
WinActorにおける API連携

サービス連携の活用方法

API連携の概要

API連携とは？

■そもそもAPIって何？

Application Programming Interfaceの略称。

あるサービスがアプリケーションのために準備した、サービスを利用できる**窓口**のようなものです。

■どうやって使うもの？

APIを使う方法はいくつかありますが、今回はHTTPを利用したAPIについて説明します。

HTTPを利用したAPIは、特に**REST API**と呼ばれます。

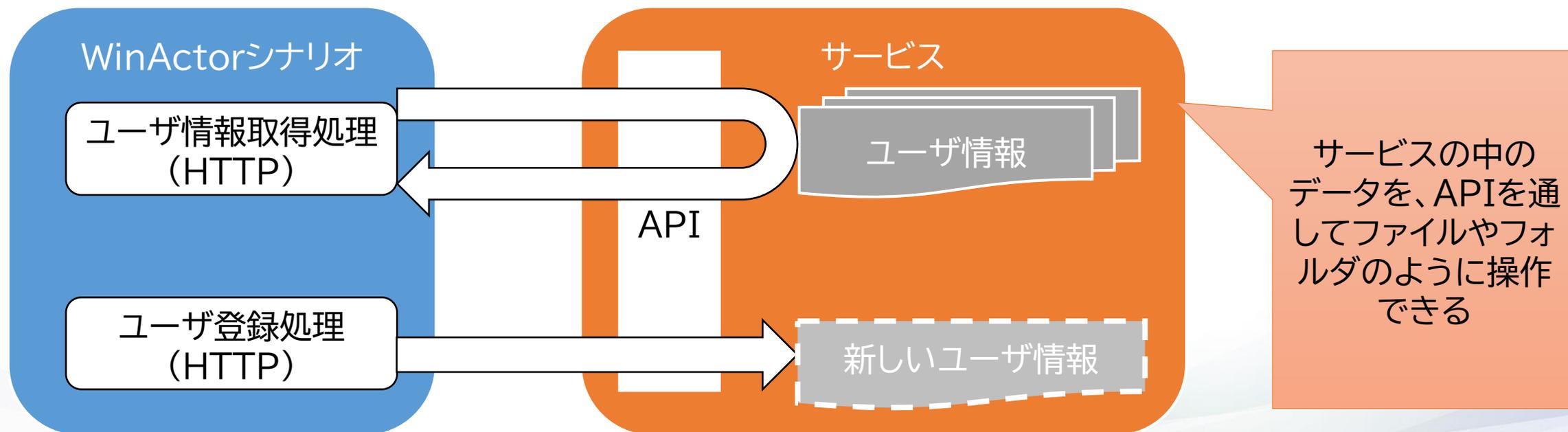
RESTとは設計思想の名称で、平たく言えば「**リソースごとにURLをもつ設計**」のことを指します。

パソコン上のファイルパスと同じような構造と考えてみてください。

■つまりAPI連携って何？

HTTPを使って、特定のサービス上のファイルや情報を読み込んだり、アップロードしたりして操作することと定義します。

WinActor的な表現で言えば、HTTPライブラリを利用したシナリオのことになります。



■なぜAPI連携？

API連携を使う理由は、とにかく**シナリオの動作が安定する**ことに尽きます。

サービスと連携する方法はいくつかありますが、API連携でシナリオを作成してしまえば、その後サービスの画面が更新されたり、バージョンアップしたりしてもシナリオを更新することなく使い続けることができます。

■サービス連携の種類

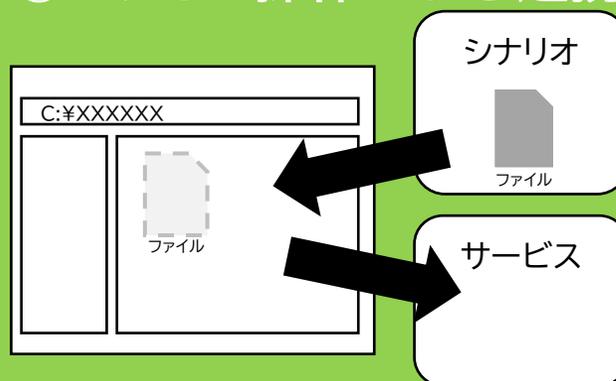
そもそも、WinActorで他のサービスと連携する方法はいくつか存在します。

①画面操作による連携



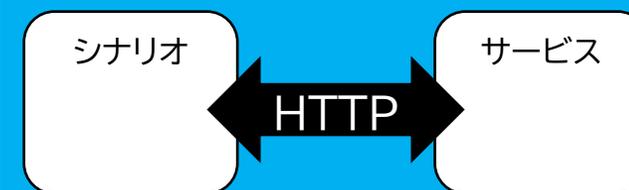
ブラウザ操作ライブラリでウェブページ上のボタンやフォームを操作する方法

②ファイル操作による連携



シナリオなどで作成したファイルを所定の場所に配置することで連携する方法

③APIによる連携

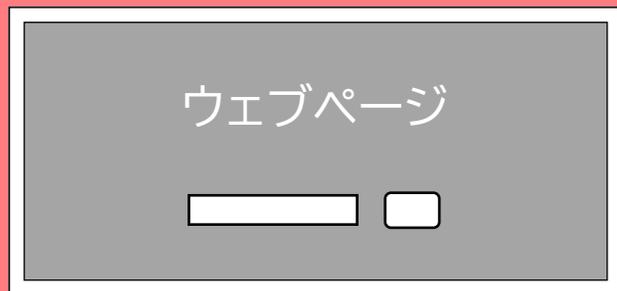


シナリオとサービスをHTTPで連携する方法

■①画面操作による連携

画面操作による連携は、普段から利用している画面をシナリオで実現するので直感的にわかりやすく、また操作の記録機能などである程度自動でも作れるというお手軽さが魅力です。

