記事
Toshihiko Minamoto · 2021年11月1日 9m read

Open Exchange

REST経由でファイル転送しプロパティに格納する - パート2

この連載の第1回目の記事では、「大規模な」チャンクのデータをHTTP POSTメソッドのRaw本体から読み取って、それをクラスのストリームプロパティとしてデータベースに格納する方法について説明しました。
では、そのようなデータとメタデータをJSON形式で格納する方法について見てみましょう。

残念ながら、Advanced REST Clientでは、バイナリーデータをキーの値としてJSONオブジェクトを作成することはできません（もしかすると、私がまだ知らないだけかもしれません）。そこで、サーバーにデータを送信する単純なクライアントをObjectScriptで記述することにしました。

新しいRestTransfer.Clientというクラスを作成し、私のWebサーバーを指定するServer = "localhost"パラメーターとPort = 52773パラメーターをそれに追加しました。
そしてGetLink
クラスメソッドを作成しました。この中で、クラス%Net.HttpRequest
の新しいインスタンスを作成し、プロパティを前述のパラメーターに設定します。

実際にPOSTリクエストをサーバーに送信
するために、クラスメソッドSendFileDirect

EntityBodyプロパティに書き込みます。

この後、メソッドPost("/RestTransfer/file")
を呼び出し、正常に完了すれば、レスポンスにHttpResponseプロパティが含まれます。

サーバーが返す結果を確認するために、レスポンスのOutputToDeviceメソッドを呼び出します。

以下は、そのクラスメソッドです。

```
Class RestTransfer.Client
{
  Parameter Server = "localhost";
  Parameter Port = 52773;
  Parameter Https = 0;

  ClassMethod GetLink() As %Net.HttpRequest
  {
      set request = ##class(%Net.HttpRequest).%New()
      set request.Server = ..#Server
      set request.Port = ..#Port
      set request.ContentType = "application/octet-stream"
      quit request
  }

  ClassMethod SendFileDirect(aFileName) As %Status
  {
    set sc = $$$OK
    set request = ..GetLink()
    set s = ##class(%Stream.FileBinary).%New()
```

```
    set s.Filename = aFileName
    While 's.AtEnd
    {
      do request.EntityBody.Write(s.Read(.len, .sc))
      Quit:$System.Status.IsError(sc)
    }
    Quit:$System.Status.IsError(sc)

    set sc = request.Post("/RestTransfer/file")
    Quit:$System.Status.IsError(sc)
    set response=request.HttpResponse
    do response.OutputToDevice()
    Quit sc
  }
}
```

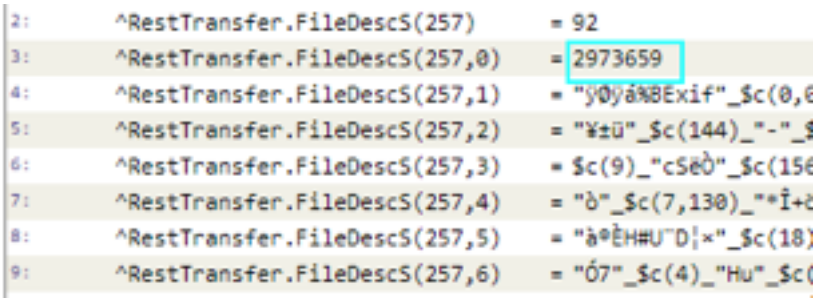
このメソッドを呼び出して、前の記事で使用したファイルを転送することができます。

```
 do ##class(RestTransfer.Client).SendFileDirect("D:\Downloads \2020_1012_114732_020.J
PG")
 do ##class(RestTransfer.Client).SendFileDirect("D:\Downloads\Archive.xml")
 do ##class(RestTransfer.Client).SendFileDirect("D:\Downloads\arc-setup.exe")
```

ファイルごとにサーバーからのレスポンスが次のように表示されます。

```
HTTP/1.1 200 OK
CACHE-CONTROL: no-cache
CONTENT-LENGTH: 15
CONTENT-TYPE: text/html; charset=utf-8
DATE: Thu, 05 Nov 2020 15:13:23 GMT
EXPIRES: Thu, 29 Oct 1998 17:04:19 GMT
PRAGMA: no-cache
SERVER: Apache
{"Status":"OK"}
```

そして、前と同じデータがグローバルにあります。



つまり、クライアントは機能しているため、JSONをサーバーに送信するように拡張することができます。

JSONを理解する

JSONは、名前/値のペアがコンマ区切りで含まれるデータで、データの格納と転送を行うための軽量な形式です。
この場合、JSONは次のようになります。

```
{
  "Name": "test.txt",
  "File": "Hello, world!"
}
```

