JavaScript： 零基础轻松学闭包

本文面向初学者，大神轻喷。

闭包是什么？

初学javascript的人，都会接触到一个东西叫做闭包，听起来感觉很高大上的。网上也有各种五花八门的解释，其实我个人感觉，没必要用太理论化的观念来看待闭包。

事实上，你每天都在用闭包，只是你不知道罢了。

比如：

```
var cheese = '奶酪';

var test = function(){
    alert(cheese);
}
```

OK，你已经写了一个闭包。

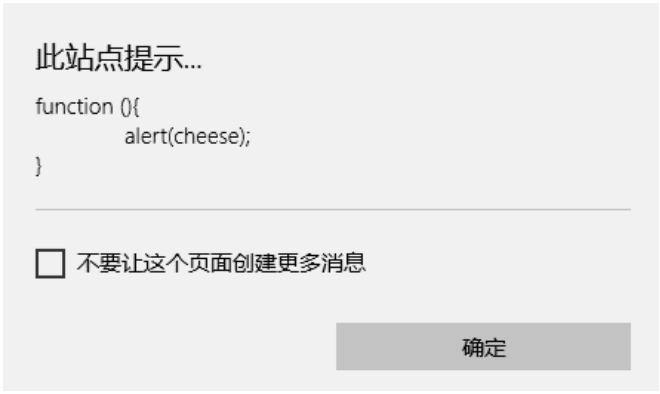
函数也是一个数据类型

变量 cheese 是在全局作用域中的一个变量，当你创建了一个 test 函数，那么，test 和 cheese 就共享一个全局作用域。

你要额外明白的一点是，在js中，函数和变量本质上是一个东西。函数也是一个数据类型。

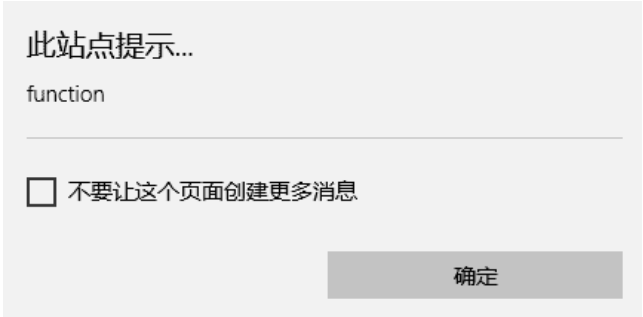
从上面的定义中也能看出来这一点。你要是不相信的话，我们来看一下咯。

```
alert(cheese);
alert(test);
```



让我们再来看看 test 和 cheese各是什么类型：

```
alert(typeof test);
```




```
alert(typeof cheese);
```

看到了吧，只是类型不同而已，他们都是数据类型。

唯一的不同点就是，函数类型的 test 可以拥有自己内部逻辑，而string类型的 cheese 只能存放一个字面值，这就是区别，仅此而已。

一目了然了，唯一不同的就是普通变量是字面值一样的存在，而函数需要打个括号才能执行而已。

你看，我现在打一个括号：

```
test();
```

打了括号，才会执行函数里面的逻辑。

作用域

让我们回到闭包，现在将之前的代码做一个小小的变动：

```
var cheese = '奶酪';
var test = function(){
    alert(cheese);
}

function test2(){
    var cheese = null;
    test();
}

test2();
```

