ななちゃんのIT教室

透明マント：スコープの秘密の巻

by nara.yasuhiro@gmail.com

ななちゃんが
JavaScript のスコープについて調べるという お話

第 0.1 版　2017 年 5 月 30 日

フリー素材
http://freeillustration.net

いらすとやフリー素材
http://www.irasutoya.com/

もくじ

第1回　スコープとは

なな：スコープって何？

先生：プログラムのどの範囲から変数や関数が参照できるかということ。それをコントロールすることよ。参照できる範囲外からは、変数や関数が、「スコープ外である」、とか、「見えない」といわれるの。

なな：見えても、見なければ良いので、見えるようにしておくのが良いのでは。

先生：そうとは言いきれないのよ。見えなければ、誤って書き換えるなどの事故が未然に防げるの。特に、プログラムを、複数人で分担開発する時に重要になるの。「Aさんが、変数 x,y,i,j,n 関数 func,sub を使っているので、他の人はこれらの名前は使わないように」というのでは、やりにくいでしょう。 便利な関数を集めたライブラリを使う時、「このライブラリでは x,y,i,j,n 関数 func,sub を使っているので、ユーザはこれらの名前は使わないように」 では、やりにくいでしょう。

なな：ライブラリを使い始める前に、その名前を使ってしまっていたら、プログラムを書き換えないといけないわね。それに、うっかり名前を使ってしまうこともあるかも。

先生：まず、変数名、関数名を付ける時の規則を復習しておきましょう。

なな：

- 使用できる文字は、アンダーバー(_)、ドル記号($)、半角英数、その他のUnicode全角文字
- 先頭文字には数字の0から9は使用できない
- 大文字と小文字は区別される
- 予約語は使用できない

先生：_ や $ を使う人は少ないので、ユーザは使わない、という作戦もあるけど、口約束にすぎないから、うっかり事故の可能性があるわね。特定のライブラリの変数、関数名をすべて liba_～ として、ユーザは使わない、というのも口約束にすぎないわね。それでは、実験をしながら、スコープを調査してみましょう。

クリ：わしも参加するぞ。

なな：クリじい、なにか秘密道具を出してよ。

クリ：「myConsole.html ～～」！ 下記のソースを myConsole.html というファイルに保存し、ダブルクリックして起動んじゃ。
Internet Explorer 11、Edge、Firefox の console、Google Chrome のデベロッパーツール と類似しているが、微妙に操作が異なったり、実行結果のグローバル変数が次の実行に引き継がれたり、1 回の入力行数に制限がある。Firefox のスクラッチパッドは、入力行数に制限はないが、グローバル変数が残る。myConsole.html は、ブラウザによらず、同じ操作、入力行数に制限はなく、グローバル変数は毎回リセットされるという、すぐれものなんじゃ。

```
var x = 1;

function f() {
   log(x);
   x = 2;
   log(x);
}
f();
log(x);
```

実行

```
1
2
2
```

```
<!DOCTYPE html>
<html lang="ja">
   <head>
      <meta charset="utf-8">
   </head>
   <body>
      <textarea rows=15 cols=60 id=inp>
var x = 1;

function f() {
     log(x);
     x = 2;
     log(x);
}
f();
log(x);
</textarea><br>
       <input type=button value="実行" onclick=go()><br>
       <textarea rows=10 cols=60 id=out></textarea>
       <script>
function log(d) { out.value += d + "\n"; }
function go() {
     out.value = "";
     try { eval(inp.value); }
     catch(e) { out.value += e; }
}
       </script>
   </body>
</html>
```

第2回　グローバルスコープと関数スコープ

先生：まず、教科書で勉強したことの復習から。
グローバルスコープ。グローバル変数は？

```
var x = 1;
function f() {
    log(x);        // 1
    x = 2;
    log(x);        // 2
}
f();
log(x);            // 2
```

なな：どこからでも見える変数のことね。

クリ：myConsole で実験じゃ。 myConsole では、出力に log() を使う。その出力を、ここでは、「// …」の形で書くことにするぞ。

先生：次に関数スコープ。関数の中で宣言する変数は？

```
var x = 1;
function f() {
    log(x);        // undefined
    var x = 2;
    log(x);        // 2
}
f();
log(x);            // 1
```

なな：関数の外からは見えないやつね。

先生：関数の中では、関数外の名前は気にせず、自由に名前をつけられるということね。関数を使う人は、関数の中の変数名を気にしなくて良いし、間違ってアクセスすることもないということね。ただし、var をつけ忘れると、グローバル変数になってしまうので要注意。

なな：function の最初の log(x) は、「1」という表示にならないの？

クリ：これは、「**巻き上げ**」と言って、関数無いの var 宣言は、さかのぼって有効ということなの。プログラム実行に先立つ、先行スキャンでチェックしてしまうのじゃ。「var x = 2;」は、「var x;」が先頭になり、「x = 2;」が元々の場所にあったのと同じに扱うのじゃ。