IRISにはJSONを操作できるクラスがいくつかあります。 次のクラスを使用します。

%JSON.Adaptor :
JSON対応オブジェクトをJSONドキュメントとして、またはその逆にシリアル化する方法を提供します。
%Library.DynamicObject :
Webクライアントとサーバー間を通過できるデータを動的に作成する方法を提供します。
上記のクラスやJSONを操作できるその他のクラスの詳細については、ドキュメントのクラスリファレンスのセクションとアプリケーション開発ガイドをご覧ください。

これらのクラスの主なメリットは、IRISオブジェクトからのJSONの作成とJSONからのオブジェクトの作成を簡単に行えることです。

JSONを使ってファイルの送受信を行う方
法を2つ紹介します。1つ目は%Library.DynamicObject
を使ってクライアント側にオブジ
ェクトを作成してからJSONにシリアル化し、%JSON.Adaptorを使ってサーバー側で
RestTransfer.FileDesc
に逆シリアル化する方法です。そしてもう1つは、手動で送信する文字列を形成し、サーバー側で解析する方法です。 読み取りやすくするために、アプローチごとに、サーバーに別々のメソッドを作成します。

まずは、最初の方法に注目しましょう。

IRISによるJSONの作成

まずは、%JSON.AdaptorからRestTransfer.FileDescクラスを継承しましょう。
こうすることで、インスタンスメソッド%JSONImport()
を使用して、JSONドキュメントを直接オブジェクトに逆シリアル化できます。
このクラスは次のようになります。

```
Class RestTransfer.FileDesc Extends (%Persistent, %JSON.Adaptor)
{
  Property File As %Stream.GlobalBinary;
  Property Name As %String;
}
```

クラスブローカーにUrlMapへの新しいルートを追加しましょう。

```
<Route Url="/jsons" Method="POST" Call="InsertJSONSmall"/>
```

これは、サービスが、URL /RestTransfer/jsonsでPOSTコマンドを受け取ると、InsertJSONSmall
クラスメソッドを呼び出すことを指定しています。

このメソッドは、ファイルのコンテンツが文字列の最大長よりも小さい、key-
valueペアのJSON形式のテキストを受け取ることを期待しています。
オブジェクトのNameプロパティとFile
プロパティを設定し、データベースに保存して、ステータスと、成功またはエラーを示すJSON形式のメッセージ
を返します。

以下は、そのクラスメソッドです。

```
ClassMethod InsertJSONSmall() As %Status
{
  Set result={}
  Set st=0

  set f = ##class(RestTransfer.FileDesc).%New()
  if (f = $$$NULLOREF) {
   do result.%Set("Message", "Couldn't create an instance of class")
  } else {
     set st = f.%JSONImport(%request.Content)
     If $$$ISOK(st) {
        set st = f.%Save()
        If $$$ISOK(st) {
           do result.%Set("Status","OK")
        } else {
           do result.%Set("Message",$system.Status.GetOneErrorText(st))
        }
     } else {
        do result.%Set("Message",$system.Status.GetOneErrorText(st))
     }
  }
  write result.%ToJSON()
  Quit st
}
```

このメソッドは何をするものでしょうか。 %requestオブジェクトのContent
プロパティを取得し、クラス%JSON.Adaptorから継承したメソッド%JSONImport を使用して、
RestTransfer.FileDesc オブジェクトに変換します。

変換中に問題が発生した場合、エラーの説明を含むJSONを作成します。

```
{"Message",$system.Status.GetOneErrorText(st)}
```

そうでない場合は、オブジェクトを保存します。
問題なく保存できたら、JSONの{"Status","OK"}メッセージを作成します。
そうでない場合は、エラーの説明を含むJSONメッセージを作成します。

最後に、このJSONをレスポンスに書き込んで、ステータスstを返します。

クライアント側には、ファイルをサーバーに送信する新しいクラスメソッドSendFileを追加しました。

このコードはSendFileDirect
のコードに非常に似ていますが、ファイルのコンテ
ンツを直接EntityBodyプロパティに書き込む代わりに、クラス%Library.DynamicObject
の新しいインスタンスを作成し、プロパティName
をファイル名に等しい値に設定し、ファイルのコンテンツをプロパティFileに暗号化してコピーします。

ファイルのコンテンツを暗号化するために、 Vitaliy Serdtsevが提案したメソッド
Base64EncodeStream()を使用します。

次に、クラス%Library.DynamicObjectのメソッド%ToJSON
を使
用して、
オブジェクトをJS
ONドキュメントにシリアル化し、リ
クエストの本体に書き込んで、メソッドPost("/RestTransfer/jsons")を呼び出します。

