記事
Toshihiko Minamoto · 2021年6月8日 17m read

サポートの現場から - Raw DEFLATEの圧縮/解凍機能の探求から、どのようにRESTによるNode.jsのコールアウトサーバー構築に至ったのか

以前、WRCケースのエスカレーショ
ンを受けました。お客様は、Caché に、**raw**DEFLATE圧縮/解凍機能が組み込まれているかを尋ねていました。

DEFLATEについて話すには、Zlibについても話す必要があります。Zlibは、90年代半ばに開発された無料の圧縮/解凍ライブラリで、、デファクトスタンダードとなっているからです。

Zlibは特定のDEFLATE圧縮/解凍アルゴリズムと、ラッパー（gzip、zlibなど）内でのカプセル化するという考えの下で動作します。
https://en.wikipedia.org/wiki/Zlib

Caché Object Script（COS）ではすでにGZIPがサポートされており、gzipファイルを使用するために、ファイルデバイスまたはtcpデバイス、またはStreamclassで/GZIP=1を使用できるようになっています。
http://docs.intersystems.com/latest/csp/docbook/DocBook.UI.Page.cls?KEY=GOBJpropstreamgzip

「CSP-Gateway/Web-Gateway」Webサーバーモジュールでも、Caché-Serverから受信するhttp-data送信をGZIP圧縮/解凍するために、Zlibライブラリが使用されています。 （CSP、Zen、SOAP、RESTなど）

しかし、GZIP形式には、DEFLATEで圧縮された生の本文にラップされた追加のヘッダーとトレーラーが含まれていました。

お客様望んでいるのはこれではありません。
お客様のユースケースでは、DEFLATEで圧縮された生のコンテンツの作成・解凍**のみ**を行う必要がありました。

これはZlibライブラリではサポートされていますが、Caché API/関数内からは現在公開されていません。

追加するにはどうすればいいでしょうか？

「どうにかしてZlibライブラリにアクセスする必要があります。」

コールアウトを使って、Caché内からZlibを利用できるようにすることはできないだろうか？

「はい、可能です。」

Cachéのコールアウトを使うと、C/C++呼び出し規則をサポートするほかの言語で書かれたほかのライブラリ（WindowsのDLL、UnixのSO）から、実行可能ファイル、オペレーティングシステムのコマンド、または関数を呼び出すことができます。

Cachéコールアウトは、$ZF関数で提供されています。
http://docs.intersystems.com/latest/csp/docbook/DocBook.UI.Page.cls?KEY=BGCLをご覧ください。

たとえば、オペレーティングシステムのコマンドを発行する場合、$ZF(-1) と$ZF(-2) 関数を使用できます。

$ZF(-1, command) はプログラムまたはオペレーティングシステムのコマンドを、生成された子プロセスとして実行し、その子プロセスがexitステータスを返すまで、現在のプロセスの実行を一時的に停止しますが、$ZF(-2, command) は、非同期的に動作するため、生成された子プロセスが完了するのを待ちません。そのためそのプロセスから直接ステータス情報を受信することができません。

もう1つの方法は、オペレーティングシステムレ
ベルと同様に、**コマンド・パイプ**を使用してプロセスと通信する方法です。 ここでは、パイプを介して出力を送信してプロセスを制御し、パイプを読み取って入力を受信し、プロセス出力を取得することができます。
http://docs.intersystems.com/latest/csp/docbook/DocBook.UI.Page.cls?KEY=GIODipcpipes

注意: 今後、$ZFとパイプを使ったCachéコールアウトの仕組みのサポートを終了し、より安全なコールアウト方法に置き換える予定です。 最新情報をお楽しみに。

私はWeb専門のエンジニアなので、JavaScriptを好んで使用しています。 しかし、皆さんもよくご存知のとおり、Webページ
のコンテキストにおい
てWebブラウザでクライアントがJavaScriptを実行するのではなく、**サーバー**で実行するための何かが必要です。

