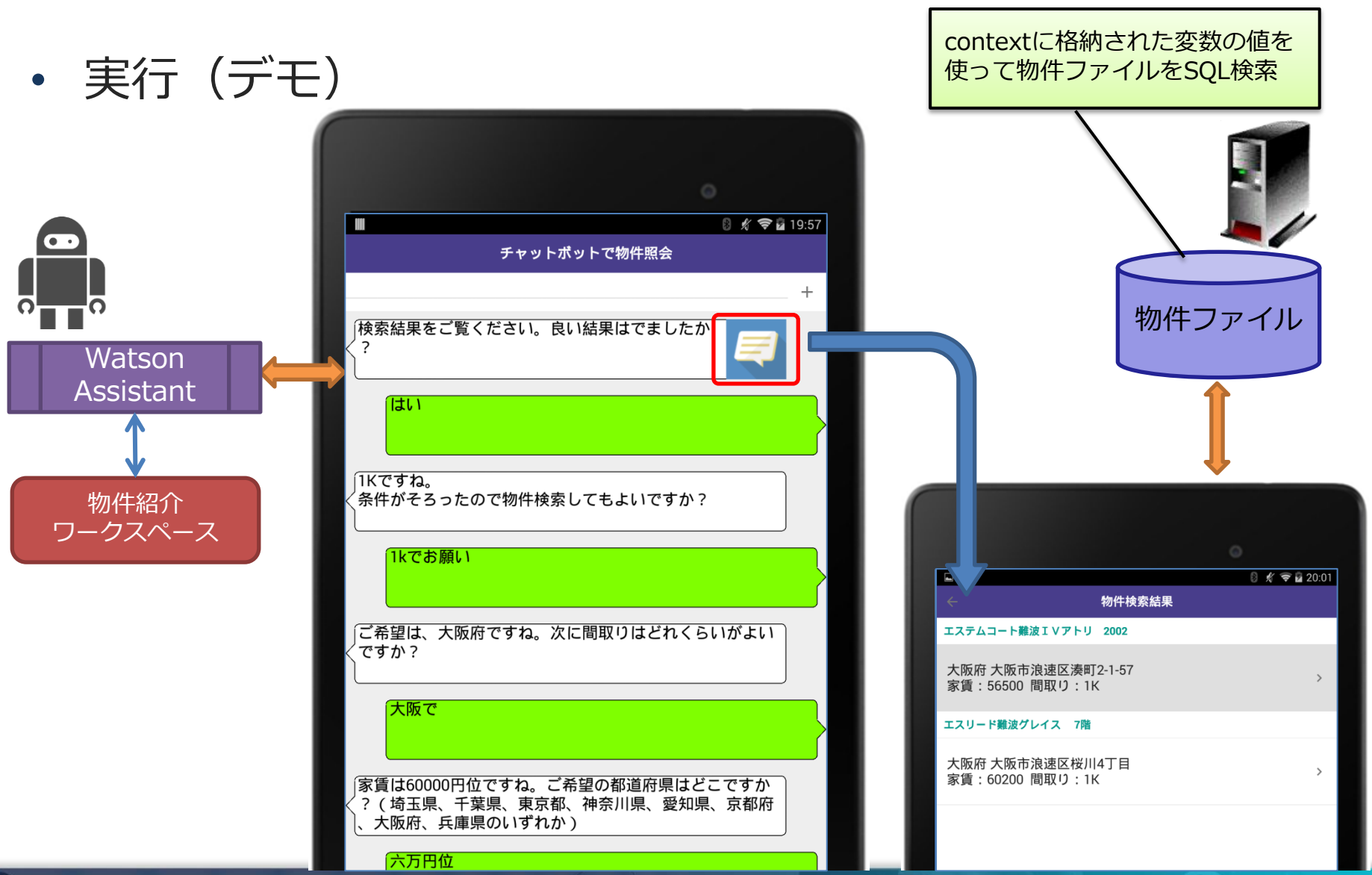
- 処理概要



補足資料に、全ソースコードを掲載

■ 物件紹介チャットアプリ

- 実行（デモ）



5．まとめ

■ まとめ

- **Watson API**
  - WatsonはIBM Cloudの一機能として使用可能
  - ライト・アカウントであれば無償で使用可能
  - Watson API は、REST/JSONサービスとして使用
  - APIリファレンスに、呼出方法が記載されている
- **Delphi/400からのWatson API呼び出し**
  - RESTコンポーネントにより呼び出しが可能
  - JSONをパース（解析）するユニットも使用可能

