# ソフトウェア開発 第 14 回目授業

平野 照比古

2016/1/15

## jQuery ファイルの違い

- jQuery のソースファイルには jQuery-<Version No>.jsと jQuery-<Version No>.min.jsの2種類が存在
- jQuery-<Version No>.min.js は jQuery-<Version No>.js から コメントや、空白を取り除き、変数名を短いものに置き換えるとい う操作を行って、ファイルサイズを小さくしたもの

# **jQuery** のソースコード (1)

```
/*!
* jQuery JavaScript Library v1.11.3
* http://jquery.com/
 * Includes Sizzle.js
  http://sizzlejs.com/
 * Copyright 2005, 2014 jQuery Foundation, Inc. and other contributors
  Released under the MIT license
 * http://jquery.org/license
 * Date: 2015-04-28T16:197
*/
```

## jQuery のソースコード(2)

```
(function( global, factory ) {
 if ( typeof module === "object" && typeof module.exports === "object" ) {
    // For CommonJS and CommonJS-like environments where a proper window is present,
    // execute the factory and get jQuery
    // For environments that do not inherently posses a window with a document
    // (such as Node.js), expose a jQuery-making factory as module.exports
    // This accentuates the need for the creation of a real window
    // e.g. var iQuerv = require("iquerv")(window);
    // See ticket #14549 for more info
   module.exports = global.document ?
     factory( global, true ) :
     function(w) {
        if (!w.document) {
         throw new Error( "¡Query requires a window with a document" );
       return factory( w );
     1:
 } else {
    factory( global ):
// Pass this if window is not defined vet
}(typeof window !== "undefined" ? window : this, function( window, noGlobal ) {
```

# jQuery のソースコード(3)

```
// Can't do this because several apps including ASP.NET trace
// the stack via arguments.caller.callee and Firefox dies if
// you try to trace through "use strict" call chains. (#13335)
// Support: Firefox 18+
//
var deletedIds = []:
var slice = deletedIds.slice:
var concat = deletedIds.concat;
var push = deletedIds.push;
var indexOf = deletedIds.indexOf;
var class2type = {};
var toString = class2type.toString;
var hasOwn = class2type.hasOwnProperty;
var support = {};
```

# jQuery のソースコード (tiny)

```
var
 version = "1.11.3",
 // Define a local copy of jQuery
  jQuery = function( selector, context ) {
   // The jQuery object is actually just the init constructor 'enhanced'
    // Need init if jQuery is called (just allow error to be thrown if not
   return new jQuery.fn.init( selector, context );
 },
  // Support: Android<4.1, IE<9
  // Make sure we trim BOM and NBSP
 rtrim = /^[\s\uFEFF\xA0]+|[\s\uFEFF\xA0]+$/g,
  // Matches dashed string for camelizing
 rmsPrefix = /^-ms-/,
 rdashAlpha = /-([\da-z])/gi,
  // Used by jQuery.camelCase as callback to replace()
 fcamelCase = function( all, letter ) {
   return letter.toUpperCase();
 };
```

### jQuery 1.12.2.js のソース

- このリストでは詳細なコメントが付いていて比較的読みやすい。
- コメントを見るとクロスブラウザ対策が行われていることも見て取れる。
- ソースコードの最小化のテクニックとしてグローバルなオブジェクトを短い変数名で置き換える
- 冒頭の定義された関数の引数に、関数を渡している。その関数の仮 引数は window となっている。これにより、開発時はわかりやすい変 数名が使えるメリットがある。

p=/-([\da-z])/gi,q=function(a,b){return b.toUpperCase()};

#### 短縮化のリスト

```
/*! jQuery v1.11.3 | (c) 2005, 2015 jQuery Foundation, Inc. | jquery.org/license */
!function(a,b){"object"==typeof module&&"object"==typeof
    module.exports?module.exports=a.document?b(a,!0):function(a){if(!a.document)thr
    new Error("jQuery requires a window with a document");
    return b(a)}:b(a)}
    ("undefined"!=typeof window?window:this,function(a,b){
        var c=[],d=c.slice,e=c.concat,f=c.push,g=c.indexOf,h={},i=h.toString,
        j=h.hasOwnProperty,k={},l="1.11.3",
        m=function(a,b){return new m.fn.init(a,b)},
        n=/^[\s\uFEFF\xA0]+|[\s\uFEFF\xA0]+$/g,o=/^-ms-/,
```

#### 短縮化の方法

- ◆ 各関数内で変数名は1文字から始めている。
- 関数の仮引数もaから付け直している。
- JavaScript の固有の関数は当然のことながら変換されていない。
- このライブラリーは一つの関数を定義して、その場で実行している。 14 行目の function() の前に(がついている。
- 更縮化されたコードではこの部分が!function()となっている。関 数オブジェクトを演算の対象とすることで実行する。
- そのほかにもキーワード true の代わりに!0 としている。
- if(){}else{} 構文は?に置き換えている。

### JavaScript ファイルの短縮化ツール

JavaScript の短縮化ツールとして、Google が提供する Closure Compiler を紹介する。このサービスは次のように説明されている。

The Closure Compiler is a tool for making JavaScript download and run faster. It is a true compiler for JavaScript. Instead of compiling from a source language to machine code, it compiles from JavaScript to better JavaScript. It parses your JavaScript, analyzes it, removes dead code and rewrites and minimizes what's left. It also checks syntax, variable references, and types, and warns about common JavaScript pitfalls.

