ソフトウェア開発
第 4 回目授業

平野 照比古

2017/10/13

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

```
let 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。

JSON

- JSON(JavaScript Object Notation) はデータ交換のための軽量なフォーマット (文字列)
- 形式は JavaScript のオブジェクトリテラルの記述法と全く同じ
- 正しく書かれた JSON フォーマットの文字列をブラウザとサーバーの間でデータ交換の手段として利用
- JSON フォーマットの文字列を JavaScript のオブジェクトに変換可能
- JavaScript 内のオブジェクトを JSON 形式の文字列に変換可能

JSON オブジェクト

JSON オブジェクトは 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
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

```
let 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 オブジェクトの利用例

次の例はこのオブジェクトを JSON に処理させたものである。

```
>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","hometown":"北海道"}]"
>o = JSON.parse(s2);
[Object, Object]
>o[0];
Object {name: "foo", hometown: "神奈川"}
```

JSON オブジェクトの利用例 (解説)

- JavaScript のオブジェクトを文字列に変更する方法は `JSON.stringify()` を用いる。
- `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
- 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。

クラスの必要性

- 同じキーを持つオブジェクトを複数作成したい場合に同じようなコードを繰り返す必要がある。
- キーの追加をするときにはそれぞれのオブジェクトに対して修正が必要でプログラムのメンテナンスが面倒

オブジェクトのひな形を作成し、それをもとにオブジェクトを構成する。

- オブジェクトのひな形は通常クラスと呼ばれる。
- ES2015 ではクラスの宣言ができる。
- クラスから作成されたオブジェクトはこのクラスのインスタンスと呼ばれる。

クラスの例

前の例をクラスを用いて書き直す。

```
1  class Person{
2    constructor(name, year, month, day, hometown="神奈川"){
3      this.name = name;
4      this.birthday = {
5        year : year,
6        month : month,
7        day : day
8      };
9      this["hometown"] = hometown;
10   }
```

クラスの例–解説

- 1 行目がクラスの宣言
  - キーワード class の後にクラス名 Person を記述
  - 通常、クラス名は大文字で始まる。
  - 関数の時と同様にクラスの宣言を変数に代入することもできる。
- その後の{と}内にクラスの記述を行う。
- クラスの定義には初期化を行う constructor() が必須
  - ここでは 5 つの引数を取るコンストラクタが定義
  - 番目の引数はデフォルトの値が指定
- キーワード this はクラスから作成されたインスタンスを指す。
- 3 行目から 9 行目で、そのインスタンスのオブジェクトのキーはプロパティと呼ばれる。プロパティの名前は前のオブジェクトと同じである。
- 9 行目の定義はコンストラクタの最後の引数はデフォルトの値が与えられている。

クラスの例–実行例

- クラスの定義からインスタンスを作成するためにはキーワード new をつけてクラスを呼び出す。

```
>p = new Person("foo",2001,8,18);
Person {name: "foo", birthday: {…}, hometown: "神奈川"}
>p.birthday
{year: 2001, month: 8, day: 18}
>p.hometown;
"神奈川"
```

- コンストラクタの最後の引数にはデフォルト値が指定されているので、この実行例のように値が指定されていない場合にはデフォルト値に設定されている

クラスメソッド

- クラス内では constructor のほかにメソッドと呼ばれる関数で定義されたプロパティが定義可能
- メソッドにはアクセッサプロパティと呼ばれるオブジェクトに値を渡すセッター、値を得るゲッターの 2 種類を指定することも可能
- ES2015 ではゲッターやセッターにはキーワード get と set を用いる。

メソッド定義の例

Person に文字列に変換する toString() と、実行時における年令を返すゲッター age を追加したもの

```
1  class Person{
2    constructor(name, year, month, day, hometown = "神奈川"){
3  .... //以前と同じなので省略
4    }
5    toString() {
6      return `${this.name}さんは`+
7        `${this.birthday.year}年${this.birthday.month}月${this.birthday.day}日に`
8       `${this.hometown}で生まれました。`;
9    }
10   get age() {
11     let today = new Date();
12     let age = today.getFullYear() - this.birthday.year;
13     if(today.getTime() <
14          new Date(today.getFullYear(),
15                   this.birthday.month-1,
16                   this.birthday.day).getTime()) age--;
17     return age;
18   }
19 }
```

メソッド定義の例–解説 (1)

- 11 行目から 15 行目でメソッド toString() が定義
  - toString() メソッドはすべてのオブジェクトに定義されていて、文字列が必要な時に呼び出される。
  - ここでは「~ さんは~ 年~ 月~ 日に~ で生まれました。」という文字列を返す。

```
p.toString();//toString() の明示的呼び出し
"foo さんは 2001 年 4 月 1 日に}神奈川で生まれました。"
‘${p}‘;          //暗黙の toString() 呼び出し
"foo さんは 2001 年 4 月 1 日に}神奈川で生まれました。"
```

メソッド定義の例–解説 (2)

16 行目から 24 行目でゲッター age が定義

- キーワード get をつけて、ゲッター age() を宣言
- 関数なので () をつける必要がある。
- ゲッターに仮引数をつけることはできない。
- 17 行目でアクセス時の時間を変数 today に保存
- 18 行目でアクセス時の年から誕生日の年を引く。
- 19 行目から 22 行目で、今年の誕生日が過ぎているかどうかをチェック
- アクセス時の年と誕生日の月を日をもとに日付を作成し、その時間 (getTime() を利用) を比較
- 誕生日前ならば 18 行目で求めた値を 1 減少

```
>p.age;
16
>p.age = 50;
50
>p.age;
16
>p.age(10);
VM87:1 Uncaught TypeError: p.age is not a function
    at <anonymous>:1:3
```