①画面操作による連携



ブラウザ操作ライブラリでウェブページ上のボタンやフォームを操作する方法

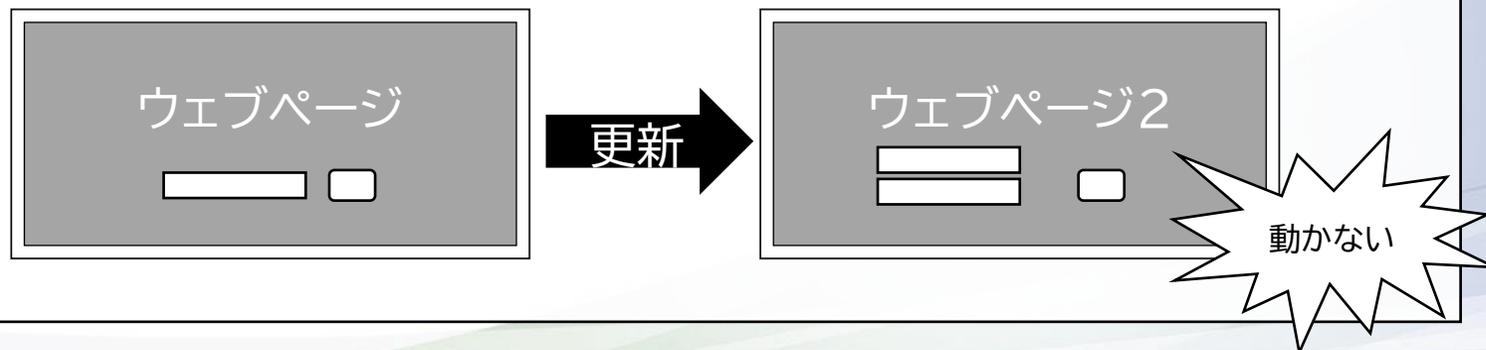
■メリット

わかりやすい！

人間の操作を再現するだけでOK

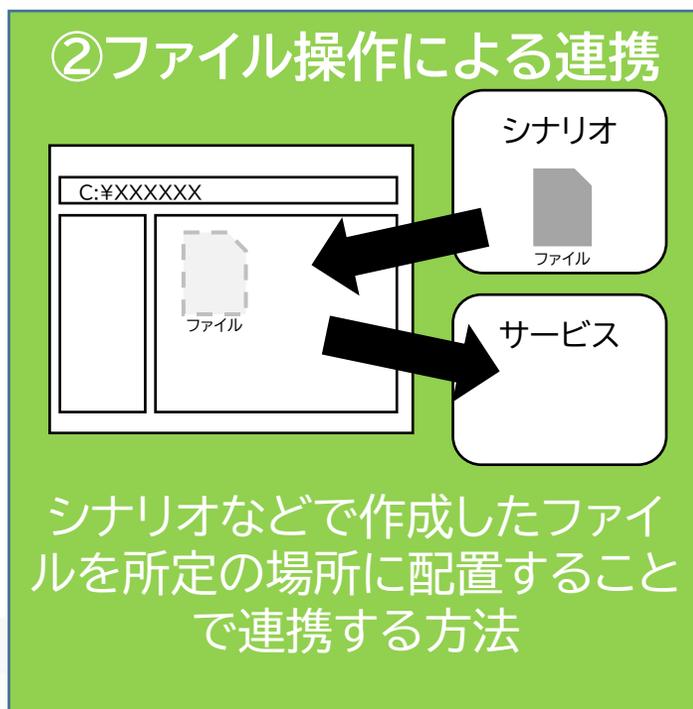
■デメリット

画面が新しくなると使えなくなることがある…



■②ファイル操作による連携

BoxやGoogleDriveなど、主にファイル共有サービスなどで使われる連携方法です。WinActorシナリオとの親和性が高く、シナリオ化し易いです。



■メリット

シナリオ化が簡単！

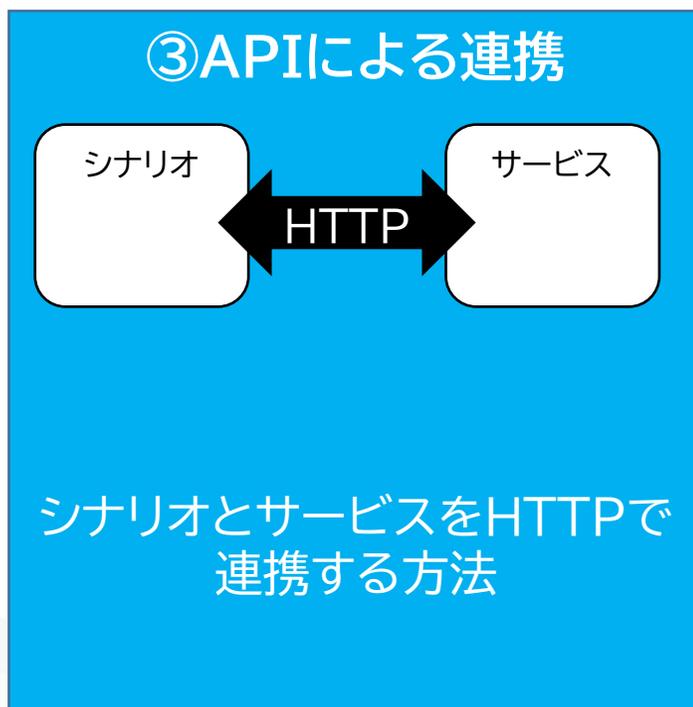
ファイル操作は得意分野

■デメリット

利用できるサービスがかなり限定される…

■③APIによる連携

API連携は、①②のような人間が操作するやり方を真似するのではなく、アプリケーションのためのやり方を実現することで連携するため、とにかく動作が安定します。



■メリット

とにかく安定！

画面やバージョンが新しくなってもちゃんと動く

■デメリット

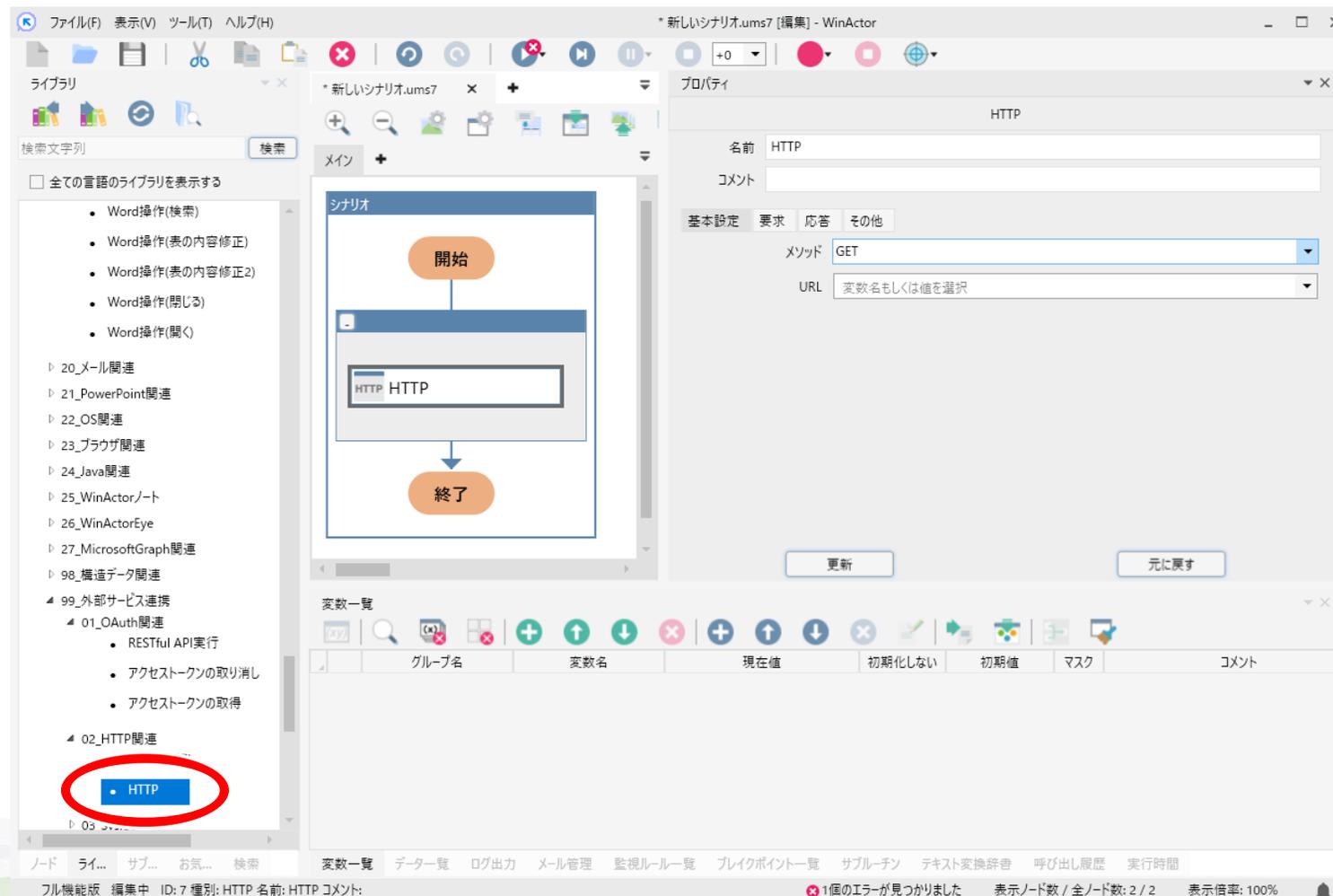
ちょっとむずかしい…

API連携を試す

■まずはHTTPライブラリをつかってみよう

HTTPライブラリは「99_外部サービス連携」→「02_HTTP関連」の中にあります。

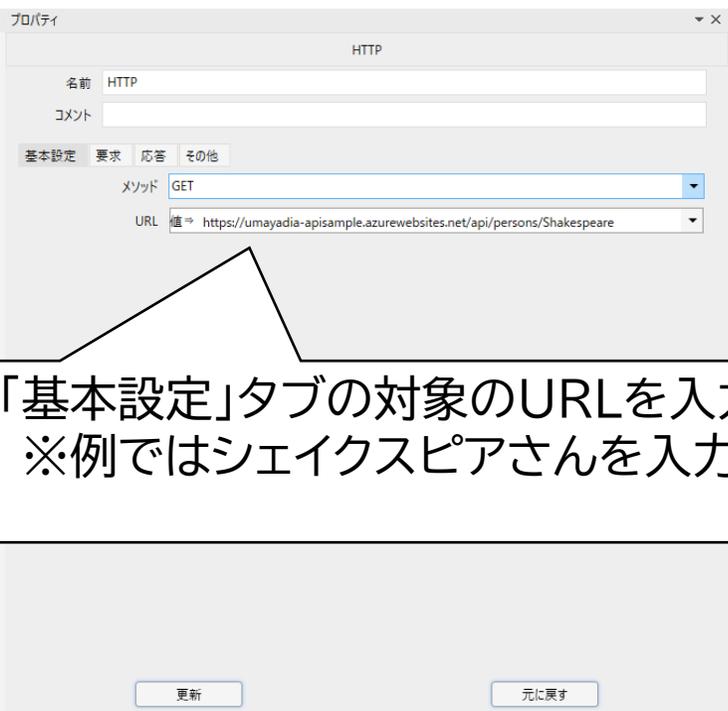
HTTPライブラリは、シナリオからサービスに送る「要求」と、要求に対してサービスが返してくれる「応答」を設定することで動かします。



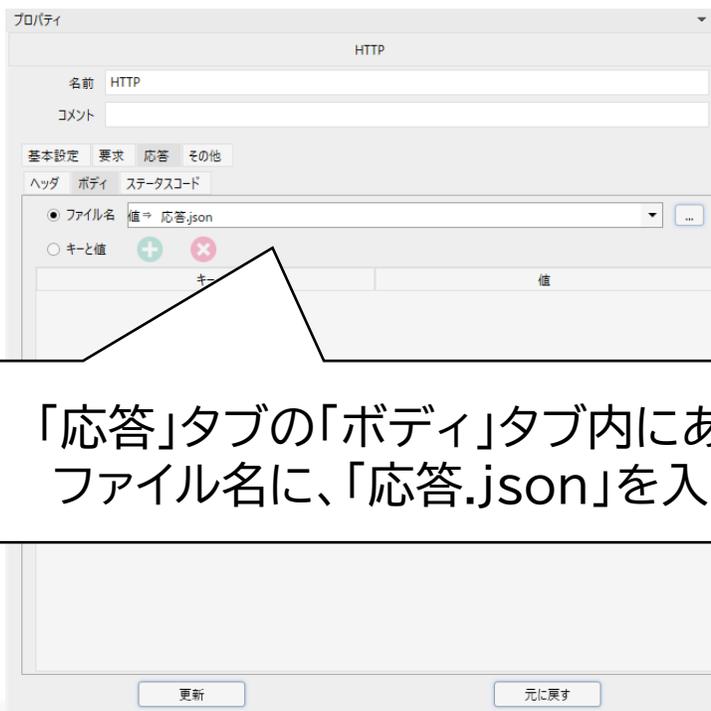
■要求と応答を実践してみる

今回はお試しということで、「**すぐ呼び出し可能なWebAPIのサンプル**」※を利用させていただきます。

HTTPライブラリのプロパティに、それぞれ以下の設定を行います。



「基本設定」タブの対象のURLを入力
※例ではシェイクスピアさんを入力



「応答」タブの「ボディ」タブ内にある
ファイル名に、「応答.json」を入力

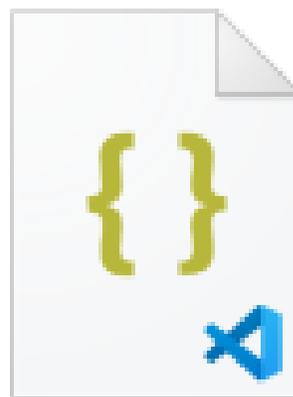
※すぐ呼び出し可能なWebAPIのサンプル <https://www.umayadia.com/Note/Note028WebAPISample.htm>

※シェイクスピアさんのユーザ情報アドレス <https://umayadia-apisample.azurewebsites.net/api/persons/Shakespeare>

■準備完了！

さっそくシナリオ実行してみましょう。

実行すると、シナリオファイルのある場所に「**応答.json**」というファイルができています。



応答.json

JSON概要

■応答の内容は？

「応答.json」をテキストエディタなどで開いてみると、次のような内容になっています。

```
{"success":true,"data":{"name":"Shakespeare","note":"Hamlet","age":13,"registerDate":"1564-04-26T20:21:00"}}
```

これは「**JSON形式**」と呼ばれるフォーマットです。APIの応答でよく使われる、アプリケーションが読みやすいように加工されたデータで、人間が読むにはちょっとわかりにくいです。

上記の内容をわかりやすく書き換えるなら、

「要求は成功しました。このデータに登録されている情報は次の通りです。名前はShakespeare、メモにはHamletと書いてあり、年齢は13、登録日は1564年4月26日の20時21分です。」

となります。

■少しだけJSONの解説

JSON形式のデータを加工するにあたり、少しだけJSONの構造について解説します。

JSONの構造は、データの構造をまとめる「オブジェクト」と呼ばれるデータのかたまりと、データそのものを記録する「キー」「値」のセットをカンマ(,)でつないで組み合わせでつくりあげます。

「オブジェクト」は{}で囲まれた部分のことを指し、「キー」「値」のセットは”キー”:”値”のように書かれた部分を指します。

```
{ "success": true, "data": { "name": "Shakespeare", "note": "Hamlet", "age": 13, "registerDate": "1564-04-26T20:21:00" } }
```

■少しだけJSONの解説

JSON形式のデータを加工するにあたり、少しだけJSONの構造について解説します。

JSONの構造は、データの構造をまとめる「オブジェクト」と呼ばれるデータのかたまりと、データそのものを記録する「キー」「値」のセットをカンマ(,)でつないで組み合わせてつくりあげます。

「オブジェクト」は{}で囲まれた部分のことを指し、「キー」「値」のセットは”キー”:”値”のように書かれた部分を指します。

```
{"success":true,"data":{"name":"Shakespeare","note":"Hamlet","age":13,"registerDate":"1564-04-26T20:21:00"}}
```

最初の{と最後の}で囲まれた、大きなオブジェクト。
オブジェクトの中にはいくつかの「キー」「値」の
セットがある。

■少しだけJSONの解説

JSON形式のデータを加工するにあたり、少しだけJSONの構造について解説します。

JSONの構造は、データの構造をまとめる「オブジェクト」と呼ばれるデータのかたまりと、データそのものを記録する「キー」「値」のセットをカンマ(,)でつないで組み合わせでつくりあげます。

「オブジェクト」は{}で囲まれた部分のことを指し、「キー」「値」のセットは”キー”:”値”のように書かれた部分を指します。

```
{"success":true, "data":{"name":"Shakespeare", "note":"Hamlet", "age":13, "registerDate":"1564-04-26T20:21:00"}}
```

successというキーと、trueという値。このセットが1つのデータになる。また、数字や真偽値は””で囲まない。

■少しだけJSONの解説

JSON形式のデータを加工するにあたり、少しだけJSONの構造について解説します。

JSONの構造は、データの構造をまとめる「オブジェクト」と呼ばれるデータのかたまりと、データそのものを記録する「キー」「値」のセットをカンマ(,)でつないで組み合わせでつくりあげます。

「オブジェクト」は{}で囲まれた部分のことを指し、「キー」「値」のセットは”キー”:”値”のように書かれた部分を指します。

```
{ "success": true, "data": { "name": "Shakespeare", "note": "Hamlet", "age": 13, "registerDate": "1564-04-26T20:21:00" } }
```

dataというキーと、{}で囲まれたオブジェクト。
dataというキーに対応する値は1つではなく、
複数あることになる。

■少しだけJSONの解説

JSON形式のデータを加工するにあたり、少しだけJSONの構造について解説します。

JSONの構造は、データの構造をまとめる「オブジェクト」と呼ばれるデータのかたまりと、データそのものを記録する「キー」「値」のセットをカンマ(,)でつないで組み合わせでつくりあげます。

「オブジェクト」は{}で囲まれた部分のことを指し、「キー」「値」のセットは”キー”:”値”のように書かれた部分を指します。

```
{ "success": true, "data": { "name": "Shakespeare", "note": "Hamlet", "age": 13, "registerDate": "1564-04-26T20:21:00" } }
```

nameというキーと、Shakespeare
という値。

■少しだけJSONの解説

JSON形式のデータを加工するにあたり、少しだけJSONの構造について解説します。

JSONの構造は、データの構造をまとめる「オブジェクト」と呼ばれるデータのかたまりと、データそのものを記録する「キー」「値」のセットをカンマ(,)でつないで組み合わせでつくりあげます。

「オブジェクト」は{}で囲まれた部分のことを指し、「キー」「値」のセットは”キー”:”値”のように書かれた部分を指します。

```
{ "success": true, "data": { "name": "Shakespeare", "note": "Hamlet", "age": 13, "registerDate": "1564-04-26T20:21:00" } }
```

noteというキーと、Hamletという値。

■少しだけJSONの解説

JSON形式のデータを加工するにあたり、少しだけJSONの構造について解説します。

JSONの構造は、データの構造をまとめる「オブジェクト」と呼ばれるデータのかたまりと、データそのものを記録する「キー」「値」のセットをカンマ(,)でつないで組み合わせでつくりあげます。

「オブジェクト」は{}で囲まれた部分のことを指し、「キー」「値」のセットは”キー”:”値”のように書かれた部分を指します。

```
{ "success": true, "data": { "name": "Shakespeare", "note": "Hamlet", "age": 13, "registerDate": "1564-04-26T20:21:00" } }
```

ageというキーと、13という値。

■少しだけJSONの解説

JSON形式のデータを加工するにあたり、少しだけJSONの構造について解説します。

JSONの構造は、データの構造をまとめる「オブジェクト」と呼ばれるデータのかたまりと、データそのものを記録する「キー」「値」のセットをカンマ(,)でつないで組み合わせでつくりあげます。

「オブジェクト」は{}で囲まれた部分のことを指し、「キー」「値」のセットは”キー”:”値”のように書かれた部分を指します。

```
{ "success": true, "data": { "name": "Shakespeare", "note": "Hamlet", "age": 13, "registerDate": "1564-04-26T20:21:00" } }
```

