JavaScript에서의 this

이은규

언어별 this의 의미

대부분의 객체지향 언어

클래스로 생성한 **인스턴스 객체**

JavaScript

어디서든 사용 가능

상황에 따라 **바라보는 대상이 달라짐**

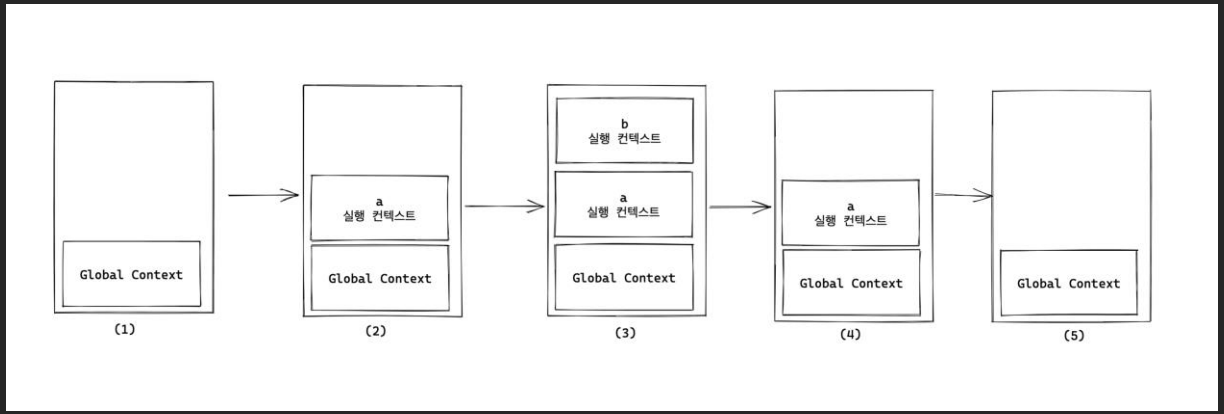
this - JavaScript MDN

this

JavaScript에서 **함수의** `this` **키워드**는 다른 언어와 조금 다르게 동작합니다. 또한 엄격 모드와 비엄격 모드에서도 일부 차이가 있습니다.

대부분의 경우 `this`의 값은 함수를 호출한 방법에 의해 결정됩니다. 실행중에는 할당으로 설정할 수 없고 함수를 호출할 때 마다 다를 수 있습니다. ES5는 **`함수를 어떻게 호출했는지 상관하지 않고 this 값을 설정할 수 있는`** `bind` 메서드를 도입했고, ES2015는 스스로의 `this` 바인딩을 제공하지 않는 화살표 함수를 추가했습니다(이는 렉시컬 컨텍스트안의 `this`값을 유지합니다).

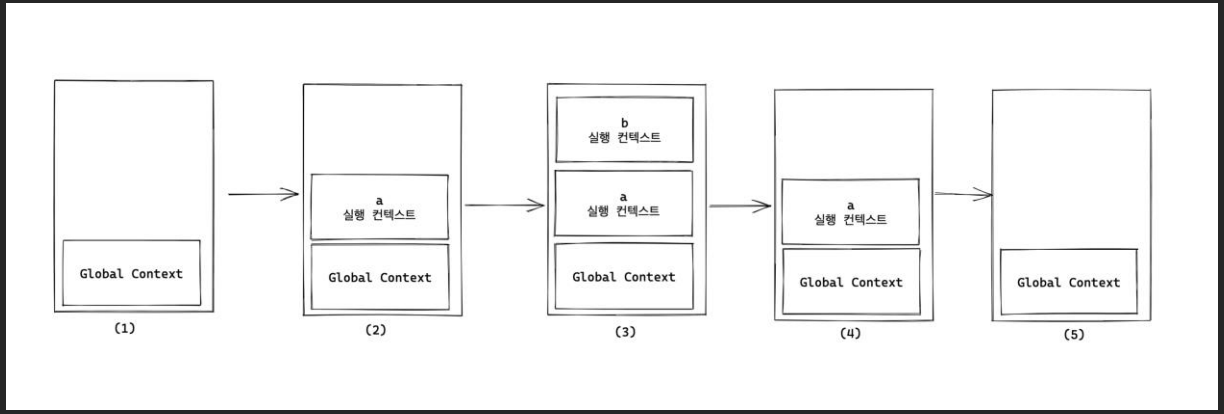
상황에 따라 달라지는 this



JavaScript에서 **this**는 **실행 컨텍스트**(Execution Context)가 **생성될 때** 함께 결정된다.

