記事
Toshihiko Minamoto · 2021年11月3日 12m 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3

この連載の最初の記事では、大き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クラスとしてデータベースに格納する方法を説明しました。
2つ目の記事では、ファイルとファイル名をJSON形式にラップして送信する方法を説明しました。

それでは、大きなファイルを分割してサーバーに送るという構想を詳しく見ていきましょう。
これを行うために使用できるアプローチにはいくつかあるのですが、 この記事では、Transfer-Encodingヘッダーを使用してチャンク転送を指示する方法を説明します。 Transfer-EncodingヘッダーはHTTP/1.1仕様で導入されたものです。RFC 7230第4.1項では説明されているものの、HTTP/2仕様からはその説明が無くなっています。

Transfer-Encoding（転送符号法）ヘッダー

Transfer-Encodingヘッダーは、ペイロード本体をユーザーに安全に転送するために使用されるエンコードの形式を指定することを目的としています。 主に、動的に生成されたペイロードを正確に区切るため、そして選択されたリソースの特性から、転送効率のためのペイロードエンコードであるのか、セキュリティのためのペイロードエンコードであるのかを区別するために使用します。

このヘッダーでは次の値を使用できます。

- Chunked
- Compress
- Deflate
- gzip

Transfer-EncodingがChunkedである場合

Transfer-EncodingをChunkedに設定した場合、メッセージの本文は不特定の数の通常のチャンク、終了チャンク、トレーラー、および最後の行頭復帰・改行（CRLF）シーケンスで構成されます。

各部分は、16進数で表現されるチャンクサイズで始まり、オプションの拡張とCRLFが続きます。
その後には、チャンクの本体と最後にCRLFが続きます。 拡張にはチャンクのメタデータが含まれます。
たとえば、メタデータには署名、ハッシュ、メッセージの中途を制御する情報などが含まれることがあります。
終了チャンクは長さがゼロの通常のチャンクです。
（おそらく空の）ヘッダーフィールドで構成されるトレーラーは、終了チャンクの後に続きます。

想像しやすくするために、以下に「Transfer-Encoding = chunked」を使ってメッセージの構造を示します。

chunked_body	*chunk last_chunk trailer_part CRLF
chunk	chunk_size [chunk_ext] CRLF chunk_data CRLF
chunk_size	*size-of-current-chunk-in-HEX*
chunk_ext	*(";" chunk_ext_name ["=" chunk_ext_val])
chunk_ext_name	*token*
chunk_ext_val	*token / quoted-string*
chunk_data	*contents-of-current-chunk*
last_chunk	1*("0") [chunk_ext] CRLF
trailer_part	*(header_field CRLF)

簡単なチャンク化メッセージの例は次のようになります。

```
13\r\n
Transferring Files \r\n
4\r\n
on\r\n
1A\r\n
community.intersystems.com
0\r\n
\r\n
```

このメッセージ本文は、3つの有意義なチャンクで構成されています。
最初のチャンクの長さは19オクテット、2つ目は4オクテット、そして3つ目は26オクテットです。
チャンクの終わりを示す末尾のCRLFはこのチャンクサイズに含まれないことがわかるでしょう。 ただし、CRLFを行末（EOL）マーカーとして使用する場合は、そのCRLFはメッセージの一部として考慮され、2オクテットとなります。 デコードされたメッセージは次のようになります。

```
Transferring Files on
community.intersystems.com
```

IRISでのチャンク化メッセージの作成

このチュートリアルでは、最初の記事で作成したサーバーのメソッドを使用します。
つまり、ファイルのコンテンツを直接POSTメソッドの本体に送信することになります。
ファイルのコンテンツを本体で送信するため、POSTをhttp://webserver/RestTransfer/fileに送信します。

では、IRISでチャンク化メッセージを作成する方法を見てみましょう。
HTTP/1.1を使用しているのであれば、「チャンク化リクエストの送信
」のセクションの「HTTPリクエストの送信」で説明されるとおり、HTTPリクエストをチャンクで送信することができます。
このプロセスの最も良いところは、%Net.HttpRequest
がメッセージ本文全体のコンテンツの長さをサーバー側で自動的に計算するため、サーバー側で何かを変更する必要がまったくないことです。
したがって、チャンク化されたリクエストを送信するには、クライアントでのみ次の手順に従う必要があります。

最初のステップは、%Net.ChunkedWriterのサブクラスを作成してOutputStreamメソッドを実装することです。 このメソッドはデータのストリームを取得し、それを調べて、分割するかどうかと分割の方法を決定し、継承されたクラスのメソッドを呼び出して出力に書き込みます。
この場合、クラスRestTransfer.ChunkedWriterを呼び出します。

