ななちゃんのIT教室

This is not a pineapple pen の巻

by nara.yasuhiro@gmail.com

ななちゃんが、JavaScript の
this の使い方を学ぶという お話

第 0.1 版 2017 年 6 月 18 日

フリー素材
http://freeillustration.net

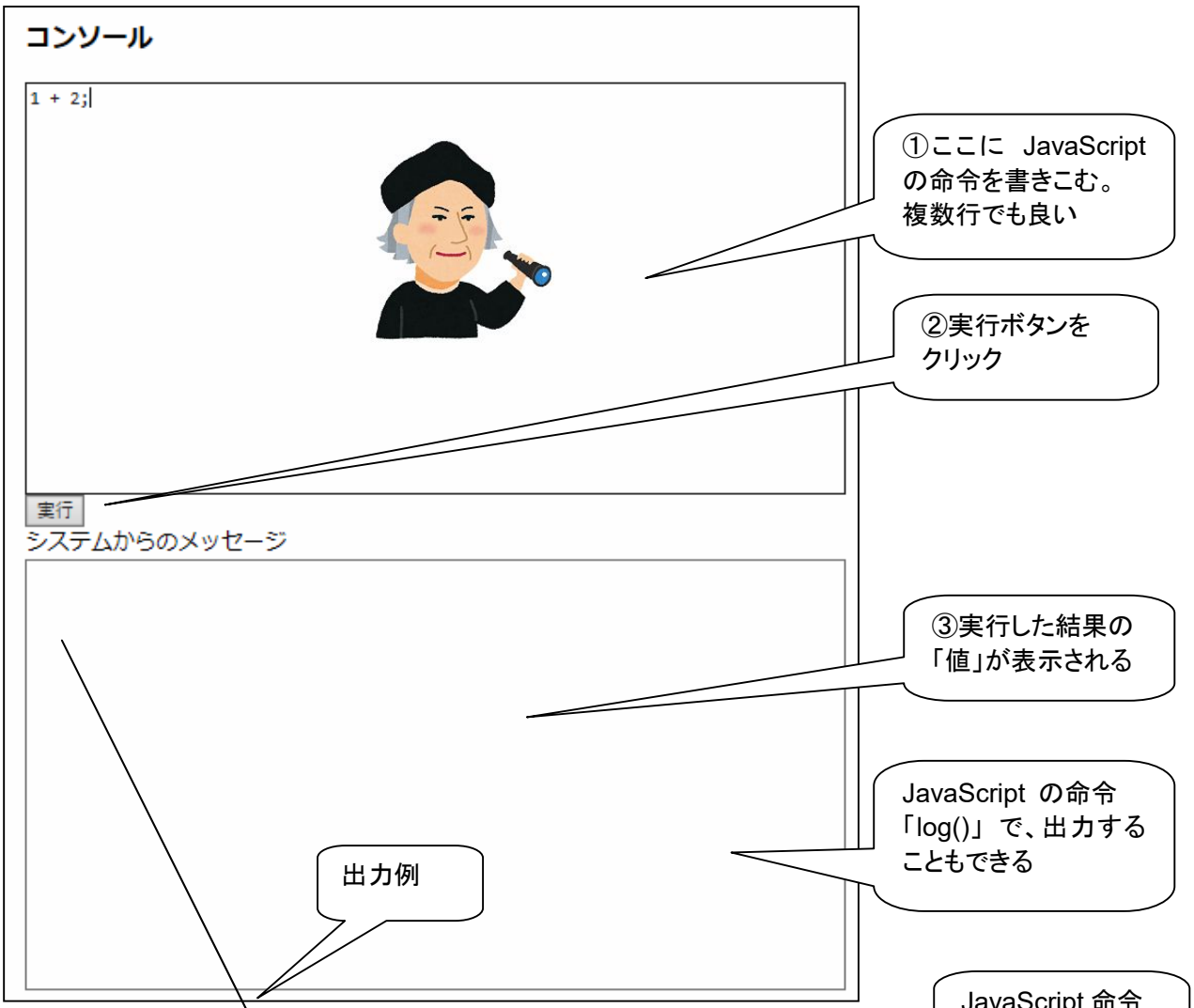
いらすとやフリー素材
http://www.irasutoya.com/

もくじ

第1回 秘密道具:マイ・コンソール

なな: クリじい、「This is a point」の勉強をするんだけど、便利な秘密道具はない？

クリ: あるぞ、あるぞ。定番秘密道具の「マイ・コンソール」。他の巻を読んでない読者のために、説明しよう。




```
<= 1 + 2;
=> Number number 3
<= "1" + "2"
=> String string "12"
<= 1;2;
=> Number number 2
<= var x = 1;
=> Undefined undefined undefined
<= x
=> Number number 1
<= var x; x= 1;
=> Number number 1
```

JavaScript 命令「1+2」を入力した

```
<= 1 + 2;
=> Number number 3
```

実行結果の「値」は 3

注意:var x = 1; の値は「undefined」

実行結果の「型」は Number

本 教材ではこのように圧縮表示しています

```
1 + 2;          // Number number 3
"1" + "2"       // String string "12"
1;2;            // Number number 2
var x = 1;      // Undefined undefined undefined
x               // Number number 1
var x; x= 1;    // Number number 1
```

「型」の判定方法は 2 種類 r

「○; ○」のように、複数の JavaScript 命令がある場合、一番右の命令の型、値だけ表示される

JavaScript 命令

実行結果の「型」と「値」

```
<!DOCTYPE html>
<html>
 <head>
    <meta charset="utf-8">
    <title>コンソール</title>
 </head>
 <body>
    <h3>コンソール</h3>
    <textarea rows="19" cols="80" id=pg autofocus>1 + 2;</textarea>
    <br><input type=button onClick=go() value="実行">
    <br>システムからのメッセージ
    <br><textarea rows="20" cols="80" id=log></textarea>
    <script>
var geval = eval;

var logp = document.getElementById("log");
var pgp    = document.getElementById("pg");
var logd;

function clog(s) { logp.value += s; }
function log(s) { logd += s; }
function typeIs(obj) {
      return(Object.prototype.toString.call(obj).slice(8, -1)); }
function isPrimitive(x) {
