**介绍**

本篇主要是介绍Function方面使用的一些技巧（上篇），利用Function特性可以编写出很多非常有意思的代码，本篇主要包括：回调模式、配置对象、返回函数、分布程序、柯里化（Currying）。

**回调函数**

在JavaScript中，当一个函数A作为另外一个函数B的其中一个参数时，则函数A称为回调函数，即A可以在函数B的周期内执行（开始、中间、结束时均可）。

举例来说，有一个函数用于生成node

var complexComputation = function () { /\* 内部处理，并返回一个node\*/};

有一个findNodes函数声明用于查找所有的节点，然后通过callback回调进行执行代码。

var findNodes = function (callback) {

var nodes = [];

var node = complexComputation();

// 如果回调函数可用，则执行它

if (typeof callback === "function") {

callback(node);

}

nodes.push(node);

return nodes;

};

关于callback的定义，我们可以事先定义好来用：

// 定义callback

var hide = function (node) {

node.style.display = "none";

};

// 查找node，然后隐藏所有的node

var hiddenNodes = findNodes(hide);

也可以直接在调用的时候使用匿名定义，如下：

// 使用匿名函数定义callback

var blockNodes = findNodes(function (node) {

node.style.display = 'block';

});

我们平时用的最多的，估计就数jQuery的ajax方法的调用了，通过在done/faild上定义callback，以便在ajax调用成功或者失败的时候做进一步处理，代码如下(本代码基于jquery1.8版)：

var menuId = $("ul.nav").first().attr("id");

var request = $.ajax({

url: "script.php",

type: "POST",

data: {id : menuId},

dataType: "html"

});

//调用成功时的回调处理

request.done(function(msg) {

$("#log").html( msg );

});

//调用失败时的回调处理

request.fail(function(jqXHR, textStatus) {

alert( "Request failed: " + textStatus );

});

**配置对象**

如果一个函数（或方法）的参数只有一个参数，并且参数为对象字面量，我们则称这种模式为配置对象模式。例如，如下代码：

var conf = {

username:"shichuan",

first:"Chuan",

last:"Shi"

};

addPerson(conf);

则在addPerson内部，就可以随意使用conf的值了，一般用于初始化工作，例如jquery里的ajaxSetup也就是这种方式来实现的：

// 事先设置好初始值

$.ajaxSetup({

url: "/xmlhttp/",

global: false,

type: "POST"

});

// 然后再调用

$.ajax({ data: myData });

另外，很多jquery的插件也有这种形式的传参，只不过也可以不传，不传的时候则就使用默认值了。

**返回函数**

返回函数，则是指在一个函数的返回值为另外一个函数，或者根据特定的条件灵活创建的新函数，示例代码如下：

var setup = function () {

console.log(1);

return function () {

console.log(2);

};

};

// 调用setup 函数

var my = setup(); // 输出 1

my(); // 输出 2  
// 或者直接调用也可  
setup()();

或者你可以利用闭包的特性，在setup函数里记录一个私有的计数器数字，通过每次调用来增加计数器，代码如下：

var setup = function () {

var count = 0;

return function () {

return ++count;

};

};

// 用法

var next = setup();

next(); // 返回 1

next(); // 返回 2

next(); // 返回 3

**偏应用**

这里的偏应用，其实是将参数的传入工作分开进行，在有的时候一系列的操作可能会有某一个或几个参数始终完全一样，那么我们就可以先定义一个偏函数，然后再去执行这个函数（执行时传入剩余的不同参数）。

举个例子，代码如下：

var partialAny = (function (aps) {

// 该函数是你们自执行函数表达式的结果，并且赋值给了partialAny变量

function func(fn) {

var argsOrig = aps.call(arguments, 1);

return function () {

var args = [],

argsPartial = aps.call(arguments),

i = 0;

