Core JavaScript #8

Chap.03 this (1)

Kim Donghee

目次

今日のゴール

メソッドの `this` と関数の `this` を理解しましょうー！

概要

- 大体のプログラミング言語での `this` は、クラスから生成されたインスタンスオブジェクトを指す。
- JavaScriptにおける `this` は、関数とオブジェクトを区分する唯一の機能で、状況によって中身が変わる。

状況によって変わるthis

- this実行コンテキストが生成される時合わせて結合される。
- 言い換えると、**関数を呼び出す時に決定される。**

グローバル領域のthis

グローバル領域での `this` はグローバルオブジェクトを指す。
ブラウザ環境では `window` 、Node.jsでは `global` を指すことになる。

グルーバル変数とグローバルオブジェクト

```
var hello = 'こんにちは';
console.log(hello);          //こんにちは
console.log(window.hello); //こんにちは
console.log(this.hello);   //こんにちは
```

グローバル変数を宣言すると、JavaScriptエンジンがグローバルオブジェクトのプロパーティとして割り当てる。
実は、すべての変数は特定のオブジェクトのプロパーティとして動作する。
その特定のオブジェクトが`LexicalEnvironment`だ。
ある弁数を呼び出すと`LexicalEnvironment`の中から検索し、その値を返却する。
なのでグローバルオブジェクトをプローパティとして直接割り当てても`var`で宣言したのと同じ結果になる。

メソッドを呼び出す時そのメソッド内部のthis

関数 vs. メソッド

- 両方予め定義した動作を行うコードの束
- **独立性**で区分することができる

```
var func = function (x) {
  console.log(this, x);
};
func(1); // window { ... } 1

var obj = {
  method: func
}
obj.method(2); // { method: f } 2
```

両方`func`を実行しているが、それぞれ出力される`this`の結果が異なってくる。
シンプルに言うと、`.`を付けて実行すると、メソッドとして関数が呼び出される。

メソッド内部でのthis

`this` に呼び出し元の情報が割り当てられる。

`.` が付いているオブジェクトを指す。

```
var obj = {
  methodA: function () { console.log(this); },
  inner: {
    methodB: function () { console.log(this); }
  }
};
obj.methodA(); // { methodA: f, inner: {...} } (=== obj)
obj.inner.methodB(); // { methodB: f } (=== obj.inner)
```

関数を呼び出す時その関数内部のthis

関数内部のthis

関数を呼び出す場合は`this`が指定されない。
実行コンテキストを構成する時`this`が指定されていない場合`this`はグローバルオブジェクトを指す。（Chap2で言及された内容）
この振る舞いをJavaSCript設計ミスだと指摘する有識者も居る。

メソッドの内部関数でのthis

```
var obj1 = {
  outer: function () {
    console.log(this); // (1)
    var innerFunc = function () {
      console.log(this); // (2) (3)
    };
    innerFunc();

    var obj2 = {
      innerMethod: innerFunc
    };
    obj2.innerMethod();
  }
};
obj1.outer();
```

`(1)`で`obj1`が、`(2)`では`window`、そして`(3)`では`obj2`が出力される。
`(1)`と`(3)`はそれぞれ呼び出し元のオブジェクトが出力されたが、`(2)`ではグローバルオブジェクトを出力している。

```
var obj1 = {
  outer: function () {
    console.log(this); // (1)
    var innerFunc = function () {
      console.log(this); // (2) (3)
    };
    innerFunc();

    var obj2 = {
      innerMethod: innerFunc
    };
    obj2.innerMethod();
  }
};
obj1.outer();
```

`innerFunc()` で関数として呼び出しているので、 `this` が指定されておらずスコープチェーンの最上位オブジェクトである `window` がバインドされるからだ。

上位スコープの `this` を指定する方法

- 上位のスコープの `this` を変数として定義する。

```
var obj = {
  outer: function () {
    console.log(this); // (1) { outer: f }
    var innerFunc = function () {
      console.log(this); // (2) window
    };
    innerFunc1();

    var self = this;
    var innerFunc2 = function () {
      console.log(self); // (3) { outer: f }
    }
    innerFunc2();
  }
};
obj.outer();
```

- **アロー関数**を使用する

```
var obj = {
  outer: function () {
    console.log(this); // (1) { outer: f }
    var innerFunc = () => {
      console.log(this); // (2) { outer: f }
    };
    innerFunc();
  }
};
obj.outer();
```

ES2015(ES6)で導入されたアロー関数は、実行コンテキスト生成時 `this` をバインドするステップ自体をスキップするので上位スコープの `this` をそのまま流用することができる。

今日のゴール確認

> メソッドの`this`と関数の`this`を理解しましょうー！

- メソッドの`this`
  - 呼び出し元（オブジェクト）を指す
- 関数の`this`
  - グロバールオブエクトを指す
  - ブラウザの場合、`window`


ボーーーン!!!
ストレート
141 km/h
©Konami Digital Entertainment

予告

- 状況によって変わるthis
  - グローバル領域のthis
  - メソッドを呼び出す時そのthis
  - 関数を呼び出す時のthis
  - **コールバック関数呼び出す時のthis**
- **thisをbindする方法**
- **まとめ**

参考文献

- 書籍
  - Core JavaScript
    - https://wikibook.co.kr/corejs/
- ドキュメント
  - 関数とthis（JavaScript Premier）
    - https://jsprimer.net/basic/function-this/#execution-context-this
  - JavaScriptの関数とメソッド（CodeZine）
    - https://codezine.jp/article/detail/221
  - this（MDN）
    - https://developer.mozilla.org/ja/docs/Web/JavaScript/Reference/Operators/this