ななちゃんのIT教室

ハトは平和のシンボル？の巻

by nara.yasuhiro@gmail.com

ななちゃんが
シンボルオブジェクトの使い方を学ぶという お話

第 0.1 版　2017 年 6 月 11 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　Symbol オブジェクトとは

なな：「Symbol オブジェクト」って何？

先生：複数人で、手分けしてプログラムを作る時に、対象オブジェクトのプロパティ名として同じ名前を異なる目的に使ってしまうと正しく動きません。また、便利プログラムをまとめたライブラリを利用する場合、ライブラリで使っているプロパティ名と同じ名前をユーザが使ってしまうと誤動作の原因になります。そのようなことを避けるために、重複のない名前（識別子）を生成する仕組みが Symbol オブジェクトです。

なな：x1542 とか、使いそうも無い名前を、乱数を使って決めるの？

先生：乱数でも、偶然に一致するという心配はあるのね。Symbol オブジェクトでは、プログラム実行中に、名前を自動生成し、重複しないようにチェックしているの。

なな：注意・は？

先生：Symbol オブジェクトはプリミティブ データ型です。参照型ではありません。new 演算子を使用して作成することはできません。文字列同様に、オブジェクトのプロパティのキーとして使えます。文字列ではないので、既存の文字列のプロパティ名と衝突する可能性がありません。

なな：どうやって作るの？

先生：obj = Symbol ([desc]) という形で生成します。desc は、シンボルの注釈で、省略可能です。

```
var key; key = Symbol();        // Symbol symbol undefined
var obj; obj = { }              // Object object {}
obj[key] = 100                  // Number number 100
obj[key]                        // Number number 100
```

なな：一意的って？ 先生：実験してみましょう。

```
var sym0; sym0 = Symbol()       // Symbol symbol undefined
var sym1; sym1 = Symbol()       // Symbol symbol undefined
sym0 == sym1                    // Boolean boolean false
Symbol() == Symbol()            // Boolean boolean false
sym1 === sym1.valueOf()         // Boolean boolean true
```

なな：注釈って？　先生：こんなふうに使います。

```
var sym2; sym2 = Symbol('abc')          // Symbol symbol undefined
Symbol('abc') === Symbol('abc')         // Boolean boolean false
Symbol('abc') == Symbol('abc')/         / Boolean boolean false
```

なな：注釈って、痕跡は残らないの？　先生：to.String() で調べられます。””+ ではダメなので注意。

```
var sym0; sym0 = Symbol();      // Symbol symbol undefined
var sym1; sym1 = Symbol('abc')          // Symbol symbol undefined

sym0.toString()                 // String string "Symbol()"
sym1.toString()                 // String string "Symbol(abc)"

""+sym1             // ! TypeError: Object doesn't support property or method 'ToString'
String(sym1)                    // String string "Symbol(abc)"
Number(sym1)                    // ! TypeError: Number expected
+sym1                           // ! TypeError: Number expected
Boolean(sym1)                   // Boolean boolean true
!!sym1                          // Boolean boolean true
```

なな：Symbol.for() って？

先生：Symbol.for('abc') を最初に実行した時は、新しい Symbolオブジェクトを返すけど、2 回目以降に実行した時は、前回と同じ Symbolオブジェクトを返します。つまり、'abc' という名前で、Symbolオブジェクトを公開したことになるの。Symbol('abc') とは別物なので、注意してね。

```
var sym1; sym1 = Symbol('abc')              // Symbol symbol undefined
var sym2; sym2 = Symbol.for('abc')    // Symbol symbol undefined
var sym3; sym3 = Symbol.for('abc')    // Symbol symbol undefined
sym1 === sym2                                  // Boolean boolean false
sym2 === sym3                                  // Boolean boolean true
sym1.toString()                               // String string "Symbol(abc)"
sym2.toString()                               // String string "Symbol(abc)"
Symbol.keyFor(sym1)                       // Undefined undefined undefined
Symbol.keyFor(sym2)                       // String string "abc"
```

なな：Symbol オブジェクトは、他のプログラマには見えないの？

先生：例：Symbol.for() 以外の方法で生成した Symbolオブジェクトは、一意的なので、それを公開しない限り、衝突はおきません。でも、生成した Symbolオブジェクトを変数に格納し、変数を公開することはできます。

```
function MyClass(privateData) {
    var key = Symbol();
    this[key] = privateData;
    this.getData = function() { return this[key]; }
}                   // Undefined undefined undefined
var c; c = new MyClass("private data") // Object object {}
c.getData()      // String string "private data"
c[key]             // ! ReferenceError: 'key' is undefined
Object.keys(c)  // Array object ["getData"]
Object.getOwnPropertySymbols(c) // Array object [null]
var idsym; idsym = Object.getOwnPropertySymbols(c)[0] // Symbol symbol undefined
c[idsym]                                  // String string "private data"
```

なな：結局、Symbol オブジェクトは、変数に記憶しないといけないわね。

先生：Symbol オブジェクトは秘匿性が目的ではないのよ。秘匿性が目的なら、下記のように、別の方法があるわ。

```
function MyClass(privateData) {
    var key = privateData;
    this.getData = function() { return key; }
}                   // Undefined undefined undefined
var c; c = new MyClass("private data") // Object object {}
c.getData()      // String string "private data"
c[key]             // ! ReferenceError: 'key' is undefined
Object.keys(c)  // Array object ["getData"]
Object.getOwnPropertySymbols(c) // Array object []
```

なな：秘匿性が目的でないなら、本当の目的は？

先生：共同開発者の Symbol オブジェクトは使わず、自分で生成するようにすれば、衝突の心配が無いということね。

なな：ビルトインシンボルというのもあると聞いたけど。

先生：たとえば、@@iterator。 Symbol.iterator　という形で使います。自分で作ったオブジェクトに、展開演算子（「…」）が適用できるようにする時に、展開内容を作るメソッドを定義します。

```
var obj; obj = { a: 1, b: 2, c: 3 } // Object object {"a":1,"b":2,"c":3}
obj[Symbol.iterator] = function () { return Object.keys(this).values() }
                                        // Function function undefined
[...obj]                                // Array object ["a","b","c"]
```

なな：他には？

先生：実装済のビルトインシンボルの一覧よ。気になるものがあれば、自分で調べてみてね。

・@@hasInstance
・@@iterator
・@@toPrimitive

・@@isConcatSpreadable
・@@toStringTag
・@@unscopables

・@@match @@replace @@search @@split
・@@species

シンボル	説明
Symbol.hasInstance	メソッド。コンストラクター オブジェクトがオブジェクトをコンストラクターのいずれかのインスタンスとして認識するかどうかを判別。instanceof 演算子によって内部的に使用される。
Symbol.iterator	メソッド。オブジェクトの既定の反復子を返す。for…of ステートメントによって内部的に使用される。
Symbol.toPrimitive	メソッド。オブジェクトを変換して、対応するプリミティブ型の値にする。ToPrimitive 抽象化操作で内部的に使用される。
Symbol.isConcatSpreadable	プロパティ。オブジェクトが Array.concat によって配列要素にフラット化される必要があるかどうかを示すブール値を返す。
Symbol.toStringTag	プロパティ。オブジェクトの既定の文字列説明を作成するために使用する文字列値を返す。Object.toString 組み込みメソッドによって内部的に使用される。
Symbol.unscopables	プロパティ。関連オブジェクトの with 環境のバインドから除外するプロパティを持つオブジェクトを返す。