Delphi/400アプリに色々なWatson APIと連携させることで、
より便利なアプリ構築が可能に！

ご清聴ありがとうございました。

- 補足資料：　自動翻訳機アプリ　ソースコード

■ 補足：自動翻訳機アプリ　ソースコード

フォーム　宣言部

```
  private
    { Private 宣言 }
    FWavFileName: String;
    FMp3FileName: String;
    procedure RecStart;
    procedure RecEnd(AWavFileName: String);
    function SpeechToText(AWavFileName: String): String;
    function Transrate(AInputText: String): String;
    procedure TextToSpeech(AText, AMp3FileName: String);
    procedure PlaySpeech(AMp3FileName: String);
  public
    { Public 宣言 }
  end;
```

録音ファイル名を保持

再生ファイル名を保持

フォーム　実装部（１）

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  MemoSource.Lines.Clear;
  MemoTarget.Lines.Clear;
end;

procedure TForm1.MediaPlayer2Notify(Sender: TObject);
begin
  MediaPlayer2.Close;

  //音声ファイルを削除
  if FileExists(FMp3FileName) then
    DeleteFile(FMp3FileName);
end;
```

再生終了時イベント
音声ファイルを削除

```
procedure TForm1.PlaySpeech(AMp3FileName: String);
begin
  //MP3ファイルをメディアプレーヤーで再生
  MediaPlayer2.DeviceType := dtAutoSelect;
  MediaPlayer2.FileName := AMp3FileName;
  MediaPlayer2.Open;
  MediaPlayer2.Notify := True;
  MediaPlayer2.Play;
end;

procedure TForm1.RecEnd(AWavFileName: String);
begin
  //音声wavファイルに保存
  MediaPlayer1.FileName:= AWavFileName;
  MediaPlayer1.Stop;
  MediaPlayer1.Save;
  MediaPlayer1.Close;
end;

procedure TForm1.RecStart;
begin
  //録音処理の開始
  MediaPlayer1.DeviceType := dtWaveAudio;
  MediaPlayer1.FileName := #0;
  MediaPlayer1.Open;
  MediaPlayer1.StartRecording;
end;
```

音声再生処理

録音終了処理
録音ファイルに保存

録音開始処理

■ 補足：自動翻訳機アプリ　ソースコード

フォーム　実装部（２）

マウス押した時のイベント
録音開始

マウス離した時のイベント
録音終了から翻訳、音声再生
まで実施（メイン処理）

```
procedure TForm1.Shape1MouseDown(Sender: TObject; Buttor
  Shift: TShiftState; X, Y: Integer);
begin
  Shape1.Brush.Color := clRed;
  Application.ProcessMessages;
  //録音開始
  RecStart;
end;

procedure TForm1.Shape1MouseUp(Sender: TObject; Button:
  Shift: TShiftState; X, Y: Integer);
var
  sSourceText: String;
  sTargetText: String;
begin
  Shape1.Brush.Color := clWindow;
  Application.ProcessMessages;

  //ファイル名の取得
  FWavFileName := ChangeFileExt(ParamStr(0), '.wav');
  FMp3FileName := ChangeFileExt(ParamStr(0), '.mp3');

  //録音終了
  RecEnd(FWavFileName);

  //テキスト変換
  sSourceText := SpeechToText(FWavFileName);
  MemoSource.Lines.Add(sSourceText);

  //翻訳処理
  sTargetText := Transrate(sSourceText);
  MemoTarget.Lines.Add(sTargetText);

  //音声変換
  TextToSpeech(sTargetText, FMp3FileName);

  //音声再生
  PlaySpeech(FMp3FileName);
end;
```

音声ファイルを文字列に
変換する処理

```
function TForm1.SpeechToText(AWavFileName: String): String;
var
  memStream: TMemoryStream;
  JSONValue: TJSONValue;
begin
  memStream := TMemoryStream.Create;
  try
    memStream.LoadFromFile(AWavFileName);

    with RESTRequest_SpToTx do
    begin
      //変換パラメータの指定
      Params.AddItem('Content-Type', 'audio/wav'
        , pkHTTPHEADER, [poDoNotEncode]);
      Params.AddItem('audio', memStream, pkREQUESTBODY, [], ctAUDIO_BASIC);
      //リクエスト実行
      Execute;
    end;

    //処理結果JSONのパース
    JSONValue := TJSONObject.ParseJSONValue(RESTResponse_SpToTx.Content);
    try
      Result := JSONValue.GetValue<String>
        ('results[0].alternatives[0].transcript');
    except
      //エラー
      raise Exception.Create('文字取得できる音声データではありません');
    end;
  finally
    memstream.Free;
  end;

