3 ways to define a JavaScript class

September 29th, 2006. Tagged: JavaScript

Introduction

JavaScript is a very flexible object-oriented language when it comes to syntax. In this article you can find three ways of defining and instantiating an object. Even if you have already picked your favorite way of doing it, it helps to know some alternatives in order to read other people's code.

It's important to note that there are no classes in JavaScript. Functions can be used to somewhat simulate classes, but in general JavaScript is a class-less language. Everything is an object. And when it comes to inheritance, objects inherit from objects, not classes from classes as in the "class"-ical languages.

1. Using a function

This is probably one of the most common ways. You define a normal JavaScript function and then create an object by using the `new` keyword. To define properties and methods for an object created using `function()`, you use the `this` keyword, as seen in the following example.

```
function Apple (type) {
    this.type = type;
    this.color = "red";
    this.getInfo = getAppleInfo;
}

// anti-pattern! keep reading...
function getAppleInfo() {
    return this.color + ' ' + this.type + ' apple';
}
```

To instantiate an object using the Apple *constructor function*, set some properties and call methods you can do the following:

```
var apple = new Apple('macintosh');
apple.color = "reddish";
alert(apple.getInfo());
```

1.1. Methods defined internally

In the example above you see that the method getInfo() of the Apple "class" was defined in a separate function getAppleInfo(). While this works fine, it has one drawback – you may end up defining a lot of these functions and they are all in the "global namespece". This means you may have naming conflicts if you (or another library you are using) decide to create another function with the same name. The way to prevent pollution of the global namespace, you can define your methods within the constructor function, like this:

```
function Apple (type) {
    this.type = type;
    this.color = "red";
    this.getInfo = function() {
        return this.color + ' ' + this.type + ' apple';
    };
}
```

Using this syntax changes nothing in the way you instantiate the object and use its properties and methods.

1.2. Methods added to the prototype

A drawback of 1.1. is that the method getInfo() is recreated every time you create a new object. Sometimes that may be what you want, but it's rare. A more inexpensive way is to add getInfo() to the *prototype* of the constructor function.

```
function Apple (type) {
    this.type = type;
    this.color = "red";
}

Apple.prototype.getInfo = function() {
    return this.color + ' ' + this.type + ' apple';
};
```

Again, you can use the new objects exactly the same way as in 1. and 1.1.

2. Using object literals

Literals are shorter way to define objects and arrays in JavaScript. To create an empty object using you can do:
`var o = {};`
instead of the "normal" way:
`var o = new Object();`
For arrays you can do:
`var a = [];`
instead of:
`var a = new Array();`
So you can skip the class-like stuff and create an instance (object) immediately. Here's the same functionality as described in the previous examples, but using object literal syntax this time:

```
var apple = {
    type: "macintosh",
    color: "red",
    getInfo: function () {
        return this.color + ' ' + this.type + ' apple';
    }
}
```

In this case you don't need to (and cannot) create an instance of the class, it already exists. So you simply start using this instance.

```
apple.color = "reddish";
alert(apple.getInfo());
```

Such an object is also sometimes called *singleton*. In "classical" languages such as Java, *singleton* means that you can have only one single instance of this class at any time, you cannot create more objects of the same class. In JavaScript (no classes, remember?) this concept makes no sense anymore since all objects are singletons to begin with.

3. Singleton using a function

Again with the singleton, eh? 🙂

The third way presented in this article is a combination of the other two you already saw. You can use a function to define a singleton object. Here's the syntax:

```
var apple = new function() {
    this.type = "macintosh";
    this.color = "red";
    this.getInfo = function () {
        return this.color + ' ' + this.type + ' apple';
    };
}
```

So you see that this is very similar to 1.1. discussed above, but the way to use the object is exactly like in 2.

```
apple.color = "reddish";
alert(apple.getInfo());
```

`new function(){...}` does two things at the same time: define a function (an anonymous constructor function) and invoke it with `new`. It might look a bit confusing if you're not used to it and it's not too common, but hey, it's an option, when you really want a constructor function that you'll use only once and there's no sense of giving it a name.

Summary

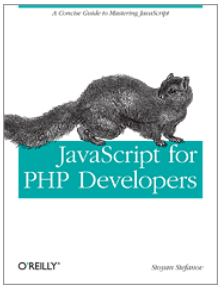
You saw three (plus one) ways of creating objects in JavaScript. Remember that (despite the article's title) there's no such thing as a class in JavaScript. Looking forward to start coding using the new knowledge? Happy JavaScript-ing!

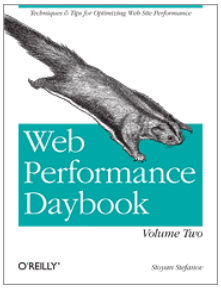
Tell your friends about this post: Facebook, Twitter, Google+

« Uncategorized
Rendering styles »

Sorry, comments disabled and hidden due to excessive spam. Working on restoring the existing comments...

Meanwhile, hit me up on twitter @stoyanstefanov





phpied.com is powered by WordPress, RSS feed

Search