一般的に使用されている人気の高いJavaScriptサーバーのランタイム環境/エンジンは、**Node.js**です。
これは、ChromeのV8
JavaScriptエンジンを基礎に構築された、コミュニティが主導するJavaScriptランタイム環境です。
Node.jsは、イベント駆動型のノンブロッキング非同期I/Oモデルを使用しており、軽量で非常に効率的です。
https://nodejs.org/en/

幸いなことに、Node.jsにはzlibモジュールが含まれており、私たちの計画に最適です。
https://nodejs.org/api/zlib.html

CachéもNode.jsをサポートしていますが、わずかに異なります。
強力な**cache.node**
コネクタ/インターフェースが備わっているため、Caché内のデータとメソッドをNode.jsから簡単に利用することができます。 http://docs.intersystems.com/latest/csp/docbook/DocBook.UI.Page.cls?KEY=BXJS

この特定のユースケースと要件の場合、私たちが求めているものは**このことではありません**。

Node.jsを介して純粋なJavaScriptを実行し、Cachéに結果を返す必要があるからです。
つまり、これは逆方向になります。

前述のコマンドパイプのコールアウトの仕組みが適しているようです。

Node.jsをダウンロードしてインストールしたら、このプランが機能するかどうかを試してみましょう。

```
USER>set cmd="node -v",oldIO=$IO open cmd:"QR" use cmd read result close prog use old
IO
USER>w result
v8.9.1
```

このテストでわかるように、期待どおりに動作しています。 「node-v」コマンドによって、現在インストールされているNode.jsのランタイム環境に関するバージョン情報が返されています。

「できました！」

ここで、指定されたコマンドラインの引数から、zlibモジュールと生のDEFLATE/INFLATEアルゴリズムを使って、ファイルのコンテンツを圧縮/解凍するノードスクリプトをコーディングしましょう。

これは簡単に行えます。 次のコードを使って、プロジェクトフォルダに*zlib.js*を作成します。

```
//zlib.js
Const
  func = process.argv[2],
  infile = process.argv[3],
  outfile = process.argv[4];
```

```
const zlib = require('zlib');
const fs = require('fs');

if (func=='DEFLATERAW') {
  var wobj = zlib.createDeflateRaw();
}
else  {
  var wobj = zlib.createInflateRaw();
}

const instream = fs.createReadStream(infile);
const outstream = fs.createWriteStream(outfile);

instream.pipe(wobj).pipe(outstream);

console.log(func + ' ' + infile + ' -> ' + outfile);
```

このスクリプトは、次のようなコマンドを使ってOSコンソールから実行し、生のDEFLATEを使用して、既存のinput.txtファイルをoutput.zzに圧縮できます。

```
C:\projects\zlib>node zlib.js DEFLATERAW input.txt output.zz
DEFLATERAW input.txt -> output.zz
```

注意: 便宜上、このコードは、ノードスクリプトが実行しているフォルダ（c:\projects\zlibなど）にあるファイルの圧縮/解凍のみをサポートしています。
そのため、少なくともinput.txtファイルをこの場所に作成するかコピーするようにしてください。

最初に、スクリプトコードは、「zlib」（Zlibライブラリノード）と「fs」（ファイルアクセス/操作用のファイルシステムノード）モジュールの機能を使用するために、これらを配置しています。

その後で、*process.argv*を使って、受信するコマンドライン引数にアクセスしています。
*argv*は「引数ベクトル（argument vector）」の略で、最初の2つの要素である「node」とスクリプトファイルへのフルパスを含む配列です。 3つ目の要素（インデックス2）は「関数/アルゴリズム名」、4つ目と5つ目の要素（インデックス3と4）は、入力ファイルの引数「infile」と出力ファイルの引数「outfile」です。

