JS中的柯里化(currying)

这篇文章发布于 2013年02月26日，星期二，23:33，归类于 JS实例。 阅读 172048 次, 今日 24 次 62 条评论

by zhangxinxu from http://www.zhangxinxu.com
本文地址：http://www.zhangxinxu.com/wordpress/?p=3048

一、柯里化和柯南的关系是?

回答：如果我说“柯里化 == 柯南”呢?

众人：博主，r u ok!? 是不是钓鱼钓久了脑袋秀逗了哈？柯里化可是函数式编程中的一个技巧，而柯南是到哪儿哪儿死人、10年不老的神话般的存在。八竿子都打不到的，怎会相等呢  ？？

回答：诸位，眼睛睁大点，是 `==`，不是 `===` 哦~

众人：嗯哪，我眼睛已经瞪得灯泡大了  ，粑粑并没有变冰淇淋啊?

回答：这不就结了嘛。我说的是弱等于 `==`，又不是强等于 `===`. 你想那，JS的世界里， `0==false`. 你看，这阿拉伯届的数字 `0`，和英美届的单词 `false` 不就相等了，这两个家伙也是八竿子都达不到的哦。

众人：  这……好吧，村妇吵架道理多，恕我们愚钝，还是看不出来“柯里化==柯南”，喔~除了那个“柯”字是一样的，求解释~~

回答：百科中柯里化解释为：“@#￥#￥#%*&……%@#￥[①]”，太术语了，我镜片看裂才知大概。若非要套用定义（见下面的①标注的引用），个人觉得：“柯里化”就像某些官员的把戏，官员要弄7个老婆，碍于国策（一夫一妻）以及年老弟衰，表面上就1个老婆，实际上剩下的6个暗地里消化。代码表示就是：

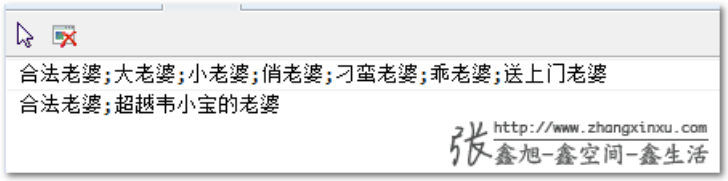
```
var currying = function(fn) {
    // fn 指官员消化老婆的手段
    var args = [].slice.call(arguments, 1);
    // args 指的是那个合法老婆
    return function() {
        // 已经有的老婆和新搞定的老婆们合成一体，方便控制
        var newArgs = args.concat([].slice.call(arguments));
        // 这些老婆们用 fn 这个手段消化利用，完成韦小宝前辈的壮举并返回
        return fn.apply(null, newArgs);
    };
};

// 下为官员如何搞定7个老婆的测试
// 获得合法老婆
var getWife = currying(function() {
    var allWife = [].slice.call(arguments);
    // allwife 就是所有的老婆的，包括暗渡陈仓进来的老婆
    console.log(allWife.join(";"));
}, "合法老婆");

// 获得其他6个老婆
getWife("大老婆","小老婆","俏老婆","刁蛮老婆","乖老婆","送上门老婆");

// 换一批老婆
getWife("超越韦小宝的老婆");
```

于是，结果就是：



众人：这与“柯南”童鞋有嘛关系？

> ① 柯里化（Currying），又称部分求值（Partial Evaluation），是把接受多个参数的函数变换成接受一个单一参数（最初函数的第一个参数）的函数，并且返回接受余下的参数而且返回结果的新函数的技术。

回答：莫急莫急，且听臣妾一曲。百科上的定义是针对众多函数式语言而言的，按照Stoyan Stefanov(《JavaScript Pattern》作者)的说法，所谓“柯里化”就是*使函数理解并处理部分应用*，套用上面官员例子，就是只要有合法老婆和见不得的情妇就行，数目什么的随意。不过，如果这样理解，老婆的例子就不符合国情，反而更加契合的理解是“柯南”。

