記事
Mihoko Iijima · 2020年10月27日 12m read

【はじめてのInterSystems IRIS】Interoperability（相互運用性）：コンポーネントの作成（ビジネス・オペレーション）

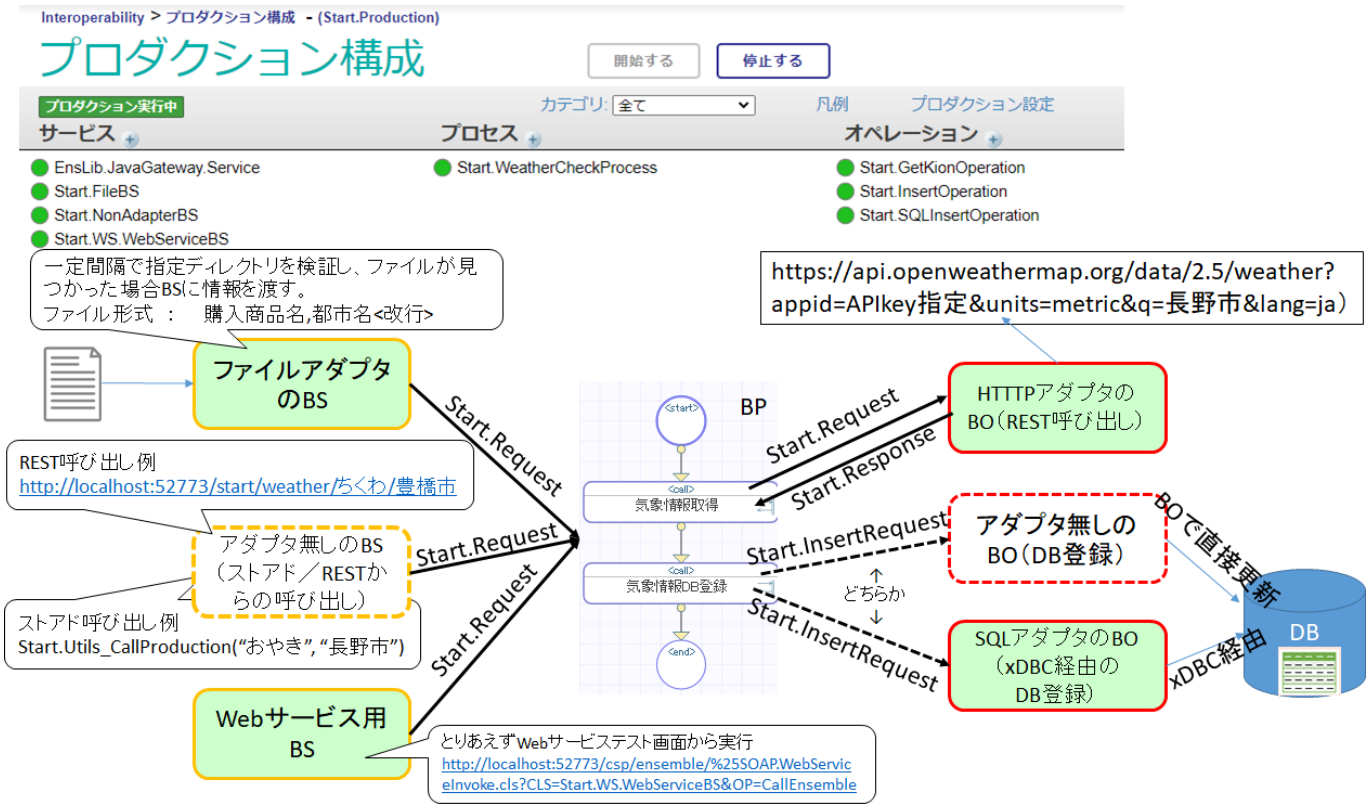
この記事はこちらの投稿の続きの内容です。

前回の記事
では、コンポーネント間のデータ送受信に使用される **メッセージ**
について、作成するときの考え方や定義方法を確認しました。

今回の記事では、コンポーネントの作成方法の中から、ビジネス・オペレーションの作成について解説します。

- プロダクション
- メッセージ（前回の記事）
- **コンポーネント**
  - ビジネス・サービス
  - ビジネス・プロセス
  - **ビジネス・オペレーション**

早速サンプルを参照しながらコードを確認します。



コンポーネント名	役割

Start.FileBS	指定ディレクトリに置かれたファイルを一定間隔で読 インバウンドアダプタを利用している**ビジネス**
Start.NonAdapterBS	アダプタを利用せず直接アプリケーションやユーザか てもらうための**ビジネス・サービス**
Start.WS.WebServiceBS	Web サービスを利用して情報を入力してもらうための**ビジ**
Start.WeatherCheckProcess	気 象情 報を取得 してからデータベ ースの登録を行う手順を制御する**ビジネス・**
Start.GetKionOperation	気象情報提供先のWeb サービスへ都市名 を渡し気象情報を返送してもらう**ビジネス・オペ**
Start.SQLInsertOperation	データベ ースへ気象情報と 購入情報の登録依頼を行うため、SQL アウトバウンドアダプタを使用する**ビジネス・オ**
Start.InsertOperation	アダプタを使用せず、InterSystems IRIS内テーブル対して更新処理を行う