    return (typeof x)!="object";
}
function toLiteral(x) {
    if (typeIs(x)=="Number" && isNaN(x)) return "NaN";
    if (x === Infinity)                  return "Infinity";
    if ((typeIs(x)!="Symbol")&&(-x === Infinity)) return "-Infinity";
    if (typeIs(x)=="Set") return "Set("+JSON.stringify([...x])+")";
    if (typeIs(x)=="Map") return "Map("+JSON.stringify([...x])+")";
    return JSON.stringify(x);
}
function type(x) { return "" + (typeof x); }
function isInteger(n) { return n%1 === 0; }
function keys(obj) { return Object.keys(obj); }
function go() {
 logd = "";
 try {
            var v = geval(pgp.value);
            clog("<= " + pgp.value + "\n=> "
               + typeIs(v) + " " + type(v) + " " + toLiteral(v) + "\n");
            pgp.value = "";
            logp.scrollTop = logp.scrollHeight;
            pgp.focus();
 }
 catch(e) { clog("<= " + pgp.value + "\n=>! " + e + "\n");
                pgp.value = "";
                logp.scrollTop = logp.scrollHeight;
                pgp.focus();
 }
 if (logd != "") clog(logd + "\n");
}
   </script>
 </body>
</html>
```

第2回 this の初期値は window、関数（メソッド）呼び出しでは変化しない

なな：this って何？

先生：this はキーワード。変数に似ているけど、プログラムで書き換えることはできません。書き換えようとするとエラーになります。

```
this = { }          // ! ReferenceError: Invalid left-hand side in assignment
```

this は、基本的に、プログラムが現在どのオブジェクトの中で実行されているかを示しています。関数内ではない環境では、キーワード this の値は キーワード window の値と同じになっています。これは、プログラムがグローバル環境で動作していて、ブラウザで一番外側の Window オブジェクトの中で動作していることを表しています。

```
typeIs(this)       // String string "Window"
this===window // Boolean boolean true
var x; x = 123   // Number number 123
window.x          // Number number 123
this.x             // Number number 123
```