那么，你觉得现在 alert 出来的是 null 还是奶酪呢？

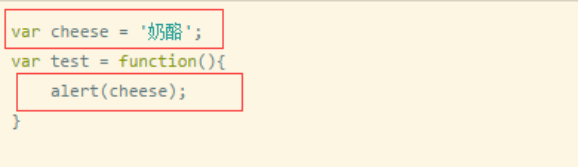
思考一下。。。

对的，弹出来的还是奶酪。

之前已经说过了，函数 test 和 变量 cheese 同处于一片蓝天下 -- 同一个作用域。

函数 test 和 变量 cheese 共同享有的作用域叫做全局作用域，就好像地球一样，我们所有的人都享有这个地球，能够在这里呼吸，吃饭，玩耍。

对test而言，他能访问到的作用域只有它本身的闭包和全局作用域：



也就是说，正常情况下他访问不到其他闭包里的内容，在 test2 里面定义的变量跟它没有半毛钱关系，所以弹出来的 cheese 依旧是全局作

用域里的 cheese。

函数可以创造自己的作用域。

我们刚才定义了一个 test 函数，{ } 包裹起来的部分就形成了一个新的作用域，也就是所谓的闭包。

其实你深刻了解了作用域的原理后，闭包也就理解了。

就好比地球是一个全局作用域，你自己家的房子是一个函数，你的房子是私人空间，就是一个局部作用域，也就是你自己建了一个闭包！

你透过窗户可以看见外边的景色，比如院子里的一棵芭蕉树，你于是通过眼镜观察看到了芭蕉树的颜色，高度，枝干的粗细等等。

这一棵芭蕉树相当于一个全局变量，你在自己的闭包内可以访问到它的数据。

所以，在这个例子中，test 就是一个房子，在里面可以通过窗户访问到全局作用域中的奶酪 —— 变量 cheese。

也就是说，cheese 在被 test 访问到的时候，就进入了它的闭包。

这样解释，你是否觉得好理解一点呢？

现在你是否可以理解一开始我说，闭包这东西其实我们天天都在用的意思了呢？

我们给出闭包的第一个注解：

1. 闭包就是在函数被创建的时候，存在的一个私有作用域，并且能够访问所有的父级作用域。

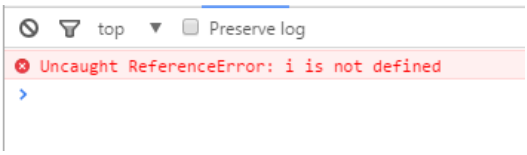
回到刚才的例子：

```
var cheese = '奶酪';
var test = function(){
    alert(cheese);
}

function test2(){
    var cheese = null;
    test();
}
```

在这个例子中，test 和 test2 各自享有一个作用域，对不对？而且他们互相不能访问。比如，我在 test 中定义的一个变量，test2就无法直接访问。

```
var test = function(){
    var i = 10;
}

function test2(){
    alert(i);
}

test2();
```

像这样，一旦执行 test2 函数，编译就不通过，因为在 test2的闭包内，根本找不到变量 i 。它首先会在自己的闭包内寻找 i，找不到的话就去父级作用域里找，这边的父级就是全局作用域，很遗憾，还是没有。这就是所谓的作用域链，它会一级一级往上找。如果找到最顶层，还是找不到的话，就会报错了。



在这里，还有一个需要注意的点就是：如果某一个闭包中对全局作用域（或父级作用域）中的变量进行了修改，那么任何引用该变量的闭包都会受到牵连。

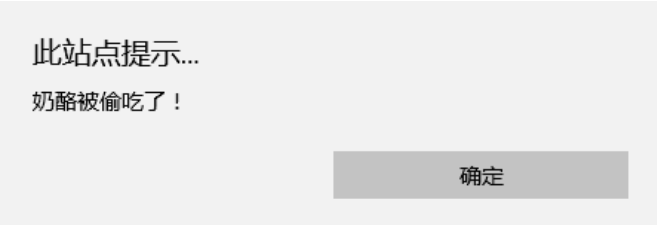
这的确是一个需要注意的地方。

举个例子

```
var cheese = '奶酪';

var test = function(){
    cheese = '奶酪被偷吃了！'
}

function test2(){
    alert(cheese);
```

```
}
test();
test2();
```

结果是：



很有趣，是不是呢？

当我们在定义一个函数，就产生了一个闭包，如果这个函数里面又有若干的内部函数，就是闭包嵌套着闭包。

像这样：

```
function house(){

    var footBall = '足球';
    /* 客厅 */
    function livingRoom(){
        var table = '餐桌';
        var sofa = '沙发';
        alert(footBall);
    }

    /* 卧室 */
    function bedRoom(){
        var bed = '大床';
    }

    livingRoom();
}

house();
```

函数house是一个闭包，里面又定义了两个函数，分别是livingRoom客厅，和bedRoom卧室，它们各自形成一个自己的闭包。对它们而言，父级作用域就是house。

如果我们希望在客厅里踢足球，在livingRoom函数执行的时候，它会先在自己的闭包中找足球，如果没找到，就去house里面找。一层一层往上找，直至找到了为止。当然，这个例子可能不是很恰当。但起码展示了作用域，闭包之间的联系。

再说明一下， 闭包就是在函数被创建的时候，存在的一个私有作用域，并且能够访问所有的父级作用域。因此，从理论上讲，任何函数都是一个闭包！

2. 如何将私有数据暴露出去

之前有这样一个例子

```
var test = function(){
    var i = 10;
}

function test2(){
    alert(i);
}

test2();
```

