ソフトウェア開発
第 4 回目授業

平野 照比古

2016/10/14

配列とオブジェクトの特徴

配列

- 配列はいくつかのデータをまとめて一つの変数に格納
- 各データを利用するためには `foo[1]` のように数による添え字を使用

オブジェクト

- オブジェクトでは添え字に任意の文字列を使うことができる

オブジェクトの例

```
var person = {
  name : "foo",
  birthday :{
    year : 2001,
    month : 4,
    day : 1
  },
  "hometown" : "神奈川",
}
```

この形式で表したデータをオブジェクトリテラルという。

コードの解説

- オブジェクトは全体を {} で囲む。
- 各要素はキーと値の組で表される。両者の間は : で区切る。
- キーは任意の文字列でよい。キー全体を "" で囲わなくてもよい。
- 値は JavaScript で取り扱えるデータなあらば何でもよい。上の例ではキー `birthday` の値がまたオブジェクトとなっている。
- 各要素の値を取り出す方法は 2 通りある。
  一つは . 演算子を用いてオブジェクトのキーをそのあとに書く。もう一つは配列と同様に [] 内にキーを文字列として指定する方法である。

データへのアクセスの例

```
>person.name;
"foo"
>person["name"];
"foo"
```

データへのアクセスの例–解説

- オブジェクトの中にあるキーをすべて網羅するようなループを書く場合や変数名として利用できないキーを参照する場合には後者の方法が利用される。
- キーの値が再びオブジェクトであれば、前と同様の方法で値を取り出せる。

```
>person.birthday;
Object {year: 2001, month: 4, day: 1}
>person.birthday.year;
2001
```

データへのアクセスの例–解説

- 取り出し方は混在してもよい。

```
>person.birthday["year"];
2001
```

- キーの値は代入して変更できる。

```
>person.hometown;
"神奈川"
>person.hometown="北海道";
"北海道"
>person.hometown;
"北海道"
```

データへのアクセスの例–解説

- 存在しないキーを指定すると値として undefined が返る。

```
>person.mother;
undefined
```

- 存在しないキーに値を代入すると、キーが自動で生成される。

```
>person.mother = "aaa";
"aaa"
>person.mother;
"aaa"
```

オブジェクトのキーをすべて渡るループ

- オブジェクトのキーをすべて渡るループは for-in で実現できる。
  - for(v in obj) の形で使用する。変数 v はループ内でキーの値が代入される変数、obj はキーが走査されるオブジェクトである。
  - キーの値は obj[v] で得られる。

```
>for(i in person) { console.log(i+" "+person[i]);};
name foo
birthday [object Object]
hometown 北海道
mother aaa
undefined
```

最後の undefined は for ループの戻り値である。

コンストラクタ関数の定義

コンストラクタ関数を用いて、前の例と同じオブジェクト (インスタンス) を構成している。

```
function Person(){
  this.name = "foo";
  this.birthday = {
    year : 2001,
    month : 4,
    day : 1
  };
  this["hometown"] = "神奈川";
}
```

コンストラクタ関数の定義–解説

- 通常、コンストラクタ関数は大文字で始まる名前を付ける。
- そのオブジェクト内にメンバーを定義するために、`this` をつけて定義する。ここでは、前の例と同じメンバー名で同じ値を設定している。
- この関数には `return` がないことに注意すること。

コンストラクタ関数を用いたオブジェクトの生成

- この関数を用いてオブジェクトを作成するためには、new をつけて関数を呼び出す。

```
>var person = new Person();
undefined
```

- 元来、戻り値がないので undefined が表示されているが、オブジェクトは作成されている。
- 前と同じ文を実行すれば同じ結果が得られる。

引数を持つコンストラクタ関数

- 引数を持つコンストラクタ関数も定義が可能
- 同じメンバーを持つオブジェクトをいくつか作る必要がある場合にプログラムが簡単になる

new を用いないで実行すると...

```
>p = Person();
undefined
>p.name
VM88:2 Uncaught TypeError: Cannot read property 'name' of undefi
>p;
undefined
>name;
"foo"
>window.name;
"foo"
>name === window.name
true
```

