[ExtendScript] 高速化へのアプローチ

Posted by 10 A in アプリケーション自動化総合コミュニティフォーラム (Japan) on Apr 22, 2018 8:35:00 PM

正直に言います。Adobe社のアプリケーションに搭載されているJavaScriptはExtendScriptというECMA Scriptのver.3あたりを拡張した代物ですからアーキテクチャーがとても古くてインタプリタの性能も低く実行速度がとても遅いです。
しかしながら、そんな事ばかり言っているだけでは前に進めませんので色々と試す事はあります。今回はExtendScriptの弱点を踏まえた上でどう対応していくかの一例を用意しました。

Base64エンコーディング

Base64エンコーディングというのはバイナリファイルをAsciiテキストとして表現する処理です。これはメール送信時等原則的にテキストしか扱えない処理系でバイナリファイルを扱うため には必須のエンコーディング処理です。
処理の基本的な部分はとても単純で３byte毎のコードを並べて24bitにして、それを６bitづつに切り分けた上で特定のキャラクタに当てはめてテキスト化するという処理になります。です から
3byte（3×8bit=24bit）のバイナリはAsciiキャラクタ４コの表現となり32bitのデータとなります。メールに添付した画像の容量が実際の送信時に容量が大きくなるの はこの処理の為で、Base64処理を通るとファイル容量は4/3倍に増加します。このあたりの情報はウィキ等検索をかけると沢山出てきますから興味のある方は調べてみて下さい。

ExtendScriptでBase64処理を行う

では本題です。このBase64エンコーディング処理をExtendScriptでやってみるとどうなるか見てみます。
サンプルとして用意したコードはAdobe社のサンプルに含まれるものです。シンプルに書かれていますので順番に追っていけば容易に理解できると思います。

```
01. EmailWithAttachment.prototype.encodeAttachment = function(binaryString)
02. {
03.     // Accepted b64 chars
04.     var keyStr = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=";
05. 
06. 
07.     var encoded = "";
08.     var c1, c2, c3;
09.     var e1, e2, e3, e4;
10.     var i = 0;
11. 
12.     while(i < binaryString.length)
13.     {
14.         // Get 3 chars starting at the begining of the binary string
15.         c1 = binaryString.charCodeAt(i++);
16.         c2 = binaryString.charCodeAt(i++);
```

```
        c3 = binaryString.charCodeAt(i++);


        //  Create 4 sets of 6 bits from the 3 ascii bytes
        e1 = c1 >> 2;
        e2 = ((c1 & 3) << 4) | (c2 >> 4);
        e3 = ((c2 & 15) << 2) | (c3 >> 6);
        e4 = c3 & 63;


        // Check that chrs 2 and 3 are correct, if not replace with special padding char
        if (isNaN(c2))
        {
            e3 = e4 = 64;
        }
        else if (isNaN(c3))
        {
            e4 = 64;
        }


        // Add encoded chars to the return string
        encoded = encoded + keyStr.charAt(e1) + keyStr.charAt(e2) +
            keyStr.charAt(e3) + keyStr.charAt(e4);
    }


    return encoded;
}
```