函数 test 和 test2 各自形成一个闭包，两个闭包之间无法访问对方的私有数据。比如，在 test 中定义的变量，在 test2 里面是无法直接访问到的。

那么问题来了， 当然，这边和挖掘机没关系。这里的问题是，有没有什么办法让 test2 可以访问到其他闭包中的私有变量呢？

办法当然是有的，最直接的想法就是，大不了我定义一个全局变量，在 test 中将私有数据赋给全局变量，然后在 test2 里面就能访问到了。

是的，因为两个函数共同享有一个全局作用域，所以这个办法确实可行。我在很多项目里也的确看到很多人就是这么做的。

那么，有没有一种更好的方法呢？要知道，全局作用域是一个比较敏感的地方，一不小心就会出现变量名重复的问题。顺便说一句，在全局作用域中，尽量不要使用诸如 temp , a , b , c 这一类的大众化变量。

于是，这就牵扯到返回值的相关知识了，你在C语言的教材中肯定见惯了类似于这样的代码

```
int sum(int a,int b)
{
    return a + b;
}

int all = sum(3,5);
```

这是一个简单的求和函数，很多人慢慢地养成了这样一个观念，就是函数的返回值就是一个字面值，要么是数字类型，要么是布尔类型，或者是字符串。

在很多强类型的语言，诸如 Java，C，C++， 确实如此。但是 return 在 JavaScript 中却大有来头。

在上一节已经说明了，js 的函数也是一种数据类型，你可以把函数看成是和int , float , double 一样的东西。

那么，既然int可以当做函数的参数或者返回值，函数当然也可以！

请看下面两句话：

在js中

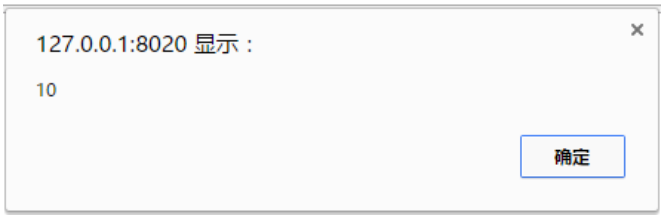
如果函数被当做参数传进去了，它就是所谓的回调函数。

如果函数被当做返回值return出去了，它就是把一个闭包return出去了。

这一章不讲回调函数，如果你不清楚啥叫回调函数，可以去看看这个小例子：
（浅谈js回调函数）

还是上面的那个例子，我们希望在 test2 中可以访问到 test 里面的变量，可以这样做：

```
var test = function(){
    var i = 10;
    /* 定义一个函数将变量i暴露出去  */
    var get = function(){
        return i ;
    }
    return get; //将函数暴露出去
}

function test2(){

    var fn= test();//接收test暴露出来的函数
    alert(fn()); //获得test中的私有数据
}

test2();
```