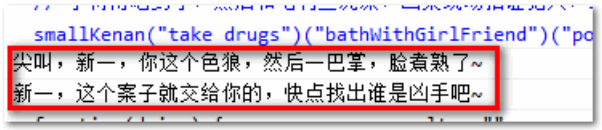
众人：哦？洗耳恭听！

回答：柯南身子虽小，但是里面住的却是大柯南，也就是一个`function`里面还有个`function`。不同柯南处理不同情况，例如，小柯南可以和…稍等，他女朋友叫什么的忘了，我查查…哦，毛利兰一起洗澡澡；但是大柯南就不行。小柯南不能当面指正犯人，需借助小五郎；但是，大柯南就可以直接质问指出凶手。就类似于，内外`function`处理不同的参数。如果代码表示就是(小柯南=`smallKenan`；大柯南=`bigKenan`；小柯南嗑药会变大柯南)：

```
var smallKenan = function(action) {
    var bigKenan = function(doing) {
        var result = "";
        if (action === "take drugs") {
            if (doing === "bathWithGirlFriend") {
                result = "尖叫，新一，你这个色狼，然后一巴掌，脸煮熟了~";
            } else if (doing === "pointOutKiller") {
                result = "新一，这个案子就交给你的，快点找出谁是凶手吧~";
            }
        } else {
            if (doing === "bathWithGirlFriend") {
                result = "来吧，柯南，一起洗澡吧~";
            } else if (doing === "pointOutKiller") {
                result = "小孩子家，滚一边去！";
            }
        }
        console.log(result);
        return arguments.callee; // 等同于return bigKenan
    };
    return bigKenan;
};

// 小柯南吃药了，然后和毛利兰洗澡，凶案现场指证犯人；结果是……
smallKenan("take drugs")("bathWithGirlFriend")("pointOutKiller");
```

结果如下截图：



“吃药”、“洗澡”、“指出凶手”就可以看成三个参数，其中，“吃药”确实是小柯南使用的，而后面的是“洗澡”、“指出凶手”虽然跟在`smallKenan()`后面，实际上是大柯南使用的。这个就是柯里化，参数部分使用。外部函数处理部分应用，剩下的由外部函数的返回函数处理。

所以，我说“柯里化==柯南”不是胡扯吧~~
众人：我们擦~~，博主扯淡的功力TM现在更上一层楼啦，说得还真那么回事！

二、柯里化有什么作用?

众人：整这么多神乎其神的东西，有个毛线用？意淫一些所谓的技术，为了技术而技术最让人诟病了，我老老实实`if else`照样脚本跑得杠杠的！

回答：唉，大家所言极是。苍蝇蚊子嗡嗡不停也让人诟病，但是，也是有其存在的理由的，我们不妨看看~~
众人：

回答：貌似柯里化有3个常见作用：**1. 参数复用；2. 提前返回；3. 延迟计算/运行。**
1. “参数复用”上面已经展示过了，官员老婆的例子就是，无论哪个官员，都是需要一个合法老婆；通过柯里化过程，`getWife()`无需添加这个多余的“合法老婆”参数。
2. “提前返回”，很常见的一个例子，兼容现代浏览器以及IE浏览器的事件添加方法。我们正常情况可能会这样写：

```
var addEvent = function(el, type, fn, capture) {
    if (window.addEventListener) {
        el.addEventListener(type, function(e) {
            fn.call(el, e);
        }, capture);
    } else if (window.attachEvent) {
        el.attachEvent("on" + type, function(e) {
            fn.call(el, e);
        });
    }
};
```

上面的方法有什么问题呢？很显然，我们每次使用`addEvent`为元素添加事件的时候，(eg. IE6/IE7)都会走一遍`if...else if ...`，其实只要一次判定就可以了，怎么做？–柯里化。改为下面这样子的代码：

```
var addEvent = (function(){
    if (window.addEventListener) {
        return function(el, sType, fn, capture) {
            el.addEventListener(sType, function(e) {
                fn.call(el, e);
            }, (capture));
        };
    } else if (window.attachEvent) {
        return function(el, sType, fn, capture) {
            el.attachEvent("on" + sType, function(e) {
                fn.call(el, e);
            });
        };
    }
})();
```

初始`addEvent`的执行其实值实现了部分的应用（只有一次的`if...else if...`判定），而剩余的参数应用都是其返回函数实现的，典型的柯里化。

3.“延迟计算”，一般而言，延迟计算或运行是没有必要的，因为一天花10块钱和月末花300块钱没什么本质区别——只是心里好受点（温水炖青蛙）。嘛，毕竟只是个人看法，您可能会不这么认为。举个例子，我每周末都要去钓鱼，我想知道我12月份4个周末总共钓了几斤鱼，把一些所谓的模式、概念抛开，我们可能就会下面这样实现：

```
var fishweight = 0;
var addweight = function(weight) {
    fishweight += weight;
};

addweight(2.3);
addweight(6.5);
addweight(1.2);
addweight(2.5);

console.log(fishweight);   // 12.5
```

每次`addWeight`都会累加鱼的总重量。

若是有柯里化实现，则会是下面这样：

```
var curryweight = function(fn) {
    var _fishweight = [];
    return function() {
        if (arguments.length === 0) {
            return fn.apply(null, _fishweight);
        } else {
            _fishweight = _fishweight.concat([].slice.call(arguments));
        }
    }
};
var fishweight = 0;
var addweight = curryweight(function() {
    var i=0; len = arguments.length;
    for (i; i<len; i+=1) {
        fishweight += arguments[i];
    }
});

addweight(2.3);
addweight(6.5);
addweight(1.2);
addweight(2.5);
addweight();    //  这里才计算

console.log(fishweight);    // 12.5
```