// 变量所有的原始参数集，

// 如果参数是partialAny.\_ 占位符，则使用下一个函数参数对应的值

// 否则使用原始参数里的值

for (; i < argsOrig.length; i++) {

args[i] = argsOrig[i] === func.\_

? argsPartial.shift()

: argsOrig[i];

}

// 如果有任何多余的参数，则添加到尾部

return fn.apply(this, args.concat(argsPartial));

};

}

// 用于占位符设置

func.\_ = {};

return func;

})(Array.prototype.slice);

使用方式如下：

// 定义处理函数

function hex(r, g, b) {

return '#' + r + g + b;

}

//定义偏函数, 将hex的第一个参数r作为不变的参数值ff

var redMax = partialAny(hex, 'ff', partialAny.\_, partialAny.\_);

// 新函数redMax的调用方式如下，只需要传入2个参数了：

console.log(redMax('11', '22')); // "#ff1122"

如果觉得partialAny.\_太长，可以用\_\_代替哦。

var \_\_ = partialAny.\_;

var greenMax = partialAny(hex, \_\_, 'ff');

console.log(greenMax('33', '44'));

var blueMax = partialAny(hex, \_\_, \_\_, 'ff');

console.log(blueMax('55', '66'));

var magentaMax = partialAny(hex, 'ff', \_\_, 'ff');

console.log(magentaMax('77'));

这样使用，就简洁多了吧。

**Currying**

Currying是函数式编程的一个特性，将多个参数的处理转化成单个参数的处理，类似链式调用。

举一个简单的add函数的例子：

function add(x, y) {

var oldx = x, oldy = y;

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

return function (newy) {

return oldx + newy;

}

}

return x + y;

}

这样调用方式就可以有多种了，比如：

// 测试

typeof add(5); // "function"

add(3)(4); // 7

// 也可以这样调用

var add2000 = add(2000);

add2000(10); // 2010

接下来，我们来定义一个比较通用的currying函数：

// 第一个参数为要应用的function，第二个参数是需要传入的最少参数个数

function curry(func, minArgs) {

if (minArgs == undefined) {

minArgs = 1;

}

function funcWithArgsFrozen(frozenargs) {

return function () {

// 优化处理，如果调用时没有参数，返回该函数本身

var args = Array.prototype.slice.call(arguments);

var newArgs = frozenargs.concat(args);

if (newArgs.length >= minArgs) {

return func.apply(this, newArgs);

} else {

return funcWithArgsFrozen(newArgs);

}

};

}

return funcWithArgsFrozen([]);

}

这样，我们就可以随意定义我们的业务行为了，比如定义加法：

var plus = curry(function () {

var result = 0;

for (var i = 0; i < arguments.length; ++i) {

result += arguments[i];

}

return result;

}, 2);

使用方式，真实多种多样哇。

plus(3, 2) // 正常调用

plus(3) // 偏应用，返回一个函数（返回值为3+参数值）

plus(3)(2) // 完整应用（返回5）

plus()(3)()()(2) // 返回 5

plus(3, 2, 4, 5) // 可以接收多个参数

plus(3)(2, 3, 5) // 同理

如下是减法的例子

var minus = curry(function (x) {

var result = x;

for (var i = 1; i < arguments.length; ++i) {

result -= arguments[i];

}

return result;

}, 2);

或者如果你想交换参数的顺序，你可以这样定义

var flip = curry(function (func) {

return curry(function (a, b) {

return func(b, a);

}, 2);

});

更多资料，可以参考如下地址：

<http://www.cnblogs.com/rubylouvre/archive/2009/11/09/1598761.html>

<http://www.cnblogs.com/sanshi/archive/2009/02/17/javascript_currying.html>

**总结**

JavaScript里的Function有很多特殊的功效，可以利用闭包以及arguments参数特性实现很多不同的技巧，下一篇我们将继续介绍利用Function进行初始化的技巧。

参考地址：http://shichuan.github.com/javascript-patterns/#function-patterns