次に、データの送信を行うクライアント側のメソッド（ここでは「SendFileChunked」と呼びます）で、RestTransfer.ChunkedWriterクラスのインスタンスを作成して、送信するリクエストデータを入力する必要があります。 ファイルを送信しようとしているので、面倒な作業はすべてRestTransfer.ChunkedWriterクラスで行うようにします。 Filename As %Stringというプロパティと「MAXSIZEOFCHUNK = 10000」というパラメーターを追加します。 もちろん、チャンクの最大許容サイズをプロパティとして設定し、ファイルまたはメッセージごとに設定することもできます。

最後に、%Net.HttpRequestのEntityBodyプロパティが作成したRestTransfer.ChunkedWriterクラスのインスタンスと等しくなるように設定すれば、準備完了です。

これらの手順は、ファイルをサーバーに送信する既存のメソッドに書き込んだり置換したりする必要のある新しいコードにすぎません。

このメソッドは次のようになります。

```
ClassMethod SendFileChunked(aFileName) As %Status
{
  Set sc = $$$OK
  Set request = ..GetLink()
  set cw = ##class(RestTransfer.ChunkedWriter).%New()
  set cw.Filename = aFileName
  set request.EntityBody = cw
  set sc = request.Post("/RestTransfer/file")
  Quit:$System.Status.IsError(sc) sc
  Set response=request.HttpResponse
  do response.OutputToDevice()
  Quit sc
}
```

%Net.ChunkedWriterクラスは、インターフェースを提供し、いくつかの実装済みメソッドとプロパティを持つ抽象ストリームクラスです。 ここでは、次のプロパティとメソッドを使用します。

- プロパティTranslateTable as %Stringは、チャンクを出力ストリーム（EntityBody）に書き込むときに、チャンクの自動変換を強制します。 Rawデータを受け取ることを期待しているため、TranslateTableを“ RAW ”に設定する必要があります。
- メソッドOutputStreamは、すべてのチャンク化操作を行うために、サブクラスによってオーバーライドされる抽象メソッドです。

- メソッドWriteSingleChunk(buffer As %String)は、Content-Length HTTPヘッダーとそれに続くエンティティ本体を単一のチャンクとして書き込みます。 ファイルのサイズがMAXSIZEOFCHUNKメソッドよりも小さいかどうかを確認し、小さい場合には、このメソッドを使用します。
- メソッドWriteFirstChunk(buffer As %String)は、Transfer-Encodingヘッダーとそれに続く最初のチャンクを書き込みます。 必ず存在する必要があります。 この後にさらにチャンクを書き込むため、0回以上の呼び出しが行われる可能性がありますが、その後、空の文字列を持つ最後のチャンクを書き込む強制的な呼び出しが行われます。 ファイルの長さがMAXSIZEOFCHUNKメソッドを超えることを確認したら、このメソッドを呼び出します。
- メソッドWriteChunk(buffer As %String)は結果として得たチャンクを書き込みます。 最初のチャンクの後の残りのファイルが依然としてMAXSIZEOFCHUNKを上回るかを確認してから、このメソッドを使用してデータを送信します。
  ファイルの最後の部分のサイズがMAXSIZEOFCHUNKよりも小さくなるまで、この作業を繰り返します。

- メソッドWriteLastChunk(buffer As %String)は、最後のチャンクと、それに続く長さゼロのチャンクを書き込み、データの終わりをマークします。

上記のすべてを基にすると、クラスRestTransfer.ChunkedWriterは次のようになります。

```
Class RestTransfer.ChunkedWriter Extends %Net.ChunkedWriter
{
  Parameter MAXSIZEOFCHUNK = 10000;
  Property Filename As %String;
  Method OutputStream()
  {
    set ..TranslateTable = "RAW"
    set cTime = $zdatetime($Now(), 8, 1)
    set fStream = ##class(%Stream.FileBinary).%New()
    set fStream.Filename = ..Filename
    set size = fStream.Size
    if size &lt; ..#MAXSIZEOFCHUNK {
      set buf = fStream.Read(.size, .st)
      if $$$ISERR(st)
      {
        THROW st
      } else {
        set ^log(cTime, ..Filename) = size
        do ..WriteSingleChunk(buf)
      }
    } else {
      set ^log(cTime, ..Filename, 0) = size
      set len = ..#MAXSIZEOFCHUNK
      set buf = fStream.Read(.len, .st)
      if $$$ISERR(st)
      {
        THROW st
      } else {
        set ^log(cTime, ..Filename, 1) = len
        do ..WriteFirstChunk(buf)
      }
      set i = 2
      While 'fStream.AtEnd {
        set len = ..#MAXSIZEOFCHUNK
        set temp = fStream.Read(.len, .sc)
    if len&lt;..#MAXSIZEOFCHUNK
    {
      do ..WriteLastChunk(temp)
    } else {
          do ..WriteChunk(temp)
        }
        set ^log(cTime, ..Filename, i) = len
        set i = $increment(i)
      }
    }
  }
}
```