最後に、パイプ処理を使用して、入力ファイルストリームと出力ファイルストリームの両方に適切なzlibメソッドを使用しています。

関数の結果を戻すには、コンソールで結果メッセージを出力するだけです。

「以上です。」

これがCachéから動作するか試してみましょう。

```
USER>set cmd="node c:\projects\zlib\zlib.js DEFLATERAW input.txt output.zz",oldIO=$IO
 open cmd:"QR" use cmd read result close cmd use oldIO
USER>w result
DEFLATERAW input.txt -> output.zz
```

「期待どおりに動作しました。」

次のコマンドを使用して、圧縮済みのoutput.zzファイルをoutput.txtに解凍（inflate）することができます。

```
USER>Set cmd="node c:\projects\zlib\zlib.js INFLATERAW output.zz output.txt",...
```

こうすると、output.txtファイルのコンテンツとサイズは、input.txtファイルと全く同じになります。

「問題は解決しました。」

コマンドパイプを使用して、ノードスクリプトへのコールアウトにより、Caché でファイルの生のDEFLATE圧縮/解凍を利用できるようになりました。

しかし、パフォーマンスの観点から、考慮すべきことがあります。コールアウトの仕組みには、コールアウトのたびに新しい子プロセスが起動されるというオーバーヘッドが伴います。

パフォーマンスが重要でない場合、または完了すべき処理作業が時間のかかるものである場合、ファイルサイズが大きくなっても圧縮と解凍は問題はないでしょうし、プロセスの開始にかかる時間のオーバーヘッドも無視できます。 しかし、多数の比較的小さなファイルを次々と集中して圧縮/解凍するのであれば、このオーバーヘッドは回避した方が良いと言えます。

では、どのようにして回避すればよいのでしょうか。

コールアウトが行われるたびに新しい子プロセスが作成されないようにする必要があります。

「どうすればよいですか？」

受信するリクエストをリスンし、要求どおりに目
的の操作を行う**サーバー**として実行するように、スクリプトを準備する必要があります。

もっともらしく聞き覚えのある方法でしょうか。そうです。これが現在、**RESTful HTTP API/サービス**で行えるとされている方法です。

Node.jsであれば、HTTPプロトコルに基づく単純なサーバーを非常に簡単に作成できます。

Node.jsには、組み込みの「http」モジュールを使用して、すぐに使用できるオーバーヘッドの低いHTTPサーバーがサポートされています。

「http」モジュールを含めるには、以下に示されるよ
うに、*simplehttps.js* スクリプトファイルに、通常どおりノードのrequire() メソッドを使用します。

```
//simple_https.js
const
  http = require('http'),
  server = http.createServer(function (request, response) {
    response.writeHead(200, {'Content-Type' : 'text/plain'});
    response.end('Hello World!\n');
  });
server.listen(3000, function(){
    console.log('ready captain!');
});
```

この単純なhttpサーバーをOSコンソールから起動するには、次のコマンドを使用します。

```
C:\projects\zlib>node simple_http.js
ready captain!
```

ここでは、「**curl**」を使って、これをテストしています。curlは一般的に使用される便利なコマンドラインツールで、HTTPリクエ

ストを特定のサーバーに発行します。
https://curl.haxx.se/

「-i」フラグを追加して、レスポンス本文のほかにHTTPヘッダーを出力する必要があるということを、curlに指示しています。

```
C:\curl>curl -i http://localhost:3000
HTTP/1.1 200 OK
Content-Type: text/plain
Date: Mon, 22 Jan 2018 13:07:06 GMT
Connection: keep-alive
Transfer-Encoding: chunked

Hello World!
```

これはうまく動作しますが、低レベルの「http」モジュールに対してhttpサービスを直接記述するのは面倒であり、時間のかかる作業です。

Node.jsには活発に活動しているオープンソースコミュニティがあり、Node.jsアプリケーションに機能を追加できる優れたモジュールがたくさん作成されているため、このケースのRESTful APIを開発するには「**Express**」を使用することにします。

