7.1 Singleton

The idea of the singleton pattern is to have only one instance of a specific class. This means that the second time you use the same class to create a new object, you should get the same object that was created the first time.

[复制代码](javascript:void(0);)

var obj = {

myprop: 'my value'

};

var obj2 = {

myprop: 'my value'

};

obj === obj2; // false

obj == obj2; // false

[复制代码](javascript:void(0);)

In JavaScript every time you create an object using the object literal, you’re actually creating a singleton, and there’s no special syntax involved.

7.1.1 Using New

when you use new  to create several objects using the same constructor, you should get only new pointers to the exact same object.

• You can use a global variable to store the instance. This is not recommended because of the general principle that globals are bad. Plus, anyone can overwrite this global variable, even by accident.

• You can cache in a static property of the constructor. Functions in JavaScript are objects,  so  they  can  have  properties.  You  can  have  something  like Universe.instance and cache the object there. This is a nice, clean solution with the only drawback that the instance property is publicly accessible, and code outside of yours might change it, so you lose the instance.

• You can wrap the instance in a closure. This keeps the instance private and not available for modifications outside of your constructor at the expense of an extra closure.

7.1.2 Instance in a Static Property

[复制代码](javascript:void(0);)

function Universe() {

// do we have an existing instance?

if (typeof Universe.instance = = = "object") {

return Universe.instance;

}

// proceed as normal

this.start\_time = 0;

this.bang = "Big";

// cache

Universe.instance = this;

// implicit return:

// return this;

}

// testing

var uni = new Universe();

var uni2 = new Universe();

uni === uni2; // true

[复制代码](javascript:void(0);)

**Drawback**

Instance is public

7.1.3 Instance in a Closure

[复制代码](javascript:void(0);)

// 7.1 Strington - Instance in closure

function Universe() {

// the cached instance

var instance = this;

// proceed as normal

this.start\_time = 0;

this.bang = "Big";

// rewrite the constructor

Universe = function () {

return instance;

};

}

[复制代码](javascript:void(0);)

**Drawback**

The rewritten function (in this case the constructor  Universe()) will lose any properties added to it between the moment of initial definition and the redefinition.

[复制代码](javascript:void(0);)

// adding to the prototype

Universe.prototype.nothing = true;

var uni = new Universe();

// again adding to the prototype after the initial object is created

Universe.prototype.everything = true;

var uni2 = new Universe();

// only the original prototype was linked to the objects

uni.nothing; // true

uni2.nothing; // true

uni.everything; // undefined

uni2.everything; // undefined

// that sounds right:

uni.constructor.name; // "Universe"

// but that's odd:

uni.constructor === Universe; // false

[复制代码](javascript:void(0);)

The reason that uni.constructor is no longer the same as the  Universe() constructor is because  uni.constructor still points to the original constructor, not the redefined one.

[复制代码](javascript:void(0);)

// 7.1 Singleton - Advanced Instance in closure

function Universe() {

// the cached instance

var instance;

// rewrite the constructor

Universe = function Universe() {

return instance;

};

// carry over the prototype properties

Universe.prototype = this; // this is point to the origin function

// the instance

instance = new Universe(); // This is initialized by the origin Universe() constructor.

instance.constructor = Universe; // Rewrite the constructor of the instance object.

// all the functionality

instance.start\_time = 0;

instance.bang = "Big";

return instance;

}

// update prototype and create instance

Universe.prototype.nothing = true; // true

var uni = new Universe();

Universe.prototype.everything = true; // true

var uni2 = new Universe();

// it's the same single instance

uni === uni2; // true

// all prototype properties work

// no matter when they were defined

uni.nothing && uni.everything && uni2.nothing && uni2.everything; // true

// the normal properties work

uni.bang; // "Big"

// the constructor points correctly

uni.constructor === Universe; // true

[复制代码](javascript:void(0);)

Alternative solution

[复制代码](javascript:void(0);)

var Universe;

(function () {

var instance;

Universe = function Universe() {

if (instance) {

return instance;

}

instance = this;

// all the functionality

this.start\_time = 0;

this.bang = "Big";

};

}());

[复制代码](javascript:void(0);)

**References*:***

*JavaScript Patterns -*by Stoyan Stefanov (O`Reilly)