これらのメソッドがファイルをどのように分割しているかを確認するために、次の構造でグローバル^logを追加します。

```
//??????????????
```

```
^log(time, filename) = size_of_the_file
//?????????????
^log(time, filename, 0) = size_of_the_file
^log(time, filename, idx) = size_of_the_idx’s_chunk
```

プログラミングが完了したので、これら3つのアプローチがさまざまなファイルでどのように機能するのかを見てみましょう。 サーバーを呼び出すための単純なクラスメソッドを記述します。

```
ClassMethod Run()
{
  // ??????????????
  kill ^RestTransfer.FileDescD
  kill ^RestTransfer.FileDescS
  // ????????????????????
  for filename = "D:\Downloads\wiresharkOutput.txt", // 856 ???
     "D:\Downloads\wiresharkOutput.pdf", // 60 134 ???
     "D:\Downloads\Wireshark-win64-3.4.7.exe", // 71 354 272 ???
     "D:\Downloads\IRIS_Community-2021.1.0.215.0-win_x64.exe" //542 370 224 bytes
  {
    write !, !, filename, !, !
    // ??????????????????3???????????????????
    set resp1=##class(RestTransfer.Client).SendFileChunked(filename)
    if $$$ISERR(resp1) do $System.OBJ.DisplayError(resp1)
    set resp1=##class(RestTransfer.Client).SendFile(filename)
    if $$$ISERR(resp1) do $System.OBJ.DisplayError(resp1)
    set resp1=##class(RestTransfer.Client).SendFileDirect(filename)
    if $$$ISERR(resp1) do $System.OBJ.DisplayError(resp1)
  }
}
```

クラスメソッドRunを実行した後、最初の3つのファイルの出力では、ステータスは正常となりました。
しかし、最後のファイルでは、最初と最後の呼び出しは動作するにもかかわらず、真ん中の呼び出しはエラー: 5922を返しました。これは応答待ちのタイムアウトです。
globalsメソッドを見ると、コードが11番目のファイルを保存しなかったことがわかります。 つまり、##class(RestTransfer.Client).SendFile(filename)が失敗しています。正確に言えば、JSONからデータを取り出すメソッドが成功しなかったということです。

ここで、ストリームを見ると、正常に保存されたファイルのサイズがすべて正しいことがわかります。

^logグローバルを見ると、各ファイルに対してコードが作成したチャンク数がわかります。

おそらく、実際のメッセージの本文を確認したいところでしょう。 Eduard

Lebedyukは、「Webをデバッグする」という記事の中で、CSP
ゲートウェイロギングとトレーシングを使用できると提案しています。

イベントログで2つ目のチャンクファイルを見ると、Transfer-
Encodingヘッダーの値が実際に「chunked」となっていることがわかります。 残念ながら、サーバーはすでにメッセージを接合してしまっているため、実際のチャンクを確認することはできません。

トレース機能を使用しても、それ以上の情報はあまり表示されませんが、最後から2番目と最後のリクエストの間にギャップがあることが明らかになります。

メッセージの実際の部分を確認するには、クライアントを別のコンピューターにコピーして、ネットワークスニファーを使用します。
ここでは、Wiresharkを選択しました。これは無料のツールで、必要な機能が揃っているためです。 コードがファイルをチャンクに分割する方法をわかりやすく示すには、MAXSIZEOFCHUNKの値を100に変更して、小さなファイルを送信することができます。 すると、次のような結果が表示されます。

最後の2つのチャンクを除くすべてのチャンクの長さがHEXの64（DECの100）に等しく、データのある最後のチャンクは21 DEC（HEXでは15）であることがわかります。また、最後のチャンクのサイズがゼロであることを確認できます。 すべては正常にみえるため、仕様に合致しています。
ファイルの全長さは421（4x100+1x21）に等しく、これをグロバールで確認することもできます。

まとめ

総合的に、このアプローチは動作し、大きなファイルを問題なくサーバーに送信できることがわかります。
さらに、大量のデータをクライアントに送信する場合は、Web ゲートウェイの動作と構成
の「アプリケーション・パスの構成パラメータ」セクションにあるパラメーター「応答サイズの通知」をよく読むことをお勧めします。 これは、使用するHTTPのバージョンに応じて、大量のデータを送信する際のWebゲートウェイの動作を指定するパラメーターです。

このアプローチのコードは、GitHubとInterSystems Open Exchange
にある、この例の前のバージョンに追加されています。

ファイルをチャンクで送信するというトピックでは、Transfer-Encodingヘッダーの有無に関わらずContent-Rangeヘッダーを使用して、データのどの部分が転送されているのかを示すことも可能です。
さらに、HTTP/2仕様で利用できる、まったく新しいストリームの概念を使用することができます。

いつものように、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%883