先生：関数定義の function name() { ... } も先行スキャンするので、定義より前に呼び出しがあっても良いの。でも、「var name = function() { ... };」は、先行スキャンの対象にならないわ。「var name;」だけが先頭になり、「name = function() { ... };」が元々の場所にあったのと同じに扱うの。だから、「name = function() { ... };」より手前に書かれた呼び出しはエラーになる。

なな：グローバルの変数宣言は省略して、いきなり使っても良いのよね。

先生：そうね。そういうのを、無宣言というのね。でも、間違いの原因になりやすので、厳密モードにすると、エラー検出できます。

```
"use strict";
x = 1;              // ReferenceError: Variable undefined in strict mode
log(x);
x = 2;
log(x);
```

でも、あとのほうに var 宣言があれば、「巻き上げ」によって、エラーになりません。

```
"use strict";
x = 1;
log(x);             // 1
var x;
log(x);             // 1
x = 2;
log(x);             // 2
```

変数宣言は、実行されない部分にあっても有効です。

```
"use strict";
log(x);                    // undefined
if (1 == 2) var x = 1;
```

先生：関数スコープは、入れ子の関数では、階層的なスコープになります。

```
var x = 0;                  // グローバル変数

function f() {
    var x = 1;              // 中間層
    function g() {
        log(x);             // undefined 巻き上げ
        var x = 2;          // 内層
        log(x);             // 2（内層が見える）
    }
    g();
    log(x);                 // 1（中間層が見える）
}
f();
log(x);                     // 0（グローバル変数が見える）
```

第3回　ブロックスコープ（Ecma Script 2015 の新機能！）

先生：Ecma Script（ES）は、JavaScript の仕様。ES 6、俗称 ES2015 の機能追加のひとつにブロックスコープがあります。ブロックスコープの変数宣言には、let と const があります。まず、let から。

```
var x = 1;
{
    let x = 2;
    log(x);        // 2
}
log(x);            // 1
```

{ … } の中の変数 x は、外側の x とは別物になっています。

```
var x = 1;
{
    log(x);        // ReferenceError: Use before declaration
    let x = 2;
}
```

let による宣言が、参照より後ろにあると、特異な動作になります。外側の x を参照するのでもなく、定義済み値無しの undefined、つまり巻き上げになるのでもなく、未定義状態になります。

```
var x = 1;
{
    log(x);        // 1
}
```

もちろん、let 宣言がなければ、{ … } の外側の x が見えるようになります。

なな：なるほど。こんな使い方ができるのね。

```
for (let i=0; i<3; i++) {
    for (let i=0; i<3; i++) {
        log(i);    // 0 1 2 0 1 2 0 1 2
    }
}
```

先生：次は、const

```
var x = 1;（または、let x = 1;）
{
    log(x);        // ReferenceError: Use before declaration
    const x = 2;
    log(x);        // 2
    x = 3;         // TypeError: Assignment to const
}
log(x);            // 1
```

const は、let の書き換え禁止版ということです。

なな：var の書き換え禁止版は無いの。

先生：ありません。もともと、C言語や、Java言語では、通常の変数宣言で、ブロックスコープが可能になっています。JavaScript にその機能が無いのが、ユーザの不満になっていたのね。ES2015 で、ブロックスコープが導入されることになったけど、var のままで導入すると、互換性が保てなくて、過去に作って、使用中のプログラムが動作しなくなってしまいます。そこで、あらたに let が使われることになったと考えられるの。

なな：let って、「させる」という意味？

先生：let は、BASIC 言語や Lisp 言語の、代入文のキーワードとして使われてきたものを参考にしたものと考えられるわ。

なな：ようするに、JavaScript の let は、var の強化版。その、let の書き換え禁止版が const ということね。そして、ブロックスコープを使えない、過去の遺産みたいな var に対する書き換え禁止版は作られなかったということね。。

先生：もうひとつ、var の問題点を、let で解決したというものがあるの、それは「再宣言」。var は、同じ変数名を、何回も宣言して良いという仕様になっているの。

```
var x = 1;
log(x);          // 1
var x;
log(x);          // 1
x = 2;
log(x);          // 2
```

2 回目の宣言で、「おかしいよ」と、エラー扱いにしたほうが良いのではないかと考えられていたの。そこで、let では、再宣言を禁止し、エラーにすることになったの。const は、その書き換え禁止版の拡張版だし、もともと、書き換え禁止＝再宣言禁止という意味合いなので、こちらも再宣言禁止。

```
let x = 1;
log(x);          // 1
let x;           // SyntaxError: Let/Const redeclaration
log(x);          // 1
x = 2;
log(x);          // 2
```