众人：我勒个去，好高的楼啊，相比之下。

回答：确实，柯里化的实现似乎啰嗦了点。老妈的啰嗦显然不是用来消耗多余的口水的。这里的`curryWeight`方法啰嗦的意义在于**柯里化的复用**。比方说，我还想知道平均每次钓货的重量，则：

```
var averageweight = 0;
```

```
var addWeight = curryWeight(function() {
    var i=0; len = arguments.length;
    for (i; i<len; i+=1) {
        averageWeight += arguments[i]/len;
    }
});

addWeight(2.3);
addWeight(6.5);
addWeight(1.2);
addWeight(2.5);
addWeight();    //  这里才计算

console.log(averageWeight);    // 3.125
```

虽然延迟计算听上去很高级，但是，恕我愚钝，我想破了脑袋也没想出哪种情况非要柯里化延迟计算实现才能显著提高性能。能想到的好处就是参数个数随意，比方说：

```
addWeight(2.3, 6.5);
addWeight(1.2, 2.5);
```

也是可以的。

补充于翌日：经人提点，发现自己忘了个东西，ES5中的`bind`方法，用来改变`Function`执行时候的上下文（函数主体本身不执行，与`call`/`apply`直接执行并改变不同），本质上就是延迟执行。例如：

```
var obj = {
    "name": "currying"
},
fun = function() {
    console.log(this.name);
}.bind(obj);

fun(); // currying
```

从IE6~8的自定义扩展来看，其实现的机制就是柯里化（不考虑执行时的新增参数）：

```
if (!function() {}.bind) {
    Function.prototype.bind = function(context) {
        var self = this
            , args = Array.prototype.slice.call(arguments);

        return function() {
            return self.apply(context, args.slice(1));
        }
    };
}
```

关于ES5中`bind`方法自定义可参见我之前的“ECMAScript 5中bind方法、自定义及小拓展”一文。

三、没有问答的结束语

最近在看《JavaScript模式》一书，天哪，里面出现的各种设计模式（如工厂模式、外观模式、观察者模式），一双手都数不过来。而且这些名词又很抽象，书一合，马上大眼瞪小眼了。这种感觉就像是，房

间里来了10个黑人，每个黑人都有一个“￥%#…￥”的名字，好不容易勉强记住了，灯一关房间一黑，等再开灯的时候，每个黑人的名字都变成…..”hello”了。

其实这些模式在实际使用的时候，或多或少都使用过，当看到“**模式”概念的时候，我们就会猛然惊起：“哦，原来这个就叫做‘观察者模式’等”。现在要讨论的问题是，我们有没有必要把这些“**模式”都记住呢，都理解其对应的核心呢？这个问题类似于，我可以看懂NBA的篮球比赛，那我有没有必要把各个球队以及球队的队员都记住呢？

如果想成为JS大神，从这个目标来看，这是需要的；好比优秀的篮球解说员必须要知道每个球队的名字、球员甚至周边八卦。但是，现实很重要。如果连JS函数相关的基本东西都驾驭不好，显然，硬是啃这些似懂非懂的概念只会造成混乱。如果你觉得可以更近一步，先通透几个自己习惯的熟悉的使用模式，足够应付实际项目；其他一些概念什么的，更多的只是噱头，实用性其实并不大。正如本文的柯里化，看上去很高级，似乎也有点用处，然而JS的灵活性使得很多实现完全摆脱“柯里化”这个概念的束缚，以更通俗易懂的方式实现。

然而，即使实用性不高，我们还是要有所了解，因为，你不知道什么时候会用到它。比方说CSS中的`display:table;`某些情况下可以解决一些棘手问题(secret!).

因此，本文的柯里化至少要知道个大概，如果你囫囵吞枣式的看到此处，只记得“柯里化==柯南”，你可以再稍微静点心重新看一下。我的笔头要比嘴巴犀利的多，因为，文字我可以静心琢磨：“如何讲述才能提起兴趣，才能让新手也能知其意会其形？”，久而久之，就形成了这种菊紧的风格——更适合新手阅读，同时被“高人”不屑：“写得太罗嗦啦，直接一句话不就完事啦！”还时不时来一句“鉴定完毕”，恩，挺适合去验尸。

柯里化函数是有一个通用的方法的，在官员那个例子(`currying()`方法就是)其实已经展示了，不重复展示，至于代码的含义嘛，知者自知，不啰嗦。

好久没在结尾扯这么多乱七八糟的东西了，扯得心里像喝了常润茶般舒畅。本文内容，边学习变整理的，JS的概念我接触的也不多，文中错误难免，欢迎指正，欢迎交流，互相学习。

> 《CSS世界》签名版独家发售，包邮，可指定寄语，点击显示购买码

（本篇完） // 想要打赏？点击这里。有话要说？点击这里。

« HTML5 progress元素的样式控制、兼容与实例

翻译-盲人如何使用互联网的8个误区 »