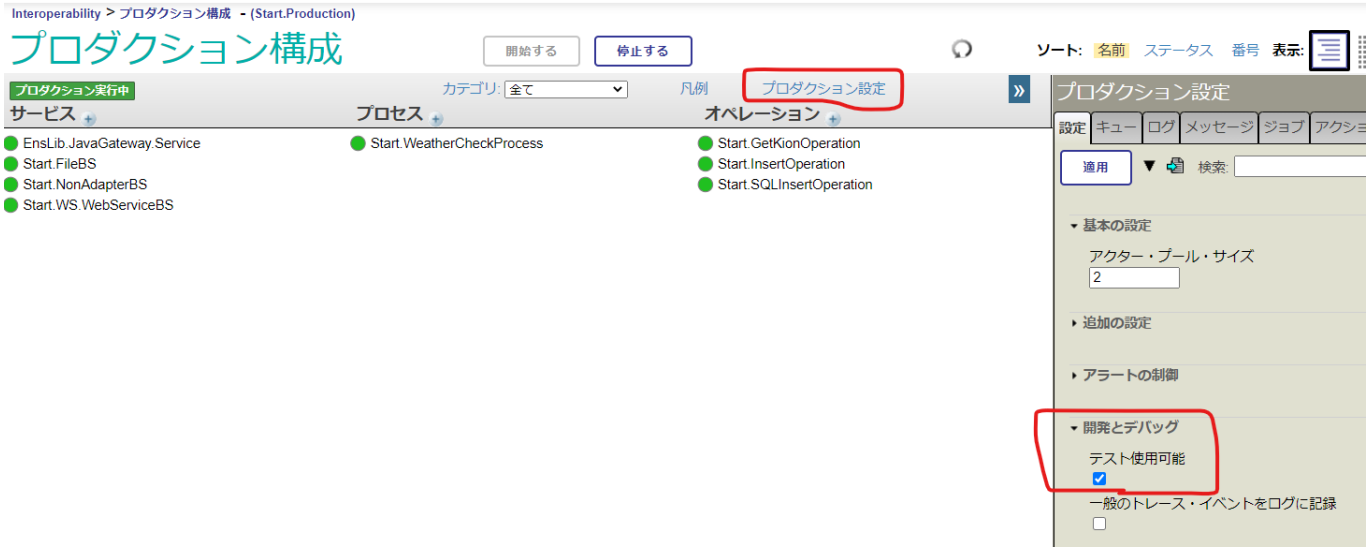
メモ：BSはビジネス・サービス、BPはビジネス・プロセス、BOはビジネス・オペレーションの略です。

ビジネス・サービスとビジネス・オペレーションはスクリプトの記述があるため、VSCode または スタジオで作成しま
す。ビジネス
・プロセスは管理ポータルでも作成できます（VSCodeの使い方については、こちらの記事をご参照ください）。

作成順は特にありませんが、今回のサンプルで考えると、接続先の外部サイトは公開サイトであるためすぐに利用できます。こういった場合ビジネス・オペレーションから作成し始めるとテストが簡単に行えて便利です。

コンポーネント作成後のテストですが、ビジネス・プロセスとビジネス・オペレーションについては、プロダクションにテスト画面が用意されています。
ただ、本番環境で誤ってテストしないように、プロダクション定義のデフォルト設定でテストは無効化されています。

プロダクションで「テスト使用可」とするかどうかの設定方法は、以下の通りです（サンプルプロダクションは予め「テスト使用可」に設定しています）。



1) ビジネス・オペレーション

サンプルでは、2種類のビジネス・オペレーションを用意しています。

1つは、REST で外部の Web API
に都市名を渡し、気象情報の取得を依頼するオペレーションで、もう1つは、InterSystems IRIS
内のデータベースへ気象情報と購入商品名を渡し更新処理を依頼するオペレーションです。

1)-1 REST ビジネス・オペレーション

REST で外部の Web API を呼び出すオペレーションの作成から確認していきましょう 。
このオペレーションは、Start.Request メッセージが入力されると GetKion()
メソッドが起動し、外部サイトへ問い合わせを行い、気象情報を Start.Response メッセージに格納し返送します.

