

JavaScript

this – 김근형 강사

This

- this

- this는 보통은 global을 가리키지만 this가 어디에 위치해 있는가에 따라 바인딩되는 곳이 달라진다.

전역공간에서	window / global
함수 내부에서	window / global
메소드 호출시	메소드 호출 주체 (메소드명 앞)
callback에서	기본적으로는 함수내부에서와 동일
생성자함수에서	인스턴스

This

○ 전역 공간에서의 This

```
/* 브라우저 콘솔에서 */  
console.log(this);
```

```
▼ Window ⓘ  
  ▶ $: function (e,t)  
  ▶ $Add: Object  
  ▶ $App: Object  
  ▼ $Edit: Object  
    ▶ ini: function ()  
    ▶ loadEdit: function ()  
    ▶ onedit: function ()  
    ▶ __proto__: Object  
  ▶ $Historys: Object  
  ▶ $Notes: Object  
  ▶ $Setting: Object  
  ▶ $Slide: function (t,e,n,i)  
  ▶ $Todos: Object
```

```
/* node.js에서 */  
console.log(this);
```

```
{ DTRACE_NET_SERVER_CONNECTION: [Function],  
  DTRACE_NET_STREAM_END: [Function],  
  DTRACE_HTTP_SERVER_REQUEST: [Function],  
  DTRACE_HTTP_SERVER_RESPONSE: [Function],  
  DTRACE_HTTP_CLIENT_REQUEST: [Function],  
  DTRACE_HTTP_CLIENT_RESPONSE: [Function],  
  global: [Circular],  
  process:  
    process {  
      title: 'node',  
      version: 'v7.9.0',  
      moduleLoadList:  
        [ 'Binding contextify',  
          'Binding natives',
```

This

○ 함수 내부에서 This

```
function a(){
  console.log(this);
}
a();
```

```
function b(){
  function c(){
    console.log(this);
  }
  c();
}
b();
```

```
var d = {
  e : function(){
    function f(){
      console.log(this);
    }
    f();
  }
}
d.e();
```

▼ Window

- ▶ `$: function (e,t)`
- ▶ `$Add: Object`
- ▶ `$App: Object`
- ▼ `$Edit: Object`
 - ▶ `ini: function ()`
 - ▶ `loadEdit: function ()`
 - ▶ `onedit: function ()`
 - ▶ `__proto__: Object`
- ▶ `$Historys: Object`
- ▶ `$Notes: Object`
- ▶ `$Setting: Object`
- ▶ `$Slide: function (t,e,n,i)`
- ▶ `$Todos: Object`
- ▶ `$addButton: Object`
- ▶ `$addIcon: Object`
- ▶ `$bookmarks: Object`

This

○ 메소드 호출 시 This

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

```
▼ {b: f} ⓘ  
  ▶ b: f ()  
  ▶ __proto__: Object  
  
▼ {c: f} ⓘ  
  ▶ c: f ()  
  ▶ __proto__: Object
```

This

○ 내부 함수에 대한 우회법

```
var a = 10;
var obj = {
  a: 20,
  b: function() {
    console.log(this.a);
  },
  c: function() {
    console.log(this.a);
  }
};
obj.b();
```

20

10



```
var a = 10;
var obj = {
  a: 20,
  b: function() {
    var self = this;
    console.log(this.a);
  },
  c: function() {
    console.log(self.a);
  }
};
obj.b();
```

This

- call, apply, bind
 - 함수의 기본 프로퍼티 일부
 - 함수를 호출하는 방법은 두가지 방법이 존재한다
 - 함수이름 뒤 ();
 - 함수이름.call/apply/bind(객체, 인수/[인수])
 - 부모인 Function.prototype으로 부터 call,apply,bind을 물려받았기 때문에 모든 함수(function)는 call,apply,bind(프로퍼티)를 호출 할 수 있다.
 - call,apply,bind메서드는 외부에서 할당하는 첫번째 인자가 그 함수의 this가 된다. 함수 자신의 실행"환경"을 외부 this로 설정할 수 있는 것이 주요한 특징이다.

This

- call, apply, bind

- 아래 실행 결과는 옆과 같이 동일하며 각각의 함수는 아래와 같이 쓸 수 있다.
- 여기서 thisArg는 this로 지정하고자 하는 객체를 의미한다.

```
function a(x, y, z) {  
  console.log(this, x, y, z);  
}  
var b = {  
  c: 'eee'  
};  
  
a.call(b, 1, 2, 3);  
  
a.apply(b, [1, 2, 3]);  
  
var c = a.bind(b);  
c(1, 2, 3);  
  
var d = a.bind(b, 1, 2);  
d(3);
```

```
▶ Object {c: "eee"} 1 2 3  
▶ Object {c: "eee"} 1 2 3  
▶ Object {c: "eee"} 1 2 3  
▶ Object {c: "eee"} 1 2 3
```

```
function.call(thisArg[, arg1[, arg2[, ...]]])  
function.apply(thisArg, [argsArray])  
function.bind(thisArg[, arg1[, arg2[, ...]]])
```


This

- call, apply, bind
 - call과 apply의 차이점은 매개변수를 일일이 받아서 실행할 것인가 아니면 배열로 받아서 실행할 것인가의 차이가 있다.
 - call과 apply는 즉시 호출을 하는 명령이지만 bind는 새로운 함수를 생성할 뿐 호출을 하진 않는다.

This

- call 에서의 this

- 메서드 안에서 call의 this를 따로 호출하게 될 경우 해당 function의 this가 바뀔 수 있다.

```
var callback = function() {  
  console.dir(this);  
};  
var obj = {  
  a: 1,  
  b: function(cb) {  
    cb();  
  }  
};  
obj.b(callback);
```

▶ Window



```
var callback = function() {  
  console.dir(this);  
};  
var obj = {  
  a: 1,  
  b: function(cb) {  
    cb.call(this);  
  }  
};  
obj.b(callback);
```

▼ Object ⓘ
 a: 1
 ▶ b: function (cb)
 ▶ __proto__: Object

This

- bind 에서의 this
 - bind를 통해 obj를 this 로 선언

```
var callback = function() {  
  console.dir(this);  
};  
var obj = {  
  a: 1  
};  
setTimeout(callback, 100);
```

▶ Window




```
var callback = function() {  
  console.dir(this);  
};  
var obj = {  
  a: 1  
};  
setTimeout(callback.bind(obj), 100);
```

▼ Object ⓘ
 a: 1
 ▶ __proto__: Object

This

○ bind를 통한 이벤트 리스너 예제

```
document.body.innerHTML += '<div id="a">클릭하세요</div>';  
  
document.getElementById('a')  
  .addEventListener('click', function() {  
    console.dir(this);  
  });
```



```
▼ div#a ⓘ  
  accessKey: ""  
  align: ""  
  assignedSlot: null  
  ▶ attributes: NamedNodeMap  
  baseURI: "chrome-extension://c  
  childElementCount: 0  
  ▶ childNodes: NodeList(1)
```

```
document.body.innerHTML += '<div id="a">클릭하세요</div>';  
var obj = { a: 1 };  
  
document.getElementById('a')  
  .addEventListener('click', function() {  
    console.dir(this);  
  }).bind(obj));
```

```
▼ Object ⓘ  
  a: 1  
  ▶ __proto__: Object
```

This

- apply call bind 를 통한 this 정리

정리

- ▶ 기본적으로는 함수의 this와 같다.
- ▶ 제어권을 가진 함수가 callback의 this를 명시한 경우 그에 따른다.
- ▶ 개발자가 this를 바인딩한 채로 callback을 넘기면 그에 따른다.