new を用いないで実行すると...–解説

- この関数は戻り値がないので、undefined が変数 p に代入される。
- このとき、キーワード this が指すのは非 strict モードではグローバルオブジェクト
- 現在の実行環境はブラウザ上なので、このときのグローバルオブジェクトは window である (非 strict モード)。
- strict モードでは指すものがない。
  - このとき、グローバル変数はすべてグローバルオブジェクトのメンバーとしてアクセス可能である。この例では this.name に値を代入した時点で変数 name が定義されている。
  - 最後の例からも、name と window.name が同じものであることがわかる。

constructor プロパティ

オブジェクトが作成されると、constructor というプロパティも設定
これはオブジェクトを作成したときに使われたコンストラクタ関数を返す。

constructor プロパティの確認

```
>var p = new Person();
undefined
>p.name;
"foo"
>p.constructor;
function Person(){
  this.name = "foo";
  this.birthday = {
    year : 2001,
    month : 4,
    day : 1
  };
  this["hometown"] = "神奈川";
}
```

Opera では定義全体が表示される。

オブジェクトのコンストラクタが呼び出せる

このプロパティに含まれるものは関数なので、コンストラクタの名前を知らなくても、元と同じオブジェクトのコピーが作成できる。

```
>np = new p.constructor();
Person {name: "foo", birthday: Object, hometown: "神奈川"}
>np.constructor;
function Person(){
  this.name = "foo";
  this.birthday = {
    year : 2001,
    month : 4,
    day : 1
  };
  this["hometown"] = "神奈川";
}
```

オブジェクトリテラルのコンストラクタ

- オブジェクトリテラルを使ってオブジェクトを作ると、組み込み関数の Object() コンストラクタ関数がセット
- このプロパティは for-in ループ内では表示されない。

```
>o = {}
Object {}
>o.constructor;
function Object() { [native code] }
```

instanceof 演算子

instanceof 演算でオブジェクトを生成したコンストラクタ関数が指定されたものかを判定できる。
次の結果は Opera で実行例

```
>p instanceof Person
true
>p instanceof Object;
true
>o instanceof Object;
true
>o instanceof Person
false
```

コンストラクタ関数で共通の変数を持つ

オブジェクト指向言語では同じコンストラクタ関数 (クラス) から生成されたオブジェクトに重複のない番号を付けることがある。JavaScript でそのようなことを実現するためには工夫が必要である。

失敗例 (1)

次のように単純にメンバーを追加しても、コンストラクタ関数が呼ばれるごとに、変数 ID が初期化されてしまい、目的を果たすことができない。

```
function Person(){
  var ID = 0;
  this.ID = ID++;
  this.name = "foo";
}

>p1 = new Person();
Person {ID: 0, name: "foo"}
>p2 = new Person();
Person {ID: 0, name: "foo"}
```

これから 2 つオブジェクトを作成しても、ID がともに 0 となっている

失敗例 (2)

クロージャを用いて関数ににすると一見、うまくいくように見える。

```
var Person2 = (function (){
  var ID = 0;
  return function(){
    this.ID = ID++;
    this.name = "foo";
  }
})();
```

失敗例 (2)–実行結果

これを実行すると次のようになる。

```
>p1 = new Person2();
Object {ID: 0, name: "foo"}
>p2 = new Person2();
Object {ID: 1, name: "foo"}
>p1.ID;
0
>p2.ID;
1
```

失敗とする理由

このプロパティは外部から変更が可能となってしまっている。

```
>p1.ID=10;
10
>p1.ID
10
```

改良版

クロージャ内の変数を参照するようなメソッドに変更

```
var Person3 = (function (){
  var ID = 0;
  return function(){
    this.getID = function() {
      return ID++;
    }
    this.name = "foo";
  }
})();
```

改良版も失敗

このコードでは getID() メソッドを実行するごとに、クロージャ内の変数 ID が増加してしまい、失敗である。

```
>p1 = new Person3();
Object {name: "foo"}
>p1.getID();
0
>p2 = new Person3();
Object {name: "foo"}
>p2.getID();
1
>p1.getID();
2
>p1.getID();
3
```

改良版の完成

呼ばれた時点での ID の値を保存するためには、いったん、スコープチェインを切る必要がある。

```
var Person4 = (function (){
  var ID = 0;
  return function(){
    this.getID = (function(x) {
      return function(){ return x;}
    })(ID++);
    this.name = "foo";
  }
})();
```