コード詳細はこちらをご参照ください。

REST 用ビジネス・オペレーションを作成する場合は、**EnsLib.REST.Operation** を継承します。

```
Class Start.GetKionOperation Extends EnsLib.REST.Operation
```

このクラスの継承により HTTP メソッドに合わせた IRIS
内の以下メソッドが提供されます。詳細はドキュメントもご参照ください。

- GetURL()— HTTP GET オペレーションで使用します。
- PostURL()— HTTP POST オペレーションで使用します。
- PutURL()— HTTP PUT オペレーションをで使用します。
- DeleteURL()— HTTP DELETE オペレーションで使用します。

REST の場合、アダプタは **EnsLib.HTTP.OutboundAdapter** を使用します。アダプタ名を例のように **ADAPTER** パラメータと **Adapter** プロパティに設定します。

また **INVOCATION** パラメータは **Queue** を設定します。

```
Parameter ADAPTER = "EnsLib.HTTP.OutboundAdapter";
Property Adapter As EnsLib.HTTP.OutboundAdapter;
Parameter INVOCATION = "Queue";
```

接続先の OpenWeather の API は実行時に各自で入手する API Key の指定が必要になります。環境に合わせて設定値が変動するような項目は、プロダクションの設定項目として表示させる方法があります。

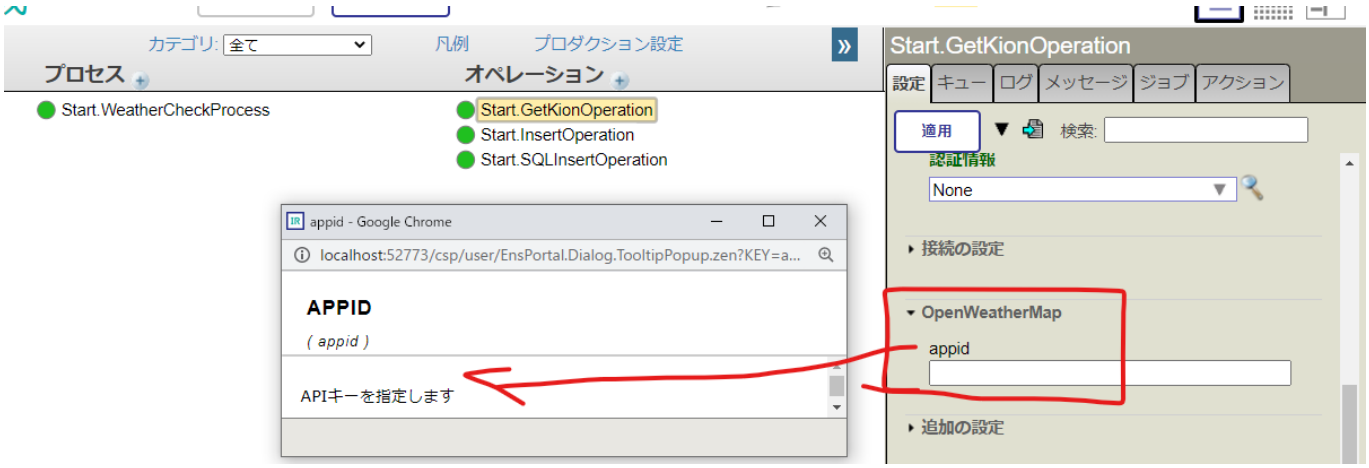
手順は以下の通りです。

1. プロパティを定義する
2. SETTINGSパラメータに作成したプロパティ名を指定する（複数ある場合はカンマで区切って指定します）。 任意でカテゴリも指定できます（プロパティ名:カテゴリ名 で指定します）。

コード例は以下の通りです。

```
/// API????????
Property appid As %String;
Parameter SETTINGS = "appid:OpenWeatherMap";
```

プロダクション設定画面では以下のように表示されます。また、プロパティ定義の直前の行に記載した説明文は、図のようにプロダクション設定画面にも表示されます。



次に、ビジネス・オペレーションにとって大事な設定である、メッセージマップについて確認します。

```
80  XData MessageMap
81  {
82  <MapItems>
83      <MapItem MessageType="Start.Request">
84          <Method>GetKion</Method>
85      </MapItem>
86  </MapItems>
87  }
```