以下は、そのクラスメソッドです。

```
ClassMethod SendFile(aFileName) As %Status
{
  Set sc = $$$OK
  Set request = ..GetLink()
  set s = ##class(%Stream.FileBinary).%New()
  set s.Filename = aFileName
  set p = {}
  set p.Name = s.Filename
  set sc = ..Base64EncodeStream(s, .t)
  Quit:$System.Status.IsError(sc)

  While 't.AtEnd {
    set p.File = p.File_t.Read(.len, .sc)
    Quit:$System.Status.IsError(sc)
  }
  do p.%ToJSON(request.EntityBody)
  Quit:$System.Status.IsError(sc)

  set sc = request.Post("/RestTransfer/jsons")
  Quit:$System.Status.IsError(sc)

  Set response=request.HttpResponse
  do response.OutputToDevice()
  Quit sc
}
```

このメソッドとメソッドSendFileDirectを呼び出して、いくつかの小さなファイルを転送します。

```
 do ##class(RestTransfer.Client).SendFile("D:\Downloads\Outline Template.pdf")
 do ##class(RestTransfer.Client).SendFileDirect("D:\Downloads\Outline Template.pdf")

 do ##class(RestTransfer.Client).SendFile("D:\Downloads\pic3.png")
 do ##class(RestTransfer.Client).SendFileDirect("D:\Downloads\pic3.png")
 do ##class(RestTransfer.Client).SendFile("D:\Downloads\Archive (1).xml")
 do ##class(RestTransfer.Client).SendFileDirect("D:\Downloads\Archive (1).xml")
```

結果は次のようになります。

ご覧のとおり、長さは同じで、これらのファイルをハードドライブに保存すれdば、変更されていないことがわかります。

JSONの手動作成

では、JSONを手動で作成するという2つ目のアプローチに注目しましょう。
これを行うには、クラスブローカーにUrlMapへの新しいルートを追加します。

```
<Route Url="/json" Method="POST" Call="InsertJSON"/>
```

これは、サービスが、URL /RestTransfer/jsonでPOSTコマンドを受け取ると、InsertJSON
クラスメソッドを呼び出すことを指定しています。
このメソッドでは、同じJSONを受け取ると期待していますが、ファイルの長さに制限を掛けません。
以下はそのクラスメソッドです。

```
ClassMethod InsertJSON() As %Status
{
  Set result={}
  Set st=0

  set t = ##class(%Stream.TmpBinary).%New()
  While '%request.Content.AtEnd
  {
    set len = 32000
    set temp = %request.Content.Read(.len, .sc)
    set:len<32000 temp = $extract(temp,1,*-2)
    set st = t.Write($ZCONVERT(temp, "I", "RAW"))

  }
  do t.Rewind()

  set f = ##class(RestTransfer.FileDesc).%New()
  if (f = $$$NULLOREF)
  {
    do result.%Set("Message", "Couldn't create an instance of class")
  } else {
    set str = t.Read()
    set pos = $LOCATE(str,"""","")
    set f.Name = $extract(str, 10, pos-1)
    do f.File.Write($extract(str, pos+11, *))
    While 't.AtEnd {
      do f.File.Write(t.Read(.len, .sc))
    }

    If $$$ISOK(st)
    {
      set st = f.%Save()
      If $$$ISOK(st)
      {
```

```
        do result.%Set("Status","OK")
      } else {
        do result.%Set("Message",$system.Status.GetOneErrorText(st))
      }
    } else {
       do result.%Set("Message",$system.Status.GetOneErrorText(st))
    }
  }
  write result.%ToJSON()
  Quit st
}
```

このメソッドは何をするものでしょうか。
まず、一時ストリームの新しいインスタンス%Stream.TmpBinary
を作成し、それをリクエストのコンテンツにコピーしています。

文字列としてそれを使用するつもりなので、末尾の二重引用符（"）と中括弧（}）を取り除く必要があります。
これを行うために、ス
トリームの最後のチャンクで、最後の2文字を取り残しておきます（$extract(temp,1,*-2)）。

同時に、「RAW」変換テーブルを使用して、文字列を変換します（$ZCONVERT(temp, "I", "RAW")）。

次に、クラスRestTransfer.FileDescの新しいインスタンスを作成し、他のメソッドと同じチェックを行います。
自分の文字列の構造をわかっているため、ファイルの名前とファイル自体を抽出し、それらを対応するプロパティに設定します。