  //変換終了した後ファイルを削除
  if FileExists(AWavFileName) then
    DeleteFile(AWavFileName);
end;
```

■ 補足：自動翻訳機アプリ　ソースコード

フォーム　実装部（３）

テキスト文字列を音声ファイルに変換する処理

```
procedure TForm1.TextToSpeech(AText, AMp3FileName: String);
var
  memStream: TMemoryStream;
  sFileName: string;
begin
  //エラーチェック
  if Trim(AText) = '' then
    raise Exception.Create('音声変換するテキストがありません');

  with RESTRequest_TxToSp do
  begin
    //変換パラメータの指定
    Params.AddItem('text', AText, pkGETorPOST);
    //リクエスト実行
    Execute;
  end;

  //レスポンスのMP3データをファイルに保管する
  memStream := TMemoryStream.Create;
  try
    memStream.WriteData(RESTResponse_TxToSp.RawBytes
      , Length(RESTResponse_TxToSp.RawBytes));

    memStream.SaveToFile(AMp3FileName);
  finally
    memStream.Free;
  end;
end;
```

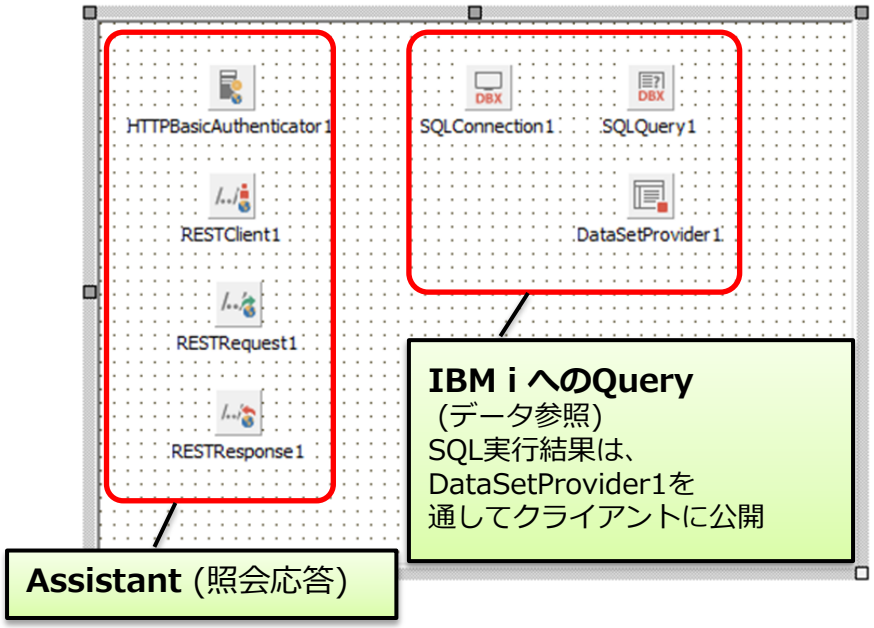
入力文字列を翻訳する処理

```
function TForm1.Transrate(AInputText: String): String;
var
  JSONValue: TJSONValue;
  sRet: String;
begin
  //エラーチェック
  if Trim(AInputText) = '' then
    raise Exception.Create('翻訳元テキストがありません');

  with RESTRequest_Trans do
  begin
    //翻訳パラメータの指定
    Params.AddItem('text', AInputText, pkGETorPOST); //翻訳元テキスト
    //リクエスト実行
    Execute;
  end;

  //翻訳テキストの取得
  JSONValue := TJSONObject.ParseJSONValue(RESTResponse_Trans.Content);
  Result := JSONValue.GetValue<string>('translations[0].translation');
end;
```

- 補足資料：物件紹介チャットアプリ　ソースコード

■ 補足：物件紹介チャットアプリソースコード

- DataSnap Server アプリ
  - AssistantサービスとIBM iへの問い合わせ



クライアントに公開するサブルーチン
Conversation：入力文字を入れると応答文字を返却
Price：　家賃の現在値を取得
State：　都道府県の現在値を取得
Layout：　間取りの現在値を取得
ShowData：検索の条件がそろった場合、True

TServerMethods1【宣言部】