上記定義は、**Start.Request メッセージ**が送信されると、**GetKion()メソッドが動作する**ように定義されています。

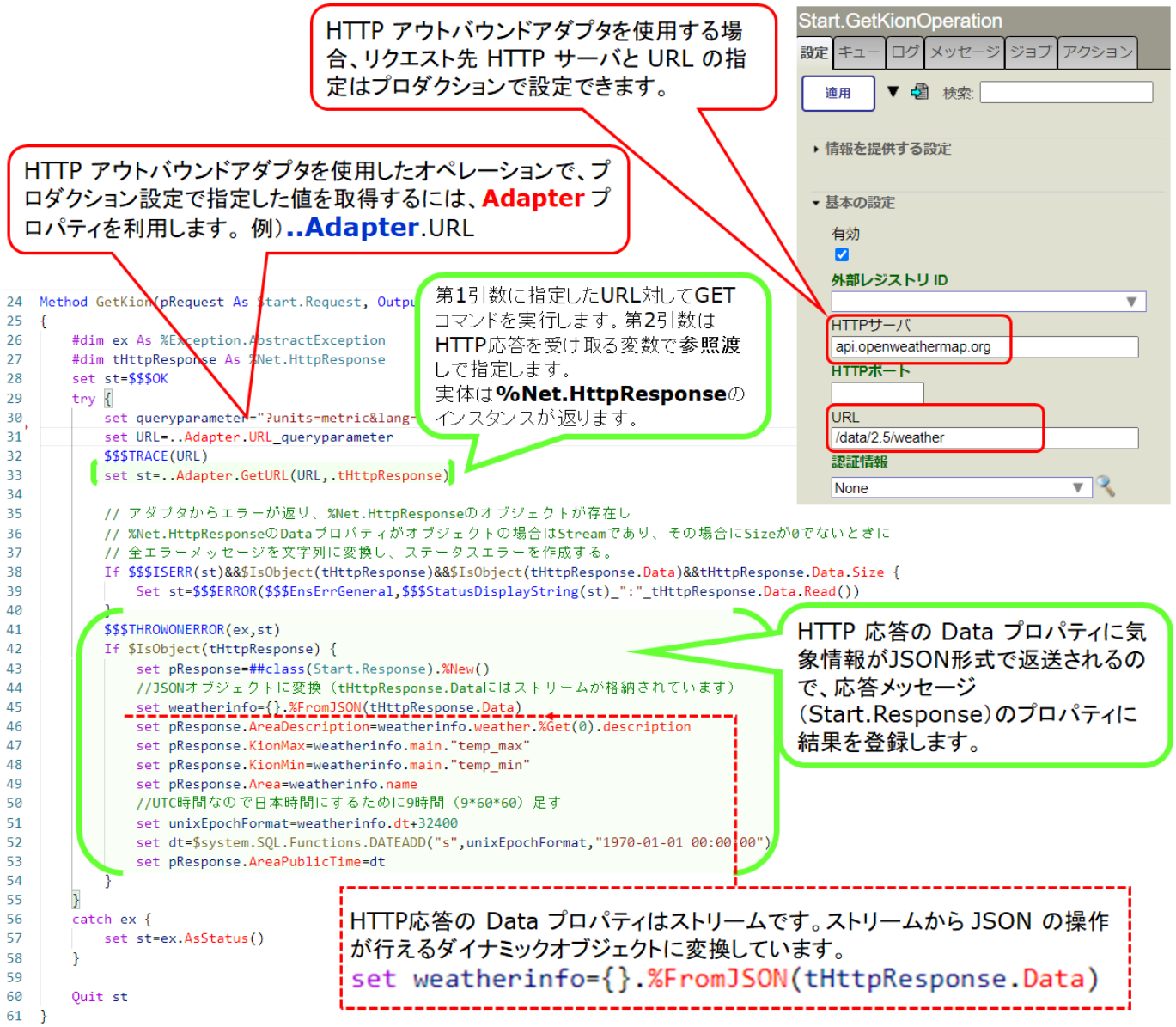
GetKion() メソッドでは、入力情報として渡される要求メッセージの Area
プロパティから**都市名**が取得できます。
この**都市名**を外部の Web API が公開している URL
のパラメータに設定し呼び出せば、**気象情報**が取得できます。

なお、HTTP サーバや URL
の設定は管理ポータルのプロダクション画面で設定します。設定内容を取得するには、HTTP
アウトバウンドアダプタから提供される **Adapter** プロパティを使用します。

例） URLの指定の場合は、 **..Adapter.URL**

REST 用ビジネス・オペレーションが提供する GetURL()
メソッドを使用して、外部サイトを呼び出します。第1引数に実行したい
URL（＝**都市名**など必要なパラメータに指定した URL）、第2引数に参照渡しで引数を用意し HTTP
応答を受け取ります。

HTTP 応答にはJSON 形式で気象情報が格納されるので、応答メッセージ（＝pResponse）に情報を登録したらオペレーションは完成です。




```
24  Method GetKion(pRequest As Start.Request, Outpu
25  {
26      #dim ex As %Exception.AbstractException
27      #dim tHttpResponse As %Net.HttpResponse
28      set st=$$$OK
29      try {
30          set queryparameter="?units=metric&lang=
31          set URL=..Adapter.URL_queryparameter
32          $$$TRACE(URL)
33          set st=..Adapter.GetURL(URL,.tHttpResponse)
34
35          // アダプタからエラーが返り、%Net.HttpResponseのオブジェクトが存在し
36          // %Net.HttpResponseのDataプロパティがオブジェクトの場合はStreamであり、その場合にSizeが0でないときに
37          // 全エラーメッセージを文字列に変換し、ステータスエラーを作成する。
38          If $$$ISERR(st)&&$IsObject(tHttpResponse)&&$IsObject(tHttpResponse.Data)&&tHttpResponse.Data.Size {
39              Set st=$$$ERROR($$$EnsErrGeneral,$$$StatusDisplayString(st)_":"_tHttpResponse.Data.Read())
40          }
41          $$$THROWONERROR(ex,st)
42          If $IsObject(tHttpResponse) {
43              set pResponse=##class(Start.Response).%New()
44              //JSONオブジェクトに変換（tHttpResponse.Datalにはストリームが格納されています）
45              set weatherinfo={}.%FromJSON(tHttpResponse.Data)
46              set pResponse.AreaDescription=weatherinfo.weather.%Get(0).description
47              set pResponse.KionMax=weatherinfo.main."temp_max"
48              set pResponse.KionMin=weatherinfo.main."temp_min"
49              set pResponse.Area=weatherinfo.name
50              //UTC時間なので日本時間にするために9時間（9*60*60）足す
51              set unixEpochFormat=weatherinfo.dt+32400
52              set dt=$system.SQL.Functions.DATEADD("s",unixEpochFormat,"1970-01-01 00:00:00")
53              set pResponse.AreaPublicTime=dt
54          }
55      }
56      catch ex {
57          set st=ex.AsStatus()
58      }
59
60      Quit st
61  }
```