猜你喜欢

- 翻译 - 解释JavaScript的“预解析(置顶解析)”
- 翻译-高质量JavaScript代码书写基本要点
- 翻编-JavaScript有关的10个怪癖和秘密
- 小折腾：JavaScript与元素间的抛物线轨迹运动
- 如何使用Tween.js各类原生动画运动缓动算法
- canvas 2D炫酷动效的实现套路和需要的技术积累
- jQuery-很酷的弹出层效果js插件
- reflection.js-实现图片投影倒影效果js插件

- JavaScript实现新浪微博文字放大显示动画效果
- IE下css bug集合-翻译自haslayout.net
- jQuery - 鼠标经过(hover)事件的延时处理

分享到：1

标签： currying, js相关, 函数, 柯里化

发表评论（目前62条评论）

名称 (必须)

邮件地址(不会被公开) (必须)

网站

提交评论

« 先前评论

1. 本末说道：
2018年12月12日 22:05

为什么我感觉网上的节流demo 全是假的啊？ ！还是姿势不对啊？ ！

为什么我这有这样才能使用， 用柯里化 怎么显得优雅一些

```
/**
* 节流函数
* @param method 事件触发的操作
* @param delay 间隔多少毫秒需要触发一次事件
*/
let timer,start ;
// = getSen(‘congestion’)
export function throttle(method, delay) {
let args = arguments;
return function loop() {
let self = this, now = Date.now();
start = start || now;
clearTimeout(timer);
if(now – start >= delay) {
method.apply(self, args);
start = now;
// setSen(‘congestion’, now);
}else {
timer = setTimeout(function() {
loop.apply(self, args);
}, delay + 10);
}
};
}
```

回复

2. **D.M**说道：
2018年11月6日 09:16

真的优秀！

回复

3. 推广说道：
2018年05月30日 10:59

增强理解

回复

« 先前评论

最新文章

» 常见的CSS图形绘制合集
» 粉丝群第1期CSS小测点评与答疑
» 分享三个纯CSS实现26个英文字母的案例
» 小tips: 纯CSS实现打字动画效果
» CSS/CSS3 box-decoration-break属性简介
» CSS :placeholder-shown伪类实现Material Design占位符交互效果
» 从天猫某活动视频不必要的3次请求说起
» CSS vector-effect与SVG stroke描边缩放
» CSS ::backdrop伪元素是干嘛用的?
» 周知：CSS -webkit-伪元素选择器不再导致整行无效

今日热门

» 常见的CSS图形绘制合集 (193)
» 未来必热：SVG Sprite技术介绍 (120)
» 粉丝群第1期CSS小测点评与答疑 (115)
» HTML5终极备忘大全（图片版+文字版） (93)
» 让所有浏览器支持HTML5 video视频标签 (86)
» Selectivizr-让IE6~8支持CSS3伪类和属性选择器 (82)
» CSS3下的147个颜色名称及对应颜色值 (80)
» 视区相关单位vw, vh..简介以及可实际应用场景 (76)
» 写给自己看的display: flex布局教程 (76)
» 小tips: 纯CSS实现打字动画效果 (76)

今年热议

» 《CSS世界》女主角诚寻靠谱一起奋斗之人 (76)
» 不借助Echarts等图形框架原生JS快速实现折线图效果 (64)
» 看，for..in和for..of在那里吵架！ (60)
» 是时候好好安利下LuLu UI框架了！ (47)
» 原来浏览器原生支持JS Base64编码解码 (35)
» 妙法攻略：渐变虚框及边框滚动动画的纯CSS实现 (33)

» 炫酷H5中序列图片视频化播放的高性能实现 (31)
» CSS scroll-behavior和JS scrollIntoView让页面滚动平滑 (30)
» windows系统下批量删除OS X系统.DS_Store文件 (26)
» 写给自己看的display: flex布局教程 (26)

猜你喜欢

- 翻译 - 解释JavaScript的“预解析(置顶解析)”
- 翻译-高质量JavaScript代码书写基本要点
- 翻编-JavaScript有关的10个怪癖和秘密
- 小折腾：JavaScript与元素间的抛物线轨迹运动
- 如何使用Tween.js各类原生动画运动缓动算法
- canvas 2D炫酷动效的实现套路和需要的技术积累
- jQuery-很酷的弹出层效果js插件
- reflection.js-实现图片投影倒影效果js插件
- JavaScript实现新浪微博文字放大显示动画效果
- IE下css bug集合-翻译自haslayout.net
- jQuery - 鼠标经过(hover)事件的延时处理

Designed & Powerd by zhangxinxu
Copyright© 2009-2019 张鑫旭-鑫空间-鑫生活
鄂ICP备09015569号