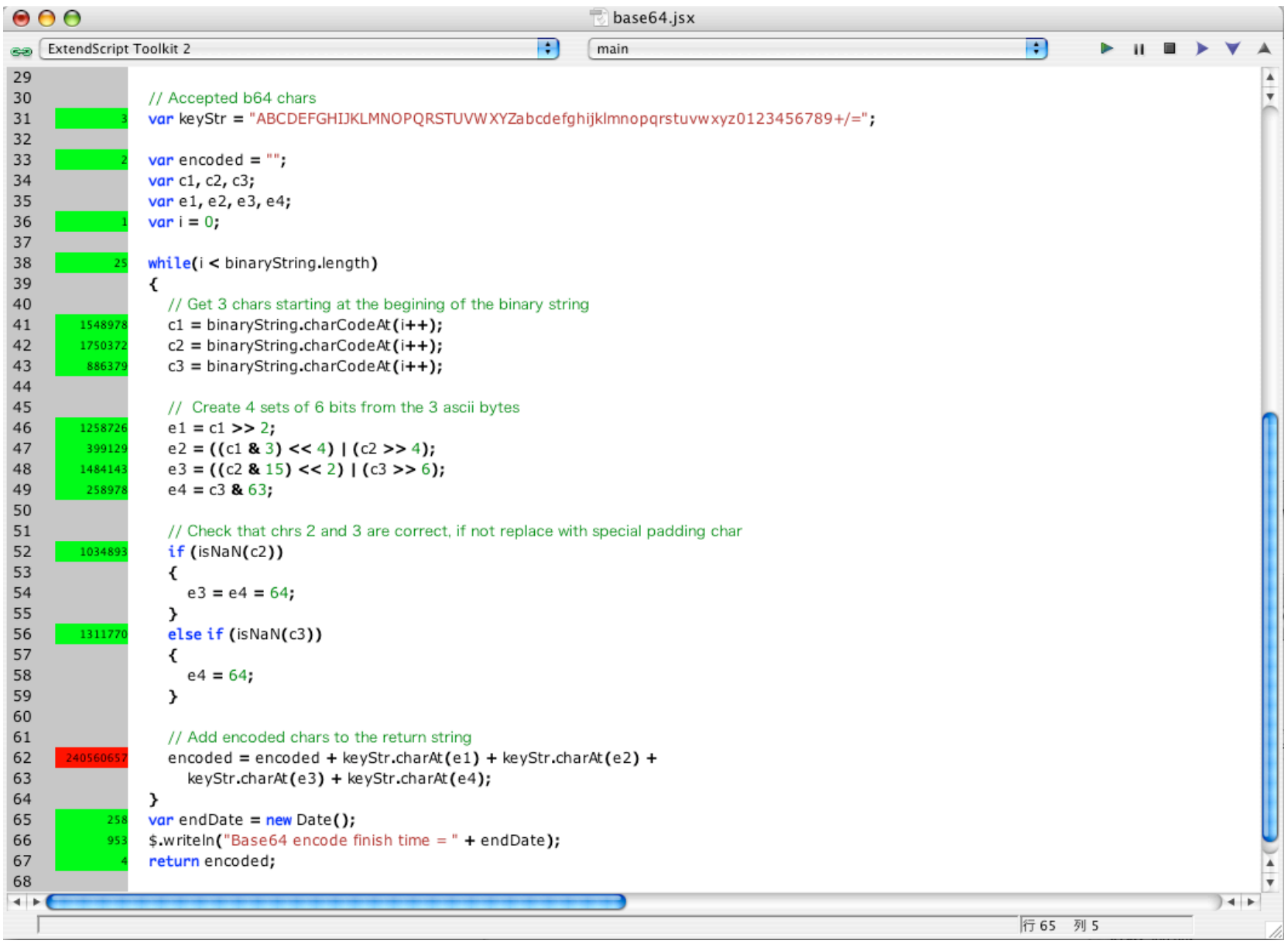
```
EmailWithAttachment.prototype.encodeAttachment = function(binaryString)
{
    // Accepted b64 chars
    var keyStr = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=";


    var encoded = "";
    var c1, c2, c3;
    var e1, e2, e3, e4;
    var i = 0;

    while(i < binaryString.length)
    {
        // Get 3 chars starting at the begining of the binary string
        c1 = binaryString.charCodeAt(i++);
        c2 = binaryString.charCodeAt(i++);
        c3 = binaryString.charCodeAt(i++);


        //  Create 4 sets of 6 bits from the 3 ascii bytes
        e1 = c1 >> 2;
        e2 = ((c1 & 3) << 4) | (c2 >> 4);
        e3 = ((c2 & 15) << 2) | (c3 >> 6);
        e4 = c3 & 63;


        // Check that chrs 2 and 3 are correct, if not replace with special padding char
        if (isNaN(c2))
        {
            e3 = e4 = 64;
        }
```

```
        else if (isNaN(c3))
        {
            e4 = 64;
        }


        // Add encoded chars to the return string
        encoded = encoded + keyStr.charAt(e1) + keyStr.charAt(e2) +
            keyStr.charAt(e3) + keyStr.charAt(e4);
    }


    return encoded;
}
```

whileループでバイナリを順番に処理しencoded変数に収容するといった処理になっています。
今回用意したテスト用の画像が300kbです。これを上記のスクリプトを使いESTK上で処理してみます。

Base64 encode execution time = 251(sec)