「**Express.js**」または単に「**Express**」とも呼ばれるモジュールは、WebアプリとAPIを構築するために設計された、Node.js用のWebアプリケーションフレームワークです。

無ければ結局は自分で書くことになるのですが、その手間を省くことのできる多数の配管コードが提供されています。 URLパスに基づいて受信リクエストをルーティングしたり、受信データを解析したり、不正な形式のリクエストを拒否したりなどすることができます。

Expressフレームワークを使えば、こういったタスクやその他無数のタスクを実現できるようになります。
Node.jsの標準的なサーバーフレームワークであるのも、当然でしょう。
http://expressjs.com/

すべてのNodeモジュールと同様に、「Express」を使用するには、まずnpm（ノードパッケージマネージャー）を使ってインストールしなければ、使うことはできません。

```
C:\projects\zlib>node install express
...
```

「express」とそのほかに必要なモジュールを含めるには、以下に示されるように、*zlibserver.js* スクリプトファイルに、通常どおりノードのrequire() メソッドを使用します。

```
//zslibserver.js
const express = require('express');
const zlib = require('zlib');
const fs = require('fs');

var app = express();

 app.get('/zlibapi/:func/:infile/:outfile', function(req, res) {
    res.type('application/json');

    var infile=req.params.infile;
```

```
    var outfile=req.params.outfile;

    try {

        var stats = fs.statSync(infile);
        var infileSize = stats.size;

        switch(req.params.func) {
          case "DEFLATERAW":
            var wobj = zlib.createDeflateRaw();
            break;
          case "INFLATERAW":
            var wobj = zlib.createInflateRaw();
            break;
          case "DEFLATE":
            var wobj = zlib.createDeflate();
            break;
          case "INFLATE":
            var wobj = zlib.createInflate();
            break;
          case "GZIP":
            var wobj=zlib.createGzip();
            break;
          case "GUNZIP":
            var wobj=zlib.createGunzip();
            break;
          default:
            res.status(500).json({ "error" : "bad function" });
            return;
        }

        const instream = fs.createReadStream(infile);
        const outstream = fs.createWriteStream(outfile);

        var d = new Date();
        console.log(d.toLocaleDateString() + ' ' + d.toLocaleTimeString() + ' : ' + r
eq.params.func + ' ' + infile + ' -> ' + outfile + '...');

        instream.pipe(wobj).pipe(outstream).on('finish', function(){

            var d = new Date();
            console.log(d.toLocaleDateString() + ' ' + d.toLocaleTimeString() + ' : '
 + 'finished!');

            var stats = fs.statSync(outfile);
            var outfileSize = stats.size

            res.status(200).json( { "result" : "OK" , "infileSize" : infileSize, "out
fileSize" : outfileSize, "ratio" : (outfileSize / infileSize * 100).toFixed(2) + "%"
} );
            return;
      });

    }
    catch(err) {
      res.status(500).json({ "error" : err.message});
      return;
    }
```

```
});
app.listen(3000, function(){
    console.log("zlibserver is ready captain.");
});
```

まず、「zlib」、「fs」、および「express」モジュールを取り込み、expressの「app」（アプリケーション）コンテキストを作成しています。

Expressの機能は、リクエストオブジェクトとレスポンスオブジェクトを操作して処理を行える非同期のミドルウェア関数を介して提供されます。

app.get() では、Expressに、*ルートの /zlibapi/:func/:infile/:outfile*パスへのHTTP GETリクエストを**処理**するのかを指示しています。 app.get() を使用すると、ルート/パスに複数のハンドラを登録することができます。
パスの「*:variable*」のチャンクは「名前付きルートパラメーター」と呼ばれます。
APIがヒットしたら、expressはURLのその部分を取得し、req.paramsで使用できるようにします。

