Function Application

apply() takes two parameters: the first one is an object to bind to this inside of the function, the second is an array or arguments, which then becomes the array-like arguments object available inside the function. If the first parameter is null, then this points to the global object, which is exactly what happens when you call a function that is not a method of a specific object.

[复制代码](javascript:void(0);)

// define a function

var sayHi = function(who) {

return "Hello" + ( who ? ", " + who : "") + "!";

};

// invoke a function

sayHi();

// "Hello"

sayHi('world');

// "Hello, world!"

// apply a function

sayHi.apply(null, ["hello"]);

// "Hello, hello!"

//-----------------------------------------------------------------------

var alien = {

sayHi : function(who) {

return "Hello" + ( who ? ", " + who : "") + "!";

}

};

alien.sayHi('world');

// "Hello, world!"

sayHi.apply(alien, ["humans"]);

// "Hello, humans!"

[复制代码](javascript:void(0);)

In the preceding snippet,  this inside of  sayHi() points to  alien. In the previous example this points to the global object.

When you have a function that takes only one parameter, you can save the work of creating arrays with just one element:

// the second is more efficient, saves an array

sayHi.apply(alien, ["humans"]); // "Hello, humans!"

sayHi.call(alien, "humans"); // "Hello, humans!"

Partial Application

[复制代码](javascript:void(0);)

var add = function(x, y) {

return x + y;

};

// full application

add.apply(null, [5, 4]);

// 9

// partial application

var newadd = add.partialApply(null, [5]);

// applying an argument to the new function

newadd.apply(null, [4]);

// 9

[复制代码](javascript:void(0);)

Here’s no  partialApply() method and functions in JavaScript don’t behave like this by default. But you can make them, because JavaScript is dynamic enough to allow this. The process of making a function understand and handle partial application is called currying.

[复制代码](javascript:void(0);)

// a curried add()

// accepts partial list of arguments

function add(x, y) {

var oldx = x, oldy = y;

if ( typeof oldy === "undefined") {// partial

return function(newy) {

return oldx + newy;

};

}

// full application

return x + y;

}

// test

typeof add(5);

// "function"

add(3)(4);

// 7

// create and store a new function

var add2000 = add(2000);

add2000(10);

// 2010

[复制代码](javascript:void(0);)

General-purpose currying function

[复制代码](javascript:void(0);)

function schonfinkelize(fn) {

var slice = Array.prototype.slice, stored\_args = slice.call(arguments, 1);

return function() {

var new\_args = slice.call(arguments), args = stored\_args.concat(new\_args);

return fn.apply(null, args);

};

}

// a normal function

function add(x, y) {

return x + y;

}

// curry a function to get a new function

var newadd = schonfinkelize(add, 5);

newadd(4);

// 9

// another option -- call the new function directly

schonfinkelize(add, 6)(7);

// 13

// a normal function

function add(a, b, c, d, e) {

return a + b + c + d + e;

}

// works with any number of arguments

schonfinkelize(add, 1, 2, 3)(5, 5);

// 16

// two-step currying

var addOne = schonfinkelize(add, 1);

addOne(10, 10, 10, 10);

// 41

var addSix = schonfinkelize(addOne, 2, 3);

addSix(5, 5);

// 16

[复制代码](javascript:void(0);)

When to Use Currying

When you find yourself calling the same function and passing mostly the same parameters, then the function is probably a good candidate for currying. You can create a new function dynamically by partially applying a set of arguments to your function. The new function will keep the repeated parameters stored (so you don’t have to pass them every time) and will use them to pre-fill the full list of arguments that the original function expects.

**References*:***

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