作成したメソッドの第 2 引数には、参照渡しで応答メッセージクラス名が指定されています。

```
Method GetKion(pRequest As Start.Request, Output pResponse As
Start.Response) As %Status
```

呼び出し元に応答メッセージを返す場合は、応答メッセージのインスタンスを生成し、第 2 引数の変数（***pResponse***）に格納し、プロパティに必要な情報を設定します。

```
set pResponse=##class(Start.Response).%New()
set weatherinfo={}.%FromJSON(tHttpResponse.Data)
set pResponse.AreaDescription=weatherinfo.weather.%Get(0).description
set pResponse.KionMax=weatherinfo.main."temp_max"
set pResponse.KionMin=weatherinfo.main."temp_min"
set pResponse.Area=weatherinfo.name
//UTC????????????????9???9*60*60???set unixEpochFormat=weatherinfo.dt+32400
set dt=$system.SQL.Functions.DATEADD("s",unixEpochFormat,"1970-01-01 00:00:00")
```

```
set pResponse.AreaPublicTime=dt
```

外部サイトからのHTTP応答がJSON形式で返送されるため、HTTP
応答で取得できたストリームを利用して、IRIS 内の JSON
操作に便利なダイナミックオブジェクトに変換しています。

```
set weatherinfo={}.%FromJSON(tHttpResponse.Data)
```

返送される JSON 文字列の例は以下の通りです。

{
"coord": {
"lon": 138.1,
"lat": 36.64
},
"weather": [
{
"id": 801,
"main": "Clouds",
"description": "薄い雲",
"icon": "02d"
}
],
"base": "stations",
"main": {
"temp": 10.12,
"feelslike": 9.1,
"tempmin": 8.33,
"tempmax": 11.67,
"pressure": 1013,
"humidity": 86
},
"visibility": 10000,
"wind": {
"speed": 0.76,
"deg": 233
},
"clouds": {
"all": 24
},
"dt": 1603584269,
"sys": {
"type": 3,
"id": 19237,
"country": "JP",
"sunrise": 1603573442,
"sunset": 1603612743
},
"timezone": 32400,
"id": 1856215,
"name": "Nagano",
"cod": 200
}

