Pass correct “this” context to setTimeout callback? [duplicate]

Asked 9 years, 6 months ago Active 1 month ago Viewed 162k times

218

55

This question already has an answer here:
How to access the correct `this` inside a callback? *10 answers*

How do I pass context into `setTimeout` ? I want to call `this.tip.destroy()` if `this.options.destroyOnHide` after 1000 ms. How can I do that?

```
if (this.options.destroyOnHide) {
     setTimeout(function() { this.tip.destroy() }, 1000);
}
```

When I try the above, `this` refers to the window.

javascript callback this settimeout

edited Jun 4 '15 at 6:21
Nayuki
14.5k 5 40 65

asked Jan 25 '10 at 4:44
JamesBrownIsDead
1,785 5 22 29

marked as duplicate by Oriol javascript May 14 '16 at 21:59

This question has been asked before and already has an answer. If those answers do not fully address your question, please ask a new question.

2 Is duplicate flag really valid? This question was actually asked earlier. – Sui Dream Oct 30 '17 at 11:34

1 if (this.options.destroyOnHide) { setTimeout(function() { this.tip.destroy() }.bind(this), 1000); } – Zibri Mar 4 '18 at 10:02

5 Answers

288

EDIT: In summary, back in 2010 when this question was asked the most common way to solve this problem was to save a reference to the context where the `setTimeout` function call is made, because `setTimeout` executes the function with `this` pointing to the global object:

```
var that = this;
if (this.options.destroyOnHide) {
     setTimeout(function(){ that.tip.destroy() }, 1000);
}
```

In the ES5 spec, just released a year before that time, it introduced the `bind` method, this wasn't suggested in the original answer because it wasn't yet widely supported and you needed polyfills to use it but now it's everywhere:

```
if (this.options.destroyOnHide) {
     setTimeout(function(){ this.tip.destroy() }.bind(this), 1000);
}
```

The `bind` function creates a new function with the `this` value pre-filled.

Now in modern JS, this is exactly the problem arrow functions solve in ES6:

```
if (this.options.destroyOnHide) {
     setTimeout(() => { this.tip.destroy() }, 1000);
}
```

Arrow functions do not have a `this` value of its own, when you access it, you are accessing the `this` value of the enclosing lexical scope.

HTML5 also standardized timers back in 2011, and you can pass now arguments to the callback function:

```
if (this.options.destroyOnHide) {
     setTimeout(function(that){ that.tip.destroy() }, 1000, this);
}
```

See also:

- setTimeout - The 'this' problem

edited Jun 11 at 4:33

answered Jan 25 '10 at 5:40

CMS
621k 166 857 819

3 It works. I tested the concept with a jsbin script: jsbin.com/etise/7/edit – John K Jan 25 '10 at 6:00

1 This code involves making an unnecessary variable (which has function-wide scope); if you correctly passed in `this` to the function, you'd have solved this problem for this case, for map(), for forEach(), etc., etc., using less code, fewer CPU cycles, and less memory. ***See: Misha Reyzlin's answer. – HoldOffHunger Oct 20 '17 at 17:06

221

There are ready-made shortcuts (syntactic sugar) to the function wrapper @CMS answered with. (Below assuming that the context you want is `this.tip` .)

ECMAScript 5 (current browsers, Node.js) and Prototype.js

If you target browser compatible with ECMA-262, 5th edition (ECMAScript 5) or Node.js, you could use `Function.prototype.bind` . You can optionally pass any function arguments to create partial functions.

```
fun.bind(thisArg[, arg1[, arg2[, ...]]])
```

Again, in your case, try this:

```
if (this.options.destroyOnHide) {
    setTimeout(this.tip.destroy.bind(this.tip), 1000);
}
```

The same functionality has also been implemented in Prototype (any other libraries?).

`Function.prototype.bind` can be implemented like this if you want custom backwards compatibility (but please observe the notes).

ECMAScript 2015 (some browsers, Node.js 5.0.0+)

For cutting-edge development (2015) you can use *fat arrow functions*, which are part of the ECMAScript 2015 (Harmony/ES6/ES2015) specification (examples).