改良版の問題点

```
>p1 = new Person4();
Object {name: "foo"}
>p1.getID();
0
>p2 = new Person4();
Object {name: "foo"}
>p2.getID();
1
>p1.getID();
0
```

しかし、これでも getID() メソッドは書き直すことが可能である。

オブジェクトのプロパティやメソッドの分類

- インスタンスフィールド
  インスタンスごとに異なる値を保持できるプロパティ
- インスタンスメソッド
  クラスのすべてのインスタンスで共有されるメソッド
- クラスフィールド
  クラスに関連付けられたプロパティ
- クラスメソッド
  クラスに関連付けられたメソッド

JavaScript では関数もデータなのでフィールドとメソッドに厳密な区別はないのでフィールドとメソッドは同一視する。
`prototype` を用いるればクラスフィールドなども作成可能 (次回、解説)

プライベートプロパティとゲッター、セッター

- オブジェクト指向の言語ではフィールドをかってに操作されないようにするために、フィールドを直接操作できなくする
- 値を設定や取得するメソッドを用意
- フィールドにアクセスするため記述が面倒になる
- プロパティの代入の形をとっても実際はゲッターやセッター関数を呼ぶ形になっている言語も存在
- JavaScript の最新版 1.8.1 以降ではそれが可能

ECMAScript

- JavaScript は Ecma International が定義している ECMAScript の仕様に基づく
- 2016 年現在、最新バージョンの ECMAScript7 の仕様は ECMA-262 としてで公開
- オブジェクトのプロパティやメソッドにプロパティ属性という機能が追加

オブジェクトのプロパティの属性

属性名	値の型	説明	デフォルト値
value	任意のデータ	プロパティの値	undefined
writable	Boolean	false のときは value の変更ができない	true
enumerable	Boolean	true のときは for-in ループでプロパティが現れる。	false
configurable	Boolean	false のときはプロパティを消去したり、value 以外の値の変化ができない	false

メソッドのプロパティ属性

属性名	値の型	説明	デフォルト値
get	オブジェクトまたは未定義	関数オブジェクトだけ。プロパティの値が読みだし時に呼び出される	undefined
set	オブジェクトまたは未定義	関数オブジェクトだけ。プロパティの値を設定時に呼び出される	undefined
enumerable	Boolean	true のときは for-in ループでメソッドが現れる。	false
configurable	Boolean	false のときはメソッドを消去したり、value 以外の値の変化が不可	false

プロパティ属性の利用目的

これらの属性のうち、get や set を使うとオブジェクトのプロパティの呼び出しや変更に関して、いわゆるゲッター関数やセッター関数を意識しないで呼び出すことが可能
これらの属性は Object.defineProperty() や Object.defineProperties() 関数を用いて設定する。

オブジェクト属性の例

```
1  var Person = (function (){
2    var ID = 0;
3    return function(name, year, month, day, hometown){
4      this.name = name;
5      var getID = (function(x) {
6        return function(){ return x;}
7      })(ID++);
8      this.birthday = {};
9      this.birthday.year = year;
10     this.birthday.month = month;
11     this.birthday.day = day;
12     this.name = name;
13     Object.defineProperty(this, "ID",
14       {get: getID,
15        enumerable:true,
16        configurable:false
17       });
18     Object.defineProperties(this.birthday,
19       {year  : {enumerable : true},
20        month : {enumerable : true},
21        day   : {enumerable : false, writable : false}
22     });
23   }
24 })();
```

コードの解説

- 3 行目から 23 行目までが、Person() のコンストラクタ関数
- オブジェクトのメンバーを、コンストラクタ関数の引数で定義できる
- 5 行目から 7 行目で呼び出されたときの ID の値を保存し、参照するためのローカルな関数 getID() を定義
- 8 行目では this.birthday をオブジェクトとして初期化し、9 行目から 10 行目でそのオブジェクトに値を設定
- 13 行目から 22 行目で、このオブジェクトの ID を設定
  - get で参照される関数を 5 行目で定義された getID 設定している (14 行目)。
  - 15 行目で for in ループで表示されるように、15 行目ではプロパティを変更不可に defineProperty メソッドで設定 defineProperty メソッドの 3 番目の引数は設定しようとするプロパティを列挙したオブジェクトになっている。
- 18 行目から 22 行目でプロパティbirthday の各項目に必要な設定をしている

