JavaScript: How to create a new instance of a class without using the new keyword?

Asked 9 years, 11 months ago Active 2 years, 4 months ago Viewed 52k times

21

7

I think the following code will make the question clear.

```
// My class
var Class = function() { console.log("Constructor"); };
Class.prototype = { method: function() { console.log("Method");} }

// Creating an instance with new
var object1 = new Class();
object1.method();
console.log("New returned", object1);

// How to write a factory which can't use the new keyword?
function factory(clazz) {
    // Assume this function can't see "Class", but only sees its parameter "clazz".
    return clazz.call(); // Calls the constructor, but no new object is created
    return clazz.new();  // Doesn't work because there is new() method
};

var object2 = factory(Class);
object2.method();
console.log("Factory returned", object2);
```

javascript oop constructor instance

edited Sep 28 '16 at 11:51

community wiki
5 revs, 4 users 100%
avernet

9 Why can't you use the "new" keyword? – Daniel Pryden Oct 16 '09 at 23:20

Daniel, in fact you can, and I incorrectly assumed I couldn't in this case. Friday afternoon fatigue, I guess ;). – avernet Oct 16 '09 at 23:44

J-P, so others can clarify the question if necessary. – avernet Oct 16 '09 at 23:45

1 Strangest 'still-open' and community question ever. – Léon Pelletier Jan 8 '13 at 18:30

6 Answers

18

Doesn't this work?

```
function factory(class_) {
    return new class_();
}
```

I don't understand why you can't use `new` .

answered Oct 16 '09 at 23:22 community wiki dave4420

2 the stipulation was no use of "new" – Jimmy Oct 16 '09 at 23:23

1 Darn yes! This works great. I thought (incorrectly!) that I couldn't use the "new" keyword in this case, but of course it doesn't matter if new takes a "class" that was just defined, or if that "class" is passed in parameter to a function and the "new" is called later. Me stupid. – avernet Oct 16 '09 at 23:43

1 Doesn't work for me. All I'm getting on FF22 is "TypeError: class_ is not a constructor" – andig Jun 27 '13 at 16:21

2 okay hot shot, how do you apply an argument list to this newly created class? ;) new class_().apply(this,array) i don't think so – Lpc_dark May 24 '14 at 14:57

@Lpc_dark hits the nail on the head! Maybe you could do `function factory(class_, ...args) {return new class_(...args)}` ? – Stijn de Witt Apr 6 '17 at 22:28

27

```
function Person(name) {
  if (!(this instanceof Person)) return new Person(name);
  this.name = name;
}

var p1 = new Person('Fred');
var p2 = Person('Barney');

p1 instanceof Person  //=> true
p2 instanceof Person  //=> true
```

answered May 31 '12 at 15:23 community wiki Cory Martin

7

If you *really* don't want to use the `new` keyword, and you don't mind only supporting Firefox, you can set the prototype yourself. There's not really any point to this though, since you can just use Dave Hinton's answer.

```
// This is essentially what the new keyword does
function factory(clazz) {
    var obj = {};
    obj.__proto__ = clazz.prototype;
    var result = clazz.call(obj);
    return (typeof result !== 'undefined') ? result : obj;
};
```

answered Oct 16 '09 at 23:57 community wiki Matthew Crumley

3

I guess browser independent solution would be better

```
function empty() {}

function factory(clazz /*, some more arguments for constructor */) {
```

```
    return obj;
}
```

answered Apr 11 '10 at 13:06

community wiki
Dima Vidmich

Dima, why is this more browser independent than the solution proposed by Dave? – avernet Apr 12 '10 at 6:13

cause "obj.__proto__ = something" doesn't work in all browsers and Dave's solution doesn't support arguments for constructor – Dima Vidmich Apr 12 '10 at 12:24

3

Because JavaScript doesn't have classes, let me reword your question: How to create a new object based on an existing object without using the new keyword?

Here is a method that doesn't use "new". It's not strictly a "new instance of" but it's the only way I could think of that doesn't use "new" (and doesn't use any ECMAScript 5 features).

```
//a very basic version that doesn't use 'new'
function factory(clazz) {
    var o = {};
    for (var prop in clazz) {
        o[prop] = clazz[prop];
    }
    return o;
};

//test
var clazz = { prop1: "hello clazz" };
var testObj1 = factory(clazz);
console.log(testObj1.prop1);    //"hello clazz"
```

You could get fancy and set the prototype, but then you get into cross-browser issues and I'm trying to keep this simple. Also you may want to use "hasOwnProperty" to filter which properties you add to the new object.

There are other ways that use "new" but sort of hide it. Here is one that borrows from the Object.create function in JavaScript: The Good Parts by Douglas Crockford:

```
    var F = function() {};
    F.prototype = clazz;
    return new F();

};

//Test
var orig = { prop1: "hello orig" };
var testObj2 = factory(orig);
console.log(testObj2.prop1);  //"hello orig"
```

EcmaScript 5 has the Object.create method which will do this much better but is only supported in newer browsers (e.g., IE9, FF4), but you can use a polyfill (something that fills in the cracks), such as ES5 Shim, to get an implementation for older browsers. (See John Resig's article on new ES5 features including Object.create).

In ES5 you can do it like this:

```
//using Object.create - doesn't use "new"
var baseObj = { prop1: "hello base" };
var testObj3 = Object.create(baseObj);
console.log(testObj3.prop1);
```

I hope that helps

answered Apr 22 '11 at 15:00 community wiki grahamesd

2

Another way:

```
var factory = function(clazz /*, arguments*/) {
    var args = [].slice.call(arguments, 1);
    return new function() {
        clazz.apply(this, args)
    }
}
```