となります。４分以上処理にかかるという結果となりました。実際にヒットカウンターを表示させると以下のようになります。



encoded変数へのアクセスで非常に負荷がかかっているのがわかります。

処理を変更してみる

上のスクリプトでenocded変数へ文字列を追加していく部分で大きな負荷がかかっている事がわかりました。300kbという容量は文字数にしてみると30万文字となります。もちろんアプリ ケーションキャッシュ等の環境にも関係する部分なのですが、基本的にはスクリプトエンジンに扱わせるには荷が重いデータです。数キロバイト程度の容量のファイルでテストしてみるとこういったパ フォーマンスの低下は見られません。
これらを踏まえて書いたのが以下のコードです。

```
function base64(fileNm) {
    var theFile = new File(fileNm);
    theFile.encoding = "binary";
    theFile.open("r");
    var binStr = theFile.read();
    theFile.close();
    var flNm = "/tmpfile.txt";
    var tmpFile = new File(flNm);
    tmpFile.open("w");
    szB64 = new Array;
    var b64Str = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
    for (j=0;j<64;j++){
        szB64[j] = keyStr.charAt(j);
        }
    var ecdStr = "";
    var cntr = 0;
    var binLen = binStr.length - (binStr.length % 3);
    for (i=0;i<binLen;i+=3) {
        ecdStr += szB64[binStr.charCodeAt (i) >> 2];
        ecdStr += szB64[((binStr.charCodeAt (i) & 0x3) << 4) | (binStr.charCodeAt (i+1) >>
        ecdStr += szB64[((binStr.charCodeAt (i+1) & 15) << 2) | (binStr.charCodeAt (i+2) >
        ecdStr += szB64[binStr.charCodeAt (i+2) & 0x3f];
        cntr +=4;
        if(cntr > 75) {
            tmpFile.writeln (ecdStr);
            ecdStr = "";
            cntr = 0;
            }
        }
    switch (binStr.length % 3){
        case 2: //padding 2 characters.
        ecdStr += szB64[binStr.charCodeAt (i) >> 2];
        ecdStr += szB64[((binStr.charCodeAt (i) & 0x3) << 4) | (binStr.charCodeAt (i+1) >>
        ecdStr += szB64[((binStr.charCodeAt (i+1) & 15) << 2) | (binStr.charCodeAt (i+2) >
        ecdStr += "=";
        tmpFile.writeln (ecdStr);
        break;
        case 1: //padding 1 character.
        ecdStr += szB64[binStr.charCodeAt (i) >> 2];
        ecdStr += szB64[((binStr.charCodeAt (i) & 0x3) << 4) | (binStr.charCodeAt (i+1) >>
        ecdStr += "=="
        tmpFile.writeln (ecdStr);
        break;
        default:
        tmpFile.writeln (ecdStr);
        break;
        }
    tmpFile.close();
    }
}
```

```
    tmpFile.close();
    }
function base64(fileNm) {
    var theFile = new File(fileNm);
    theFile.encoding = "binary";
    theFile.open("r");
    var binStr = theFile.read();
    theFile.close();
    var flNm = "/tmpfile.txt";
    var tmpFile = new File(flNm);
    tmpFile.open("w");
    szB64 = new Array;
    var b64Str = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
    for (j=0;j<64;j++){
        szB64[j] = keyStr.charAt(j);
        }
    var ecdStr = "";
    var cntr = 0;
    var binLen = binStr.length - (binStr.length % 3);
    for (i=0;i<binLen;i+=3) {
        ecdStr += szB64[binStr.charCodeAt (i) >> 2];
        ecdStr += szB64[((binStr.charCodeAt (i) & 0x3) << 4) | (binStr.charCodeAt (i+1) >>
        ecdStr += szB64[((binStr.charCodeAt (i+1) & 15) << 2) | (binStr.charCodeAt (i+2) >
        ecdStr += szB64[binStr.charCodeAt (i+2) & 0x3f];
        cntr +=4;
        if(cntr > 75) {
            tmpFile.writeln (ecdStr);
            ecdStr = "";
            cntr = 0;
            }
        }
    switch (binStr.length % 3){
        case 2: //padding 2 characters.
        ecdStr += szB64[binStr.charCodeAt (i) >> 2];
        ecdStr += szB64[((binStr.charCodeAt (i) & 0x3) << 4) | (binStr.charCodeAt (i+1) >>
        ecdStr += szB64[((binStr.charCodeAt (i+1) & 15) << 2) | (binStr.charCodeAt (i+2) >
        ecdStr += "=";
        tmpFile.writeln (ecdStr);
        break;
        case 1: //padding 1 character.
        ecdStr += szB64[binStr.charCodeAt (i) >> 2];
        ecdStr += szB64[((binStr.charCodeAt (i) & 0x3) << 4) | (binStr.charCodeAt (i+1) >>
        ecdStr += "=="
        tmpFile.writeln (ecdStr);
        break;
        default:
        tmpFile.writeln (ecdStr);
        break;
        }
    tmpFile.close();
    }
   }
    tmpFile.close();
    }
```

