### 标题

#### 1.执行上下文

JavaScript引擎通过执行上下文来跟踪代码，基于单线程的执行模式，通过执行上下文栈，来确定执行顺序。

JavaScript代码有两种类型：

全局代码在所有函数外部定义，对应全局执行上下文，只有一个；函数代码在函数内部定义，对应函数执行上下文，每次调用函数，就会创建一个新的。

    function skulk(ninja){

      report(ninja + 'skulking')

    }

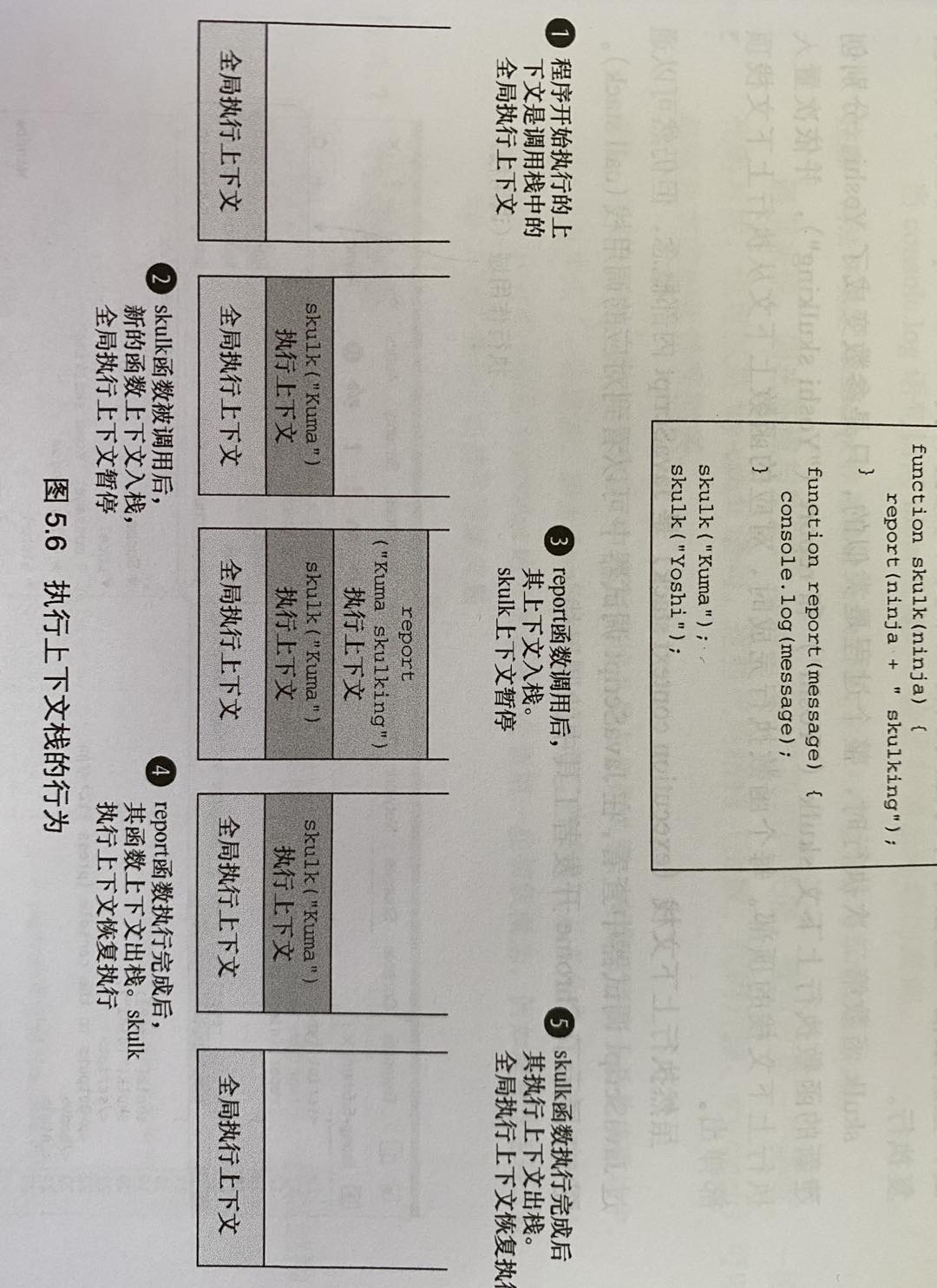
    function report(message){

      console.log(message);

    }

    skulk('kuma')

    skulk('yoshi')



#### 2.作用域

作用域是JavaScript引擎内部用来跟踪标识符与特定变量之间的映射关系，也叫词法环境。

    var ninja = 'lynch'

    console.log(ninja)

当console.log语句访问ninja变量时，会进行作用域的查询

#### 3.闭包

闭包是有权访问另一个函数作用域中的变量的函数。**闭包封闭的是外部状态，当外部状态的scope消失的时候，通过闭包还能访问到原始作用域（我自己感觉闭包更像一个空间，里面包括函数和变量）。**

三个条件：闭包是一个函数；闭包允许函数访问并操作函数创建时所在的作用域内的变量；闭包存在于定义该变量的作用域中。

    var outerValue = 'ninja'

    function outerFunction(){

      assert(outerValue === 'ninja', 'I can see the ninja')

    }

    outerFunction()