最高気温、最低気温、天気 は以下のように取得できます。

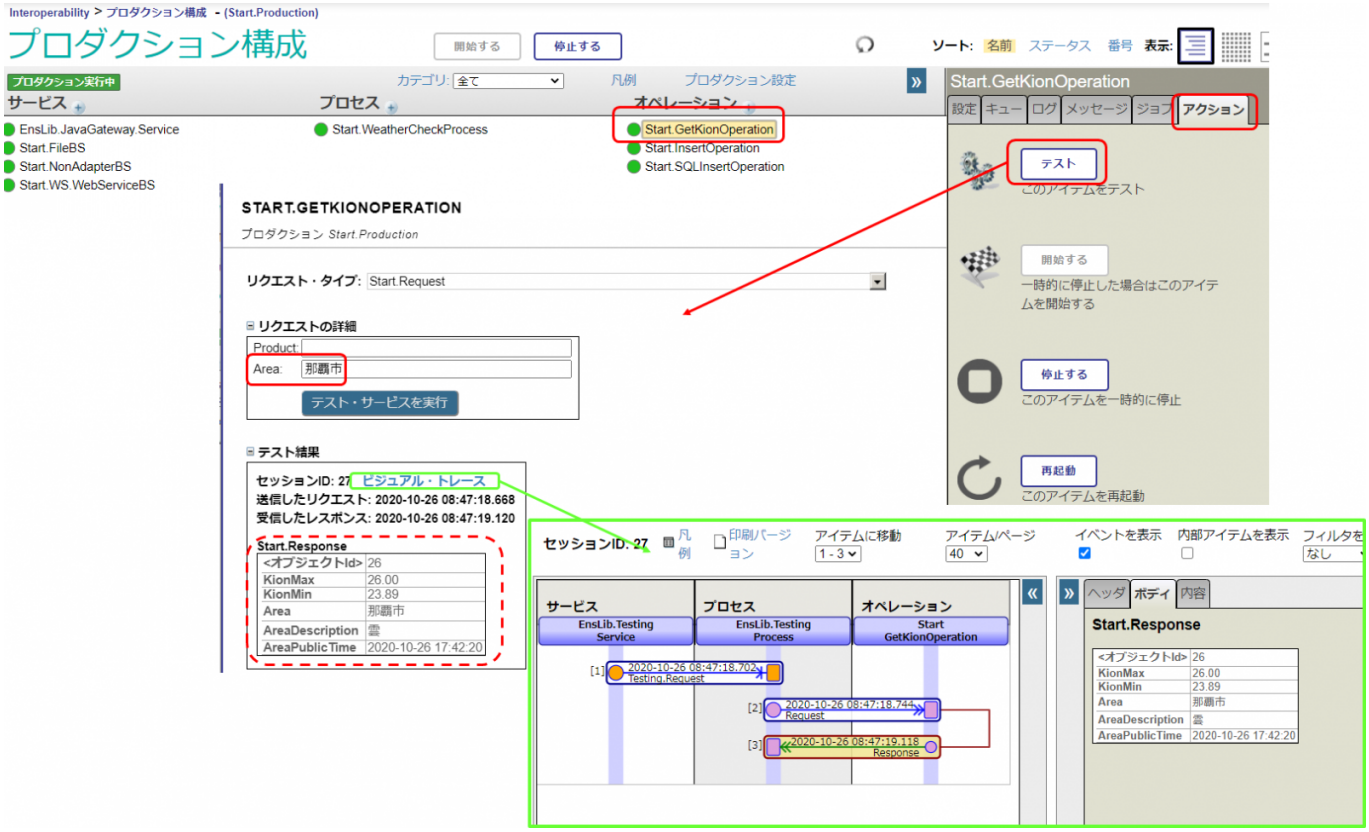
```
set pResponse.KionMax=weatherinfo.main."temp_max"
set pResponse.KionMin=weatherinfo.main."temp_min"
set pResponse.AreaDescription=weatherinfo.weather.%Get(0).description
```

IRIS 内の JSON 操作について確認されたい場合は、こちらの記事やドキュメントもご参照ください。

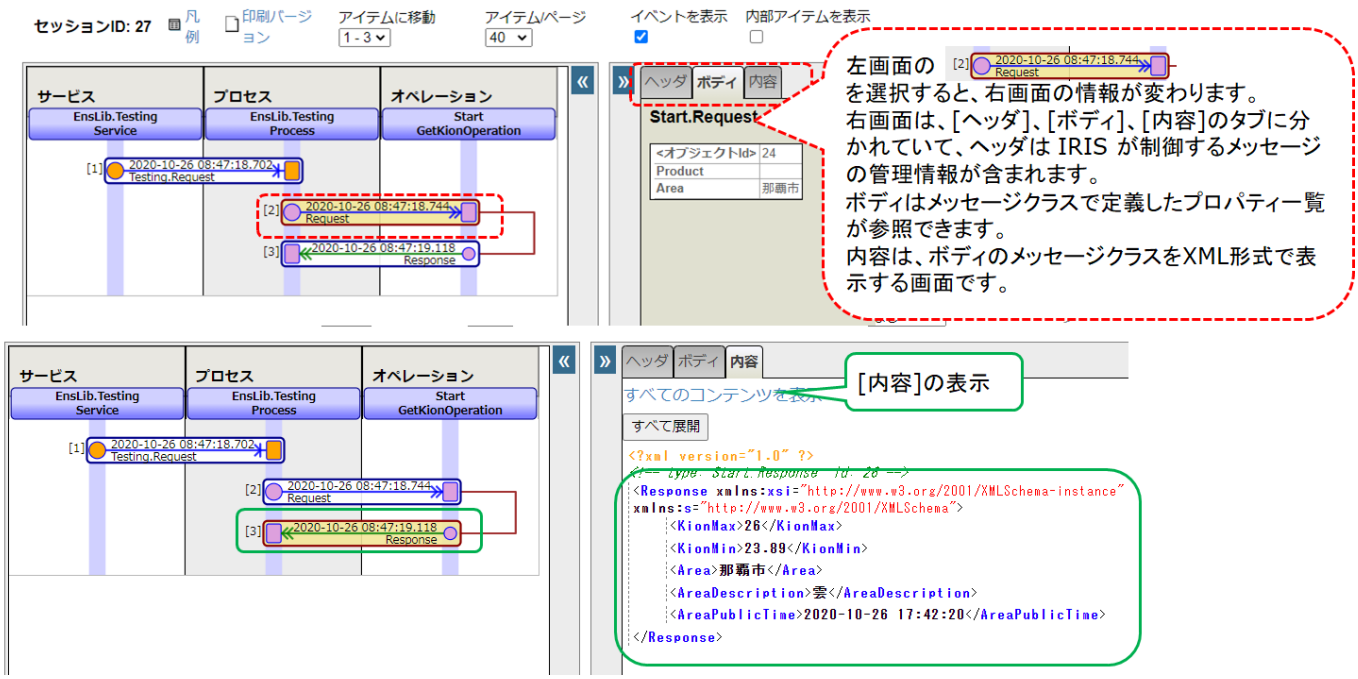
それでは、天気情報をちゃんと取得できるかどうか、プロダクションのテストツールを使って確認しましょう。