> An **arrow function expression** (also known as **fat arrow function**) has a shorter syntax compared to function expressions and lexically binds the `this` value [...].

```
(param1, param2, ...rest) => { statements }
```

In your case, try this:

```
if (this.options.destroyOnHide) {
    setTimeout(() => { this.tip.destroy(); }, 1000);
}
```

jQuery

If you are already using jQuery 1.4+, there's a ready-made function for explicitly setting the `this` context of a function.

> jQuery.proxy(): Takes a function and returns a new one that will always have a particular context.

```
$.proxy(function, context[, additionalArguments])
```

In your case, try this:

```
if (this.options.destroyOnHide) {
    setTimeout($.proxy(this.tip.destroy, this.tip), 1000);
}
```

Underscore.js, lodash

It's available in Underscore.js, as well as lodash, as `_.bind(...)` [1,2]

> **bind** Bind a function to an object, meaning that whenever the function is called, the value of `this` will be the object. Optionally, bind arguments to the function to pre-fill them, also known as partial application.

```
_.bind(function, object, [*arguments])
```

In your case, try this:

```
if (this.options.destroyOnHide) {
    setTimeout(_.bind(this.tip.destroy, this.tip), 1000);
}
```

bind | jquery | underscore.js | ecmascript-5 | prototypejs | node.js

edited Dec 20 '15 at 17:51

answered Jan 10 '12 at 8:01

Joel Purra
17.8k 7 50 56

Why not default `func.bind(context...)` ? Do I miss something? – aTei Jul 12 '15 at 13:23

@aTei: it's already listed. – Joel Purra Jul 16 '15 at 14:44

Is it performant to constantly keep creating a new function (which bind does) every time you call this? I have a search timeout that resets after every keypress, and it just seems like I should be caching this 'bound' method somewhere for reuse. – Triynko Dec 2 '15 at 14:38

@Triynko: I wouldn't see binding a function as an expensive operation, but if you call the same bound function multiple times you might as well keep a reference: `var boundFn = fn.bind(this); boundFn(); boundFn();` for example. – Joel Purra Dec 5 '15 at 14:54

30

In browsers other than Internet Explorer, you can pass parameters to the function together after the delay:

```
var timeoutID = window.setTimeout(func, delay, [param1, param2, ...]);
```

So, you can do this:

```
var timeoutID = window.setTimeout(function (self) {
  console.log(self);
}, 500, this);
```

This is better in terms of performance than a scope lookup (caching `this` into a variable outside of the timeout / interval expression), and then creating a closure (by using `$.proxy` or `Function.prototype.bind`).

The code to make it work in IEs from Webreflection:

```
/*@cc_on
(function (modifierFn) {
  // you have to invoke it as `window`'s property so, `window.setTimeout`
  window.setTimeout = modifierFn(window.setTimeout);
  window.setInterval = modifierFn(window.setInterval);
})(function (originalTimerFn) {
    return function (callback, timeout){
      var args = [].slice.call(arguments, 2);
      return originalTimerFn(function () {
        callback.apply(this, args)
      }, timeout);
    }
});
@*/
```

edited Jan 22 '14 at 8:27

answered Feb 15 '12 at 17:35

Misha Reyzlin

11.8k 4 45 61

1 When creating a class by using the prototype chain and your methods are prototype methods... 'bind' is the only thing that will alter what 'this' is within the method. By passing a parameter to the callback, you don't alter what 'this' is in the function, so such a prototype function could not be written using 'this' within it as any other prototype method could. That leads to inconsistency. Bind is the closest thing to what we actually want, and the closure could be cached in 'this' for higher lookup performance and not having to create it more than once. – Triynko Dec 2 '15 at 14:43

NOTE: This won't work in IE

3

```
var ob = {
    p: "ob.p"
}

var p = "window.p";

setTimeout(function(){
    console.log(this.p); // will print "window.p"
},1000);

setTimeout(function(){
    console.log(this.p); // will print "ob.p"
}.bind(ob),1000);
```

edited May 27 '14 at 10:32

answered May 26 '14 at 12:19

gumkins
3,294 5 31 51

2

If you're using `underscore` , you can use `bind` .

E.g.

```
if (this.options.destroyOnHide) {
     setTimeout(_.bind(this.tip.destroy, this), 1000);
}
```

edited Oct 20 '17 at 18:13

HoldOffHunger
4,474 2 26 55

answered Oct 20 '12 at 17:18

Sam
5,249 4 36 51