```
type
  TServerMethods1 = class(TDSServerModule)
    RESTResponse1: TRESTResponse;
    RESTRequest1: TRESTRequest;
    RESTClient1: TRESTClient;
    HTTPBasicAuthenticator1: THTTPBasicAuthenticator;
    SQLConnection1: TSQLConnection;
    SQLQuery1: TSQLQuery;
    SQLQuery1BUBKCD: TStringField;
    SQLQuery1BUBKNM: TStringField;
    SQLQuery1BUSTAT: TStringField;
    SQLQuery1BUADRS: TStringField;
    SQLQuery1BUPRIC: TIntegerField;
    SQLQuery1BULAYT: TStringField;
    DataSetProvider1: TDataSetProvider;
    procedure DSServerModuleCreate(Sender: TObject);
  private
    { private 宣言 }
    FContext: String;
    FPrice: Currency;
    FState: String;
    FLayout: String;
    FShowData: Boolean;
  public
    { public 宣言 }
    function Conversation(AInputStr: string): string;
    function Price: Currency;
    function State: String;
    function Layout: String;
    function ShowData: Boolean;
  end;
```

■ 補足：物件紹介チャットアプリソースコード

- DataSnap Server アプリ

フォーム　実装部（1）

```
function TServerMethods1.Conversation(AInputStr: string): string;
var
  JSONValue: TJSONValue;
  JSONPair: TJSONPair;
  sMsg: String;
begin
  RESTRequest1.Params.Clear;
  RESTRequest1.Params.AddItem;

  RESTRequest1.Params.Clear;
  //リクエストパラメータのセット
  if FContext = '' then  //----- 会話が初回の場合
    sMsg := '{"input": {"text": "' + AInputStr + '"}}'
  else                   //----- 会話が2回目以降の場合
    sMsg := '{"input": {"text": "' + AInputStr + '"}, '
      + FContext + '}';  //会話の状態(FContext)をセット

  RESTRequest1.Params.AddItem('body', sMsg, pkGETorPOST
    , [], ctAPPLICATION_JSON);

  //リクエストの実行
  RESTRequest1.Execute;

  //処理結果JSONのパース
  JSONValue := TJSONObject.ParseJSONValue(RESTResponse1.Content);

  //結果 (output->text)の取得
  Result := JSONValue.GetValue<String>('output.text[0]');

  //会話の状態 (context) を取得
  JSONPair := TJSONObject(JSONValue).Get('context');
  FContext := JSONPair.ToJSON;

  //会話の状態より、変数値を取得
  try
    FPrice := JSONValue.GetValue<Currency>('context.price');   // 家賃
  except
    FPrice := 0;  //キーが存在しない場合
  end;

  try
    FState := JSONValue.GetValue<String>('context.state');     // 都道府県
  except
    FState := ''; //キーが存在しない場合
  end;

  try
    FLayout := JSONValue.GetValue<String>('context.layout'); // 間取り
  except
    FLayout := ''; //キーが存在しない場合
  end;

  try
    FShowData := JSONValue.GetValue<String>
                   ('context.show_data') <> ''; //検索結果表示
  except
    FShowData := False;  //キーが存在しない場合
  end;
end;

procedure TServerMethods1.DSServerModuleCreate(Sender: TObject);
begin
  //変数初期化
  FContext := '';
  FPrice := 0;
  FState := '';
  FLayout := '';
  FShowData := False;

  //IBMi接続の初期設定
  with SQLConnection1 do
  begin
    Params.Values['RoleName'] := 'D4TEC22LIB';
    Params.Values['HostName'] := 'MIGAROI5';
    Params.Values['User_Name'] := 'D400';
    Params.Values['Password'] := 'D400';
    Params.Values['Database'] := 'MIGAROI5';
  end;

  //IBMiへ接続
  SQLConnection1.Connected := True;
end;
```

入力文字列をもとに、会話の応答を返却する処理

OnCreateイベント
初期処理　変数初期化、DB接続

IBM i の接続環境にあわせて要修正

■ 補足：物件紹介チャットアプリソースコード

- DataSnap Server アプリ

フォーム　実装部（２）