なな：はーい。

第4回　透明マント：グローバル変数を使わないための工夫

先生：チーム開発の他メンバのプログラムや、ライブラリプログラムで使われている変数や関数で、それを利用するプログラマに見える必要のないものを隠蔽する工夫がいろいろあります。ここでは、いろいろな方法を説明しますが、一番のすぐれものは最後に説明するので、忙しい人は、途中の説明を飛ばして読んでください。

まず、「即時関数」。名前を隠蔽できるけど、変数の状態を残せないという問題があります。つまり、カウンタのようなものは実現できません。これは、ユーザが呼び出すのではなく、隠蔽状態で何かを実行する方法です。

```
var x = 1;                  // 利用者の変数
(function () {              // 名前の無い、無名関数
   var x = 2;
   function f() {
      log(x);               // 2
      x = x + 1;
      log(x);               // 3
   }
   f();
})();                       // 無名関数の呼び出し＝実行
log(x);                     // 1  利用者の変数は影響を受けていない。
```

即時関数

同じことを、ES2015 で導入された arrow 関数（無名関数定義の簡略記法）を使うと下記のようになります。

```
var x = 1;
(() => {
   var x = 2;
   function f() {
      log(x);               // 2
      x = x + 1;
      log(x);               // 3
   }
   f();
})();
log(x);                     // 1
```

ここだけ異なる

関数の中の変数は、毎回リセットされるので、状態を残すにはグローバル変数を使わざるを得なません。

```
var count = 1;              // グローバル変数
function f() {
    log(count);
    count = count + 1;
}
f();                        // 1
f();                        // 2
f();                        // 3
```

```
function f(iv) {
      var a;
      if (typeof a === 'undefined') {
            a = iv;
      }
      a++;
      log(a);
}
f(0);               // 1
f(0);               // 1
f(0);               // 1
log(f.a);// undefined
```

関数内部の変数は毎回リセットするのでカウンタは作れない

先生：オブジェクトを使うと状態が残せるけど、隠蔽は不十分です。

```
var obj = {
   f : function () {
      log(this.x);
      this.x = this.x + 1;
   },
   x : 1
};
obj.f();          // 1
obj.f();          // 2
obj.f();          // 3
log(obj.x);       // 4
obj.x = 9;
obj.f();          // 9
```

カウンタを作れる

読み書きできてしまう

先生：こういうオブジェクトの使い方もあります。やはり、状態が残せるけど、隠蔽は不十分です。

```
function f(iv) {
      this.a = iv;
      this.cup = function() {
          this.a++;
          log(this.a); }
}
var g = new f(0);
g.cup();          // 1
g.cup();          // 2
g.cup();          // 3
log(g.a);         // 3
g.a = 9;
g.cup();          // 10
```

カウンタを作れる

読み書きできてしまう

先生：このような、関数の特別な使い方をすると、状態を残せるし、隠蔽もOKです。でも、ちょっと変則的。

```
function f(iv) {
      if (typeof this.a === 'undefined') {
          this.a = iv;
      }
      this.a++;
      log(this.a);
}
f(0);             // 1
f();              // 2
f();              // 3
log(f.a);         // undefined
```

```
function f() {
      var count;
      if(count == undefined) count = 0;
      this.cup = function() {
          log(++count);
      }
}
var g = new f();
g.cup();          // 1
g.cup();          // 2
g.cup();          // 3
log(g.count);     // undefined
```

先生：これが決定版。クロージャといいます。状態が残るし、隠蔽も完璧です。

```
function func() {
   var count = 0;
   function cup() {
      log(++count);
   }
   return cup;
}
var f = func();
f();              // 1
f();              // 2
f();              // 3
log(f.count);     // undefined
log(count);       // ReferenceError: 'count' is undefined
```

これもクロージャです。即時関数の形です。

```
var f = (function() {
   var count = 0;
   return {
      cup: function() {
         count++;
         return count;
      }
   };
})();
log(f.cup());     // 1
log(f.cup());     // 2
log(f.cup());     // 3
log(f.count);     // undefined
log(count);       // ReferenceError: 'count' is undefined
```

closure は、動詞 close の名詞形です。動詞 close の意味は、〈ドア、窓、目、口などを〉閉じる、閉める、という意味以外に、〈すき間・傷口などを〉〈…で〉ふさぐ、例：close a wound with stitches 傷口を縫ってふさぐ、というような使い方があります。上記のプログラム例で言えば、変数 count をふさいで、外から見えなくしているという意味合いでしょう。コンピュータ用語としての closure の日本語訳は「関数閉包」です。

クリ：ちなみに、透明マントは、ドラえもんのひみつ道具のひとつじゃ。外観は透明色の布で、これで覆った物は目に見えなくなる。 ハリー・ポッターシリーズにも登場しておる。

なな：先生、クリじい、お疲れ様！