test 函数中的 get 方法是一个内部函数，它自己也形成了一个闭包， test 是他的父级作用域，因此它可以获取i的值。

i 进入 get 方法的闭包，被包了起来，然后最终被返回了出去。

而对于 test2 来说，是可以访问到 test函数的，因此可以调用并执行 test 函数，从而获取其返回值。

你可能会说，我直接在test中把i给return出去就好了嘛，干嘛这么麻烦。

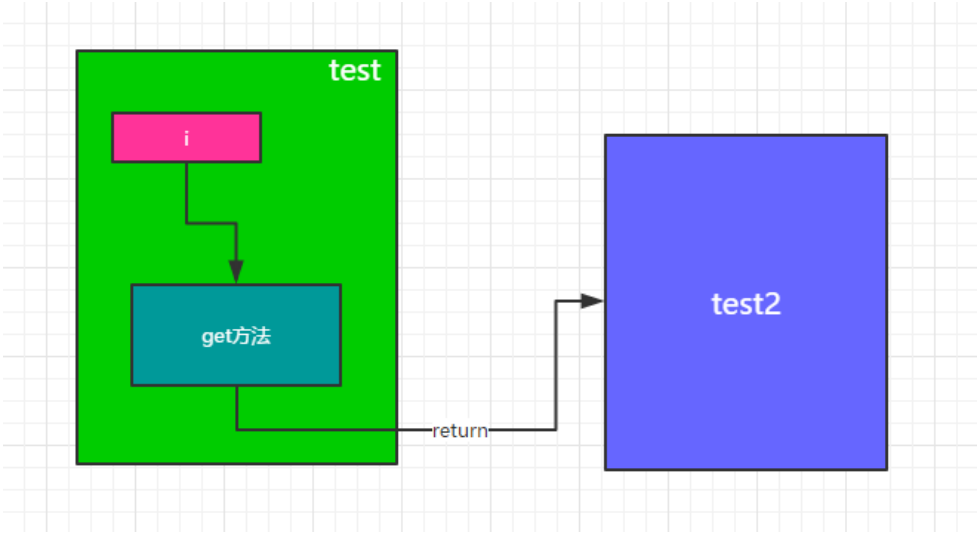
是的，言之有道理。

可是，如果我要访问 test 中多个私有数据咋办捏？

这下你可明白了吧！

现在，我们给出关于闭包的第二个注解：
（第一个注解在上一节）

从应用的角度来看，闭包可以将函数或者对象的私有数据暴露出去，而不影响全局作用域。



通过这张图，是不是好理解一些了呢？我们这一节单说函数里的私有数据。

2. 将私有数据包装成json对象

刚才的例子说明，在js中，return出去的可以是基本数据类型，也可以是函数类型。

其实，JavaScript是一种基于对象的语言，也有对象的概念，所以，我们可以把你需要的东西包裹成一个对象返回出去！

上代码：

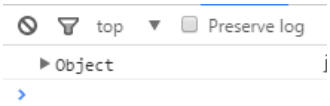
```
var test = function(){
    var apple = '苹果';
    var pear = '梨子';
    /* 定义一个函数将水果暴露出去  */
    var getFruit = {
        apple : apple ,
        pear : pear
    }

    return getFruit; //将对象暴露出去
}

function test2(){

    var getFruit = test();//接收test暴露出来的函数

    console.log(getFruit);
}

test2();
```

像这样用 { } 括起来的东西就是一个js对象，也就是所谓json。你可能经常会听到json这个词，觉得还挺高大上的。其实它就是一个用 { } 包起来的数据而已。

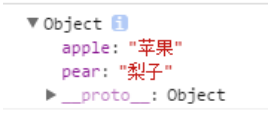
里面是键值对的形式，非常类似于Java里面的HashMap。

在这个例子中，我们可以直接把需要暴露的私有数据用一个 { } 包起来，构成一个json对象return出去就可以啦。

因为是 js 对象，alert 不能看到里面的具体内容，所以我们使用 console.log()，结果如下：



展开后：



这样是不是也可以了？多出来的 proto 是原型链，以后会讲到。

3. 我们来做一个紫金葫芦

大家都还记得西游记里孙悟空用遮天的把戏骗来的紫金葫芦吗，只要你拿着这个葫芦，叫一声别人的名字，如果答应了，别人就会被吸进去。

OK，这个紫金葫芦里面不正如一个闭包吗？

对不对嘛，所以，我们用闭包的知识来做一个好玩的东西吧。

```
<body>
    <div id='box' style='width:50px;height:50px;background:#333;color:#fff;text-align:center;line-height:50px'>小妖</div>
</body>
```

紫金葫芦里面的源码大概是这样的：

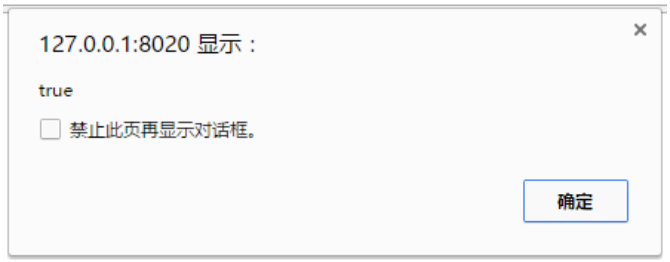
```
var 紫金葫芦 = function(id){
    var domElement = document.getElementById(id);

    var returnObject = {

        domElement : domElement ,

        backgroundColor : function(color){
            domElement.style.backgroundColor = color;
        },

        click : function(fn){
            domElement.onclick = fn;
        }
    };

    return returnObject;
}
```

注：我纯粹是为了看起来方便而采用中文定义变量，在实际开发中，千万不要使用中文变量。

我们在返回出去的对象上加了三个东西：

1.domElement
你传进来一个id，我就用 document.getElementById 来包一下，得到一个dom元素，最终要操作的也就是这个dom元素。也就是说：

```
var box1 = 紫金葫芦('box').domElement;
var box2 = document.getElementById('box');
alert(box1 === box2);
```



他们是一个东西，一样的。

```
紫金葫芦('box');
```

这行代码一旦执行，紫金葫芦就会返回 returnObject 对象，也就是说。我们喊一声 “box”，那个id为box的小妖一答应，就被装进来了，然后我们可以对它为所欲为！

比如，给它换一个背景色：

2.backgroundColor 给元素添加背景色的方法

```
var box = 紫金葫芦('box');
box.backgroundColor('red');
```

3.click 给元素添加点击事件，需要传入一个回调函数

```
var box = 紫金葫芦('box');
box.click(function(){
    alert('就没人吐槽这个无聊的作者么，小妖也有尊严的好么，啊喂！！');
});
```

结果：

也许你已经发现了，这些方法是不是和jQuery有点类似呢？