```
function TServerMethods1.Layout: String;
begin
  Result := FLayout;  // 間取り
end;

function TServerMethods1.Price: Currency;
begin
  Result := FPrice; // 家賃
end;

function TServerMethods1.ShowData: Boolean;
begin
  Result := FShowData; // 検索結果表示
end;

function TServerMethods1.State: String;
begin
  Result := FState; // 都道府県
end;
```

間取りの現在値を返却する処理

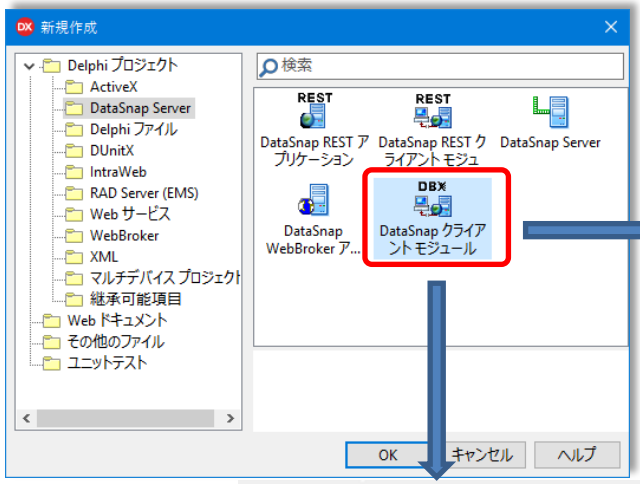
家賃の現在値を返却する処理

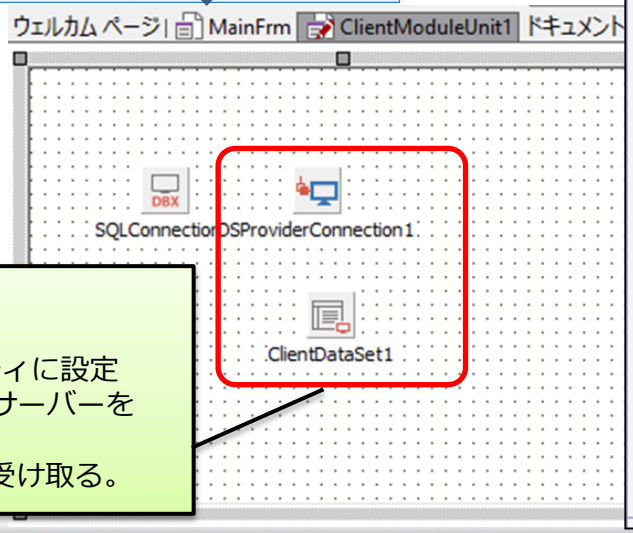
検索結果を表示するフラグの現在値を返す処理

都道府県の現在値を返却する処理

■ 補足：物件紹介チャットアプリソースコード

- DataSnap Client アプリ
  - DataSnapクライアントモジュールの作成方法





TServerMethods1Client

DataSnapサーバーに用意されたメソッドが自動的に定義。
TServerMethods1Clientクラスのオブジェクトを通して
アクセス可能。

```
//
// DataSnap プロキシ ジェネレータにより作成。
// 2018/05/12 20:53:35
//

unit ClientClassesUnit1;

interface

uses System.JSON, Data.DBXCommon, Data.DBXClient, Da

type
  TServerMethods1Client = class(TDSAdminClient)
  private
    FDSServerModuleCreateCommand: TDBXCommand;
    FConversationCommand: TDBXCommand;
    FPriceCommand: TDBXCommand;
    FStateCommand: TDBXCommand;
    FLayoutCommand: TDBXCommand;
    FShowDataCommand: TDBXCommand;
  public
    constructor Create(ADBXConnection: TDBXConnectio
    constructor Create(ADBXConnection: TDBXConnectio
    destructor Destroy; override;
    procedure DSServerModuleCreate(Sender: TObject);
    function Conversation(AInputStr: string): string
    function Price: Currency;
    function State: string;
    function Layout: string;
    function ShowData: Boolean;
  end;
```

IBM i へのQuery発行

CommandTextプロパティに設定
したSQL文をDataSnapサーバーを
通して実行。
結果をClientDataSetが受け取る。

■ 補足：物件紹介チャットアプリソースコード

- DataSnap Client アプリ
  - ClientModuleUnit1

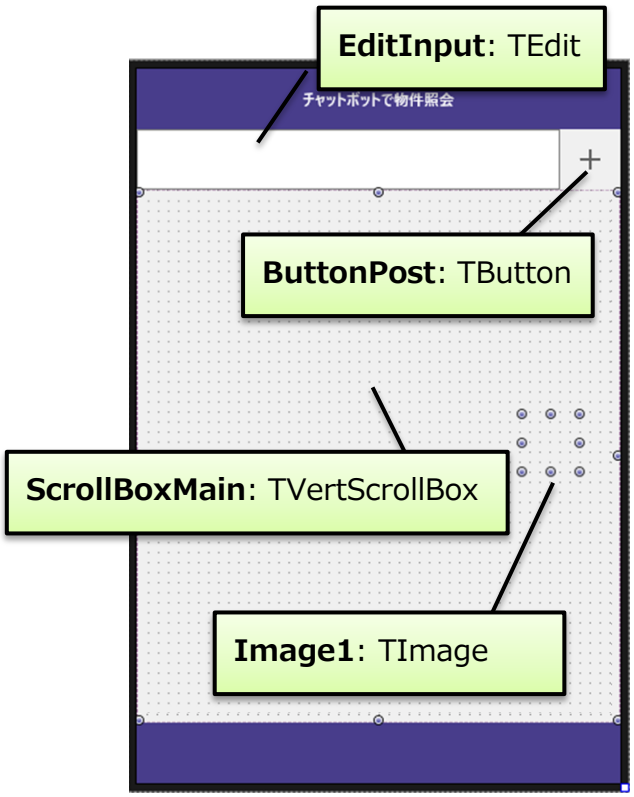
実装部

OnCalcFieldsイベント
[計算項目]に値をセット