Window は、ブラウザの一番外側のオブジェクトです。変数 x は、window.x と同じです。「window.」は省略可能というルールを適用したということになります。「変数 x は、グローバル変数である」と言います。この状態では、this.x と書いても同じことになります。

```
var x = 123;

function f() {
    var x = 456;
    return x;
}
f()                    // Number number 456
```

グローバル変数（x）と同じ名前の変数（x）を関数の中で定義すると、グローバル変数（x）は、関数の中から、x という名前ではアクセスできなくなります。

なな：this や window を使って、グローバル変数にアクセスできないかしら。

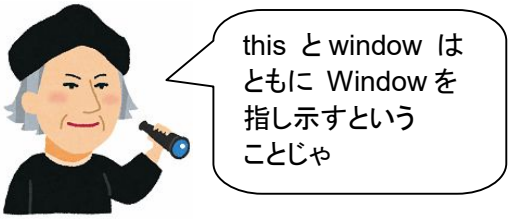
先生：するどいわね。できますよ。

```
var x = 123;

function f() {
    var x = 456;
    return window.x;
}
f()                    // Number number 123
```

window.x と書けば、関数の中からグローバル変数にアクセスすることができます。

```
function f() {
    return this === window;
}
f()                    // Boolean boolean true
```



```
var x = 123;

function f() {
    var x = 456;
    return this.x;
}
f()                  // Number number 123
```

↑普通に関数を呼び出した場合、関数の中でも、this の値は変化しません。 ですから、this.x と書いても、グローバル変数にアクセスできます。

```
var x = 123;

function f() {
    var x = 456;
    return g();
}

function g() {
    return x;
}

f()                  // Number number 123
```

↑これは、this とは関係ありませんが、関数 f() を呼び出して、それが関数 g() を呼び出した場合、g() の中からアクセスできるのは、その呼び出し元 f() の変数ではなく、グローバル変数です。

```
var x = 123;

function f() {
    var x = 456;
    return g();
    function g() {
        return x;
    }
}
f()                  // Number number 456
```

↑これも、this とは関係ありませんが、関数 g() が、関数 f() の内部で定義されている場合は、f() の変数にアクセスできます。

```
var x = 123;

function f() {
    var x = 456;
    return g();
    function g() {
        return this.x;
    }
}
f()                  // Number number 123
```

↑関数呼び出しにおいて、this の値は変換しないので、g() の中でも、this は、window と同じ値になっています。

関数内での this の値は、どこからその関数を呼び出したかに依存します。つまり、いつでも window の値と同じであることが保証されるわけではありません。ですから、関数の中からグローバル変数の内容を参照したい場合は、this より window を使ったほうが「安全」です。

第3回 HTML 要素の中の this

先生：今回は、myConsole.html は使いません。HTML要素の中の this は、要素オブジェクト自身を指し示しています。しかし、そこから呼び出した関数の this には影響しません。つまり、そこから呼び出した関数の中での this の値は、要素オブジェクトにはなりません。

```
<input type=text onclick=go(this) value=""><br>
<input type=text onclick=go(this) value=""><br>
<script>
function go(elm) {
    elm.style.backgroundColor = "red";
}
</script>
```

引数を通じて input 要素の this を伝える必要がある。function go() の中の this は Window オブジェクトを差し示している。

上記のような内容を、たとえば、test.html という名前のファイルに書き込み、そのファイルアイコンをクリックすることで、ブラウザで実行させます。

要素オブジェクトを、そこから呼び出した関数に伝えるには、引数を利用します。
上記のプログラムでは、input 枠がふたつ表示され、マウスでクリックすると、クリックしたほうの input 枠だけ、背景色が赤くなります。