在同一作用域中声明outerFunction和outerValue，outerFunction可以看见并访问outerValue，这个闭包的例子优势不明显。

每一个通过闭包访问变量的函数都具有一个作用域链，作用域链包含闭包的所有信息。

#### 3.1使用闭包

使用闭包时，所有的信息都会存储在内存中，直到JavaScript引擎确保这些信息不再使用或页面卸载时，才会清理这些信息。

##### 1.封装私有变量

原生JavaScript不支持私有变量，通过使用闭包可以实现很接近的、可接受的私有变量。

function Ninja(){

      var feints = 0

      this.getFeints = function(){

        return feints

      }

      this.feint = function(){

        feints++

      }

    }

var ninja1 = new Ninja()

ninja1.feint()

assert(ninja1.feints === undefined,

'And the private data is inaccessible to us.')

assert(ninja1.getFeints() === 1,

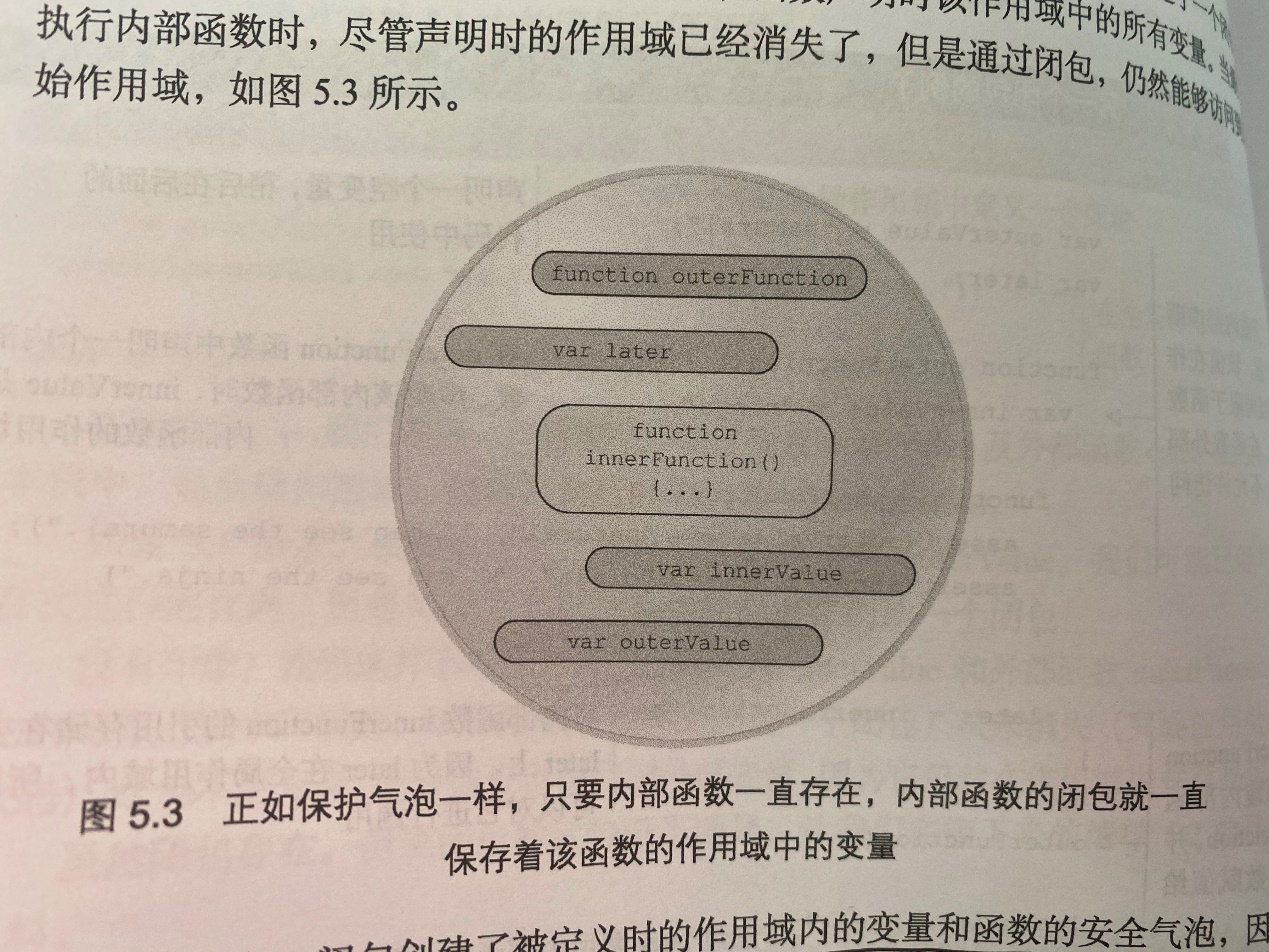
'We are able to access the internal feint count')

var ninja2 = new Ninja()

assert(ninja2.getFeints() === 0, 'The second ninja object gets its own feints variable.')

**闭包封闭的是外部状态，当外部状态的scope消失的时候，通过闭包还能访问到原始作用域。**

当在外部函数中声明内部函数时，不仅定义了函数的声明，而且还创建了一个闭包，这个‘气泡’里包含了函数的声明，还包含了函数声明时该作用域中的所有变量。



##### 2.回调函数

回调函数是需要在将来不确定的某一时刻异步调用的函数。回调函数需要频繁地访问外部数据。

使用闭包可以减少全局变量，减少传递函数的参数量。