Difference between this and self in JavaScript

Asked 6 years, 2 months ago Active 1 year ago Viewed 64k times

88

Everyone is aware of `this` in javascript, but there are also instances of `self` encountered in the wild, such as here

So, what is the difference between `this` and `self` in JavaScript?

javascript

26

edited Aug 13 '15 at 12:28

kumar_harsh
13.7k 6 58 87

asked Jun 1 '13 at 18:14
noname
2,669 8 37 73

3 And regarding this... – Denys Séguret Jun 1 '13 at 18:15

8 @dystroy: There is one: `window.self` (`=== window`). Though the OP probably means a trivial variable name… – Bergi Jun 1 '13 at 18:18

2 @dystroy: Actually I didn't think he could really mean it, but indeed in global scope (and a browser environment) `this === self` is true :-) – Bergi Jun 1 '13 at 18:20

2 Subjective aside: aliasing `this` to `self` is not a great practice nowadays when it's common to have code with many (well, more than one is bad enough) levels of callback nesting, as a consequence of asynchronous programming. Use a more descriptive name instead. Objectively speaking the name `this` itself carries no information and is only a nonbad choice of name because the lexical context of a class definition qualifies it. – millimoose Jun 1 '13 at 18:22

2 this is a valid and useful question, it should be reopened – danza Jun 2 '14 at 18:39

3 Answers

98

Unless set elsewhere, the value of `self` is `window` because *JavaScript* lets you access any property `x` of `window` as simply `x`, instead of `window.x`. Therefore, `self` is really `window.self`, which is different to `this`.

```
window.self === window; // true
```

If you're using a function that is executed in the global scope and is not in strict mode, `this` defaults to `window`, and therefore

```
function foo() {
    console.log(
        window.self === window, // is self window?
        window.self === this,   // is self this?
        this === window         // is this window?
    );
}
foo(); // true true true
```

If you're using a function in a different context, `this` will refer to that context, but `self` will still be `window`.

```
// invoke foo with context {}
foo.call({}); // true false false
```

You can find `window.self` defined in the W3C 2006 working draft for the *Window Object* **here**.

edited Jan 27 '18 at 17:43 — Antimony — **28.5k** 7 74 83

answered Jun 1 '13 at 18:56 — Paul S. — **50k** 6 81 106

24 For completeness, `self` is useful in context of WebWorker when window is not accessible (developer.mozilla.org/en-US/docs/Web/Guide/Performance/…). Using `self` instead of `window` lets you access the global object in a portable way. – lqc Jun 1 '13 at 19:05

23

Although I am late here but I came across one example which too can be helpful to understand `this` further:

```
var myObject = {
 foo: "bar",
 func: function() {
    var self = this;
    console.log("outer func:  this.foo = " + this.foo);
    console.log("outer func:  self.foo = " + self.foo);
    (function() {
        console.log("inner func:  this.foo = " + this.foo);
        console.log("inner func:  self.foo = " + self.foo);
    }());
 }
```

```
};
myObject.func();
```

O/P

```
outer func:  this.foo = bar
outer func:  self.foo = bar
inner func:  this.foo = undefined
inner func:  self.foo = bar
```

> Prior to ECMA 5, `this` in the inner function would refer to the global window object; whereas, as of ECMA 5, `this` in the inner function would be undefined.

edited Apr 18 '17 at 12:04

answered Jul 24 '16 at 6:14

Shashank Vivek

6,721 3 27 64

11

A slight addition to this as people may encounter this in the context of service workers, in which case it means something slightly different.

You might see this in a service worker module:

```
self.addEventListener('install', function(e) {
  console.log('[ServiceWorker] Install');
});
```

Here **self** refers to the WorkerGlobalScope, and this is the standard method for setting event listeners.

From Mozilla docs:

> By using self, you can refer to the global scope in a way that will work not only in a window context (self will resolve to window.self) but also in a worker context (self will then resolve to WorkerGlobalScope.self).

answered Jul 8 '18 at 10:02

andyhasit

6,590 4 30 38