registerDateというキーと、
1564-04-26T20:21:00という値。

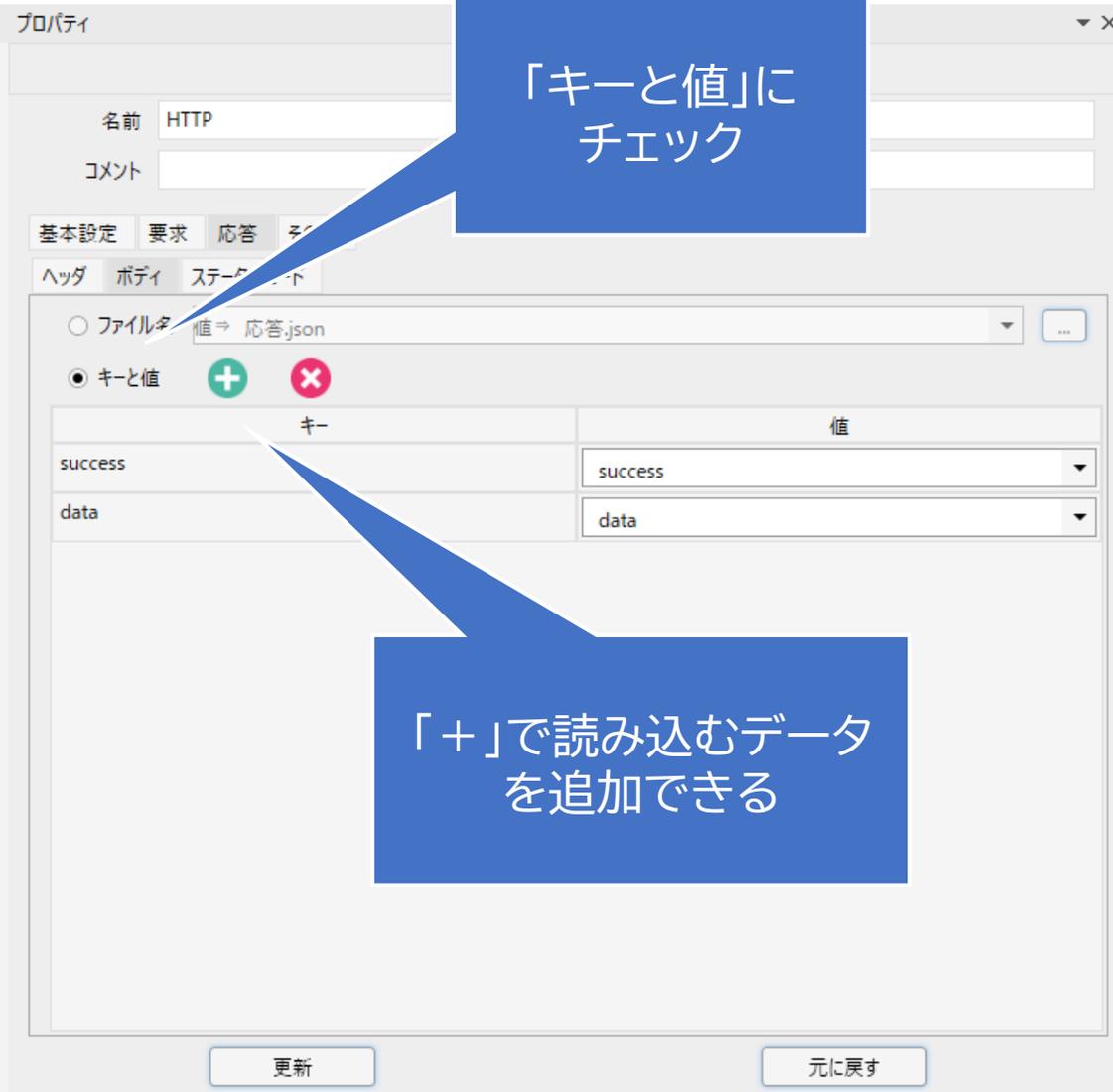
HTTPライブラリとJSON応答

■応答を解析してシナリオ化してみよう

JSON形式について理解したところで、今度はシナリオ内でJSONを解析してみます。

HTTPライブラリの「応答」タブ内を更新し、変数に応答の内容を出力するようにします。

さきほど解析した応答.jsonの内容から、
応答オブジェクトの中にある2つのキー
「success」と「data」をそれぞれ変数に読み込むよう設定します。



「キーと値」にチェック

キー	値
success	success
data	data

「+」で読み込むデータを追加できる

更新 元に戻す

■変数に読み込まれたJSONの解析

HTTPノードで変数にJSONを読み込んだ場合、右のように「キー」に対応する「値」の中身が変数の現在値として読み込まれます。

dataには{}で囲まれたJSONオブジェクトが読み込まれていることがわかります。

変数名	現在値
success	true
data	{ "name": "Shakespeare", "note": "Hamlet", "age": 13, "registerDate": "1564-04-26T20:21:00" }

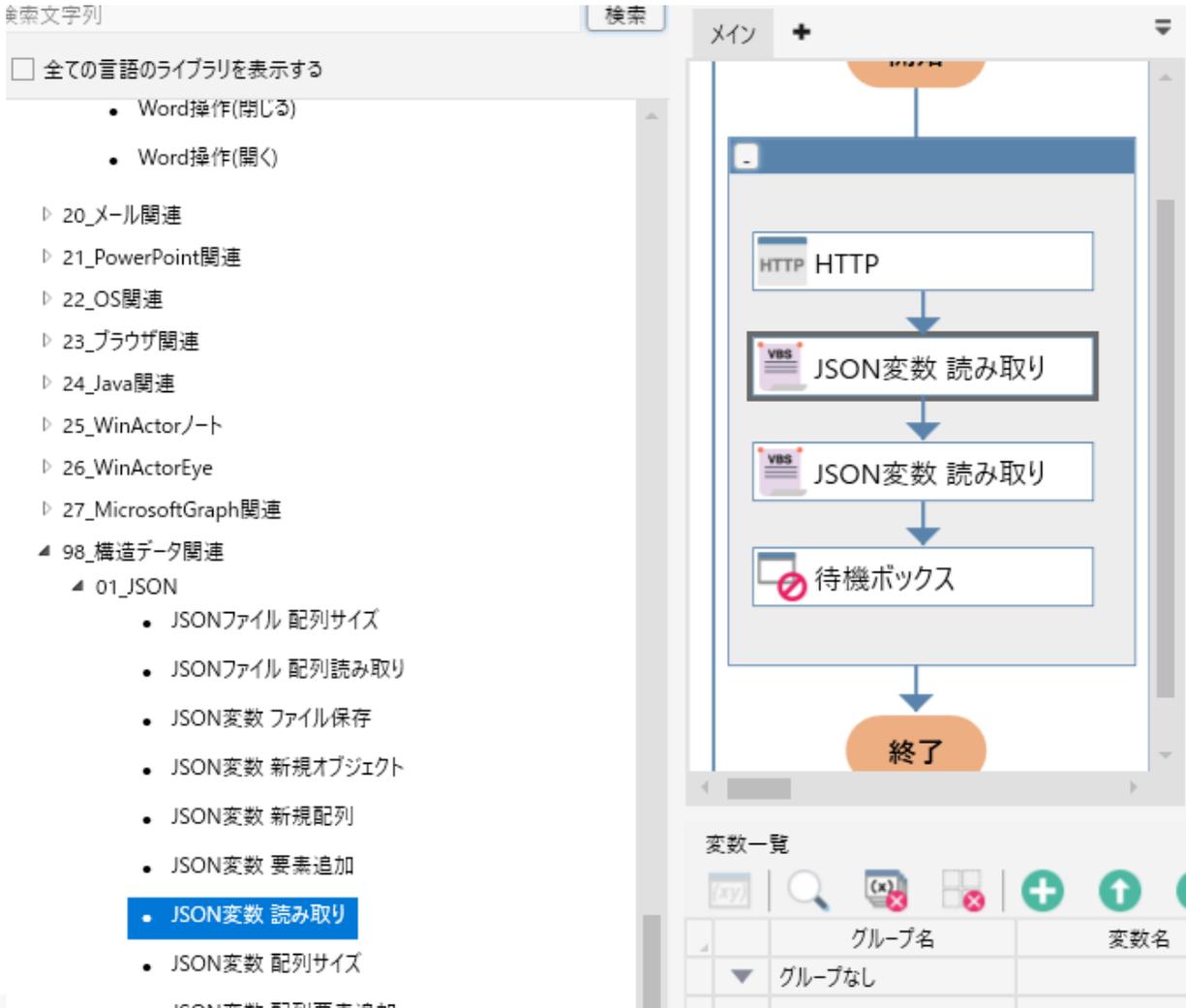
応答を解析する(2/5)

■変数に読み込まれたJSONの解析

dataの「値」のようなJSONオブジェクトを解析するために、JSON解析専用のライブラリを利用します。

今回は「98_構造データ関連」→「01_JSON」の中にある、「JSON変数 読み取り」ライブラリを利用します。

※より詳細なオプションをつけたい方は、「98_構造データ関連」→「02_JSON」の中にある、「JSONオブジェクト 値の取得」ライブラリを利用してください。



The screenshot displays a software interface for response analysis. On the left, a search bar is at the top, followed by a checkbox '全ての言語のライブラリを表示する'. Below is a tree view of libraries:

- Word操作(閉じる)
- Word操作(開く)
- 20_メール関連
- 21_PowerPoint関連
- 22_OS関連
- 23_ブラウザ関連
- 24_Java関連
- 25_WinActorノート
- 26_WinActorEye
- 27_MicrosoftGraph関連
- 98_構造データ関連
 - 01_JSON
 - JSONファイル 配列サイズ
 - JSONファイル 配列読み取り
 - JSON変数 ファイル保存
 - JSON変数 新規オブジェクト
 - JSON変数 新規配列
 - JSON変数 要素追加
 - JSON変数 読み取り** (highlighted)
 - JSON変数 配列サイズ
 - JSON変数 配列要素追加

On the right, a flowchart titled 'メイン' shows the process flow:

```

graph TD
    Start([START]) --> HTTP[HTTP HTTP]
    HTTP --> JSON1[JSON変数 読み取り]
    JSON1 --> JSON2[JSON変数 読み取り]
    JSON2 --> Wait[待機ボックス]
    Wait --> End([終了])
  
```

At the bottom right, there is a '変数一覧' (Variable List) table:

グループ名	変数名
グループなし	

■JSON変数 読み取り

「JSON変数 読み取り」ライブラリは、変数の中のJSONオブジェクトから「キー」を指定して「値」を取り出すことができます。

今回はdata変数に保存された「オブジェクト」から「キー」を指定して「値」を取り出します。



名前 JSON変数 読み取り

コメント

設定 スクリプト 注釈 バージョン情報

JSONからキーと目的を指定して値を読み取ります。

「JSON変数」：JSON文字列が格納されている変数名を設定します。
「キー」：読み取り対象のキー名を設定します。
「読み取り目的」：読み取る目的を指定します。
「値」：読み取った値を格納する変数名を設定します。

JSON変数 data

キー 値 => name

読み取り目的 転記

値 名前

応答を解析する(4/5)

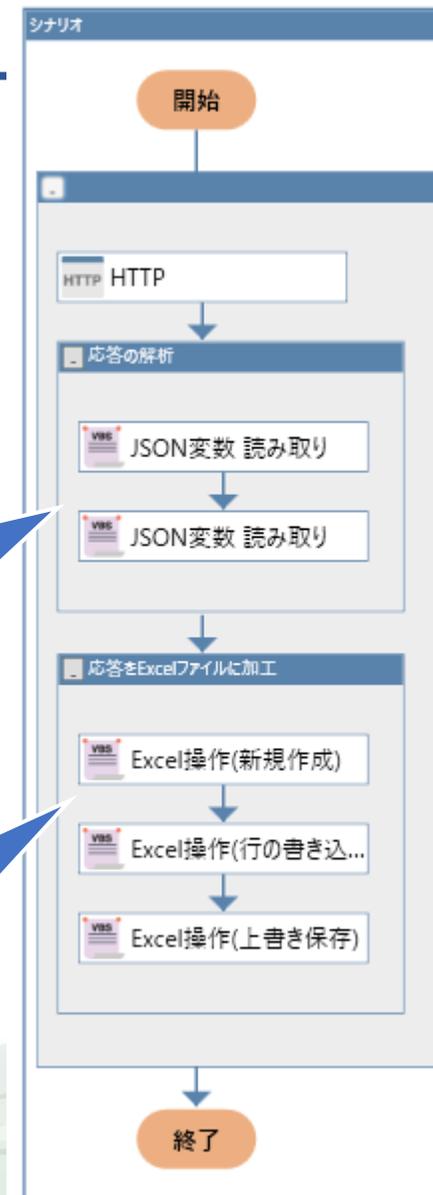
■読み取った値を加工する

変数に値を読み込めたら、あとはいつもどおりのシナリオを作るだけです。

名前と年齢をExcelファイルに書き出すようにしてみます。

dataから名前と年齢を解析して変数に入れる

名前と年齢を行の書き込みライブラリで書き込む



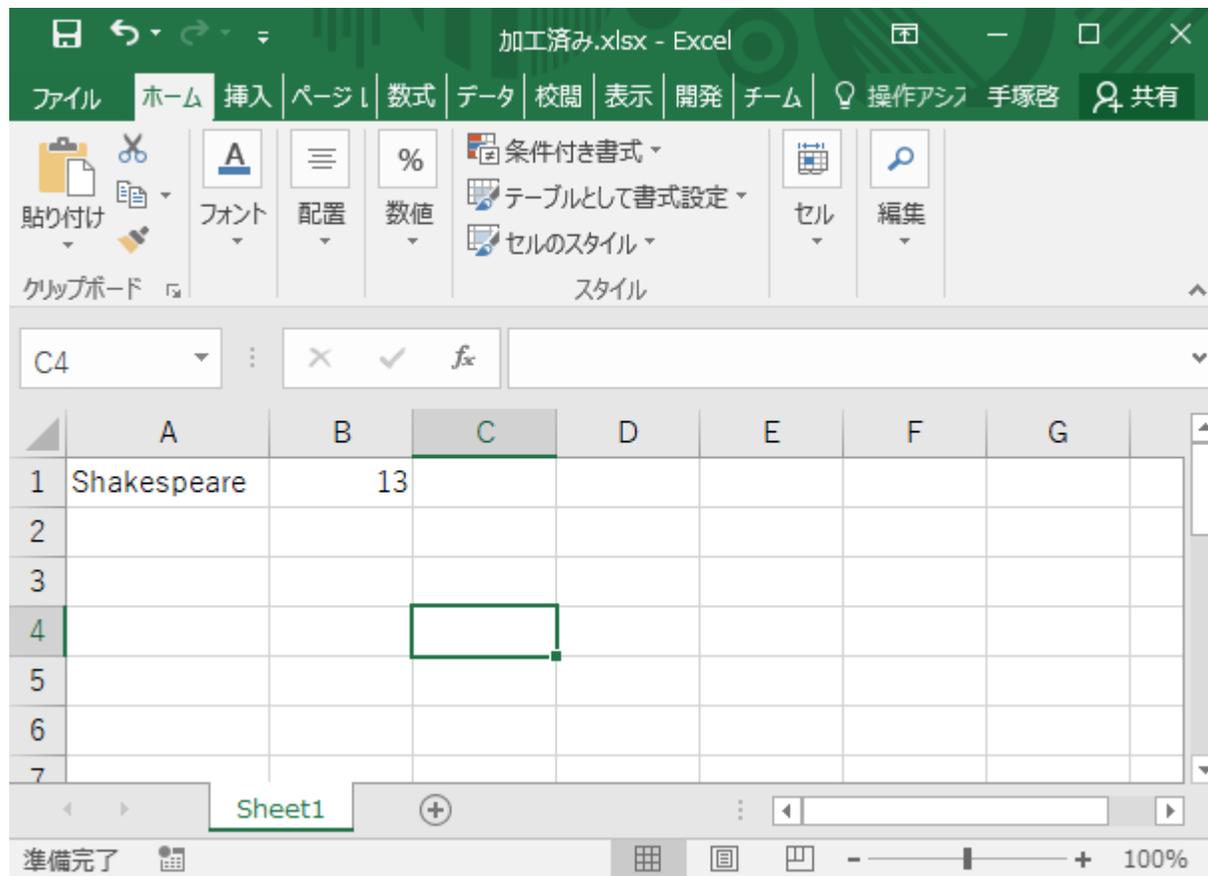
応答を解析する(5/5)