プロダクション画面を開き（**管理ポータル > [Interoperability] > [構成] > [プロダクション]**）Start.GetKionOperation をクリックし、**[アクション]タブ**の**「テスト」ボタン**をクリックします。

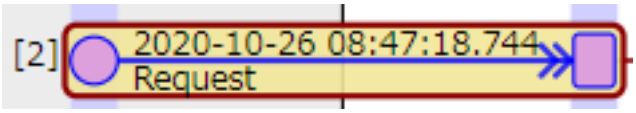
Area に **都市名**（那覇市、札幌市、長野市、新宿区など）を指定して、**「テストサービスを実行」**ボタンをクリックします。
下にテスト結果が表示され、最高気温、最低気温、天気が表示されるのが確認できます。



続いて、トレース画面の使い方をご紹介します。



左画面で



などの横長の長方形を選択すると、右画面の情報が変わります。

システム統合の流れで送受信したメッセージは自動的にデータベースに保存されます。メッセージのトレース画面を使用すると、時系列でどのようなメッセージがどのコンポーネントに渡されたか、応答があったかどうか、など詳細に確認することができます。

また、エラーが発生した場合は、

「〇から△のコンポーネントに□のメッセージを 送付中／受信中／受信後 にエラー発生」

がわかるように、エラーが発生した場所に赤いマークがつきます。もちろん、トレース以外にもイベントログページも用意しています。
（**管理ポータル > [Interoperability] > [表示] > [イベントログ]**）

続いて、データベースへ更新依頼を行うオペレーションを確認してみましょう。

1)-2 データベースへ更新依頼を行うビジネス・オペレーション

サンプルには Start.SQLInsertOperation と Start.InsertOperation の2種類のオペレーションの用意があります。

それぞれ、データベースの更新依頼を行うオペレーションですが、Start.SQLInsertOperation
は、SQLアウトバウンドアダプタを利用していますが、Start.InsertOperation は、アダプタを使用していません。

両者の違いは、

SQL アウトバウンドアダプタを使用するオペレーションは、ODBC／JDBC接続でのアクセスを前提としているため、データベースの接続先をプロダクションの設定で切り替えることができます。

アダプタを使用しないオペレーションでは、DB更新対象はプロダクション構成から見える範囲のデータベースで、接続先の切り替えが発生しないことが前提となります。

IRIS にはデータベースが含まれているため、システム統合中の任意データの格納場所に IRIS 内データベースを使用できますが、数年後、システム構成が何らかの理由で変更となり、異なるサーバ上のデータベースへ接続する必要が出てきた場合、アダプタを使用しないオペレーションをそのまま継続利用することはできません。

反対に、SQL アウトバウンドアダプタを使用するオペレーションは、接続先指定を変えることで処理内容に変更が無ければそのまま運用できます（実行する SQL
文に問題がなければ、異なる製品のデータベースにも接続できます）。

システム統合の中では、外部システムの都合で接続情報が変更となるケースも考えられるため、変更に柔軟に対応できるような設計にしておくことが重要です。そのため、外部接続に対応するコンポーネントは **疎結合**
で作成することを推奨しています。

とはいえ、将来構成に変更がなければ、わざわざ ODBC／JDBC 接続を使用しなくても IRIS
内データベースにアクセスできますので、利用用途に合わせアダプタを使用する／しないをご選択いただけます。