실행 컨텍스트는 함수를 호출할 때 생성되므로, **this는 함수를 호출할 때 결정된다!**

상황에 따라 달라지는 this



this는 **호출한 놈**

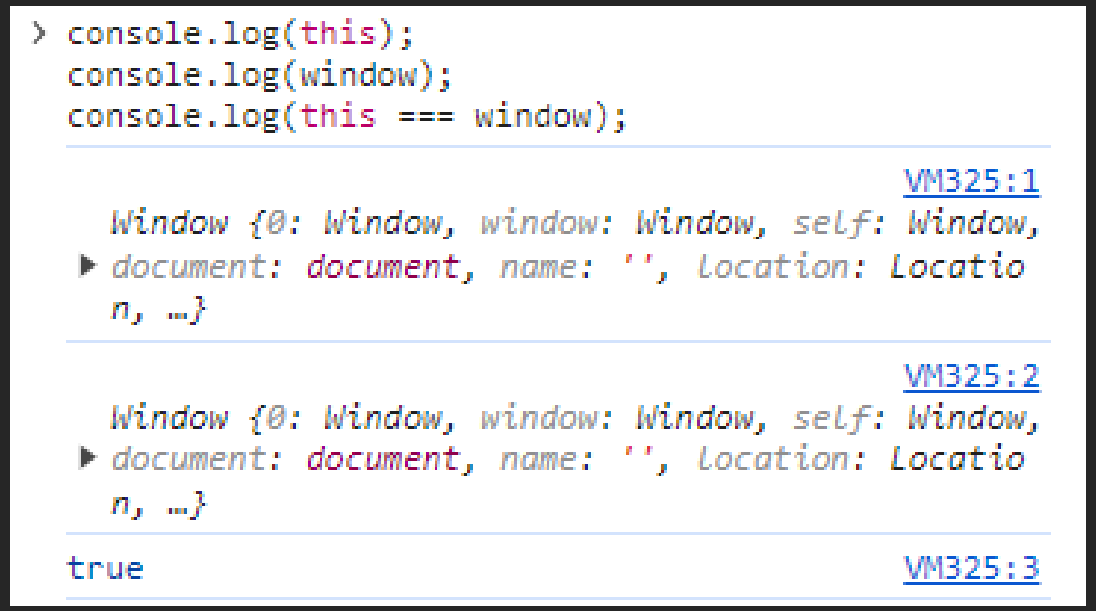
this는 호출한 주체이다.

호출한 주체가 없을 경우에는 기본값으로 **전역 객체**이다.

상황에 따라 달라지는 this

- 전역 공간에서의 this
- 메서드로써 호출할 때, 그 메서드 내부에서의 this
- 함수로써 호출할 때, 그 함수 내부에서의 this

전역 공간에서의 this



전역 공간에서 **this** === **전역 객체**

전역 컨텍스트를 생성하는 주체가 전역 객체이기 때문이다.

전역 공간에서의 this

```
> var a = 1;
  console.log(a);
  console.log(window.a);
  console.log(this.a);

  1

  1

  1
```

```
> var a = 1;
  window.b = 2;
  console.log(a, window.a, this.a);
  console.log(b, window.b, this.b);

  1 1 1

  2 2 2
```

전역 변수를 선언하면 JavaScript 엔진은 이를 **전역객체의 프로퍼티**로 할당한다.

var 변수 선언 = Window 프로퍼티에 직접 할당

전역 공간에서의 this

```
> var a = 1;
  delete window.a;
  console.log(a, window.a, this.a);
  1 1 1                                instrument.ts:113
< undefined
> var b = 2;
  delete b;
  console.log(b, window.b, this.b);
  2 2 2                                instrument.ts:113
```

```
> window.c = 3;
  delete window.c;
  console.log(c, window.c, this.c);
  ▶ Uncaught ReferenceError: c is not defined        VM923:3
      at <anonymous>:3:13
> window.d = 4;
  delete d;
  console.log(d, window.d, this.d);
  ▶ Uncaught ReferenceError: d is not defined        VM965:3
      at <anonymous>:3:13
```

전역객체의 프로퍼티로 할당한 경우 => **삭제 가능**

전역변수로 선언한 경우 => **삭제 불가**

함수 vs 메서드

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

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

함수로서 호출한 경우와 **메서드로서 호출한 경우**에 this가 가리키는 대상이 달라진다.

함수 내부에서의 this

어떤 함수를 함수로써 호출할 경우에는 this가 지정되지 않는다.

그러므로 this는 **전역 객체**를 바라본다.

메서드의 내부함수에서의 this

```
> var obj1 = {
      outer: function () {
          console.log(this); // obj1
          var innerFunc = function () {
              console.log(this);  // Window
          }                       // obj2
          innerFunc();

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

같은 함수임에도 함수로서 호출하냐, 메서드로서 호출하냐에 따라 바인딩되는 this의 대상이 달라진다.

해당 함수를 호출하는 구문 앞에 **점 또는 대괄호** 표기가 있는지 없는지가 관건이다.

메서드의 내부 함수에서 this를 우회하는 방법

```
> var obj = {
      outer: function () {
          console.log(this);      // obj
          var innerFunc1 = function () {
              console.log(this); // Window
          };
          innerFunc1();

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

호출 당시 주변 환경의 this를 그대로 상속받아 사용하는 방법은?

outer 스코프 내부에서 self라는 변수에 this를 저장한 상태로 호출한 innerFunc2의 경우 객체 obj가 출력됨

this를 바인딩하지 않는 함수

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

ES6에서는 함수 내부에서 this가 전역객체를 바라보는 문제를 보완하고자,

this를 바인딩하지 않는 **화살표 함수**를 도입했다.

상위 스코프의 this를 그대로 활용할 수 있다.

참조 :

[서적]

코어 자바스크립트(Core JavaScript), 정재남 (위키북스, 2019), 03_this

[유튜브]

https://youtu.be/GteV4zfqPIk?si=mXhYhHJkJXQr-iZo

자바스크립트 this란 무엇인가? | 웹 개발 입문자들을 위한 this 강좌!