```
procedure TClientModule1.ClientDataSet1CalcFields(DataSet: TDataSet);
begin
  ClientDataSet1DetailInfo.AsString :=
        ' ' + ClientDataSet1BUSTAT.AsString
    + ' ' + ClientDataSet1BUADRS.AsString
    + #13#10
    + '家賃：' + ClientDataSet1BUPRIC.AsString
    + '  間取り：' + ClientDataSet1BULAYT.AsString;
end;
```

■ 補足：物件紹介チャットアプリソースコード

- DataSnap Client アプリ
  - MainFrm



宣言部

```
type
  THeaderFooterForm = class(TForm)
    Header: TToolBar;
    Footer: TToolBar;
    HeaderLabel: TLabel;
    ToolBarEntry: TToolBar;
    ScrollBoxMain: TVertScrollBox;
    ButtonPost: TButton;
    EditInput: TEdit;
    Rectangle1: TRectangle;
    Rectangle2: TRectangle;
    Image1: TImage;
    procedure ButtonPostClick(Sender: TObject);
    procedure Image1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { private 宣言 }
    FPrice: Currency;          // 家賃
    FState: String;            // 都道府県
    FLayout: String;           // 間取り
    FShowData: Boolean;        // 検索画面表示フラグ
    procedure SetRightMsg(sText: String);
    procedure SetLeftMsg(sText: String);
  public
    { public 宣言 }
  end;

  //情報保持できるイメージコンポーネント
  TValueImage = class(TImage)
  private
    FPrice: Currency;          // 家賃
    FState: String;            // 都道府県
    FLayout: String;           // 間取り
  public
    constructor Create(AOwner: TComponent; APrice: Currency;
      AState, ALayout: String);          // 生成時情報をセット
    property Price: Currency read FPrice;    // プロパティ：家賃
    property State: String read FState;      // プロパティ：都道府県
    property Layout: String read FLayout;    // プロパティ：間取り
  end;
```

TImageクラスに
家賃、都道府県、間取りの情報を
追加

■ 補足：物件紹介チャットアプリソースコード

- DataSnap Client アプリ
  - MainFrm

ボタン押下時処理
（メイン処理）

```
procedure THeaderFooterForm.ButtonPostClick(Sender:
var
  sRequestMsg: String;
  sResponceMsg: String;
begin
  //入力ブランクの場合、何もしない
  if EditInput.Text = EmptyStr then
    Exit;

  //リクエストメッセージをセット
  sRequestMsg := EditInput.Text;
  EditInput.Text := '';

  //発言メッセージをセット
  SetRightMsg(sRequestMsg);
  Application.ProcessMessages;

  //Watsonに問い合わせを行い、結果を受け取る
  with ClientModule1.ServerMethods1Client do
  begin
    sResponceMsg := Conversation(sRequestMsg);
    //検索情報の取得
    FPrice := Price;
    FState := State;
    FLayout := Layout;
    FShowData := ShowData;
  end;
  //応答メッセージをセット
  SetLeftMsg(sResponceMsg);

  EditInput.SetFocus;
end;
```

画像クリック時処理
検索結果画面へ遷移

```
procedure THeaderFooterForm.Image1Click(Sender:
begin
  //検索結果画面を表示
  frmList := TfrmList.Create(Self);
  frmList.Price := TValueImage(Sender).FPrice;
  frmList.State := TValueImage(Sender).FState;
  frmList.Layout := TValueImage(Sender).FLayout;
  frmList.Show;
end;
```

左側（ワトソン側）
吹き出し表示処理

