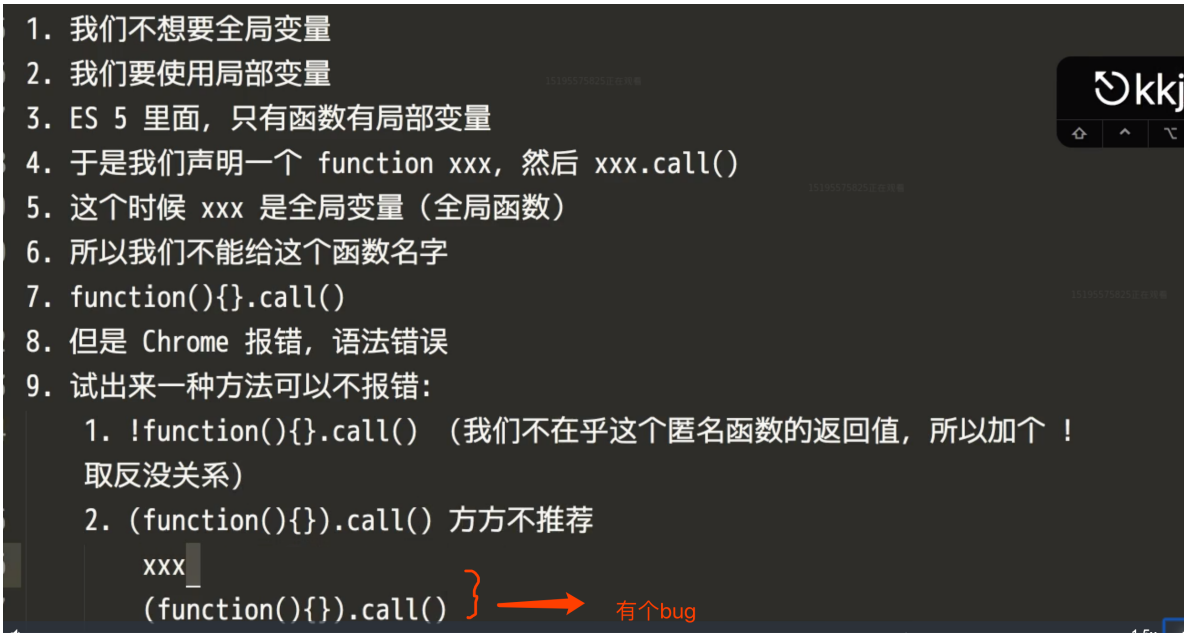
立即执行函数的使用

闭包永远和立即执行函数一起用;




# 立即执行函数的使用

使用场景

编写代码时不想用全局变量，只想使用局部变量

解决方案

ES5里面，只有函数有局部变量，可以声明一个function xxx，然后 xxx.call()。此时xxx成为了全局变量（全局函数），违背初衷，故采用匿名函数function(){}.call()。这种写法符合逻辑，但是Chrome 报错，提示语法错误。这种错误可以通过'取反'处理，即最终写法为：

```
!function(){
    //其它代码
    }.call()
```

注：上述写法并不在乎这个匿名函数的返回值，所以加个！取反没关系

访问局部变量

通过全局变量window访问

```
!function(){
    var person = window.person {//局部、全局变量都存下匿名对象的地址
        name:'tom'
    }
    console.log (person)
    console.log (window.person) //打印结果相同
}.call
```

```
!function(){
    var person = window.person
    console.log (person)
    console.log (window.person) //打印结果同上
}.call
```

通过匿名函数与闭包的结合访问

通常匿名函数会和闭包、全局变量一起使用，形成封装。

```
!function(){
    var person = {
        name:'tom',
        age:18
    }
    window.tomGrowup = function(){//闭包
        person.age = person.age + 1
        return person.age
    }
}.call

!function(){
    var newAge =  window.tomGrowup () //间接操作age，且只能增加
    console.log (newAge)
}.call
//如果没有闭包，这里的person就是全局变量
//1.立即执行函数使得person无法被外部访问
//2.闭包使得匿名函数可以操作person
//3.window.tomGrowup保存了匿名函数的地址
//4.任何地方都可以使用window.tomGrowup
//总结:任何地方都可以使用window.tomGrowup间接操作person，当不能直接访问person

```

下面代码与上同效

```
var accessor = function(){
    var person = {
        name:'tom',
        age:18
    }
    return function(){//返回匿名函数
        person.age = person.age +1
        return person.age
    }
}
var growUp = accessor.call()//立即执行，返回值为函数
```

```
growUp.call()//继续执行
//1.声明一个匿名函数并立即执行
//2.accessor匿名函数返回值是另一个匿名函数
//3.将accessor匿名函数返回值赋给growUp
//4.再次执行growUpgrowUp函数


```

注：立即执行函数用来隔离作用域

## 闭包永远和立即执行函数一起用;