■完成？

名前と年齢がExcelに書き込まれました。これで、API連携の基礎は完成しました。

とはいえ、これだけでは何に使えるのかいまいちイメージがつかめないかと思います。

なので、少し応用していきます。



シナリオ応用①

REST APIの階層構造

■RESTについてもう少し詳しく理解する

ここまで特に理由もなく使ってきましたが、そもそもシェイクスピアさんのURLってどうやって知るの？という話です。

```
https://umayadia-apisample.azurewebsites.net/api/persons/Shakespeare
```

RESTの説明に立ち戻りますが、今回の資料の一番最初で、次のような説明をしました。

HTTPを利用したAPIは、特に**REST API**と呼ばれます。RESTとは設計思想の名称で、平たく言えば「**リソースごとにURLをもつ設計**」のことを指します。
パソコン上のファイルパスと同じような構造と考えてみてください。

大事なのは「パソコン上のファイルパスと同じような構造」という点です。

■RESTについてもう少し詳しく理解する

URLの後ろの方に注目すると、ファイル・フォルダ階層に似ています。

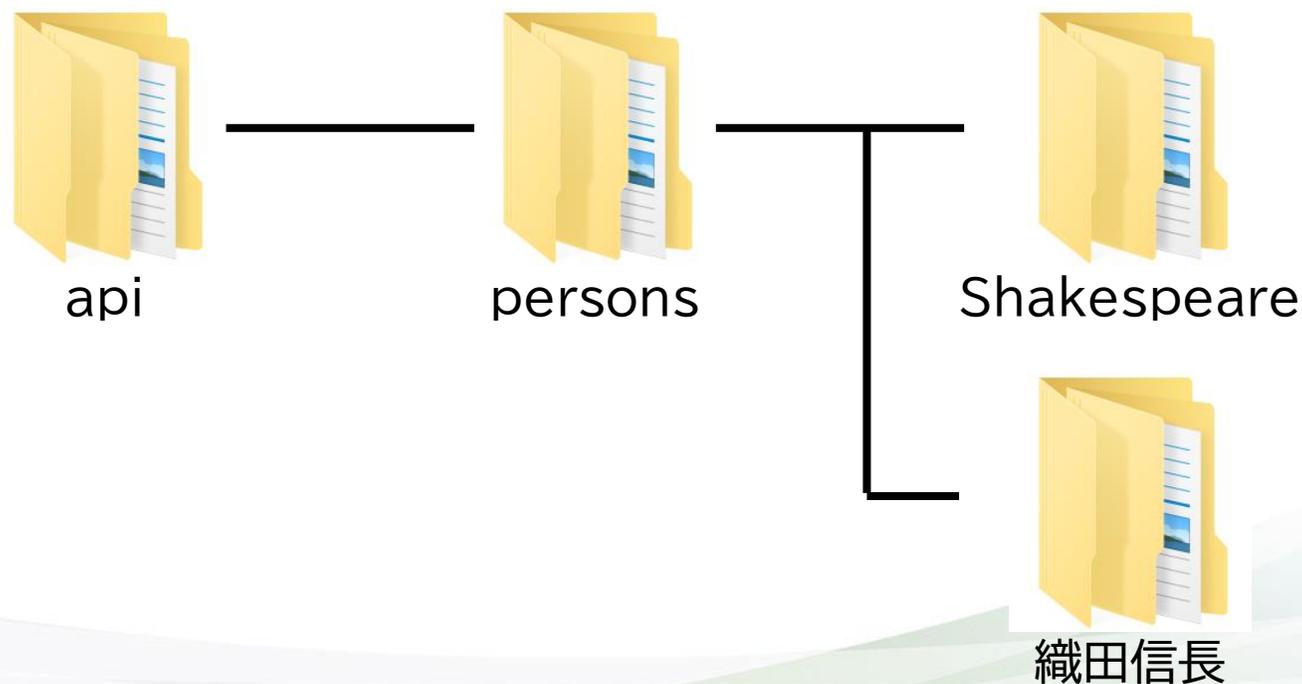
```
https://umayadia-apisample.azurewebsites.net/api/persons/Shakespeare
```

↑はシェイクスピアさんのURLですが、例えば織田信長さんのURLは次のようになっています。

```
https://umayadia-apisample.azurewebsites.net/api/persons/織田信長
```

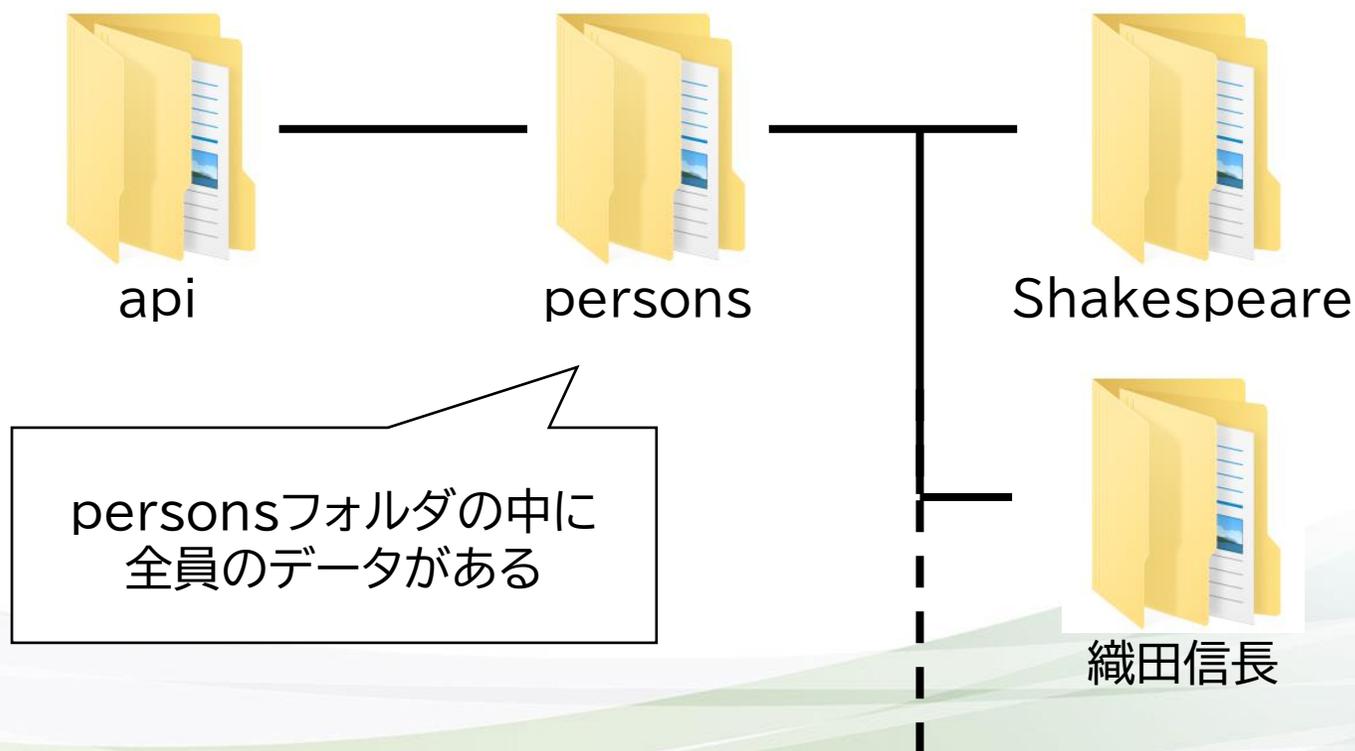
■RESTについてもう少し詳しく理解する

見通しをよくすると、以下のような階層構造になっています。REST APIは、システムが持つデータをこういった階層構造で操作できるものになります。



■階層構造ということとは？

この構造が見えてくると、例えば「すべての登録済みの人のデータがほしい」と思ったときは、personsの下の一覧を取得できればよいのでは？と考えられます。



■つまりこうすれば良い

つまり、全員のデータが欲しい場合は、personsまでのURLに対して要求すれば良いということです。

```
https://umayadia-apisample.azurewebsites.net/api/persons
```

多くのAPIではこのような要求を行うことができます。

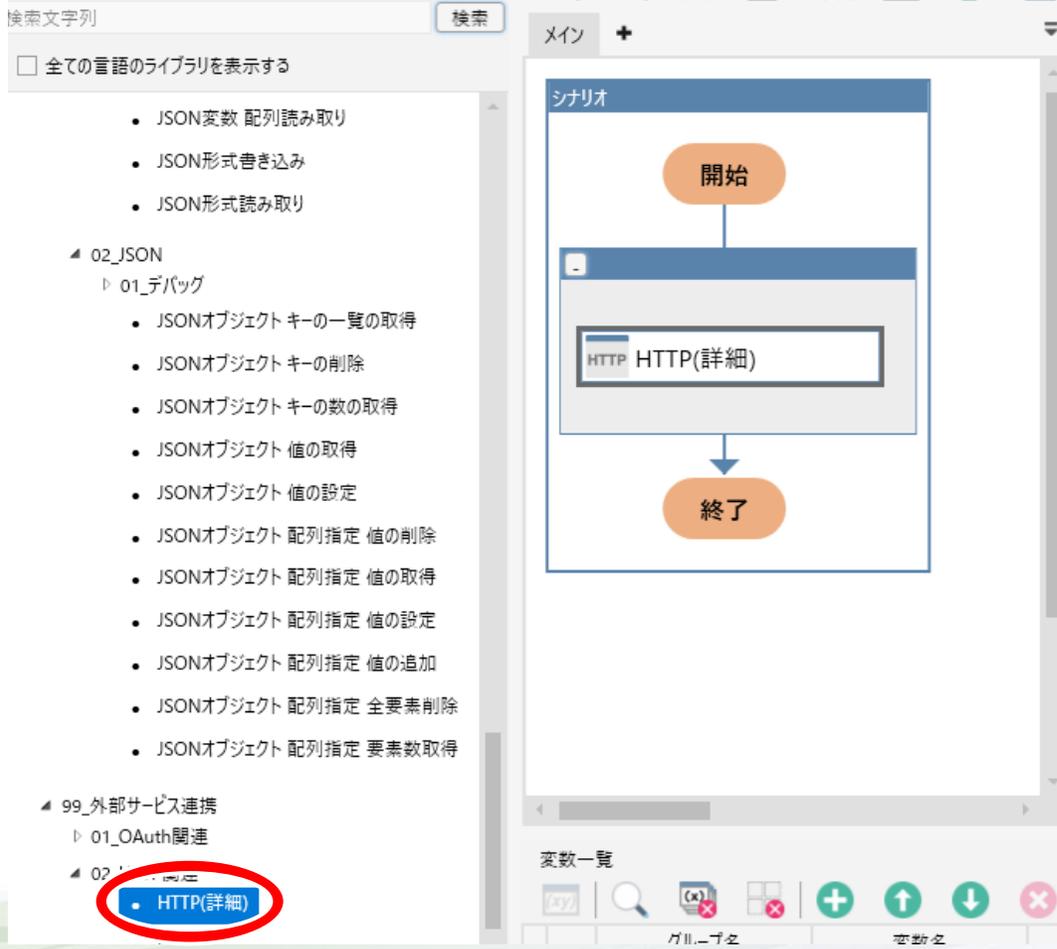
こういった形式で応答されるかはAPIによって異なりますので、その点は注意してください。

改めて、APIを試してみる

■登録されている全員分の名前と年齢をExcelに書き出してみる

今度は、特定の一人ではなくpersonsの中にある全員分のデータを取り出して見ましょう。

今回は少し難しい処理をするので、HTTPライブラリは「99_外部サービス連携」→「02_HTTP関連」の中にある「HTTP(詳細)」ライブラリを使います。

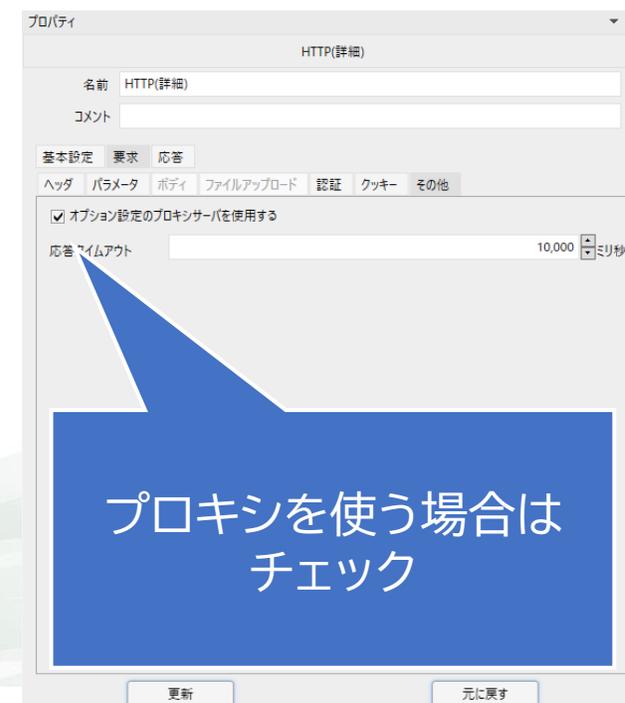
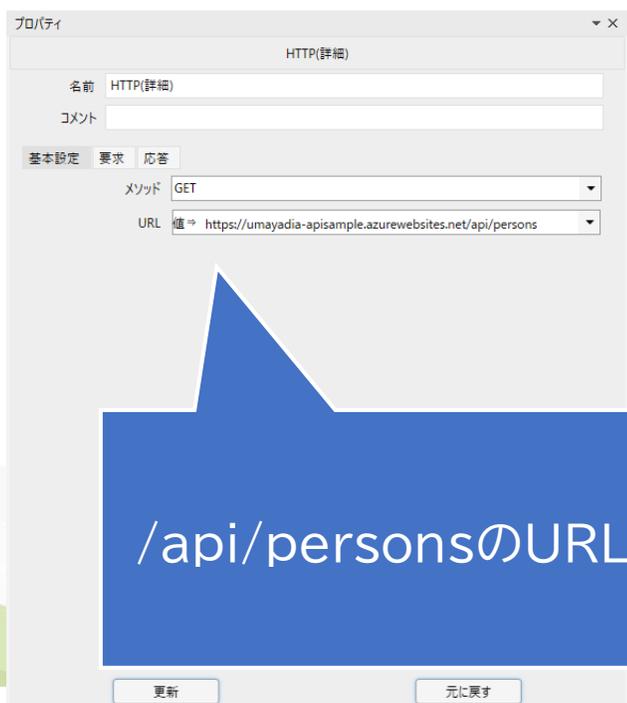


