**Scenario**

You want to use just the methods you like, without inheriting all the other methods that you’ll never need. This is possible with the borrowing methods pattern, which benefits from the function methods  call() and apply().

[复制代码](javascript:void(0);)

// call() example

notmyobj.doStuff.call(myobj, param1, p2, p3);

// apply() example

notmyobj.doStuff.apply(myobj, [param1, p2, p3]);

[复制代码](javascript:void(0);)

Example: Borrow from Array

[复制代码](javascript:void(0);)

// Example for calling the slice

function f() {

var args = [].slice.call(arguments, 1, 3);

return args;

}

// example

f(1, 2, 3, 4, 5, 6); // returns [2,3]

[复制代码](javascript:void(0);)

Borrow and Bind

When borrowing methods either through call()/apply() or through simple assignment, the object that this points to inside of the borrowed method is determined based on the call expression. But sometimes it’s best to have the value of  this “locked” or bound to a specific object and predetermined in advance.

[复制代码](javascript:void(0);)

var one = {

name: "object",

say: function (greet) {

return greet + ", " + this.name;

}

};

var two = {

name: "another object"

};

var say = one.say;

// passing as a callback

var yetanother = {

name: "Yet another object",

method: function (callback) {

return callback('Hola');

}

};

one.say('hi'); // "hi, object"

one.say.apply(two, ['hello']); // "hello, another object"

say('hoho'); // "hoho, undefined"

yetanother.method(one.say); // "Holla, undefined"

[复制代码](javascript:void(0);)

Solution

This bind() function accepts an object o and a method m, binds the two together, and then returns another function. The returned function as access to  o and  m via a closure. Therefore even after  bind() returns, the inner function will have access to  o and  m, which will always point to the original object and method.

[复制代码](javascript:void(0);)

function bind(o, m) {

return function () {

return m.apply(o, [].slice.call(arguments));

};

}

var twosay = bind(two, one.say);

twosay('yo'); // "yo, another object"

[复制代码](javascript:void(0);)

Disadvantage

The price you pay for the luxury of having a bind is the additional closure.

Function.prototype.bind()

ECMAScript 5 adds a method bind() to Function.prototype, making it just as easy to use as apply() and call().

var newFunc = obj.someFunc.bind(myobj, 1, 2, 3);

Implement Function.prototype.bind() when your program runs in pre-ES5 environments.

It’s using partial application and concatenating the list of arguments—those passed to  bind()(except the first) and those passed when the new function returned by  bind() is called later.

var twosay2 = one.say.bind(two);

twosay2('Bonjour'); // "Bonjour, another object"

**References*:***

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