クリ：実験で確かめてみよう。

```
<input type=text onclick=go(this) value=""><br>
<input type=text onclick=go(this) value=""><br>
<script>
function typeIs(obj) {
      return(Object.prototype.toString.call(obj).slice(8, -1)); }

function go(elm) {
    elm.value = typeIs(elm);      // HTMLInputElement
}
</script>
```

実験じゃ。

クリックしたほうの input 枠に HTMLInputElement と表示

引数を通して、input 要素のポイントが送られている

先生：変数の値の「型」を調べる関数 typeIs() を定義した上で、前記プログラムの go() に送られる引数の値を調べてみましょう。上記のプログラムは、クリックした input の枠内に、引数の「型」を表示します。結果は「HTMLInputElement」です。つまり、引数の値は、HTML の input 要素へのポインタになっています。

```
function go(elm) {
    elm.value = typeIs(this);      // Window
}
```

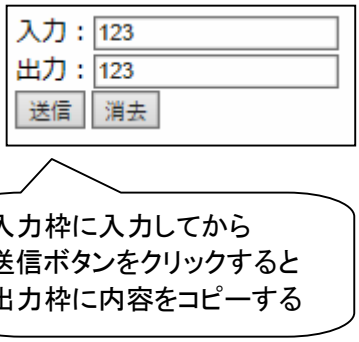
go() の内容を、上記のように書き換えると、go() 内部での、this の「型」をが分かります。結果は、Window になっています。これは、window キーワードと同じということです。

この関数 go() は、input 要素から呼び出されていますが、this の値の「型」は、HTMLInputElement ではなくて、Window になっています。

なな：実験すると、理解がすすむわね。

なな：原理だけでなく、応用例を見てみたいな。

```
<form onSubmit="go(this);return false">
    入力:<input type=text name=ipt><br>
    出力:<input type=text name=opt><br>
    <input type=submit value="送信">
    <input type=reset    value="消去">
</form>
<script>
function go(elm) {
    elm.opt.value = elm.ipt.value;
}
</script>
```



先生：form 要素の this の値を、引数を通じて go() に伝え、form 内の ipt から、opt に内容をコピーするものです。

入力枠に文字列を書き込み、「送信」ボタンをクリックすると、入力枠の内容が、出力枠にコピーされます。

なな：return false はどういう意味？

先生：本来、form の onSubmit では、ウェブサーバにデータを送信するものですが、今回の例では送信は行わないので、「return false」としています。もともとは、入力内容に問題があるなどで、データ送信を中止する時に使う設定です。

なな：input タグに、name=ipt、name=opt と書いてあるのね。id=ipt、id=opt の間違いでは？

先生：この使い方では、name= を使います。このやり方は、ちょっと古い方法です。最近は、id= を使い、document.getElementById('ipt') を使うのが主流です。

第4回 イベントハンドラ内の this は、イベントソース要素

なな: 今回は、何を勉強するの？

```
<input type=text id=i1><br>
<input type=text id=i2><br>
<script>
function go() {
      this.style.backgroundColor = "red";
}

document.getElementById("i1").onclick = go;
document.getElementById("i2").onclick = go;
</script>
```

先生: input 枠から直接呼び出すのではなく、イベントハンドラとして呼び出すと、this が、イベントを発生した HTML 要素を指し示すように設定されます。

上記のプログラムでは、input 枠がふたつ表示され、マウスでクリックすると、クリックしたほうの input 枠だけ、背景色が赤くなります。

クリ: じっけん、じっけn。

```
<input type=text id=i1><br>
<input type=text id=i2><br>
<script>
function typeIs(obj) {
      return(Object.prototype.toString.call(obj).slice(8, -1)); }

function go() {
      this.value = typeIs(this);    // HTMLInputElement
}

document.getElementById("i1").onclick = go;
document.getElementById("i2").onclick = go;
</script>
```

クリ: 上記のプログラムを実行し、どちらかの input 枠をクリックすると、その枠内に「HTMLInputElement」と表示されるんじゃ。this が、input要素を指し示していることが分かるぞ。

