

Old Lisper (/)

프로그래밍 언어 그 자체에 관심 있습니다. 항상 더 나은 생산성을 위해 노력합니다.

Javascript this, call, apply 그리고 bind

2014. 6. 18. 13:31 | 언어/Node.js (/category/%EC%96%B8%EC%96%B4/Node.js)

+ - FONT ▾



이미지를 불러올 수 없습니다

Javascript this, call, apply, bind

Javascript 에 오신것을 환영합니다. 낯선이여! 이 포스트는 글에 기반하여 작성되었습니다.

1. <http://dailyjs.com/2012/06/18/js101-this/> (<http://dailyjs.com/2012/06/18/js101-this/>)
2. <http://dailyjs.com/2012/06/25/this-binding/> (<http://dailyjs.com/2012/06/25/this-binding/>)
3. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/bind (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/bind)

Method, Function

먼저 메소드와 함수의 차이에 대해서 간단히 알아보자. ES5 Spec (<http://es5.github.io/#x4.3.27>) 에 의하면 메소드는 Object 에 붙어있는 Function 이다. new 로 Object 를 생성하고 이 오브젝트의 Method 내에서 this 는 해당 오브젝트의 인스턴스를 가리킨다. (자바스크립트에서는 클래스가 따로 없으므로 오브젝트라 부르는 것 같다.). 반면 Method 가 아니라 Function 내부에서 this 는 window, Node.js 라면 global 이다. 그리고 strict mode 라면 Function 내부에서 this 는 undefined 다.

요약하자면 아무데에도 붙어있지 않고 홀로 정의되어 있는 것은 함수요, 오브젝트의 프로퍼티로 정의된 함수는 메소드이며, 함수 내에서 this 는 global, window 또는 undefined 가 될 수 있다. 반면 오브젝트의 메소드 내에서 this 는 해당 오브젝트의 인스턴스를 가리킨다.

Self

아래는 흔히 하는 실수와, 변수 self 를 통해서 해결하는 방법이다.

```
function Shape() {
  this.x = 0;
  this.y = 0;
}

Shape.prototype = {
  move: function(x, y) {
    this.x += x;
    this.y += y;

    function checkBounds() {
      if (this.x > 100) {
        console.error('Warning: Shape out of bounds');
      }
    }

    checkBounds();
  }
};

var shape = new Shape();
shape.move(101, 1);
```

console.error 가 호출되지 않는 이유는, checkBounds Function 은 shape.move property 내에서 정의되어 있지만, 그 자체가 Object 의 메소드는 아니기 때문이다.

```
Shape.prototype = {
  move: function(x, y) {
    var self = this;

    this.x += x;
    this.y += y;

    function checkBounds() {
      if (self.x > 100) {
        console.error('Warning: Shape out of bounds');
      }
    }

    checkBounds();
  }
};
```

아무리 Object 에 붙어있는 Method 라도, 그 자체로 따로 변수로 저장하면 단순한 Function 이다. 무슨말이고 하니

```
this.x = 9;
var module = {
  x: 81,
  getX: function() { return this.x; }
};

module.getX(); // 81

var getX = module.getX;
getX(); // 9, because in this case, "this" refers to the global object
```

getX 는 단순한 Function 이다. new Contructor 로 Instance 를 만들고, Instance.method() 와 같이 호출하거나, Object.method() 처럼 호출되는 것이 아니면 해당 함수는 전역 컨텍스트에서 실행되므로 this 도 global 이거나 window 일거다.

call, apply

Function.prototype.call 을 이용해서도 문제를 해결할 수 있다. call 은 파라미터 중 첫 번째 인자를, 함수 내부에서 사용할 this 로 만들어 준다.

```
Shape.prototype = {
  move: function(x, y) {
    this.x += x;
    this.y += y;

    function checkBounds() {
      if (this.x > 100) {
        console.error('Warning: Shape out of bounds');
      }
    }

    checkBounds.call(this);
  }
};
```

apply 도 마찬가지다. 그러나 call 과는 달리 apply 는 파라미터를 **배열(Array)** 형태로 받는다.

```
Shape.prototype = {
  move: function(x, y) {
    this.x += x;
    this.y += y;

    function checkBounds(min, max) {
      if (this.x < min || this.x > max) {
        console.error('Warning: Shape out of bounds');
      }
    }

    checkBounds.call(this, 0, 100);
    checkBounds.apply(this, [0, 100]);
  }
};
```