```
procedure THeaderFooterForm.SetLeftMsg(sText: String);
var
  crLeftMsg: TCalloutRectangle;
  txtLeftMsgText: TText;
  TmpImg: TValueImage;
begin
  //左側の吹き出しセット
  crLeftMsg := TCalloutRectangle.Create(Self);
  with crLeftMsg do
  begin
    Parent := ScrollBoxMain;
    Align := TAlignLayout.Top;
    CalloutPosition := TCalloutPosition.Left;
    CornerType := TCornerType.Round;
    XRadius := 5; YRadius := 5;
    Margins.Top := 10;
    Margins.Bottom := 10;
    Margins.Right := 50;
    Height := 75;
    Fill.Color := claWhite;
  end;

  //メッセージセット
  txtLeftMsgText := TText.Create(Self);
  with txtLeftMsgText do
  begin
    Parent := crLeftMsg;
    Align := TAlignLayout.Client;
    TextSettings.HorzAlign := TTextAlign.Leading;
    TextSettings.VertAlign := TTextAlign.Leading;
    Text := sText;
    Margins.Left := 15; Margins.Right := 5;
    Width := crLeftMsg.Width - 20;
    WordWrap := True;
    AutoSize := True;
  end;

  //検索結果ありの場合、アイコンを表示する
  if FShowData then
  begin
    TmpImg := TValueImage.Create(Self, FPrice, FState, FLayout);
    TmpImg.Parent := crLeftMsg;
    TmpImg.Align := TAlignLayout.FitRight;
    TmpImg.Bitmap.Assign(Image1.Bitmap);
    TmpImg.Width := 75;
    TmpImg.OnClick := Image1Click;
  end;
end;
```

■ 補足：物件紹介チャットアプリソースコード

- DataSnap Client アプリ
  - MainFrm

```
procedure THeaderFooterForm.SetRightMsg(sText: String);
var
  crRightMsg: TCalloutRectangle;
  txtRightMsgText: TText;
begin
  //右側の吹き出しセット
  crRightMsg := TCalloutRectangle.Create(Self);
  with crRightMsg do
  begin
    Parent := ScrollBoxMain;
    Align := TAlignLayout.Top;
    CalloutPosition := TCalloutPosition.Right;
    CornerType := TCornerType.Round;
    XRadius := 5; YRadius := 5;
    Margins.Top := 10;
    Margins.Bottom := 10;
    Margins.Left := 50;
    Height := 75;
    Fill.Color := claChartreuse;
  end;
  //メッセージセット
  txtRightMsgText := TText.Create(Self);
  with txtRightMsgText do
  begin
    Parent := crRightMsg;
    Align := TAlignLayout.Client;
    TextSettings.HorzAlign := TTextAlign.Leading;
    TextSettings.VertAlign := TTextAlign.Leading;
    Text := sText;
    Margins.Right := 15; Margins.Left := 5;
    Width := crRightMsg.Width - 20;
    WordWrap := True;
    AutoSize := True;
  end;
end;
```

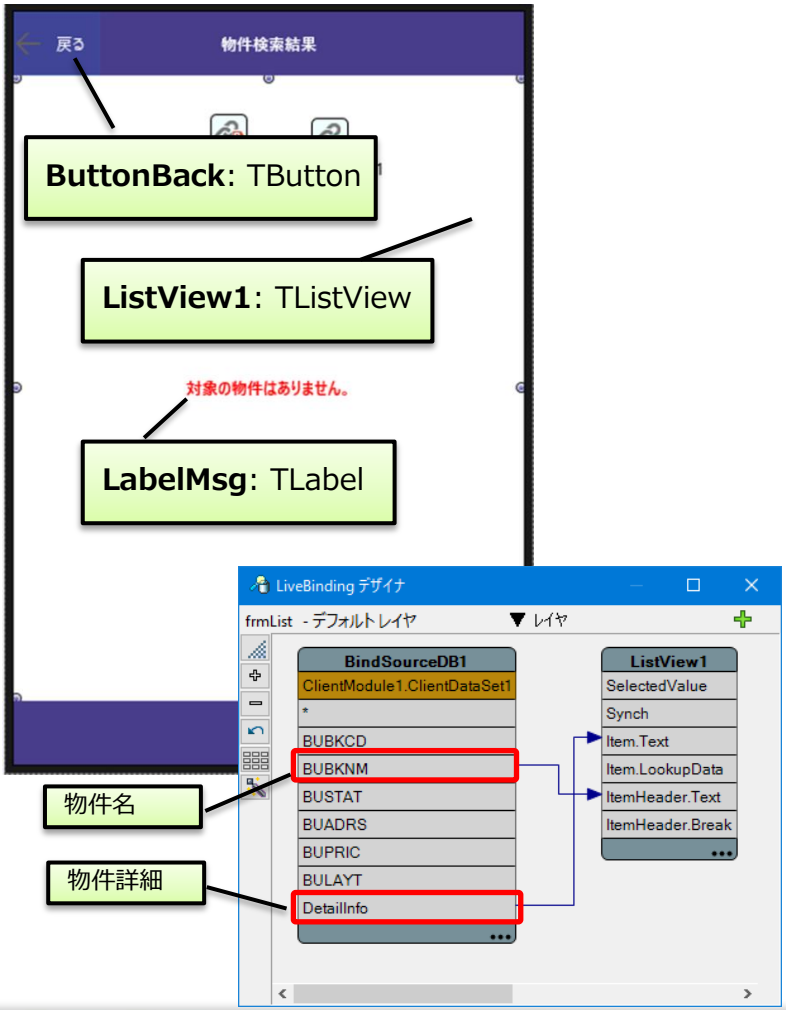
右側（人側）
吹き出し表示処理

拡張Imageクラスの
生成処理