先生: ちなみに、今回の最初のプログラムは、this を使わずに、下記のように書くこともできます。

```
<input type=text id=i1><br>
<input type=text id=i2><br>
<script>
function go(e) {
      e.target.style.backgroundColor = "red";
}

document.getElementById("i1").onclick = go;
document.getElementById("i2").onclick = go;
</script>
```

e は、イベント発生源に関するさまざまなデータを持つオブジェクトです。

なな: こっちの話は聞いたことがあるような気がするけど、this でも大丈夫というのが新知識でした！

第5回　オブジェクト内の this は、そのオブジェクト

なな：今回のテーマは何？　今回のひみつ道具は？

先生：また、myConsole.html を使います。

```
var x = 123;
var obj1 = {
      x: 456,
      fn: function() { return this.x; }
}
obj1.fn();          // Number number 456
```

「オブジェクト内にある関数（メソッド）」を呼び出すと、関数内の this は、そのオブジェクトを指し示すように設定されます。ですから、上記の例の this.x は、obj1 オブジェクトの中の変数 x を示すことになります。

```
var obj1 = {
      x: 1,
      fn: function() { return this.x; }
}
var obj2 = {
      x: 2,
      fn: function() { return obj1.fn(); }
}
obj2.fn();          // Number number 1
```

上記の例では、obj2.fn() を呼び出し、その中から obj1.fn() を呼び出しています。はじめから obj1.fn() を呼び出すのと同じです。オブジェクト内にある関数（メソッドといいます）を実行する時には、this の値がそのオブジェクトになる、ということです。

```
var obj1 = {
      x: 1,
      fn: functi                    ς; }
}
var obj2 = {
      x: 2,
      fn: obj1.f
}
obj2.fn();          // Number number 2
```

この例では、obj2.fn() を呼び出し、その中から obj1.fn() を呼び出しているのではありません。obj2.fn() を呼び出しているだけです。obj2.fn() のプログラムが、obj1.fn() のプログラムを借りているだけです。つまり、「fn: obj1.fn」は、「fn: function() { return this.x; }」と書いたのと同じだということです。

なな：関数とメソッドの違いは？

先生：オブジェクト内で、プロパティとして定義されている関数のことを特に「メソッド」と呼びます。 関数は、f() のような形で呼び出します。メソッドは、obj.f() とか、this.f() のような形で呼び出します。f() は、window.f() であると考えると、関数とメソッドの区別はあやしくなってきます。一般に、window.f() は関数、それ以外の、obj.f() をメソッドと呼んでいます。

なな：分かったような気がするけど、グローバル変数の x が、実は Window オブジェクトの x プロパティだというのがトリッキーね。

先生: 以下は、有名な問題（トラブル）です。とても不自然で、不便なのですが、「JavaScript はそういう仕様なのでしょうがない」というものです。

```
var x = 123;

var obj = {
   x: 456,
   f: function() {
         function g() {
            return this.x;
         }
      return g();
   }
};

obj.f();  // Number number 123
```

関数の内部に定義されている子供関数では、this が window にリセットされてしまう、という問題です。

```
var x = 123;

var obj = {
   x: 456,
   f: function() { return g(); }
};

function g() {
      return this.x;
}

obj.f();  // Number number 123
```

このように、g() が、子供ではなくて、外側に置かれているかのように動作するということです。g() が、f() の中からしか呼び出すことができない、f() の外には見えない、という制約があるだけです。

g() から、obj の x をアクセスさせたい場合は、 下記のように、 変数 self を用意し、それに this の値をコピーします。

```
var x = 123;

var obj = {
   x: 456,
   f: function() {
        var self = this;
           function g() {
              return self.x;
           }
      return g();
   }
};
obj.f();  // Number number 456
```

この手法を使う場合、変数名は「self」、「that」、「_this」のいずれかにしましょうという、暗黙のルールがあるほど、よく使われる、有名な手法です。

な な: どうしようもないの？ function の仕様を変えるとか。

先生: ECMAScript 2015 で導入された、アロー関数では、この問題は解決されています。次回の apply、call、bind で、this を設定することもできます。function は、仕様を変えると、過去に作ったプログラムが動かなくなるおそれがあるので、変更していません。

