Article
Eduard Lebedyuk · Dec 29, 2018 6m read

Importing code from external libraries

Intro

Recently I reread this article by @Bernd Mueller. It's about calling DELFATE function from zlib library. In this article I'll demonstrate several different approaches to callout libraries, we'll build the same functionality (compress function) in several different languages and compare them.

NodeJS

Let's start with NodeJS. I'm taking the code almost directly from Bernd's article, except it does not use files, but rather direct http connection to pass data. For production use, it would be better to pass request as a body and encode both request and response as base64. Still, here's the code:

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

var app = express();

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

    var text=req.params.text;

    try {
        zlib.deflate(text, (err, buffer) => {
           if (!err) {
                res.status(200).send(buffer.toString('binary'));
            } else {
                res.status(500).json( { "error" : err.message});
            // handle error
            }
        });
     }
    catch(err) {
      res.status(500).json({ "error" : err.message});
      return;
    }

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

To start it execute in OS bash (assuming node and npm installed):

```
cd <repo>\node
npm install
node  ./zlibserver.js
```

We're running on port 3000, reading input string from request and returning compressed data in response as is. On a Caché side http request is used to interact with this api:

```
/// NodeJS implementation
/// do ##class(isc.zlib.Test).node()
ClassMethod node(text As %String = "Hello World", Output response As %String) As %Sta
tus
{
    kill response
    set req = ##class(%Net.HttpRequest).%New()
    set req.Server = "localhost"
    set req.Port = 3000
    set req.Location = "/zlibapi/" _ text
    set sc = req.Get(,,$$$NO)
    quit:$$$ISERR(sc) sc
    set response = req.HttpResponse.Data.Read($$$MaxStringLength)
    quit sc
}
```

Note, that I'm setting the third argument set sc = req.Get(,,$$$NO) - reset to zero. If you're writing interface to the outside http(s) server it's best to reuse one request object and just modify it as needed to perform new requests.

Java

Java Gateway allows calling arbitrary Java code. Coincidently Java has Deflater class which does exactly what we need:

```
package isc.zlib;

import java.util.Arrays;
import java.util.zip.Deflater;

public abstract class Java {

    public static byte[] compress(String inputString) {
        byte[] output = new byte[inputString.length()*3];
        try {
            // Encode a String into bytes
            byte[] input = inputString.getBytes("UTF-8");

            // Compress the bytes

            Deflater compresser = new Deflater();
            compresser.setInput(input);
            compresser.finish();
            int compressedDataLength = compresser.deflate(output);
```

```
            compresser.end();
            output = Arrays.copyOfRange(output, 0, compressedDataLength);

        } catch (java.io.UnsupportedEncodingException ex) {
            // handle
        }


        return output;
    }
}
```

The problem with this implementation is that it returns byte[] which becomes a Stream on Caché side. I have tried to return a string, but hadn't been able to found how to form proper binary string from byte[]. If you have any ideas please leave a comment.
To run it place jar from releases page into <instance>/bin folder, load ObjectScript code into your instance and execute:

```
write $System.Status.GetErrorText(##class(isc.zlib.Utils).createGateway())
write $System.Status.GetErrorText(##class(isc.zlib.Utils).updateJar())
```

Check createGateway method before running the command. Second argument javaHome assumes that JAVAHOME environment variable is set. If it does not, specify home of Java 1.8 JRE as a second argument. After installing run this code to get compressed text:

```
set gateway = ##class(isc.zlib.Utils).connect()
set response = ##class(isc.zlib.Java).compress(gateway, text)
```

C

An InterSystems Callout library is a shared library that contains your custom Callout functions and the enabling code that allows Caché to use them.

Here's our Callout library:

```
#define ZF_DLL

// Ugly Windows hack
#ifndef ulong
   typedef unsigned long ulong;
#endif

#include "string.h"
#include "stdio.h"
#include "stdlib.h"
#include "zlib.h"
#include <cdzf.h>

int Compress(char* istream, CACHE_EXSTRP retval)
{
    ulong srcLen = strlen(istream)+1;      // +1 for the trailing `\0`
    ulong destLen = compressBound(srcLen); //  estimate size needed for the buffer
```

```
    char* ostream = malloc(destLen);
    int res = compress(ostream, &destLen, istream, srcLen);
    CACHEEXSTRKILL(retval);
    if (!CACHEEXSTRNEW(retval,destLen)) {return ZF_FAILURE;}
    memcpy(retval->str.ch,ostream,destLen);   // copy to retval->str.ch
    return ZF_SUCCESS;
}

ZFBEGIN
    ZFENTRY("Compress","cJ",Compress)
ZFEND
```

To run it place dll or so files from releases page into <instance>/bin folder. Repository also contains build scripts for Windows and Linux, execute them to build your own version.
Linux prerequisites:

```
apt install build-essential zlib1g zlib1g-devel
```

Windows prerequisites: WinBuilds - comes with zlib.

To interact with callout library execute:

```
set path =  ##class(isc.zlib.Test).getLibPath() //get path to library file
set response = $ZF(-3, path, "Compress", text)       // execute function
do $ZF(-3, "")                                    //unload library
```

System

A little unexpected in an article about callout mechanisms, but Caché also has built-in Compress (and Decompress function). Call it with:

```
set response = $extract($SYSTEM.Util.Compress(text), 2, *-1)
```

Remember that searching the docs or asking the questions here on the community may save you some time.

Comparison

I have run simple tests (1Kb text, 1 000 000 iterations) on Linux and Windows and got these results.

Windows:

Method ---------	Callout ---------	System---------	Java---------	Node---------
Time	22,77	33,41	152,73	622,51
Speed (Kb/s)	43912	29927	6547	1606
Overhead, %	-/-	46,73%	570,75%	2633,90%

Linux:

Method ---------	Callout ---------	System---------	Java---------	Node---------
Time	76,35	76,49	147,24	953,73
Speed (Kb/s)	13097	13072	6791	1049
Overhead, %	-/-	0,19%	92%	1149%

To run tests load code and call:

```
do ##class(isc.zlib.Test).test(textLength, iterations)
```

Conclusion

With InterSystems products, you can easily leverage existing code in other languages. However, choosing correct implementation is not always easy, you need to take several metrics into account, such as development speed, performance, and maintainability. Do you need to run on different operating systems? Finding answers to these questions can help you decide on the best implementation plan.

Links

- Repo
- Binaries
- Http requests
- Java Gateway
- Callout library
- Compress function

#Beginner #Callout #Java #Node.js #JavaScript #Caché