# ななちゃんのIT教室

## 新大陸SetMapを探れの巻

by nara.yasuhiro@gmail.com

ななちゃんが
ES2015 の Set/Map に挑戦するという お話

第 0.2 版　2017 年 6 月 12 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　Set とは
第2回　Map とは

# 第1回　Set とは

なな：Set って何？

先生：Set は、数学の「集合」の意味です。配列に似ているけど、重複するデータ、つまり、「== で true」になるようなデータの登録は無視されます。ES2015 で導入されたオブジェクトです。

クリ：EcmaScript 2015 といっても、実際にブラウザ上で使えるようになったのは、ごく最近じゃ。いわば、新大陸発見のようなものじゃ。他の人たちより先に使いこなせるようになりたいものじゃ。

なな：どうやって使うの？

先生：「new Set ( iterable )」のような形で生成します。「iterable」は、配列や、文字列などです。配列の要素は、どんなオブジェクトでかまいません。たとえば、

```
var set; set = new Set([1, 'one', [1,2],{ a:1}]);      // Set object Set([1,"one",[1,2],{"a":1}])
[...set]                                               // Array object [1,"one",[1,2],{"a":1}]
```

ここでは、配列に変換してから内容を表示しています。
それから、「new Set()」で、空の Set を作り、add() で要素を追加することもできます。

```
var set; set = new Set();          // Set object Set([])
set.add(2);                        // Set object Set([2])
set.add('a')                       // Set object Set([2,"a"])
```

なな：他に、どんな操作があるの？

先生：・値の存在確認

```
set.has(2);                        // Boolean boolean true
set.has(3);                        // Boolean boolean false
```

・値の削除

```
set                                // Set object Set([2,"a"])
set.delete(2)                      // Boolean boolean true
set                                // Set object Set(["a"])
```

・格納してあるデータ数

```
set                                // Set object Set([2,"a"])
set.size                           // Number number 2
```

・値の取得

```
set                                // Set object Set([2,"a"])
for (var value of set) log(value+"/");     // Undefined undefined undefined 2/a/
```

・values()

```
set                                // Set object Set([2,"a"])
var it; it = set.values()          // Set Iterator object {}
it.next()                          // Object object {"value":2,"done":false}
it.next()                          // Object object {"value":"a","done":false}
it.next()                          // Object object {"done":true}
```

・keys()、entries()

```
[...set]                    // Array object [2,"a"]
[...set.keys()]             // Array object [2,"a"]
[...set.values()]           // Array object [2,"a"]
[...set.entries()]          // Array object [[2,2],["a","a"]]
```

Set では、key と value が同じ。配列のような順序概念はない。

・値の取得（forEach）

```
[...set]                    // Array object [2,"a"]
set.forEach(function (v1, v2, s) { log(v1+":"+v2+":"+s+"/"); });
                            // Undefined undefined undefined
                            // 2:2:[object Set]/a:a:[object Set]/
```

・clear()

```
Set                         // Set object Set([2,"a"])
set.clear()                 // Undefined undefined undefined
set                         // Set object Set([])
```

・[Symbol.iterator]()

```
set                         // Set object Set([1,2,3])
var it; it = set[Symbol.iterator]()   // Set Iterator object {}
it.next()                   // Object object {"value":1,"done":false}
it.next()                   // Object object {"value":2,"done":false}
it.next()                   // Object object {"value":3,"done":false}
it.next()                   // Object object {"done":true}
```

なな：応用例は？

先生：積集合

```
var set1; set1 = new Set([1,2,3]);          // Set object Set([1,2,3])
var set2; set2 = new Set([2,3,4]);          // Set object Set([2,3,4])
var set3; set3 = new Set([...set1].filter(x => set2.has(x))); // Set object Set([2,3])
```

和集合

```
set1                                        // Set object Set([1,2,3])
set2                                        // Set object Set([2,3,4])
var set4; set4 = new Set([...set1,...set2]) // Set object Set([1,2,3,4])
```