コードには、RAWDEFLATE/RAWINFLATEのほかに、zlibがサポートする圧縮/解凍ラッパー形式のGZIP/GUZIPやDEFLATE/INFLATEのサポートが追加されています。

また、出発点として、基本的なTry/Catchエラー処理も追加しました。

結果とともにJSONオブジェクトを送り返すには、レスポンスオブジェクトのresと、res.sendStatus() に相当するres.status() を使用しています。
詳細については、Expressのドキュメントをご覧ください。

最後に、受信するHTTPリクエストのリスンをTCPポート3000で開始しています。

それでは、「zlibserver」アプリを実行して、実際に動作するかを見てみましょう。

```
C:\projects\zlib>node zlibserver.js
zlibserver is ready captain.
```

実行することがわかったので、サービスとして使用してみることにします。

ここではCachéから試してみますが、「curl」やその他の「Postman」などのサードパーティツールを使って、「zlibserver」RESTful APIをテストすることもできます。

*%Net.HttpRequest*を使用して、Caché COSに、GETリクエストを実行する単純なRESTクライアントを実装する必要があります。これはそれほど手間のかからない作業ですが、コードを数行、記述する必要があります。
以下は、クラスの*utils.Http:getJSON()* メソッドです。

```
Include %occErrors
Class utils.Http [ Abstract ]
{
  ClassMethod getJSON(server As %String = "localhost", port As %String = "3000", url
As %String = "",
    user As %String = "", pwd As %String = "", test As %Boolean = 0) As %DynamicAbstr
actObject
  {
     set prevSLang=##class(%Library.MessageDictionary).SetSessionLanguage("en")

     set httprequest=##class(%Net.HttpRequest).%New()
     set httprequest.Server=server
     set httprequest.Port=port

     if user'="" do httprequest.SetParam("CacheUserName",user)
```

```
     if pwd'="" do httprequest.SetParam("CachePassword",pwd)

     set sc=httprequest.SetHeader("Accept","application/json")
     if $$$ISERR(sc) $$$ThrowStatus(sc)
     set sc=httprequest.SetHeader("ContentType","application/json")
     if $$$ISERR(sc) $$$ThrowStatus(sc)

     try {
         set sc=httprequest.Get(url,test)
         if $$$ISERR(sc) $$$ThrowStatus(sc)
         if (httprequest.HttpResponse.StatusCode \ 100) = 2 {
             set response = ##class(%DynamicAbstractObject).%FromJSON(httprequest.Htt
pResponse.Data)
         }
         else {
             Throw ##class(%Exception.General).%New(httprequest.HttpResponse.ReasonPh
rase, $$$GeneralError,,httprequest.HttpResponse.StatusLine)
         }
     }
     catch exception  {
         set response = $$$NULLOREF
         throw exception
     }
     Quit response
  }
}
```

Cachéから、次のようにして使用することができます。

```
USER>try { set res="",res = ##class(utils.Http).getJSON(,,"/zlibapi/DEFLATERAW/input.
txt/output.zz"),result=res.result } catch (exc) { Set result=$system.Status.GetOneErr
orText(exc.AsStatus()) }
USER>w result
OK
USER>w res.%ToJSON()
{"result":"OK","infileSize":241243,"outfileSize":14651,"ratio":"6.07%"}
```