아래의 jQuery API 예제에서, 일반적이라면 우리가 넘겨준 콜백은 단순한 Function 이므로 this 는 window 또는 global 이어야 하지만, jQuery 가 내부적으로 콜백 내에서 사용하는 this 가 DOM 객체가 될 수 있도록 call, apply 를 적용해 주었을 것이다. ~~(소스를 뒤져봤는데 내공 부족으로 원하는 내용을 찾을 수 없었음)~~

bind

Function.prototype.bind 는 원하는 Function 에 인자로 넘긴 this 가 바인딩 된 새로운 함수를 리턴한다.

```
Shape.prototype = {
  move: function(x, y) {
    this.x += x;
    this.y += y;

    function checkBounds(min, max) {
      if (this.x < min || this.x > max) {
        console.error('Warning: Shape out of bounds');
      }
    }

    var checkBoundsThis = checkBounds.bind(this);
    checkBoundsThis(0, 100);
  }
};
```

```
Shape.prototype = {
  move: function(x, y) {
    this.x += x;
    this.y += y;

    var checkBounds = function(min, max) {
      if (this.x < min || this.x > max) {
        console.error('Warning: Shape out of bounds');
      }
    }.bind(this);

    checkBounds(0, 100);
  }
};
```

bind : default parameters

bind() 를 이용하면, default parameters 를 가진 함수를 만들 수 있다. 그 전에 잠깐 이후에 사용할 Array.prototype.slice 에 대해 간단히 살펴보자.

```
var arr = [1, 2, 3];
arr.slice() // [1, 2, 3]
arr.slice(1) // [2, 3]
arr.slice(2) // [3]
```

기본적으로 slice 는 인자가 아무것도 없으면, this 를 배열로 만들어 돌려준다. 그래서 call 을 이용하면 이런 활용이 가능하다.

```
function list() {
  return Array.prototype.slice.call(arguments);
}

list(1, 2, 3) // [1, 2, 3]
```

그리고 bind 를 이용하면, 이렇게 default parameter 를 가진 함수를 만들 수 있다.

```
function list() {
  return Array.prototype.slice.call(arguments);
}

var list1 = list(1, 2, 3); // [1, 2, 3]

// Create a function with a preset leading argument
var leadingThirtysevenList = list.bind(undefined, 37);

var list2 = leadingThirtysevenList(); // [37]
var list3 = leadingThirtysevenList(1, 2, 3); // [37, 1, 2, 3]
```

bind : setTimeout

setTimeout 의 callback 의 this 는 전역 컨텍스트에서 사용되므로 window 또는 global (Node.js) 이다. 따라서 setTimeout 의 callback 으로 오브젝트의 메소드를 주고, 그 메소드 내에서 오브젝트의 인스턴스를 this 로 참조하려면, bind 가 꼭 필요하다.

```
function LateBloomer() {
  this.petalCount = Math.ceil( Math.random() * 12 ) + 1;
}

// declare bloom after a delay of 1 second
LateBloomer.prototype.bloom = function() {
  window.setTimeout( this.declare.bind( this ), 1000 );
};

LateBloomer.prototype.declare = function() {
  console.log('I am a beautiful flower with
    ' + this.petalCount + ' petals!');
};
```

만약 bind() 가 없다면, declare() 는 this.petalCount 를 얻지 못해 undefined 가 출력된다.

bind : Advanced

구독하기

- 분류 전체보기 (135) (/category)
 - 삶 (36) (/category/%EC%82%B6)
 - 사는 이야기 (15) (/category/%EC%82%B6/%EC%82%AC%EB%8A%94%20%EC%9D%B4%EC%95%BC%EA%B8%B0)
 - 티스토리 꾸미기 (2) (/category/%EC%82%B6/%ED%8B%B0%EC%8A%A4%ED%86%A0%EB%A6%AC%20%EA%BE%B8%EB%AF%B8%EA%B8%B0)
 - 솔깃한 정보 (12) (/category/%EC%82%B6/%EC%86%94%EA%B9%83%ED%95%9C%20%EC%A0%95%EB%B3%B4)
 - 글을 읽고 (5) (/category/%EC%82%B6/%EA%B8%80%EC%9D%84%20%EC%9D%BD%EA%B3%A0)
 - 기발한 생각들 (2) (/category/%EC%82%B6/%EA%B8%B0%EB%B0%9C%ED%95%9C%20%EC%83%9D%EA%B0%81%EB%93%A4)
 - 구성요소 (30) (/category/%EA%B5%AC%EC%84%B1%EC%9A%94%EC%86%8C)
 - 떠오르는 아이디어 (2)
(/category/%EA%B5%AC%EC%84%B1%EC%9A%94%EC%86%8C/%EB%96%A0%EC%98%A4%EB%A5%B4%EB%8A%94%20%EC%95%84%EC%9D%B4%EB%
 - 코드를 위한 논리 (8)
(/category/%EA%B5%AC%EC%84%B1%EC%9A%94%EC%86%8C/%EC%BD%94%EB%93%9C%EB%A5%BC%20%EC%9C%84%ED%95%9C%20%EB%85%BC%

- 6/6