前出のものとの大きな違いは処理したエンコード済文字列は 1 行分溜まったらテンポラリファイルに出力してしまって、変数上で大きなデータを保持しないようにしたところです。こちらもトリッキー な処理は行っていませんからスクリプトの構造は容易に追えるとおもいます。
これを先ほどと同じ画像を処理してみます。

Base64 encode execution time = 7(sec)

実行時間が大幅に短縮されたことがお分かりいただけるでしょう。実際にヒットカウンターを見てみると２ケタ低下しているのが分かります。

Base64_filestream.jsx

ExtendScript Toolkit 2 | main

```
1
1   function base64(fileNm) {
10    var startDate = new Date();
355   $.writeln("Base64 encode start time= " + startDate);
183   var theFile = new File(fileNm);
81    theFile.encoding = "binary";
231   theFile.open("r");
6902  var binStr = theFile.read();
55    theFile.close();
3     var flNm = "/tmpfile.txt";
196   var tmpFile = new File(flNm);
292   tmpFile.open("w");
87    szB64 = new Array;
3     var keyStr = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";    // Accepted b64 chars
6     for (j=0;j<64;j++){
3005    szB64[j] = keyStr.charAt(j);
        }
2     var ecdStr = "";
1     var counter = 0;
26    var binLen = binStr.length - (binStr.length % 3);
5     for (i=0;i<binLen;i+=3) {     // Get 3 chars starting at the begining of the binary string
1235506   ecdStr += szB64[binStr.charCodeAt (i) >> 2];
2059832   ecdStr += szB64[((binStr.charCodeAt (i) & 0x3) << 4) | (binStr.charCodeAt (i+1) >> 4)];
2105699   ecdStr += szB64[((binStr.charCodeAt (i+1) & 15) << 2) | (binStr.charCodeAt (i+2) >> 6)];
1217694   ecdStr += szB64[binStr.charCodeAt (i+2) & 0x3f];
158222    counter +=4;
252726    if(counter > 75) {     // when hit the limit, write the line and reset paramater
240361      tmpFile.writeln (ecdStr);
11677       ecdStr = "";
9703        counter = 0;
            }

          }
18  switch (binStr.length % 3){
      case 2: //padding 2 characters.
      ecdStr += szB64[binStr.charCodeAt (i) >> 2];
      ecdStr += szB64[((binStr.charCodeAt (i) & 0x3) << 4) | (binStr.charCodeAt (i+1) >> 4)];
      ecdStr += szB64[((binStr.charCodeAt (i+1) & 15) << 2) | (binStr.charCodeAt (i+2) >> 6)];
      ecdStr += "=";
      if (ecdStr.length>75){
        tmpFile.writeln(ecdStr.slice (0, 76));
        tmpFile.writeln(ecdStr.slice (76));
        } else {
          tmpFile.writeln (ecdStr);
          }
      break;
      case 1: //padding 1 character.
```

行 28　列 36

この様にExtendScriptでは変数に大きなデータを喰わせると極端にパフォーマンスが低下する場合があります。もしスクリプトの実行時間が遅く感じたら、まずはヒットカウンターをチェ ックして、何処の処理に負荷がかかっているかを検証することが大事です。
しかしながら、ExtendScriptはインタープリタであり実行時にスクリプトエンジンがコードを逐一読み込みながら実行されるものです。基本的に実行時間が遅いものだと考えなければいけ ません。
もし、さらなる高速化を望む場合はCEPエクステンションやExternalObject等を検討する必要がある事も覚えておきましょう。

954 Views　Tags : javascript, extendscript

0 Comments