実行例

ID のプロパティでは set が設定されていないので、呼び出しの処理は行われない。

```
>p = new Person("foo",2001,4,1,"Japan");
Object {name: "foo", birthday: Object}
>p.ID;
0
>p.ID = 5;
5
>p.ID;
0
```

p.ID に値を代入してもエラーは起きていないので値が設定できたように見える。しかし、値を参照してみると変化がないことがわかる。

p.ID の消去

p.ID は消去できない。

```
>          delete p.ID
false
```

p のプロパティを列挙

3 つが表示されることがわかる。

```
>for(c in p) console.log(c+":"+p[c]);
 name:foo
 birthday:[object Object]
 ID:0
undefined
```

p.birthday のプロパティ

p.birthday のプロパティでは day が表示されない。

```
>for(c in p.birthday) console.log(c+":"+p.birthday[c]);
 year:2001
 month:4
undefined
```

これは 21 行目で enumerable を false に設定しているため

p.birthday.year の値の変更

p.birthday.year の値は変更できる。

```
>p.birthday.year = 2010
2010
>p.birthday.year;
2010
```

p.birthday.day の値の変更

p.birthday.day の値は writable が false に設定されているために変更できない。

```
>p.birthday.day = 20;
20
>p.birthday.day;
1
```

プロパティの消去

p.birthday.day は configurable が設定されていないので消去できる。

```
>delete p.birthday.day;
true
>p.birthday.day;
undefined
```

p.ID は configurable が false に設定されているので消去できない。

```
>delete p.ID;
false
>p.ID;
0
```

JSON とは

- JSON(JavaScript Object Notation) はデータ交換のための軽量なフォーマット
- 形式は JavaScript のオブジェクトリテラルの記述法と全く同じ
- JavaScript 内で、JSON フォーマットの文字列を JavaScript のオブジェクトに変換できる。
- JavaScript 内で、JSON フォーマットの文字列を JavaScript のオブジェクトに変換できる。
- JavaScript 内のオブジェクトを JSON 形式の文字列に変換できる。

JSON オブジェクト

JavaScript のオブジェクトと JSON フォーマットの文字列の相互変換の手段を提供する。

変換対象の JavaScript オブジェクト

次の例は 2 つの同じ形式からなるオブジェクトを通常の配列に入れたものを定義している。

```
var persons = [{
  name : "foo",
  birthday :{ year : 2001, month : 4, day : 1},
  "hometown" : "神奈川",
},
{
  name : "Foo",
  birthday :{ year : 2010, month : 5, day : 5},
  "hometown" : "北海道",
}];
```

JSON の処理の例

```
>s = JSON.stringify(persons);
"[{"name":"foo","birthday":{"year":2001,"month":4,"day":1},
"hometown":"神奈川"},
{"name":"Foo","birthday":{"year":2010,"month":5,"day":5},
"hometown":"北海道"}]"
>s2 = JSON.stringify(persons,["name","hometown"]);
"[{"name":"foo","hometown":"神奈川"},{"name":"Foo","hometow
>o = JSON.parse(s2);
[Object, Object]
>o[0];
Object {name: "foo", hometown: "神奈川"}
```

JSON の処理の例–解説

- JavaScript のオブジェクトを文字列に変更する方法は JSON.stringigy() を用いる。このまま見ると"がおかしいように見えるが表示の関係でそうなっているだけである。なお、結果は途中で改行を入れているが実際は一つの文字列となっている。
- JSON.stringigy() の二つ目の引数として対象のオブジェクトのキーの配列を与えることができる。このときは、指定されたキーのみが文字列に変換される。
- ここでは、"name" と "hometown"が指定されているので "birthday"のデータは変換されていない。
- JSON データを JavaScript のオブジェクトに変換するための方法は JSON.parse() を用いる。
- ここではオブジェクトの配列に変換されたことがわかる。
- 各配列の要素が正しく変換されていることがわかる。