改めて、APIを試してみる

■登録されている全員分の名前と年齢をExcelに書き出してみる

設定することはほとんど変わりません。「基本設定」タブのURLに前述の全員分を取得するURLを、「応答」タブの中の「ボディ」タブ内で、ボディを取得にチェックを入れて変数名を指定します。

また、もしプロキシ環境で利用されている場合は「要求」タブの中の「その他」タブ内で、オプション設定のプロキシサーバを使用するにチェックを入れてください。



■一度実行してみる

準備はこれで完了したので、一度実行してみます。HTTP(詳細)ライブラリ実行後、body変数の中身を待機ボックスで表示してみましょう。

The screenshot shows the WinActor interface. On the left, a workflow diagram includes a step labeled 'HTTP HTTP(詳細)' and a subsequent '待機ボックス' (Waiting Box) step. On the right, a 'WinActor' data window displays the following JSON response:

```
{
  "success": true,
  "data": [
    {
      "name": "徳川家康",
      "note": "江戸幕府の初代将軍",
      "age": 11,
      "registerDate": "1543-02-10T03:04:05"
    },
    {
      "name": "織田信長",
      "note": "本能寺の変",
      "age": 12,
      "registerDate": "1534-07-03T04:14:25"
    },
    {
      "name": "Richard I",
      "note": "Richard the Lionheart",
      "age": 12,
      "registerDate": "1157-09-08T13:00:50"
    },
    {
      "name": "Shakespeare",
      "note": "Hamlet",
      "age": 13,
      "registerDate": "1564-04-26T20:21:00"
    },
    {
      "name": "聖徳太子",
      "note": "法隆寺",
      "age": 21,
      "registerDate": "0574-02-07T23:41:35"
    },
    {
      "name": "平清盛",
      "note": "福原遷都",
      "age": 21,
      "registerDate": "1118-02-17T16:00:15"
    },
    {
      "name": "源頼朝",
      "note": "鎌倉殿",
      "age": 22,
      "registerDate": "1147-05-16T17:48:22"
    },
    {
      "name": "武田信玄",
      "note": "風林火山",
      "age": 23,
      "registerDate": "1521-12-11T11:14:29"
    },
    {
      "name": "平賀源内",
      "note": "エレキテル",
      "age": 24,
      "registerDate": "1728-01-01T00:00:00"
    },
    {
      "name": "メフメト2世",
      "note": "東ローマ帝国を滅ぼす",
      "age": 31,
      "registerDate": "1901-01-02T22:00:01"
    },
    {
      "name": "Edison",
      "note": "An American inventor",
      "age": 32,
      "registerDate": "1847-02-11T14:47:11"
    },
    {
      "name": "Kennedy",
      "note": "the 35th president of the United States",
      "age": 32,
      "registerDate": "1917-05-29T15:18:49"
    },
    {
      "name": "నారాధునుడు",
      "note": "రాంచిన బౌద్ధ ధర్మ శాస్త్రికుడు.",
      "age": 32,
      "registerDate": "0150-01-01T00:00:00"
    }
  ]
}
```

At the bottom, a '変数一覧' (Variable List) table shows the 'body' variable with its corresponding JSON value.

グループ名	変数名	現在値	初期化しない	初期値	マスク	コメント
グループなし	body	{"success":true,"data":[{"name":"徳川家康","note":"江戸幕府の初代将軍","age":11,"registerDate":"1543-02-10T03:04:05"}, {"name":"織田信長","note":"本能寺の変","age":12,"registerDate":"1534-07-03T04:14:25"}, {"name":"Richard I","note":"Richard the Lionheart","age":12,"registerDate":"1157-09-08T13:00:50"}, {"name":"Shakespeare","note":"Hamlet","age":13,"registerDate":"1564-04-26T20:21:00"}, {"name":"聖徳太子","note":"法隆寺","age":21,"registerDate":"0574-02-07T23:41:35"}, {"name":"平清盛","note":"福原遷都","age":21,"registerDate":"1118-02-17T16:00:15"}, {"name":"源頼朝","note":"鎌倉殿","age":22,"registerDate":"1147-05-16T17:48:22"}, {"name":"武田信玄","note":"風林火山","age":23,"registerDate":"1521-12-11T11:14:29"}, {"name":"平賀源内","note":"エレキテル","age":24,"registerDate":"1728-01-01T00:00:00"}, {"name":"メフメト2世","note":"東ローマ帝国を滅ぼす","age":31,"registerDate":"1901-01-02T22:00:01"}, {"name":"Edison","note":"An American inventor","age":32,"registerDate":"1847-02-11T14:47:11"}, {"name":"Kennedy","note":"the 35th president of the United States","age":32,"registerDate":"1917-05-29T15:18:49"}, {"name":"నారాధునుడు","note":"రాంచిన బౌద్ధ ధర్మ శాస్త్రికుడు.","age":32,"registerDate":"0150-01-01T00:00:00"}]}	<input type="checkbox"/>		<input type="checkbox"/>	

シナリオ応用②

JSON配列

■応答を解析する

文字数が多くて複雑そうに見えますが、こちらも丁寧に解析していきます。

```
{“success”:true,“data”:[{“name”:“徳川家康”,“note”:“江戸幕府の初代将軍”,“age”:11,“registerDate”:“1543-02-10T03:04:05”},{“name”:“織田信長”,“note”:“本能寺の変”,“age”:12,“registerDate”:“1534-07-03T04:14:25”},…………,{“name”:“?????”,“note”:“?????”,“age”:35,“registerDate”:“1684-11-27T15:18:49”}]}
```

■応答を解析する

文字数が多くて複雑そうに見えますが、こちらも丁寧に解析していきます。

```
{“success”:true,“data”:[{“name”:“徳川家康”,“note”:“江戸幕府の初代将軍”,“age”:11,“registerDate”:“1543-02-10T03:04:05”},{“name”:“織田信長”,“note”:“本能寺の変”,“age”:12,“registerDate”:“1534-07-03T04:14:25”},…………,{“name”:“?????”,“note”:“?????”,“age”:35,“registerDate”:“1684-11-27T15:18:49”}]}
```

応答データのオブジェクト

■応答を解析する

文字数が多くて複雑そうに見えますが、こちらも丁寧に解析していきます。

```
{“success”:true,“data”:[{“name”:“徳川家康”,“note”:“江戸幕府の初代将軍”,“age”:11,“registerDate”:“1543-02-10T03:04:05”},{“name”:“織田信長”,“note”:“本能寺の変”,“age”:12,“registerDate”:“1534-07-03T04:00:25”},…………,{“name”:“?????”,“note”:“?????”,“age”:35,“registerDate”:“1684-11-27T15:18:49”}]}
```

successというキーと、trueという値。

■応答を解析する

文字数が多くて複雑そうに見えますが、こちらも丁寧に解析していきます。

```
{“success”:true,“data”:[{“name”:“徳川家康”,“note”:“江戸幕府の初代将軍”,“age”:11,“registerDate”:“1543-02-10T03:04:05”},{“name”:“織田信長”,“note”:“本能寺の変”,“age”:12,“registerDate”:“1534-07-03T04:14:25”},……,……,{“name”:“?????”,“note”:“?????”,“age”:35,“registerDate”:“1680-01-27T15:18:49”}]}
```

dataというキーと、
[]で囲まれたなにか？

■ []で囲まれたなにか？

[]で囲まれたものは、JSONにおける配列です。

```
{“success”:true,“data”:[{“name”:“徳川家康”,“note”:“江戸幕府の初代将  
軍”,“age”:11,“registerDate”:“1543-02-10T03:04:05”},{“name”:“織田信  
長”,“note”:“本能寺の変”,“age”:12,“registerDate”:“1534-07-  
03T04:14:25”},…………,{“name”:“?????”,“note”:“?????”,“age”:35,“r  
egisterDate”:“1684-11-27T15:18:49”}]}
```

■ []で囲まれたなにか

配列は[“要素A”, “要素B”, “要素C”]のように、カンマ区切りで羅列することができます。要素にはオブジェクトを入れることも出来るので、以下のように読み解けます。

```
{“success”:true, “data”:[{“name”:“徳川家康”, “note”:“江戸幕府の初代将軍”, “age”:11, “registerDate”:“1543-02-10T03:04:05”}, {“name”:“織田信長”, “note”:“本能寺の変”, “age”:12, “registerDate”:“1534-07-03T04:14:25”}, ……………, {“name”:“?????”, “note”:“?????”, “age”:35, “registerDate”:“1684-11-27T15:18:49”}]}
```

1つ目の要素
(オブジェクト)

2つ目の要素
(オブジェクト)

最後の要素
(オブジェクト)

■配列の中身は？

配列の要素であるオブジェクトは、先程シェイクスピアさんのときに解析したものと同一構造です。

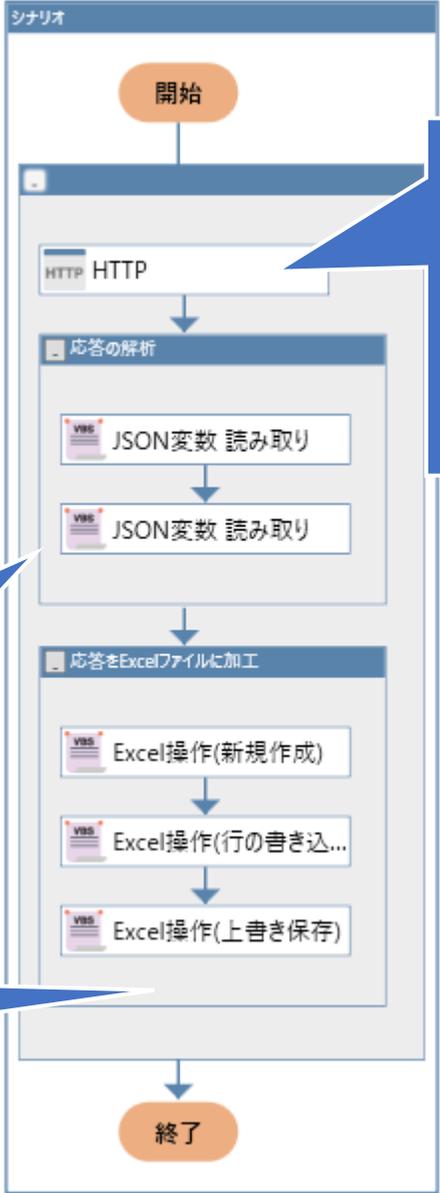
```
{“success”:true,“data”:[{“name”:“徳川家康”,“note”:“江戸幕府の初代将軍”,“age”:11,“registerDate”:“1543-02-10T03:04:05”},{“name”:“織田信長”,“note”:“本能寺の変”,“age”:12,“registerDate”:“1534-07-03T04:14:25”},…………,{“name”:“?????”,“note”:“?????”,“age”:35,“registerDate”:“1684-11-27T15:18:49”}]}
```

オブジェクトの中
身はデータの
セット

JSON配列(8/11)

■つまり、配列の要素を取り出せば...