クライアント側では、クラスメソッドSendFile
を変更し、JSONを作成する前にファイルの長さをチェックします。
2,000,000バイト未満（明らかに%JSONImportの制限）である場合は、
Post("/RestTransfer/jsons")を呼び出します。 そうでない場合は、Post("/RestTransfer/json")を呼び出します。

すると、メソッドは次のようになります。

```
ClassMethod SendFile(aFileName) As %Status
{
  Set sc = $$$OK
  Set request = ..GetLink()
  set s = ##class(%Stream.FileBinary).%New()
  set s.Filename = aFileName
  if s.Size > 2000000 //3641144 max length of the string in IRIS
  {
    do request.EntityBody.Write("{""Name"":""_s.Filename_""", ""File"":""")
    While 's.AtEnd
    {
      set temp = s.Read(.len, .sc)
      do request.EntityBody.Write($ZCONVERT(temp, "O", "RAW"))
      Quit:$System.Status.IsError(sc)
    }

    do request.EntityBody.Write("""}")
    set sc = request.Post("/RestTransfer/json")
  } else {
    set p = {}
    set p.Name = s.Filename
    set sc = ..Base64EncodeStream(s, .t)
    Quit:$System.Status.IsError(sc)
```

```
    While 't.AtEnd {
        set p.File = p.File_t.Read(.len, .sc)
        Quit:$System.Status.IsError(sc)
    }
    do p.%ToJSON(request.EntityBody)
    Quit:$System.Status.IsError(sc)
    set sc = request.Post("/RestTransfer/jsons")

  }

  Quit:$System.Status.IsError(sc)
  Set response=request.HttpResponse
  do response.OutputToDevice()
  Quit sc
}
```

2つ目のメソッドを呼び出すために、手動でJSONを含む文字列を作成します。 そしてファイル自体を転送するために、クライアント側では「RAW」変換テーブルを使用してコンテンツを変換（($ZCONVERT(temp, "O", "RAW")）します。

次に、このメソッドとメソッドSendFileDirectを呼び出して、さまざまなサイズのファイルを転送します。

```
 do ##class(RestTransfer.Client).SendFile(“D:\Downloads\pic3.png”)
 do ##class(RestTransfer.Client).SendFileDirect(“D:\Downloads\pic3.png”)
 do ##class(RestTransfer.Client).SendFile(“D:\Downloads\Archive (1).xml”)
 do ##class(RestTransfer.Client).SendFileDirect(“D:\Downloads\Archive (1).xml”)
 do ##class(RestTransfer.Client).SendFile(“D:\Downloads\Imagine Dragons-
Thunder.mp3”)
 do ##class(RestTransfer.Client).SendFileDirect(“D:\Downloads\Imagine Dragons-
Thunder.mp3”)
 do ##class(RestTransfer.Client).SendFile(“D:\Downloads\ffmpeg-win-2.2.2.exe”)
 do ##class(RestTransfer.Client).SendFileDirect(“D:\Downloads\ffmpeg-win-2.2.2.exe”)
```

結果は次のようになります。

長さが同じであるため、すべてが想定どおりに動作していることがわかります。

まとめ

繰り返しになりますが、RESTサービスの作成に関する詳細については、ドキュメントをご覧ください。
両アプローチのサンプルコードは、GitHubとInterSystems Open Exchangeにあります。

連載の最初の記事で取り上げたTWAINモジュールの結果の転送については、受信するデータのサイズによって異なります。

最大解像度が制限されており、フ
ァイルが小さいのであれば、システムクラスの%JSON.Adaptorと
%Library.DynamicObjectを使用することができますが、
念のために、POSTコマンドの本体に直接ファイルを送信するか、JSONを手動で作成することをお勧めします。
また、範囲リクエストを使用して大きなファイルを分割し、個別に送信して、サーバー側で1つのファイルに統合し直すこともお勧めします。

いずれにしても、質問や提案があれば、お気軽にコメントセクションに書き込んでください。

#REST API #InterSystems IRIS
InterSystems Open Exchangeで関連アプリケーションを確認してください

ソースURL:
https://jp.community.intersystems.com/post/rest%E7%B5%8C%E7%94%B1%E3%81%A7%E3%83%95%E3%82%A1%E3%82%A4%E3%83%AB%E8%BB%A2%E9%80%81%E3%81%97%E3%83%97%E3%83%AD%E3%83%91%E3%83%86%E3%82%A3%E3%81%AB%E6%A0%BC%E7%B4%8D%E3%81%99%E3%82%8B-%E3%83%91%E3%83%BC%E3%83%882