- 定義の方ははにはメソッドであることを示す () があるが、利用するときはプロパティと同じようになる。() を付けるとエラーになる。
- 代入の式はエラーがなく実行できるが、ゲッターの機能は変わらない。代入はセッターの呼び出しが行われる。

クラスの継承

- 継承とはあるクラスをもとに機能の追加や変更を加えた新しいクラスを作ること
- 新しいクラスはもとになるクラスを継承しているという。
- 新しいクラスはもとになるクラスのサブクラス
- もとのクラスは新しいクラスのスーパークラス
- JavaScript では複数のクラスから同時に継承する多重継承はサポートされていない

継承の例

次の例はクラス Person を継承して新しいクラス Student を作成するものである。

```
1  class Student extends Person {
2    constructor(name, id, year, month, day, hometown) {
3      super(name, year, month, day, hometown);
4      this.id = id;
5    }
6  }
```

- クラスを継承するためにはクラス名の後に、キーワード extends を付けて継承するクラス名を書く (1 行目)。
- Student クラスのコンストラクタには Person クラスで必要なパラメタのほかに id が付け加えている (2 行目)。
- 親クラス (スーパークラス) のコンストラクタを呼び出すために super() を実行する (3 行目)
- 4 行目で追加のプロパティの設定を行っている。

継承の例–実行例

```
>s = new Student("foo",1523999,2001,4,1);//Person のデフォルト引数値が設定
Student {name: "foo", birthday: {…}, hometown: "神奈川", id: 1523999}
> `${s}` //toString() も Person で定義されたものを使用される
"foo さんは 2001 年 4 月 1 日に神奈川で生まれました。"
>s2 = new Student("foo",1523999,2001,4,1,"厚木");//デフォルト以外の設定もできる
Student {name: "foo", birthday: {…}, hometown: "厚木", id: 1523999}
>`${s2}`;
"foo さんは 2001 年 4 月 1 日に厚木で生まれました。"
```

instanceof 演算子

instanceof 演算子は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する。

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

Person オブジェクトが Object を継承しているので p instanceof Object が true となっている。

- クラスに対して、インスタンスを作成しないで使用できる静的メソッドを定義できる。
- 静的メソッドはキーワード static を付ける。
- インスタンス化されたオブジェクトからは使用不可

Math オブジェクトにある各関数が Math オブジェクトの静的メソッドの例

クラスメソッドを用いて連続した id を作成

クラス Student のプロパティ id を重複がなくかつ連続な値に自動で設定しようとするとクラスにおいて変数を管理する変数 (クラス変数) が必要

```
1  class Student extends Person {
2    static getNextId(){
3      return Student.nextId++;
4    }
5    constructor(name, year, month, day, hometown) {
6      super(name, year, month, day, hometown);
7      this.id = Student.getNextId();
8    }
9  }
10 Student.nextId = 10000;
```

クラスメソッドを用いて連続した id を作成– 解説

- 2 行目から 4 行目で次の id を求めるクラスメソッドを定義している。
- ここではクラス変数 nextId を 1 増加させている (3 行目)
- nextId の初期化は 10 行目で行っている。
- 初期化の方法からも推察されるように、この値は途中で変更が可能

クラスメソッドを用いて連続した id を作成– 実行例

```
>s1 = new Student("foo1",2001,4,1);
Student {name: "foo", birthday: {…}, hometown: "神奈川", id: 10000}
>s2 = new Student("foo2",2001,5,1);
Student {name: "foo", birthday: {…}, hometown: "神奈川", id: 10001}
Student.nextId = -100;
-100
>s3 = new Student("foo2",2001,6,1);
Student {name: "foo", birthday: {…}, hometown: "神奈川", id: -100}
```

- 3 つ目のインスタンスを作成する前にこの値を上書きしている
- 最後のインスタンスは前の 2 つと異なるものに設定
- この欠点を解消するにはクラスメソッドで定義する代わりにクラスをクロージャの中で定義して、その中に変数を置く。

クラス式のクロージャ

次の例は即時実行関数内に id を管理する変数を用意して、その関数がクラス式を返すようにした。

```
1  const Student = (function(){
2    let id = 10000;
3    return class extends Person {
4      constructor(name, year, month, day, hometown) {
5        super(name, year, month, day, hometown);
6        this.id = id++;
7      }
8    }
9  })();
```

クラス式のクロージャ–解説

- 即時実行関数の戻り値はクラス式であり、それを変数 Student に代入している。同じ名前のクラスが再定義されないようにするため変数を const で宣言している。
- 2 行目で次に設定する id を保存する変数を初期化している。
- 関数の戻り値としてクラス式を返す。クラスの定義は以前のものとほとんど同じである。
- 違いはインスタンスに設定した後で id を管理する変数を増加させているところ (6 行目の右辺)。

クラス式のクロージャ–実行例

```
>s = new Student("foo",2001,4,1);  //初めのインスタンスの id は 10000
{name: "foo", birthday: {…}, hometown: "神奈川", id: 10000}
>`${s}`;
"foo さんは 2001 年 4 月 1 日に}神奈川で生まれました。"
>s2 = new Student("foo",2001,4,1,"厚木");  //次のインスタンスの id は 10001
{name: "foo", birthday: {…}, hometown: "厚木", id: 10001}
>`${s2}`;
"foo さんは 2001 年 4 月 1 日に}厚木で生まれました。"
```