先程作った処理を繰り返し実行するだけで、全員分のデータをExcelに書き出せるわけです。



配列から要素(オブジェクト)を取り出す

要素(オブジェクト)から、名前と年齢を読み取る

そして書き込む

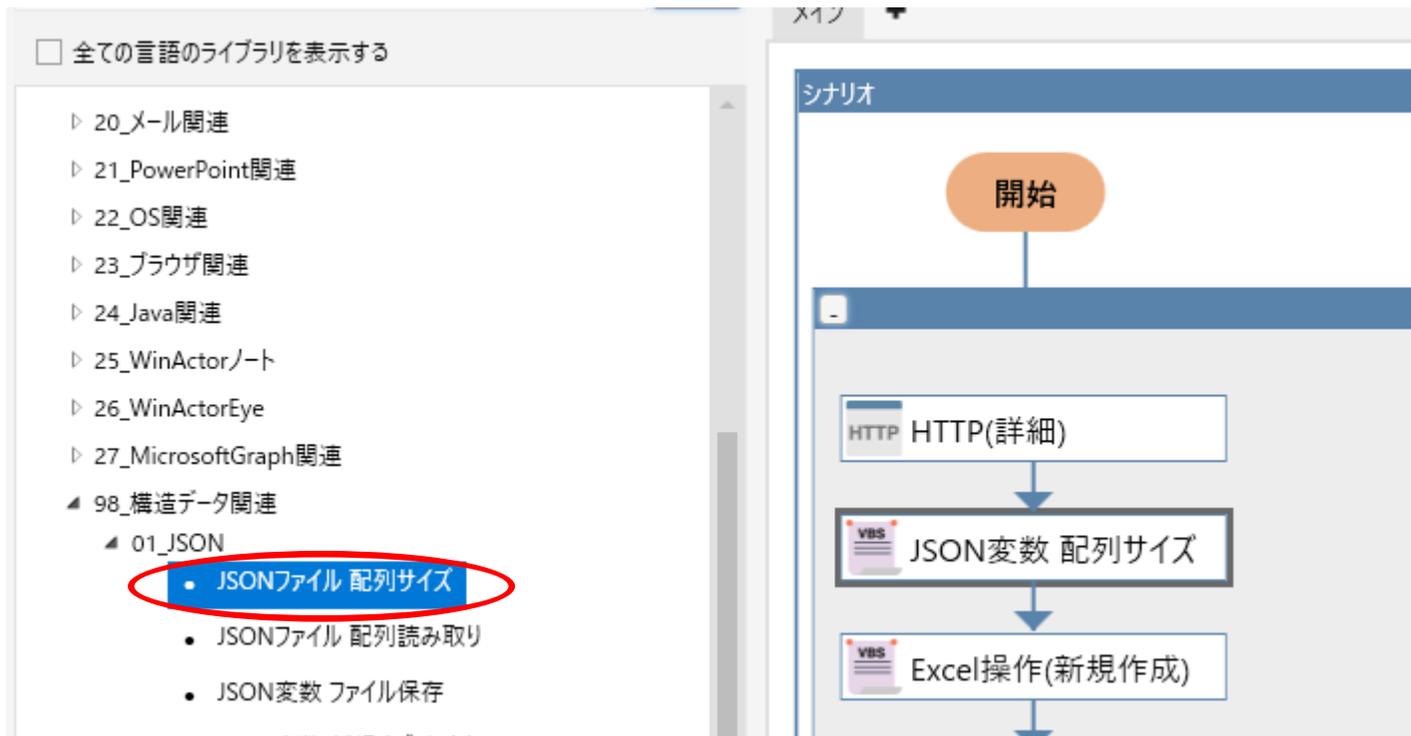


これを繰り返す

■JSON変数 配列サイズ

まず、JSONの配列サイズを読み込んで、繰り返し回数を決めます。

「98_構造データ関連」→「01_JSON」内、
「JSON変数 配列サイズ」ライブラリを
使います。

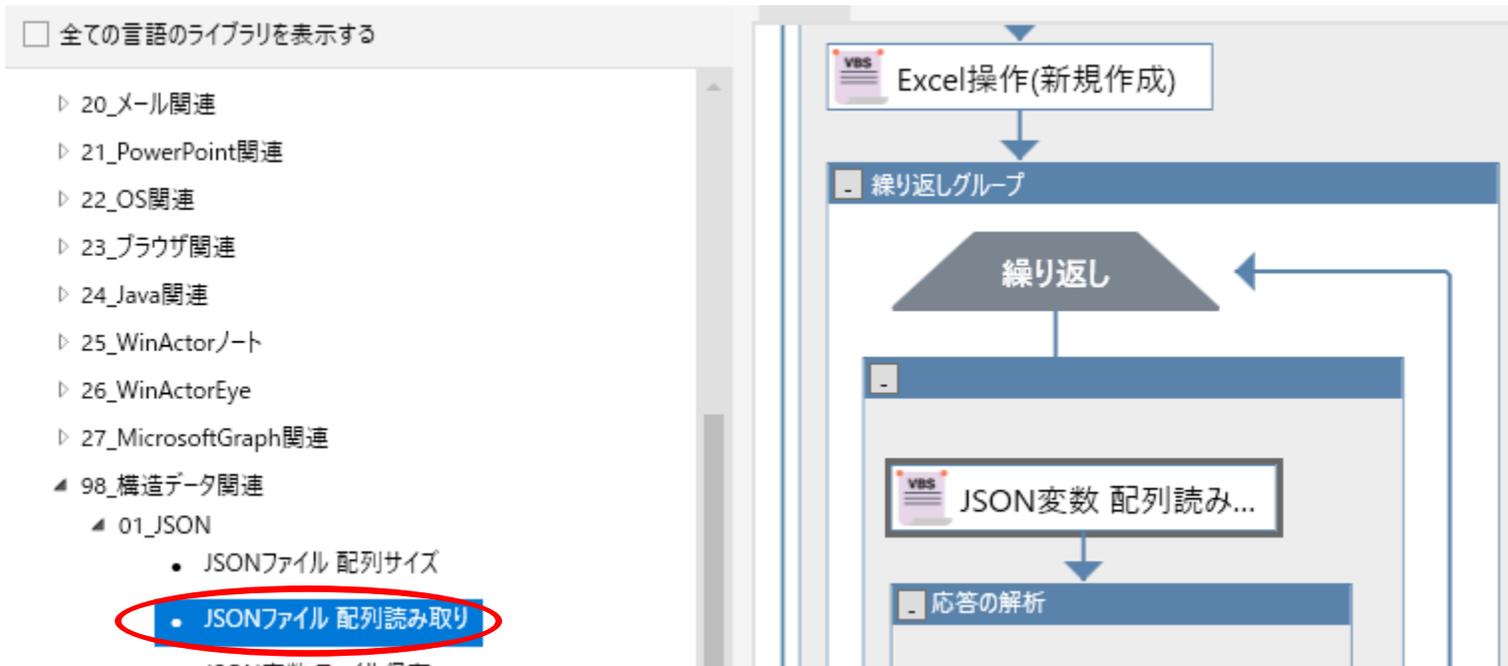


The screenshot displays two parts of a software interface. On the left is a library tree with a search filter '全ての言語のライブラリを表示する'. The tree is expanded to '98_構造データ関連' > '01_JSON', where 'JSONファイル 配列サイズ' is highlighted with a red circle. On the right is a 'シナリオ' (Scenario) flowchart. It starts with an orange oval labeled '開始' (Start), followed by a sequence of three rectangular steps: 'HTTP HTTP(詳細)', 'VBS JSON変数 配列サイズ', and 'VBS Excel操作(新規作成)'. Arrows indicate the flow from top to bottom.

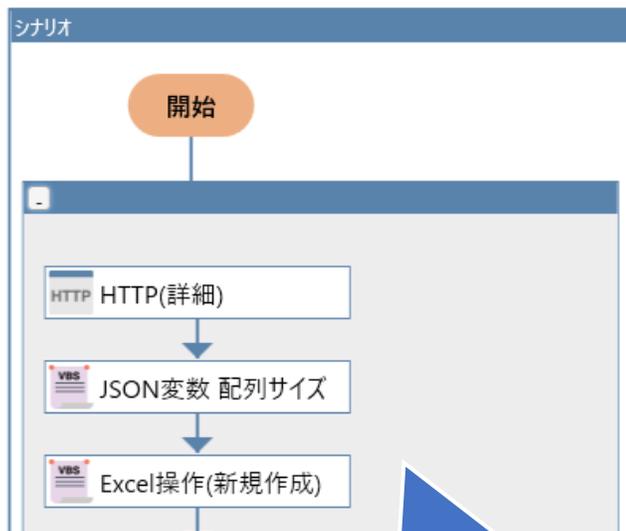
■JSON変数 配列読み込み

そして、JSONの配列から要素を読み込みます。

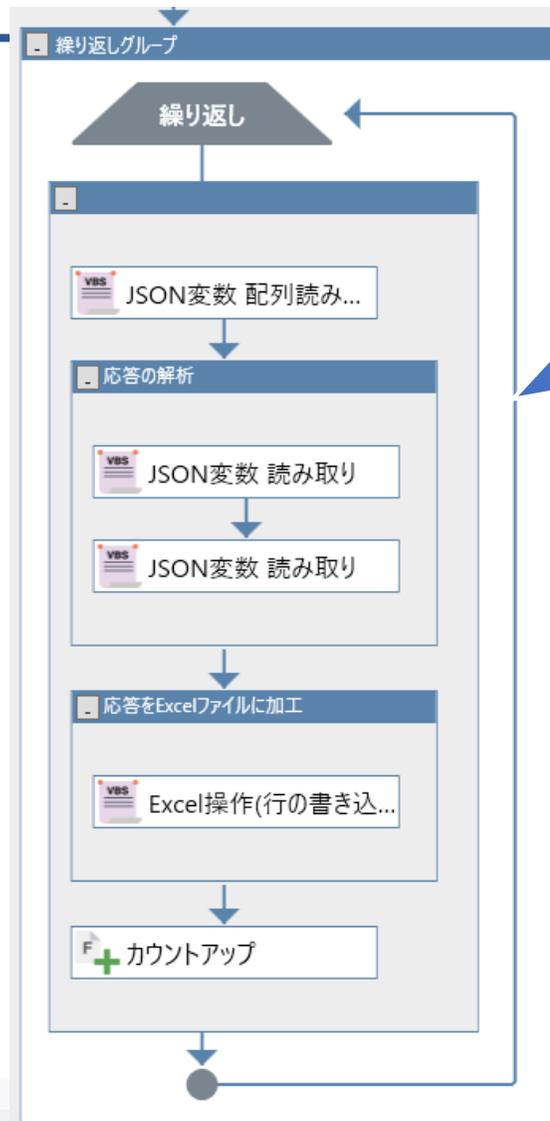
「98_構造データ関連」→「01_JSON」内、
「JSON変数 配列読み取り」ライブラリを
使います。



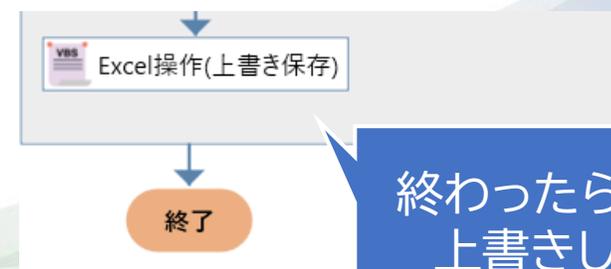
■こうなりました



APIで全員分のデータをJSON配列で取得して、そのサイズ=人数を確認



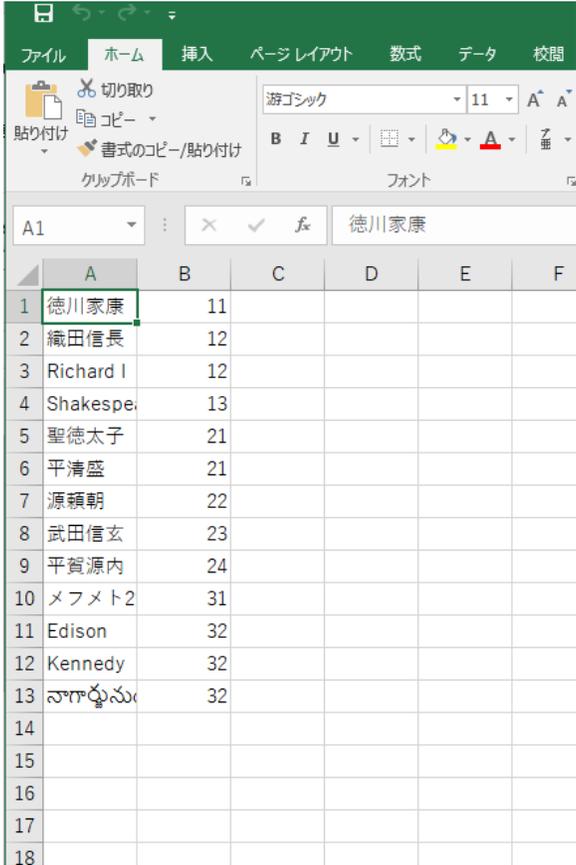
配列要素のオブジェクトから名前と年齢を取り出してExcelファイルに書き込む、を人数分繰り返す



終わったらファイルを上書きして閉じる

■結果の確認

シナリオを実行すると、登録されている全員の名前と年齢が記録できました。



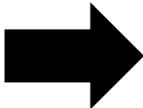
The screenshot shows an Excel spreadsheet with the following data:

	A	B	C	D	E	F
1	徳川家康	11				
2	織田信長	12				
3	Richard I	12				
4	Shakespe	13				
5	聖徳太子	21				
6	平清盛	21				
7	源頼朝	22				
8	武田信玄	23				
9	平賀源内	24				
10	メフメト2	31				
11	Edison	32				
12	Kennedy	32				
13	నాగార్జునుడు	32				
14						
15						
16						
17						
18						

ここまでのまとめ

■これまでのサービス連携は…

ブラウザ操作ライブラリ
で自由自在にシナリオが
作れる！



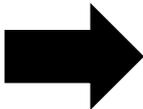
Web画面更新
シナリオが複雑

作った後の保守運用が
結構大変…

■API連携を使うと…

JSON
HTTP
REST API

作るまでに必要な知識
が多い…



一度作れば安定して運
用できる！

API連携実践編

DocuSign連携の紹介

■サンプルシナリオを例に、実際のサービスと連携したものを紹介します



「DocuSign」とAPI連携の初期設定を行うサンプルシナリオ

カテゴリ
■ 外部サービス連携

対応バージョン
・ ums6、ums7

公開日 / 2021年03月03日

「DocuSign」とAPI連携を行うために必要な初期設定を行い、認証情報を取得するサンプルシナリオになります。

詳しくはこちら [➡](#)



「DocuSign」とAPI連携でドキュメントのダウンロードをするサンプルシナリオ

カテゴリ
■ 外部サービス連携

対応バージョン
・ ums6、ums7

公開日 / 2021年03月03日

「DocuSign」とAPI連携し、署名済みドキュメントをすべてダウンロードするサンプルシナリオになります。

詳しくはこちら [➡](#)

■DocuSignとは？

DocuSignは、電子署名を行うサービスです。

サービスの中には契約書のドキュメントや契約情報などが保存されており、社内システム等と連携しやすい内容になっています。

■API連携の初期設定を行うシナリオ

DocuSign連携のサンプルシナリオは、APIを実行するための初期設定を行うシナリオと、各種操作をAPI連携で行うサンプルシナリオの2種類に分かれています。

まずは初期設定のシナリオについて概要を紹介します。



外部サービス関連

公開日 / 2021年03月03日

「DocuSign」とAPI連携でドキュメントのダウンロードをするサンプルシナリオ

カテゴリー
■ 外部サービス連携

対応バージョン
・ ums6、ums7

「DocuSign」とAPI連携し、署名済みドキュメントをすべてダウンロードするサンプルシナリオになります。

詳しくはこちら [→](#)

