Core JavaScript #9

Chap.03 this (2)

Kim Donghee

目次

状況によって変わるthis

前回の復習

```
// グローバル領域の`this`
console.log(this); // window

// メソッドを呼び出す時の`this`
var obj = { x: function () {
  console.log(this);
}};
obj.x(); // obj

// 関数を呼び出す時の`this`
function func () {
  console.log(this);
}
func(); // window
```

コールバック関数呼び出す時のthis

- コールバック関数とは
  - 関数またはメソッドに引き渡される関数
  - 基本的にグローバルオブジェクトを参照するが引き渡された関数で `this` を指定した場合はそれを参照する。

```
setTimeout(function () {
  console.log(this); // (1) windowが出力される
  }, 300);

[1,2,3,4,5].forEach(function (x) {
  console.log(this, x); // {2} windowが５回出力される
});

document.body.innerHTML += '<button id="hoge">Click!</button>';
document.body.querySelector('#hoge').addEventListener('click', function (e) {
  console.log(this, e); // {3} buttonエレメントとMouseEventが出力される
});
```

`addEventListener`は呼び出し元をthisとして指定すように定義されていて、グローバルオブジェクトのwindowでなくbuttonエレメントが出力される。

コンストラクタ内部のthis

- コンストラクタとは
  - オブジェクトを生成する関数
  - クラスを用いて作られたオブジェクトをインスタンスと呼ぶ。（詳しくは `Chap07.クラス` で！）
  - `new` キーワードでコンストラクタを実行し、インスタンスを生成する。
  - インスタンス内部の `this` はインスタンス自分自身になる。

```
var User = function (name, age) {
  this.shopName = this.name + '\'s shop';
  this.name = name;
  this.age = age;
}
var naitoh = new User('Naitoh Tetsuya', 38);
var ibushi = new User('Ibushi Kota', 38);

console.log(naitoh, ibushi);
/ *
  User { shopName: 'Naitoh Tetsuya's home', name: Naitoh Tetsuya, age: 38 }
  User { shopName: 'Ibushi Kota's home', name: Ibushi Kota, age: 38 }
* /
```

thisをbindする方法

`self = this` などで回避するのではなく、 `call`, `apply`, `bind` を使って簡潔に `this` を引き渡すことができる。

- bindとは

> 束縛（する）、拘束（する）、結びつける、関連付ける、などの意味を持つ英単語。ITの分野では、何らかの要素やデータ、ファイルなどが相互に関連付けられている状態や、そのような状態を実現する機能などのことを指すことが多い。

`this`は基本状況によって変わるが、直接バインドすることもできる。

callメソッド

- 関数を即時実行させる。第１引数を `this` にバインドする。

```
var func = function (a, b) {
  console.log(this, a, b);
};

func(1, 2); // window 1 2
func.call({x: 3}, 1, 2); // {x: 3} 1 2
```

applyメソッド

callメソッドと機能的に同一。第２引数として配列を受け取る。

```
var func = function (a, b) {
  console.log(this, a, b);
};

func.apply({x: 3}, [1, 2]); // {x: 3} 1 2
```

bindメソッド

ES5から追加されたメソッド。

引き渡されたthisと引数で新しい関数を返す。

```
var func = function (a, b, c, d) {
  console.log(this, a, b, c, d);
};

func(1, 2, 3, 4); // window 1 2 3 4

var bindFunc1 = func.bind({x: 1});
bindFunc1(5, 6, 7, 8); // {x: 1} 5 6 7 8

var bindFunc2 = func.bind({x: 1}, 4, 5);
bindFunc2(6, 7); // {x: 1} 4 5 6 7
bindFunc2(8, 9); // {x: 1} 4 5 8 9
```

name プロパーティ

bindメソッドで作った関数には `bound` というbindの受動態がプレフィックスとして付く。
callとapplyよりデバッグしやすい。

```
var func = function (a, b, c, d) {
  console.log(this, a, b, c, d);
};

var bindFunc = func.bind({ x: 1 }, 4, 5);
console.log(func.name);     // func
console.log(bindFunc.name); // bound func
```

アロー関数の例外事項

アロー関数は実行コンテキスト生成時`this`をバインドしないので、アロー関数の内部には`this`が存在しない。
`this`を参照するとスコープチェーン上一番近い`this`を参照することになる。

```
var obj = {
  outer: function() {
    var innerFunc = () => {
      console.log(this);
    };
    innerFunc();
  }
};
obj.outer(); // obj
```

`call`, `apply`, `bind`も使わずにthisを参照することができてさらに簡潔になる。

まとめ

- 状況別の `this`
  - グローバル領域
    - グローバルオブジェクトの `window`（Node.jsは `global`）
  - 関数実行時
    - `window`
  - メソッド実行時
    - メソッドの呼び出し元
  - コールバック関数の内部
    - コールバックによって異なるが、定義されてない場合は `window`
  - コンストラクタ
    - 生成されるインスタンス

- `this` バインド
  - `call`、`apply` は `this` を指定し即時実行する
  - `bind` は `this` を指定し新しい関数を生成する
  - アロー関数は `this` を持たない

Next

- `Chap.04 コールバック関数` に突入！
  - コールバック関数とは
  - 制御権
  - コールバック関数は関数だ
  - thisを別途バインドする方法
  - コールバック地獄と非同期制御
  - まとめ

参考文献

- バインド 【 bind 】 バインディング
  - http://e-words.jp/w/バインド.html
- 関数とthis
  - https://jsprimer.net/basic/function-this/