第6回　関数の apply、call 呼び出し、bind

なな：前回の、function の問題対策に役立つ apply、call、bind の予告があったけど。

先生：関数 f を呼び出す場合、f() と書く代わりに、f.apply() とか、f.call() と書くという方法があります。その時に、「this」を設定（強制的に束縛）できるんです。

```
var obj1 = {
   x: 1,
   f: function() { return this.x; }
};

var obj2 = {
   x: 3
};

obj1.f();                  // Number number 1

obj1.f.apply(obj2);        // Number number 3
obj1.f.call(obj2);         // Number number 3
```

この例では、obj1 の f を呼び出すのに、this は obj2 にしてしまう、ということを実現しています。「apply」と「call」の違いは、第二引数以降の書き方だけです。「apply」は、第二引数に配列をとり、配列の中身が引数として渡されます。「call」は、第二引数以降がそのまま渡されます。第一引数は両者とも「this」です。

```
var obj1 = {
   add: function(v1, v2) {
      return this.x + v1 + v2;
   }
};

var obj2 = {
   x: 3
};

obj1.add.apply(obj2, [1, 2]);     // Number number 6
obj1.add.call(obj2, 1, 2);        // Number number 6
```

[1, 2]

1, 2

3 + 1 + 2 = 6

なな：bind は？

先生：bind() にも、this を設定する機能があります、bind() には、第二引数以降で、対象関数の引数を挿入する機能もあります。

```
obj1.add.bind(obj2)(1, 2);        // Number number 6
obj1.add.bind(obj2,1)(2);         // Number number 6
obj1.add.bind(obj2,1,2)();        // Number number 6
```

第7回 インスタンス内の this は、そのインスタンス

なな: インスタンス？

```
function greet(n) {
    this.name = n;
    this.say = function() {
        return "My name is " + this.name;
    }
}

var j = new greet("Java");   j.say();      // String string "My name is Java"
var s = new greet("Script"); s.say();      // String string "My name is Script"
```

先生:この例では、greet() は、関数の形をしていますが、「クラス定義」(コンストラクタ関数)として使っています。そして、j、s という、ふたつの「インスタンス」を、new を使って生成しています。こういう使い方をする場合、this は、それぞれのインスタンス内部で、自分自身のインスタンスを指し示すように設定されます。このプログラムだと、関数 say() の定義は、インスタンス j、s にコピーされます。

```
function greet(n) {
   this.name = n;
}
greet.prototype.say = function() {
    return "My name is " + this.name;
}

var j = new greet("Java"); j.say();        // String string "My name is Java"
var s = new greet("Script"); s.say();      // String string "My name is Script"
```

このような書き方にすると、say() は、greet クラスのメソッド(関数)になります。インスタンス i、j にはコピーされません。インスタンス i、j が、クラス内蔵の say() を呼び出すという形になります。

応用例です。

```
function myClass() {
    this.x = 0;
    this.set = function(d) { this.x = d;          return this; }
    this.add = function(d) { this.x += d;         return this; }
    this.show = function() { return this.x; }
}

var ins = new myClass();

ins.set(10).add(5).show();     // Number number 15
```

このように、それぞれのメソッド(関数)の末尾に「return this」を入れておくと、メソッドを連ねることができるようになります。メソッドチェーンと呼んだりします。「ins.set(10).add(5).show();」は、「ins.set(10); ins.add(5); ins.show();」と同じことになります。データを次々に加工してゆくような問題に適しています。

例えば、音声データに、低音強調フィルタを掛けた上で、音量調整するとか、画像データに、色補正処理をしたうえで、輪郭強調し、最後にサイズ調整をするとか、といったプログラムに使えます。

応用例その2. HTML 要素から onclick で関数を呼び出す例です。call() を使って、input要素のポインタで this を束縛しています。

```
<input type=text onclick=go.call(this) value=""><br>
<input type=text onclick=go.call(this) value=""><br>
<script>
function go() {
    this.style.backgroundColor = "red";
}
</script>
```