注意：Set オブジェクトのリテラル値（定数表現）は存在しません。ですから、Set オブジェクトを作るには new Set() を用いたり、それに add() で項目を追加するしかありません。

先生：WeakSet オブジェクトというものもあります。Set に似ていますが、オブジェクトの参照（ポインタ、非プリミティブ）のみを、登録できる点が異なります。WeakSet オブジェクトには、値情報を列挙する手段がありません。size プロパティはありません。登録したオブジェクトが、どこからも参照されなくなった場合、ガベージコレクションの対象となって消滅します。WeakSet オブジェクトで利用中だったとしても、値情報は消滅してしまいます。こういうのを弱参照といいます。WeakSet オブジェクトは、イテレーターではありません。

```
var a1; a1 = [];                    // Array object []
var o1; o1 = {};                    // Object object {}
var r1; r1 = /a/;                   // RegExp object {}
var ws; ws = new WeakSet([ a1, o1, r1 ] );      // WeakSet object {}
ws.has(a1);                         // Boolean boolean true
ws.has([]);                         // Boolean boolean false
ws.size;                            // Undefined undefined undefined
```

# 第2回　Map とは

なな：Map って何？

先生：オブジェクトに似ているけど、キーの名前の制限がなく、文字列以外に、数値やオブジェクトなどを指定することができます。EcmaScript 2015 から導入されたクラスです。連想配列、ハッシュ、辞書ともいわれます。

クリ：EcmaScript 2015 といっても、実際にブラウザ上で使えるようになったのは、ごく最近じゃ。いわば、新大陸発見のようなものじゃ。 Map も、他の人たちより先に使いこなせるようになりたいものじゃ。

・新規作成

```
var map; map = new Map();       // Map object Map([])

var map; map    = new Map([
      ['1', 'string'],
      [1, 'number'],
      [[], 'array'],
      [{}, 'object']
]);                  // Map object Map([["1","string"],[1,"number"],[[],"array"],[{},"object"]])
```

・値の取得

```
map                // Map object Map([["1","string"],[1,"number"],[[],"array"],[{},"object"]])
map.get('1');              // String string "string"
map.get('5')               // Undefined undefined undefined
```

・値の設定

```
map.set("2","also a string")
  // Map object Map([["1","string"],[1,"number"],[[],"array"],[{},"object"],["2","also a string"]])
```

・値の設定（キーが同じものは上書きされる）

```
map              // Map object Map([["1","string"],[1,"number"],[[],"array"],[{},"object"]])
map.set("1","STRING")
                 // Map object Map([["1","STRING"],[1,"number"],[[],"array"],[{},"object"]])
```

・格納したデータ数

```
map              // Map object Map([["1","string"],[1,"number"],[[],"array"],[{},"object"]])
map.size         // Number number 4
```

・存在確認

```
map              // Map object Map([["1","string"],[1,"number"],[[],"array"],[{},"object"]])
map.has('1')     // Boolean boolean true
map.has('3')     // Boolean boolean false
```

・データの削除（1つ）

```
map              // Map object Map([["1","string"],[1,"number"],[[],"array"],[{},"object"]])
map.delete("1")          // Boolean boolean true
map              // Map object Map([[1,"number"],[[],"array"],[{},"object"]])
map.delete("3")          // Boolean boolean false
```

・データの削除（全部）

```
map             // Map object Map([[1,"number"],[[],"array"],[{},"object"]])
map.clear()     // Undefined undefined undefined
map             // Map object Map([])
```

・ keyを全部取得

```
map             // Map object Map([["1","string"],[1,"number"],[[],"array"],[{},"object"]])
map.keys()      // Map Iterator object {}
[...map.keys()] // Array object ["1",1,[],{}]
```

・値の取得（values）

```
map             //Map object Map([["1","string"],[1,"number"],[[],"array"],[{},"object"]])
map.values()    // Map Iterator object {}
[...map.values()]          // Array object ["string","number","array","object"]
```