### Closure Compiler の使い方

#### 使い方については次のように書かれている。

You can use the Closure Compiler as:

- An open source Java application that you can run from the command line.
- A simple web application.
- A RESTful API.

To get started with the compiler, see "How do I start" below.

ここでは2番目にある Web アプリケーションで行う。

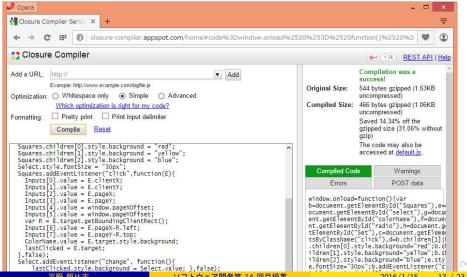
#### Closure Compiler の Web サイト

サイトは http://closure-compiler.appspot.com/home



#### 短縮化の例

#### 使用するファイルは9回目の授業で用いた event. js



### 短縮化の結果(1)

画面の右のほうで、元のファイルの大きさが 1.53KB であったのに対し、 短縮化の結果が 1.06KB となったことがわかる。

#### 短縮化の詳細

```
window.onload=function(){var b=document.getElementById("Squares"),
    e=document.getElementById("select"),
    g=document.getElementById("colorName"),
    f=document.getElementById("radio"),
    h=document.getElementById("Set"),
    c=document.getElementByClassName("click"),
    d=b.children[1];
    b.children[0].style.background="red";
    b.children[1].style.background="yellow";
    b.children[2].style.background="blue";
    e.style.fontSize="30px";
```

## 短縮化の結果 (2)

```
b.addEventListener("click",function(a){
  c[0].value=a.clientX;
  c[1].value=a.clientY;
  c[2].value=a.pageX;
  c[3].value=a.pageY;
  c[4].value=window.pageXOffset;
  c[5].value=window.pageYOffset;
  var b=a.target.getBoundingClientRect();
  c[6].value=a.pageX-b.left;
  c[7].value=a.pageY-b.top;
  g.value=a.target.style.background;
 d=a.target
},!1);
```

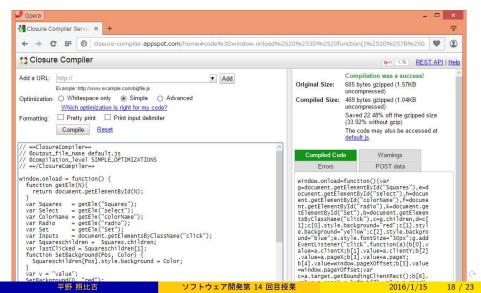
## 短縮化の結果(3)

```
e.addEventListener("change",function(){
    d.style.background=e.value},!1);
f.addEventListener("click",function(a){
    alert(a.target.tagName);
"DIV"===a.target.tagName&&(a.target.firstChild.checked=!0);
console.log("----"+f.value);
d.style.background=f.querySelector("input:checked").value
},!1);
h.addEventListener("click",function(){
    d.style.background=g.value},!1)};
```

- if 文であったものが論理式の&&で置き換えられている。
- document.getElementById などはそのまま短縮化、共通化されていない。

### document.getElementById を減らす

#### 単純に関数を置き換えただけではうまくいかない。



#### 解説

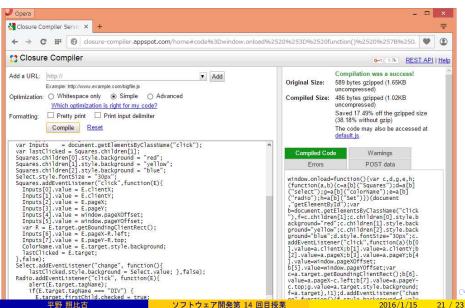
ラップした関数は消えていて、もとの document.getElementById に戻っている。

これを避けるためには関数を定義してその場で実行する関数の引数に document と"getElementById"を渡すことで解決できる。

# ソースコードの書き直し(1)

```
var Squares, Select, ColorName, Radio, Set;
(function(document,getElementById){
    Squares = document[getElementById]("Squares");
    Select = document[getElementById]("select");
    ColorName = document[getElementById]("colorName");
    Radio = document[getElementById]("radio");
    Set = document[getElementById]("Set");
})(document, "getElementById");
```

# ソースコードの書き直し(2)



## ソースコードの書き直し(3)

コンパイル後の結果が 1.02KB とわずかに小さくなっていることがわかる。

短縮化を効率よくするためには元のプログラムも考えて作らないといけない。

#### 注意

この書き直しで document.getElementById だけを渡せば十分と思われるかもしれないが、これは関数オブジェクトとなり、関数内での this が document にならないので、実行時にエラーが生ずる。 与えられた関数を別のオブジェクトのメソッドとして実行するために、call メソッドを用いる。

```
(function(document,getElementById){
    Squares = getElementById.call(document,"Squares");
    Select = getElementById.call(document,"select");
    ColorName = getElementById.call(document,"colorName");
    Radio = getElementById.call(document,"radio");
    Set = getElementById.call(document,"Set");
})(document,document.getElementById);
```