■シナリオ解説の前に…

ここでの「初期設定」とは、認証処理(=APIでサービスにログインすること)です。

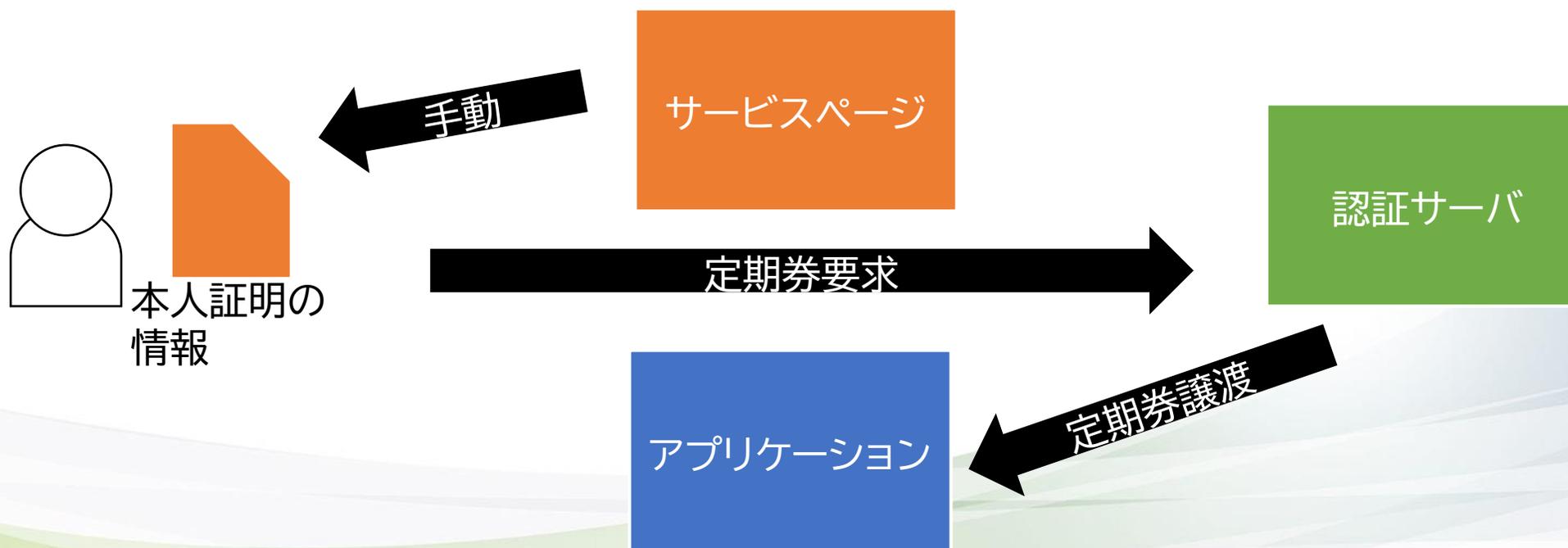
ブラウザからサービスを使うときにログインが必要なように、API連携する時もログイン処理に相当する認証処理が必要なことが多いです。このサンプルシナリオは、その認証処理を行う内容になっています。

DocuSign連携で行う認証処理は**OAuth2.0(オーオース2.0)**という方式を用いています。多くのサービスで利用されている方式なので、簡単に説明します。

■OAuth2.0

OAuth2.0を利用した認証の概念を簡単に説明すると、「アプリケーションがサービスにAPI連携するための定期券をもらう手順」です。

定期券をもらうためには一度は手動でログインしなければならない(身分証明書のようなものが必要)ですが、一度定期券をもらってしまえば何度でもAPI連携を利用できます。



■初期設定シナリオを実行すると得られるもの

サンプルシナリオの手順通りに実行すると、「**Basic認証キー**」「**アクセストークン**」「**リフレッシュトークン**」の3つのファイルが出力されます。

「**Basic認証キー**」は本人証明の情報、「**アクセストークン**」「**リフレッシュトークン**」はAPI利用の定期券に相当します。

これでAPI連携の準備は完了です。

■API連携するシナリオ

次にAPI連携のシナリオを見ていきます。



外部サービス関連

公開日 / 2021年03月03日

「DocuSign」とAPI連携でドキュメントのダウンロードをするサンプルシナリオ

カテゴリ

- 外部サービス連携

対応バージョン

- ・ ums6、ums7

「DocuSign」とAPI連携し、署名済みドキュメントをすべてダウンロードするサンプルシナリオになります。

[詳しくはこちら](#) ➔

■API連携するシナリオ

API連携するHTTPライブラリの設定を見ると、「要求」タブ内の「ヘッダ」にパラメータが追記されています。



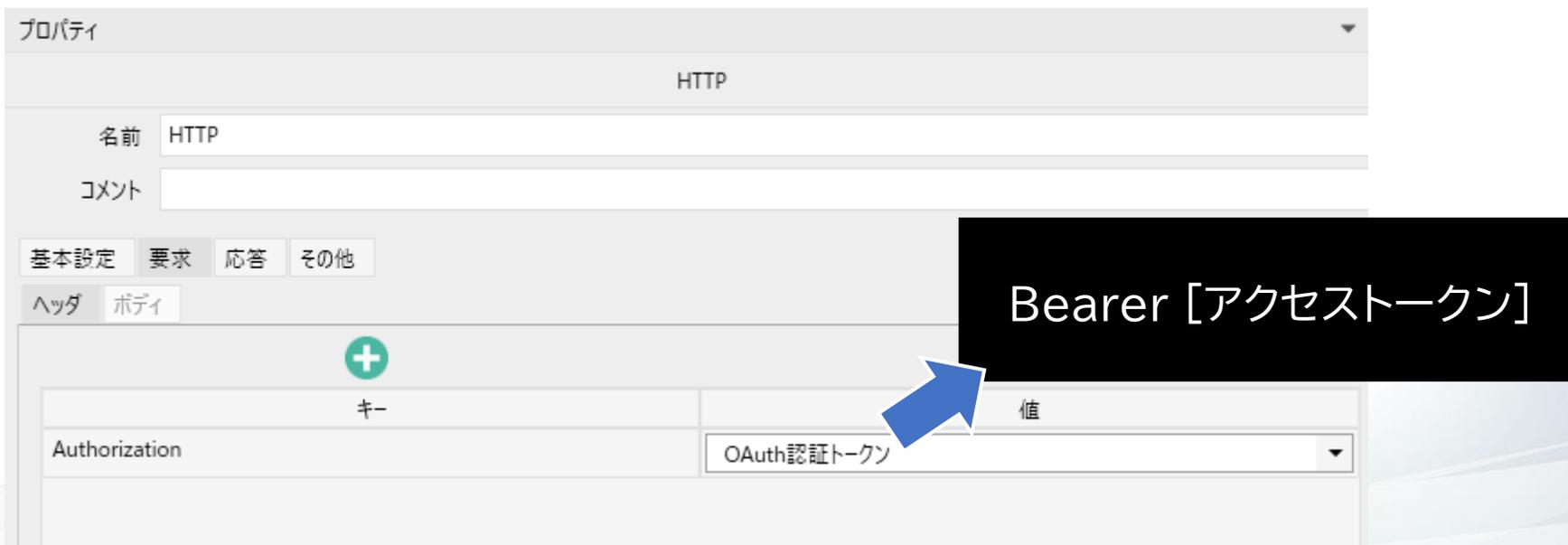
The screenshot shows a configuration window for an HTTP request. The 'Headers' tab is selected, and a table lists the headers. The 'Authorization' header is highlighted with a red box, and its value is 'OAuth認証トークン'.

キー	値
Authorization	OAuth認証トークン

■Bearer認証

ヘッダというのは、HTTPの要求につけるオプションのようなものです。認証が必要なAPIでは、このヘッダに認証情報を設定します。

今回連携するDocuSignではBearer認証という認証方式が採用されているので、それに則ったヘッダを設定しています。



プロパティ

HTTP

名前 HTTP

コメント

基本設定 要求 応答 その他

ヘッダ ボディ

キー	値
Authorization	OAuth認証トークン

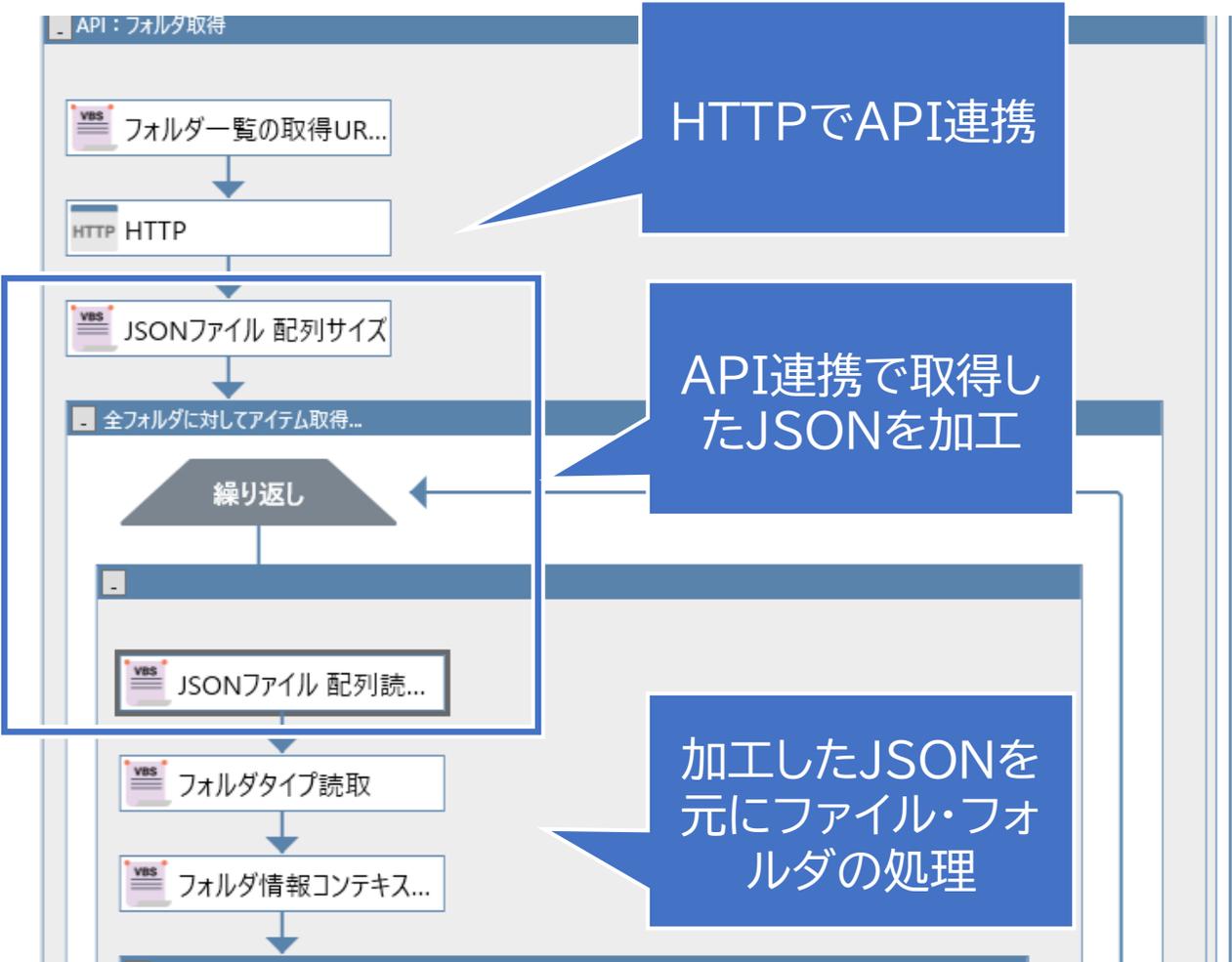
Bearer [アクセストークン]

API連携後の処理

■認証以外の処理はどうなっている？

API連携後の処理は今までと同じく、API連携で取得したJSONを加工するものです。

一見複雑に見えますが、「HTTP」と「JSON」の役割を理解するとシンプルな構造になっていることがわかります。



■シナリオの実行

API連携シナリオを実行すると、DocuSignサービス上にあるリソース(=契約文書のドキュメントファイル)がダウンロードされます。

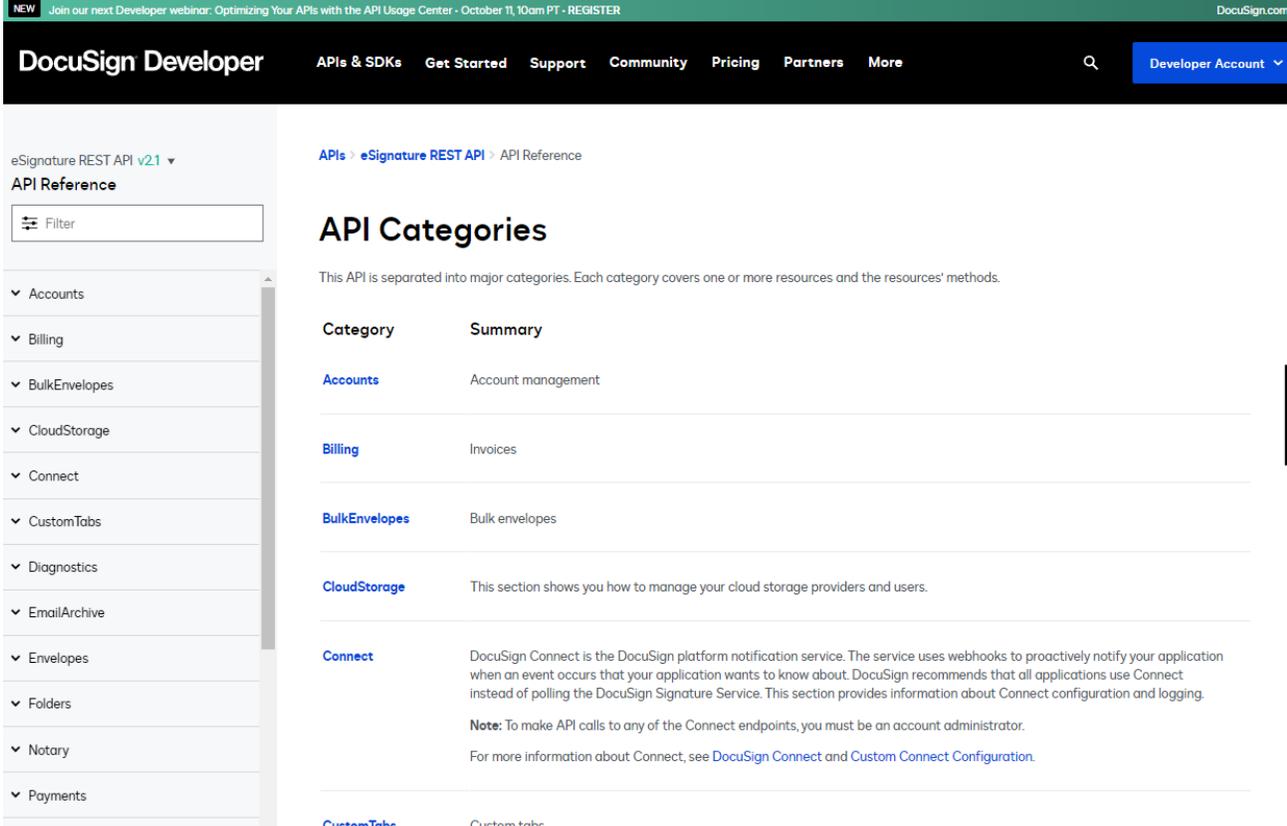
■APIを知る

API連携を作るためには、URLの情報や認証の手順などを調べる必要があります。

通常、APIを公開しているサービスはAPIリファレンス(APIの仕様書)も併せて公開されています。

まずはリファレンスを読み、何ができるのかを把握することから始めることをお勧めします。

海外のサービスは日本語のAPIリファレンスがないことが多いので、ちょっとハードルが高いです。



DocuSign Developer

NEW Join our next Developer webinar: Optimizing Your APIs with the API Usage Center - October 11, 10am PT - REGISTER

DocuSign.com

APIs & SDKs Get Started Support Community Pricing Partners More

Developer Account

eSignature REST API v2.1

API Reference

Filter

Accounts

Billing

BulkEnvelopes

CloudStorage

Connect

CustomTabs

Diagnostics

EmailArchive

Envelopes

Folders

Notary

Payments

APIs > eSignature REST API > API Reference

API Categories

This API is separated into major categories. Each category covers one or more resources and the resources' methods.