「うまく動作しました！」

以下は、curlを使ったAPIのテスト方法です（エクステントtest.logファイルを使用しています）。

```
C:\curl>curl -i http://localhost:3000/zlibapi/GZIP/test.log/test.gz

HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 76
ETag: W/"4c-iaOk5W3g6IlIEkzJaRbf3EmxrKs"
Date: Fri, 26 Jan 2018 07:43:17 GMT
Connection: keep-alive

{"result":"OK","infileSize":36771660,"outfileSize":8951176,"ratio":"24.34%"}

C:\curl>curl -i http://localhost:3000/zlibapi/GUNZIP/test.gz/test.txt

HTTP/1.1 200 OK
```

```
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 77
ETag: W/"4d-tGgowYnW3G9ctHKcpvWmnMgnUHM"
Date: Fri, 26 Jan 2018 07:43:36 GMT
Connection: keep-alive

{"result":"OK","infileSize":8951176,"outfileSize":36771660,"ratio":"410.80%"}

C:\curl>curl -i http://localhost:3000/zlibapi/DEFLATERAW/test.log/test.zz

HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 76
ETag: W/"4c-4svUs7nFvjwm/JjYrPrSSwhDklU"
Date: Fri, 26 Jan 2018 07:44:26 GMT
Connection: keep-alive

{"result":"OK","infileSize":36771660,"outfileSize":8951158,"ratio":"24.34%"}

C:\curl>curl -i http://localhost:3000/zlibapi/INFLATERAW/test.zz/test.txt

HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 77
ETag: W/"4d-7s7jwh1nxCU+6Qi7nX2TB3Q1IzA"
Date: Fri, 26 Jan 2018 07:44:42 GMT
Connection: keep-alive

{"result":"OK","infileSize":8951158,"outfileSize":36771660,"ratio":"410.80%"}
```

ここで、受信ジョブを実行/受信/処理する間のzlibserverのコンソール出力を確認できます。

```
C:\projects\zlib>node zlibserver
zlibserver is ready captain.
2018-1-26 08:43:14 : GZIP test.log -> test.gz...
2018-1-26 08:43:17 : finished!
2018-1-26 08:43:36 : GUNZIP test.gz -> test.txt...
2018-1-26 08:43:36 : finished!
2018-1-26 08:44:23 : DEFLATERAW test.log -> test.zz...
2018-1-26 08:44:26 : finished!
2018-1-26 08:44:42 : INFLATERAW test.zz -> test.txt...
2018-1-26 08:44:42 : finished!
```

お客様の問題とそれを解決する上で達成したことをまとめましょう。

RESTを使ったNode.jsのコールアウトによって、Caché を非常に簡単に拡張できることを学びました。

まず、この記事のきっかけとなった元々の特定のユースケースを「ブラインドアウト」し、優れたモードモジュールが豊富に用意され、APIによって広範な機能の可能性が提供されている、素晴らしいNode.jsエコシステムの観点から前向きに考えてみると、魅力的なソリューションで、Caché から簡単にAPIにアクセスして制御することができるようになりました。

もっとも頻繁に使用されているNode.jsモジュール/APIのリストについては、次のリンクを参照してください。
http://www.creativebloq.com/features/20-nodejs-modules-you-need-to-know

「サポートの現場からは以上です！」 :)

お役に立てられれば幸いです。

Bernd

#JavaScript #Node.js #ObjectScript #REST API #コールアウト #ベストプラクティス #Caché

ソースURL:
https://jp.community.intersystems.com/post/%E3%82%B5%E3%83%9D%E3%83%BC%E3%83%88%E3%81%AE%E7%8F%BE%E5%A0%B4%E3%81%8B%E3%82%89-raw-deflate%E3%81%AE%E5%9C%A7%E7%B8%AE%E8%A7%A3%E5%87%8D%E6%A9%9F%E8%83%BD%E3%81%AE%E6%8E%A2%E6%B1%82%E3%81%8B%E3%82%89%E3%80%81%E3%81%A9%E3%81%AE%E3%82%88%E3%81%86%E3%81%ABrest%E3%81%AB%E3%82%88%E3%82%8Bnodejs%E3%81%AE%E3%82%B3%E3%83%BC%E3%83%AB%E3%82%A2%E3%82%A6%E3%83%88%E3%82%B5%E3%83%BC%E3%83%90%E3%83%BC%E6%A7%8B%E7%AF%89%E3%81%AB%E8%87%B3%E3%81%A3%E3%81%9F%E3%81%AE%E3%81%8B