Contents

this Keyword in JavaScript

Source: MDN Web Docs

> Nilai dari `this` pada JavaScript bergantung pada bagaimana sebuah function dipanggil, bukan bagaimana dia didefinisikan.
>
> – *The value of* `this` *in JavaScript depends on how a function is invoked, not how it is defined* –.

Ketika *regular function* dipanggil sebagai method dari object (`obj.method()`), `this` merujuk ke object tersebut. Ketika dipanggil sebagai function yang berdiri sendiri (tidak menempel pada object) `this` umumnya akan merujuk pada *global object* (pada mode *non-strict*) atau bernilai *undefined* (pada mode *strict*). Method `Function.prototype.bind()`[1] dapat membuat function tanpa merubah ikatan dari `this`. Dan method `Function.prototype.apply()`[2] dan `Function.prototype.call()`[3] yang bisa menyetel nilai `this` untuk panggilan tertentu.

Arrow function berbeda dalam bagaimana menangani `this`: mereka mewarisi `this` dari *parent scope*-nya disaat mereka didefiniskan. Perilaku ini membuat *arrow function* secara khusus berguna untuk callback dan mempertahankan context. Namun tetap saja *arrow function* tidak punya kepemilikan `this`. Oleh karena itu nilai this merekan bisa dipasang dengan method `bind()`, `apply()`, atau `call()`, dan juga tidak merujuk pada object saat ini pada object method.

> Nilai `this` selalau mengacu pada sebuah object (pada mode **non-strict**). Pada mode **strict** itu bisa bernilai apa pun.

Function Context

Di dalam sebua function, nilai dari `this` bergantung bagaimana function dipanggil. Anggap `this` sebagai parameter tersembunyi dari function – seperti parameter yangg dideklarasikan pada function definition, `this` adalah pengikat yang bahasa ciptakan ketika *function body* dievaluasi.

Untuk function regular, nilai `this` adalah object dimana function tersebut sedang diakses. Dengan kata lain, jika pemanggilan function dalam bentuk `obj.f()`, maka `this` merujuk pada `obj`. Contoh:

```
function getThis() {
  return this;
}
```

[1] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/call
[2] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/apply
[3] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/call

```
const obj1 = { name: "obj1" };
const obj2 = { name: "obj2" };

obj1.getThis = getThis;
obj2.getThis = getThis;

console.log(obj1.getThis()); // { name: 'obj1', getThis: [Function: getThis] }
console.log(obj2.getThis()); // { name: 'obj2', getThis: [Function: getThis] }
```

Catat bagaimana function yang sama, namun berdasarkan bagaimana mereka dipanggil (*invoke*) nilai dari `this` menjadi berbeda.

Nilai dari `this` bukanlah object yang memiliki function tersebut sebagai propertinya, namun milik object yang digunakan untuk memanggil function tersebut. Kita bisa membuktikan dengan memanggil sebuah method dari object lain dalam prototype chain.

```
const obj3 = {
  __proto__: obj1,
  name: "obj3",
};

console.log(obj3.getThis()); // { name: 'obj3' }
```

Nilai dari `this` selalu berubah berdasarkan bagaimana function dipanggil, bahkan ketika function tersebut didefiniskan pada object saat *creation*.

```
const obj4 = {
  name: "obj4",
  getThis() {
    return this;
  },
};

const obj5 = { name: "obj5" };

obj5.getThis = obj4.getThis;
console.log(obj5.getThis()); // { name: 'obj5', getThis: [Function: getThis] }
```

Jika nilai dari method yang sedang diakses adalah tipe data primitif, `this` akan bernilai promitif pula – namun dengan syarat function tersebut berada pada *strict mode*.

```
function getThisStrict() {
  "use strict"; // Enter strict mode
  return this;
}

// Only for demonstration - you should not mutate built-in prototypes
Number.prototype.getThisStrict = getThisStrict;
console.log(typeof (1).getThisStrict()); // "number"
```

Tapi jika function dipanggil tanpa diakses dari object manapun, `this` akan bernilai `undefined` – dengan function pada *strict mode*.

```
console.log(typeof getThisStrict()); // "undefined"
```

pada *non-strict mode*, sebuah proses spesial yang dinamakan this substitution memastikan bahwa nilai dari `this` selalu bertipe object. Sehingga:

- Jika sebuah function dipanggil dengan `this` yang di-set ke `undefined` atau `null`, `this` mendapatkan substitusi dengan globalThis.
- Jika function dipanggil dengan `this` yang di-set ke nilai primitif, `this` akan disubtitusi dengan object pembungkus nilai primif.

```
function getThis() {
  return this;
}

// Only for demonstration - you should not mutate built-in prototypes
Number.prototype.getThis = getThis;
console.log(typeof (1).getThis()); // "object"
console.log(getThis() === globalThis); // true
```

Pada umumnya pemanggilan function, `this` secara tidak langsung dilewatkan seperti sebuah parameter melalui awalan function (*function's prefix*) – bagian sebelum titik. Anda juga bisa secara jelas men-set nilai dari `this` menggunakan method-method `Function.prototype.call()`, `Function.prototype.apply()`, atau `Reflect.apply()`. Menggunakan `Function.prototype.bind()`, anda bisa menciptakan function baru dengan nilai tertentu dari `this` yang tidak berubah terlepas dari bagaimana function tersebut dipanggil. Ketika menggunakan method-method ini, aturan-aturan subtitusi `this` tetap berlaku jika function dalam mode *non-strict*.

Callbacks

Ketika sebuah function dilewatkan sebagai *callback*, nilai dari `this` bergantung pada bagaimana callback dipanggil, yang mana ditentukan oleh implementor API. Callback umumnya dipanggil dengan sebuah `this` yang bernilai `undefined` (pemanggilan secara langsung tanpa terpasang pada object apapun), yang mana jika function tersebut pada mode *non-strict*, nilai dari `this` adalah global object (`globalThis`).