それでは順番に、アダプタを使用する Start.SQLInsertOperation のコードを確認しましょう。

サンプルが使用しているアダプタは SQL アウトバウンドアダプタで、データベースに対して SQL文
の実行を依頼できるアダプタです。
アダプタ
が異なると提供さ
れるメソッドも異なります。アダプタから提供されるメソッド詳細についてはドキュメントをご参照ください。

```
1  Class Start.SQLInsertOperation Extends Ens.BusinessOperation
2  {
3
4  Parameter ADAPTER = "EnsLib.SQL.OutboundAdapter";
5
6  Property Adapter As EnsLib.SQL.OutboundAdapter;
7
8  Parameter INVOCATION = "Queue";
9
10 Method Insert(pRequest As Start.InsertRequest) Output pResponse As Ens.Response) As %Status
11 {
12     set st=$$$OK
13     #dim ex As %Exception.AbstractException
14     try {
15         set sql="insert into Start.WeatherHistory (Area,AreaDescription,AreaPublicTime,KionMin,KionMax,Product) values(?,?,?,?,?,?)"
16         set st=..Adapter.ExecuteUpdate(.rows,sql,pRequest.WeatherInfo.Area,pRequest.WeatherInfo.AreaDescription,pRequest.WeatherInfo.AreaPublicTime,
17                                        pRequest.WeatherInfo.KionMin,pRequest.WeatherInfo.KionMax,pRequest.Product)
18         $$$THROWONERROR(ex,st)
19     }
20     catch ex {
21         set st=ex.AsStatus()
22     }
23     quit st
24 }
25
26 XData MessageMap
27 {
28 <MapItems>
29     <MapItem MessageType="Start.InsertRequest">
30         <Method>Insert</Method>
31     </MapItem>
32 </MapItems>
33 }
```

アダプタ経由でSQLの更新文を依頼する場合
は、..Adapter.ExecuteUpdate()メソッドを実行します。
第1引数：影響受けた行数
第2引数：実行するSQL文
第3引数：SQL文の引数

要求メッセージStart.InsertRequestが入力されると
Insert()メソッドが起動します。

続いて、アダプタを使用しない Start.InsertOperation のコードを確認しましょう。

アダプタを使用する／しないに関わらず、オペレーションのメッセージマップとメソッド定義は必要です。アダプタを使用しない場合は、アダプタ用 Paramter と Property の定義は不要です。

```
Class Start.InsertOperation Extends Ens.BusinessOperation
{
Parameter INVOCATION = "Queue";
Method Insert(pRequest As Start.InsertRequest, Output pResponse As Ens.Response) As %Status
{
    set st=$$$OK
    #dim ex As %Exception.AbstractException
    try {
        // /* objectで更新する場合 */
        /*
        set obj=##class(Start.WeatherHistory).%New()
        set obj.Product=pRequest.Product
        set obj.Area=pRequest.WeatherInfo.Area
        set obj.AreaDescription=pRequest.WeatherInfo.AreaDescription
        set obj.AreaPuclicTime=pRequest.WeatherInfo.AreaPublicTime
        set obj.KionMin=pRequest.WeatherInfo.KionMin
        set obj.KionMax=pRequest.WeatherInfo.KionMax
        $$$THROWONERROR(ex,obj.%Save())
        */
        // /* SQLで更新する場合*/
        set cols(2)=pRequest.WeatherInfo.Area
        set cols(3)=pRequest.WeatherInfo.AreaDescription
        set cols(4)=pRequest.WeatherInfo.AreaPublicTime
        set cols(6)=pRequest.WeatherInfo.KionMax
        set cols(7)=pRequest.WeatherInfo.KionMin
        set cols(8)=pRequest.Product
        &sql(insert into Start.WeatherHistory values :cols())
        if SQLCODE<0 {
            throw ##class(%Exception.SQL).CreateFromSQLCODE(SQLCODE,%msg)
        }
    }
    catch ex {
        set st=ex.AsStatus()
    }
    Quit st
}
XData MessageMap
{
<MapItems>
    <MapItem MessageType="Start.InsertRequest">
        <Method>Insert</Method>
    </MapItem>
</MapItems>
}

}

```

アダプタを使用しないビジネス・オペレーション： Start.InsertOperation では、ObjectScript を利用してSQLを実行しています（コメント文はオブジェクト操作での更新処理です）。

更新対象データベースが IRIS から切り離されることが無ければ十分な実装といえます。

アダプタを使用したオペレーションは、接続先に処理を依頼するための専用メソッドが提供されることがわかりました。また、アダプタを使用しないオペレーションも作成可能で自由にコードを記述できることが確認できました。

次の投稿では、天気情報取得用オペレーションとデータベース更新用オペレーションを順番を守って呼び出しを行うビジネス・プロセスの作り方について解説します。

#ビジネスオペレーション #初心者 #相互運用性 #InterSystems IRIS #InterSystems IRIS for Health

ソースURL:
https://jp.community.intersystems.com/post/%E3%80%90%E3%81%AF%E3%81%98%E3%82%81%E3%81%A6%E3%81%AEintersystems-iris%E3%80%91interoperability%EF%BC%88%E7%9B%B8%E4%BA%92%E9%81%8B%E7%94%A8%E6%80%A7%EF%BC%89%EF%BC%9A%E3%82%B3%E3%83%B3%E3%83%9D%E3%83%BC%E3%83%8D%E3%83%B3%E3%83%88%E3%81%AE%E4%BD%9C%E6%88%90%EF%BC%88%E3%83%93%E3%82%B8%E3%83%8D%E3%82%B9%E3%83%BB%E3%82%AA%E3%83%9A%E3%83%AC%E3%83%BC%E3%82%B7%E3%83%A7%E3%83%B3%EF%BC%89