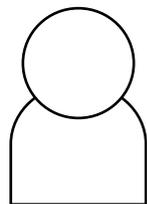
Category	Summary
Accounts	Account management
Billing	Invoices
BulkEnvelopes	Bulk envelopes
CloudStorage	This section shows you how to manage your cloud storage providers and users.
Connect	DocuSign Connect is the DocuSign platform notification service. The service uses webhooks to proactively notify your application when an event occurs that your application wants to know about. DocuSign recommends that all applications use Connect instead of polling the DocuSign Signature Service. This section provides information about Connect configuration and logging. Note: To make API calls to any of the Connect endpoints, you must be an account administrator. For more information about Connect, see DocuSign Connect and Custom Connect Configuration .
CustomTabs	Custom tabs

※<https://developers.docusign.com/docs/esign-rest-api/reference/>

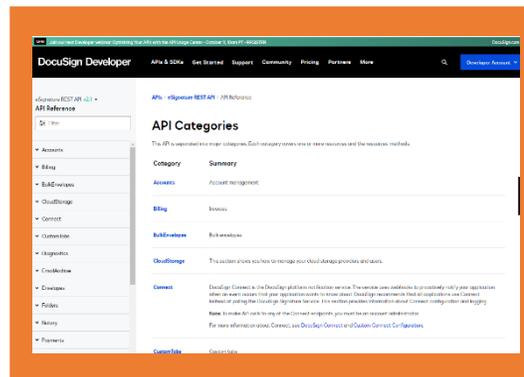
①前提知識を身につける

JSON

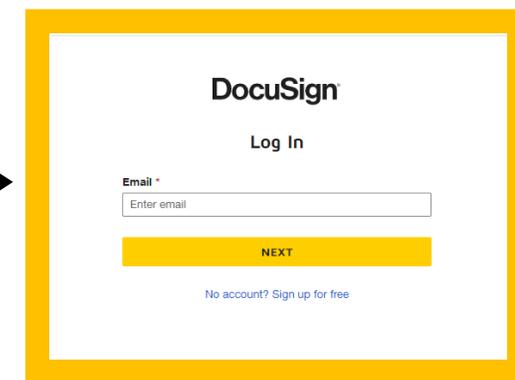
HTTP



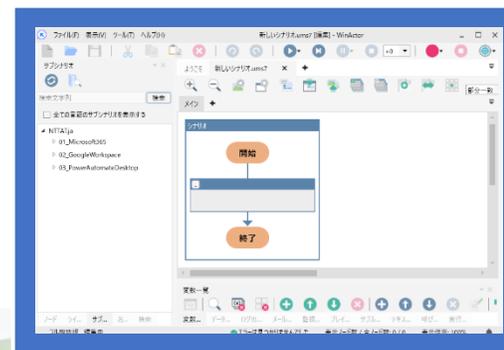
①APIリファレンスを見て
できることを理解する



②サービスにログインして
API利用の準備をする



③HTTPライブラリと
JSONライブラリで
シナリオを作る



サービスと連携する方法は
いくつかあります。

画面操作

ファイル
操作

API連携

メリットとデメリットを把握して、
目的に合った連携方法を選択しましょう。

まずHTTPライブラリを使って
みることをお勧めします。

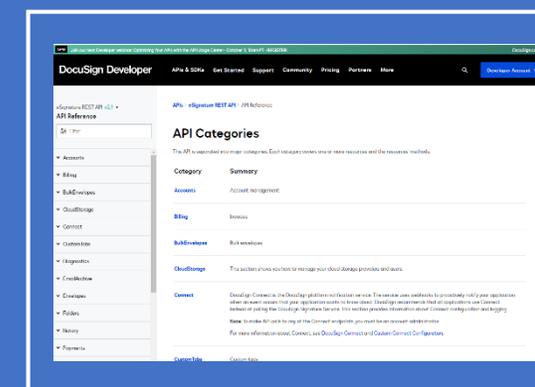
要求

応答

JSON

APIのサンプルを使って、
要求・応答の確認とJSONの
構造を理解しましょう。

シナリオを作るには、
APIの知識が必要です。



まずはリファレンスを読み、
APIで何ができるのかを
把握しましょう。

知識をつけて、よいAPI連携ライフを！

未来を拓くチカラと技術。



参考1:プロキシ環境での動作

■プロキシ環境でHTTPライブラリを動かす場合の設定

HTTPライブラリはWinActorのプロキシ設定を参照して動作します。WinActorをプロキシ環境で利用されている場合は、WinActor本体のプロキシ設定を確認してください。

「ツール」メニューの一番下、「オプション」から設定を確認できます。

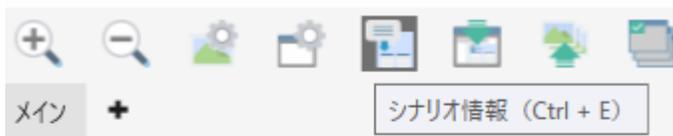


The screenshot shows the 'Options' dialog box in WinActor. The 'Proxy Server' tab is selected. The 'Use Proxy Server' checkbox is checked. Under 'Proxy Server Settings', the 'Manual Setting' radio button is selected. The 'Host' and 'Port' fields are filled with redacted text. The 'Authentication Required' checkbox is also checked, and the 'Authentication Method' is set to 'Basic/Digest'. The 'Username' and 'Password' fields are also redacted, and the 'Domain' field is empty. The 'OK' and 'Cancel' buttons are at the bottom.

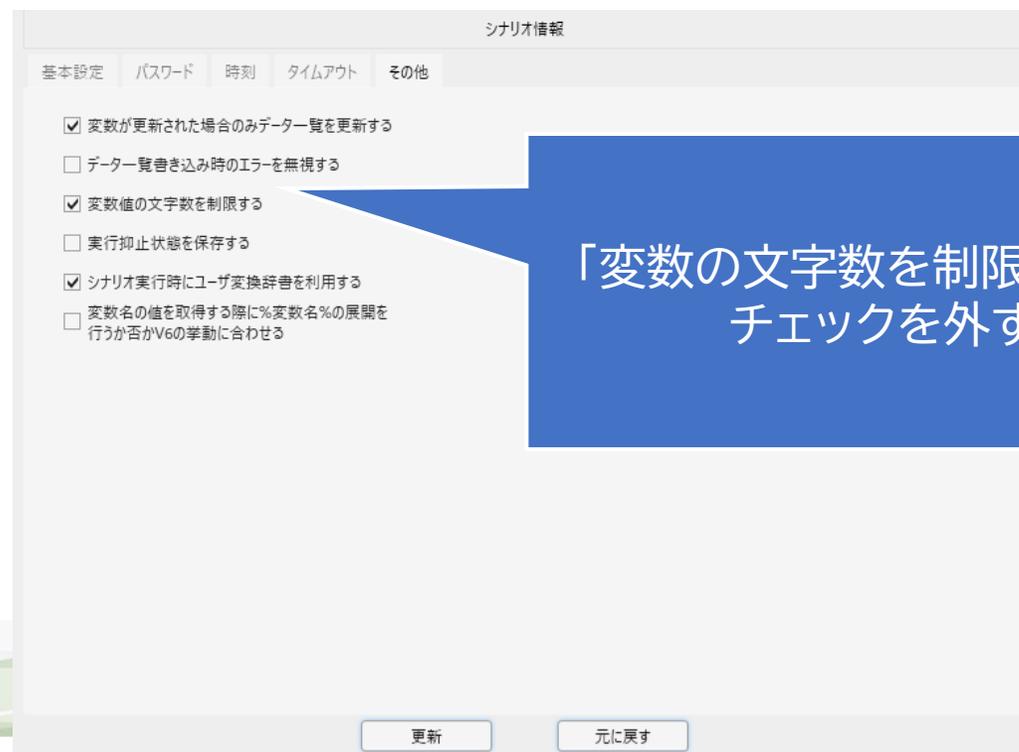
■シナリオ作成の注意点

今回紹介したAPI連携では、応答データを変数に格納する方法を紹介しています。

応答データは比較的文字数が大きくなりがちのため、デフォルトのシナリオ設定では文字制限に引っ掛かりやすいです。API連携シナリオを作成される際は、予め変数の文字数制限を解除しておくことをおすすめします。



シナリオ情報ボタンをクリック



「変数の文字数を制限する」の
チェックを外す

■サブシナリオも連携に使える

WinActor本体には、MicrosoftOffice、GoogleWorkspace、PowerAutomateと連携するサブシナリオが同梱されています。

サブシナリオは画面操作で実現されたシナリオですが、メンテナンスはNTTATで実施しているため、画面の更新を気にすることなく利用が可能です。

■GET,POST,PUT,DELETE

資料中では紹介しませんでしたでしたが、HTTPにはメソッドというものがあります。リソースを取得するにはGET、リソースを新規にアップロードするにはPOST、既存のリソースを上書きするにはPUT、リソースを削除するにはDELETEを設定します。

APIリファレンスを読むと対応するメソッドが記載されていますので、それに合わせて変更してください。

■ステータスコード

HTTPライブラリは、どのような結果になってもエラーを出すことはありません。要求の成否を確認するためには、応答の中にあるステータスコードを見る必要があります。

ステータスコードの詳細は省きますが、200番台の番号であれば成功、それ以外であれば失敗とだけ思っていればひとまずは大丈夫です。

■Basic認証キーとリフレッシュトークン

アクセストークンがあればAPIを使えますが、これには**有効期限**が存在するので、そのうち使えなくなります。

じゃあ有効期限が切れたらまた最初からやり直し…というのは面倒なので、登場するのが「**リフレッシュ**」です。リフレッシュとは、「Basic認証キー」と「リフレッシュトークン」を使うことで、新しいアクセストークンを簡単に取得できる仕組みのことです。

APIは窓口、アクセストークンがチケットとすれば、リフレッシュは定期券の更新のようなもの、と考えるとわかりやすいかもしれません。

■リフレッシュ処理

リフレッシュ要求には、ヘッダにBasic認証キー、ボディにリフレッシュトークンを設定しています。

The image shows a flowchart on the left and an API configuration interface on the right. The flowchart, titled 'トークンのリフレッシュ', starts with '開始' (Start), followed by 'リフレッシュトークン読み込み' (Load refresh token), then 'リフレッシュ要求' (Refresh request) via HTTP. It branches into '正常系' (Normal flow) and '異常系' (Exception flow). The normal flow includes '最新版アクセストークン...' (Latest access token...) and '最新版リフレッシュトークン...' (Latest refresh token...). The exception flow includes 'アクション例外' (Action exception). The API configuration interface shows the 'リフレッシュ要求' (Refresh request) configuration. The 'ヘッダ' (Header) tab is selected, showing a 'Basic認証キー' (Basic authentication key) under 'Authorization'. The 'ボディ' (Body) tab is also visible, showing a table with 'grant_type' and 'refresh_token' parameters.

ヘッダにはBasic認証キー

ボディにはリフレッシュトークン

キー	型	値
grant_type	文字列	値⇒ refresh_token
refresh_token	文字列	リフレッシュトークン

■有効期限は短い

アクセストークンの有効期限は比較的短い時間が設定されることが多いです。

これは、もしアクセストークンが第三者に知られた場合でも、悪用される前にすぐに使えなくなるというセキュリティ上の理由からです。

■有効期限が長いものもある

一方で、アクセストークンの有効期限が長めに設定されていることもあります。

これは、サービス側でリフレッシュトークンの仕組みが用意されていないため、アクセストークンの再発行という面倒な手続きを短い期間に何度も行わないように配慮されていることが多いです。

■認証情報をどうやって隠す？

参考5で説明した通り、認証情報のうちアクセストークンは有効期限があるため、外部に漏洩したり、誤って編集・削除してしまってもセキュリティや動作上の影響が少ないです。

一方、Basic認証キーはユーザ名とパスワードの情報そのものなので、編集・閲覧の権限は注意して設定する必要があります。

DocuSign連携シナリオではBasic認証キーをファイルにしていたので、ファイル自体を読み取り専用にするだけで編集されなくすることが可能です。

その他、WinActorシナリオ変数に入れておき、変数のマスク機能で閲覧できなくする、Excelファイルに記録しておき、パスワード付きにして閲覧できなくするなど、運用に合わせて適切な権限管理をすることを推奨します。

■JSONって何者？

JSON構造はもともとJavaScriptの世界で定義されたもので、あまりに便利だったので後々あらゆる部門で活用され始めたものです。

正式な仕様は「**ECMAScript 5.1**」というJavaScriptの規格の仕様書に記載されています。

とはいえAPI連携で利用する範疇であれば、本資料の内容 + α (躓いたときに調べる) ぐらいでも十分活用できると思います。