```
{ TValueImage }

constructor TValueImage.Create(AOwner: TComponent; APrice:
  Currency; AState, ALayout: String);
begin
  inherited Create(AOwner);
  //データの抽出条件を保持
  FPrice := APrice;
  FState := AState;
  FLayout := ALayout;
end;
```

■ 補足：物件紹介チャットアプリソースコード

- DataSnap Client アプリ
  - ListFrm



宣言部

```
type
  TfrmList = class(TForm)
    Footer: TToolBar;
    Rectangle1: TRectangle;
    Header: TToolBar;
    Rectangle2: TRectangle;
    HeaderLabel: TLabel;
    ButtonBack: TButton;
    ListView1: TListView;
    BindSourceDB1: TBindSourceDB;
    BindingsList1: TBindingsList;
    LinkFillControlToField1: TLinkFillControlToField;
    LabelMsg: TLabel;
    procedure ButtonBackClick(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseA
    procedure FormCreate(Sender: TObject);
  private
    { private 宣言 }
    FLayout: String;
    FPrice: Currency;
    FState: String;
    function SetBukkenData(APrice: Currency; AState,
      ALayout: String): Boolean;
  public
    { public 宣言 }
    property Price: Currency read FPrice write FPrice;
    property State: String read FState write FState;
    property Layout: String read FLayout write FLayout;
  end;
```

■ 補足：物件紹介チャットアプリソースコード

- DataSnap Client アプリ
  - ListFrm

戻るボタン押下時処理

```
procedure TfrmList.ButtonBackClick(Sender: TObject);
begin
  //画面を終了する
  Close;
end;
```

フォーム終了処理

```
procedure TfrmList.FormClose(Sender: TObject; var Action:
begin
  //フォームメモリを解放
  Action := TCloseAction.caFree;
end;
```

フォーム表示処理
物件データを取得

```
procedure TfrmList.FormShow(Sender: TObject);
begin
  LabelMsg.Visible := False;

  //データの取得（データない場合、メッセージ表示）
  if not SetBukkenData(FPrice, FState, FLayout) then
    LabelMsg.Visible := True;
end;
```

物件データ取得処理
SQLを指定して
ClientDataSet取得

```
function TfrmList.SetBukkenData(APrice: Currency;
  AState, ALayout: String): Boolean;
var
  sSQL: String;
begin
  //初期処理
  Result := False;

  //SQL文の作成
  sSQL :=  'SELECT * FROM BUKENP'
       + ' WHERE BUSTAT = ''' + AState + ''''
       + ' AND BUPRIC >= ' + CurrToStr(APrice - 20000)
       + ' AND BUPRIC <= ' + CurrToStr(APrice + 5000)
       + ' AND BULAYT = ''' + ALayout + ''''
       + ' ORDER BY BUBKCD ';

  //SQLを実行して、結果を取得
  with ClientModule1.ClientDataSet1 do
  begin
    Active := False;
    CommandText := sSQL;
    Active := True;

    if Eof and Bof then
    begin
      //対象データが無い場合、データセットを閉じる
      Active := False;
      Exit;
    end;

    //データが存在する場合True
    Result := True;
  end;
end;
```