・値の取得（entries）

```
map             //Map object Map([["1","string"],[1,"number"],[[],"array"],[{},"object"]])
map.entries()   // Map Iterator object {}
[...map.entries()] // Array object [["1","string"],[1,"number"],[[],"array"],[{},"object"]]
```

・ for of 構文

```
map             //Map object Map([["1","string"],[1,"number"],[[],"array"],[{},"object"]])
for (var e of map) log(e+"/")      // Undefined undefined undefined
                                   1,string/1,number/,array/[object Object],object/
for (var e of map) log(JSON.stringify(e)+"/") // Undefined undefined undefined
                                   ["1","string"]/[1,"number"]/[[],"array"]/[{},"object"]/
for (var e of map.values()) log(JSON.stringify(e)+"/")
                                   // Undefined undefined undefined
                                      "string"/"number"/"array"/"object"/
```

・値の取得（forEach）

```
map             // Map object Map([["1","string"],[1,"number"],[[],"array"],[{},"object"]])
map.forEach(function(value, key, map) { log(value+":"+key+":"+map+"\n");});
                // Undefined undefined undefined
                string:1:[object Map]
                number:1:[object Map]
                array::[object Map]
                object:[object Object]:[object Map]
```

なな：WeakMap って？

先生：Map に似ているけど、キーがポインタ（参照）に限定され、プリミティブ型を使おうとするとエラーになるの。値は任意。キーが文字列型に限定される一般的なオブジェクトでは、複数のプログラマが間違って同じキーを使ってしまうリスクがあります。キーがポインタなら、その場でオブジェクトを生成し、それへのポインタをキーに使え、ユニーク性を保証できます。キーの一覧を得る手段が無いので、秘密が守られるというか、間違って書き換えられてしまう可能性がありません。

```
var k1 = [ ], k2 = {}, k3 = /a/;
var wkmap; wkmap    = new WeakMap([
      [k1, 'array'],
      [k2, 'object'],
      [k3, 'regexp']
])                                  // WeakMap object {}
wkmap.get(k1)                       // String string "array"
wkmap.get([])                       // Undefined undefined undefined
wkmap.size                          // Undefined undefined undefined
[...wkmap]                          // ! TypeError: Function expected
for (k of wkmap) ("/"+k+"/");
                  // ! TypeError: Object doesn't support property or method 'Symbol.iterator'
Array.from(wkmap)                   // Array object []
wkmap.set(1,"number")               // ! TypeError: WeakMap.prototype.set: 'key' is not an object
wkmap.set(k2,"OBJECT")              // WeakMap object {}
wkmap.get(k2)                       // String string "OBJECT"
k1 = []                             // Array object [ ]
wkmap.get(k1)                       // Undefined undefined undefined
```

元の　k1　の値（ポインタの実体）は消滅！

```
var a; a = {};                      // Object object {}
var m; m = new Map();               // Map object {}
m.set(a,100);                       // Map object {}
m.get(a);                           // Number number 100
m.get({});                          // Undefined undefined undefined
m.size                              // Number number 1
Array.from(m)                       // Array object [[{},100]]
for (k of m) log("/"+k+"/");        // Undefined undefined undefined /[object Object],100/
a = {}                              // Object object {}
var key; key = Array.from(m)[0][0] // Object object {}
m.get(key)                          // Number number 100
```

元の　a　の値（ポインタの実体）の発掘成功！

Map では、m のエントリは消滅しません。キーの一覧を得て、そこから取り出すことができるので。

WeakMap では、key の一覧表示ができないので、key を無くすと（key を記憶している変数を書き換えると）、情報は取り出せなくなります。それどころか、対象情報は、どこからも参照されない状態になり、システムから捨てられてしまいます。ようするに、使われないので、削除されてしまいます。 こういうのを「ガーベッジコレクション」（ゴミ回収）といいます。メモリ（記憶装置）を回収して、他のデータ記憶用に再利用するということです。