

前端核心 JavaScript

Front-End JavaScriptCore

Unit02

内容

上午	09:00 ~ 09:30	作业讲解和回顾
	09:30 ~ 10:20	RegExp
	10:30 ~ 11:20	
	11:30 ~ 12:00	Function
下午	14:00 ~ 14:50	
	15:00 ~ 15:50	
	16:00 ~ 16:50	
	17:00 ~ 17:30	总结和答疑



Function

Function

Function 对象

函数与 Function 对象

创建函数

使用直接量方式创建函数

使用 Function 对象创建函数

三种定义方式的对比

闭包

变量作用域

作用域链

匿名函数的优点

闭包

闭包应用特征

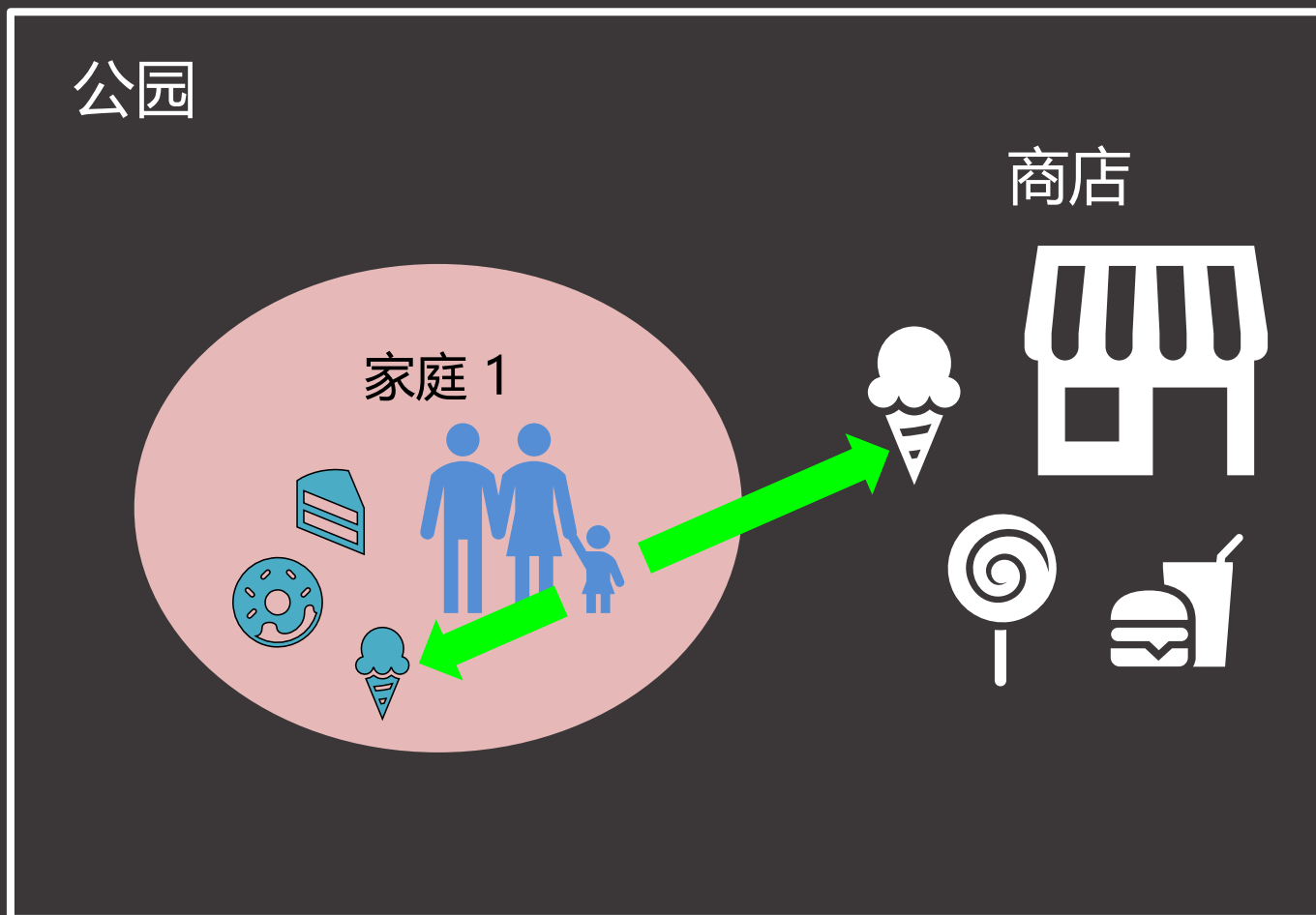
闭包的作用

作用域(scope)



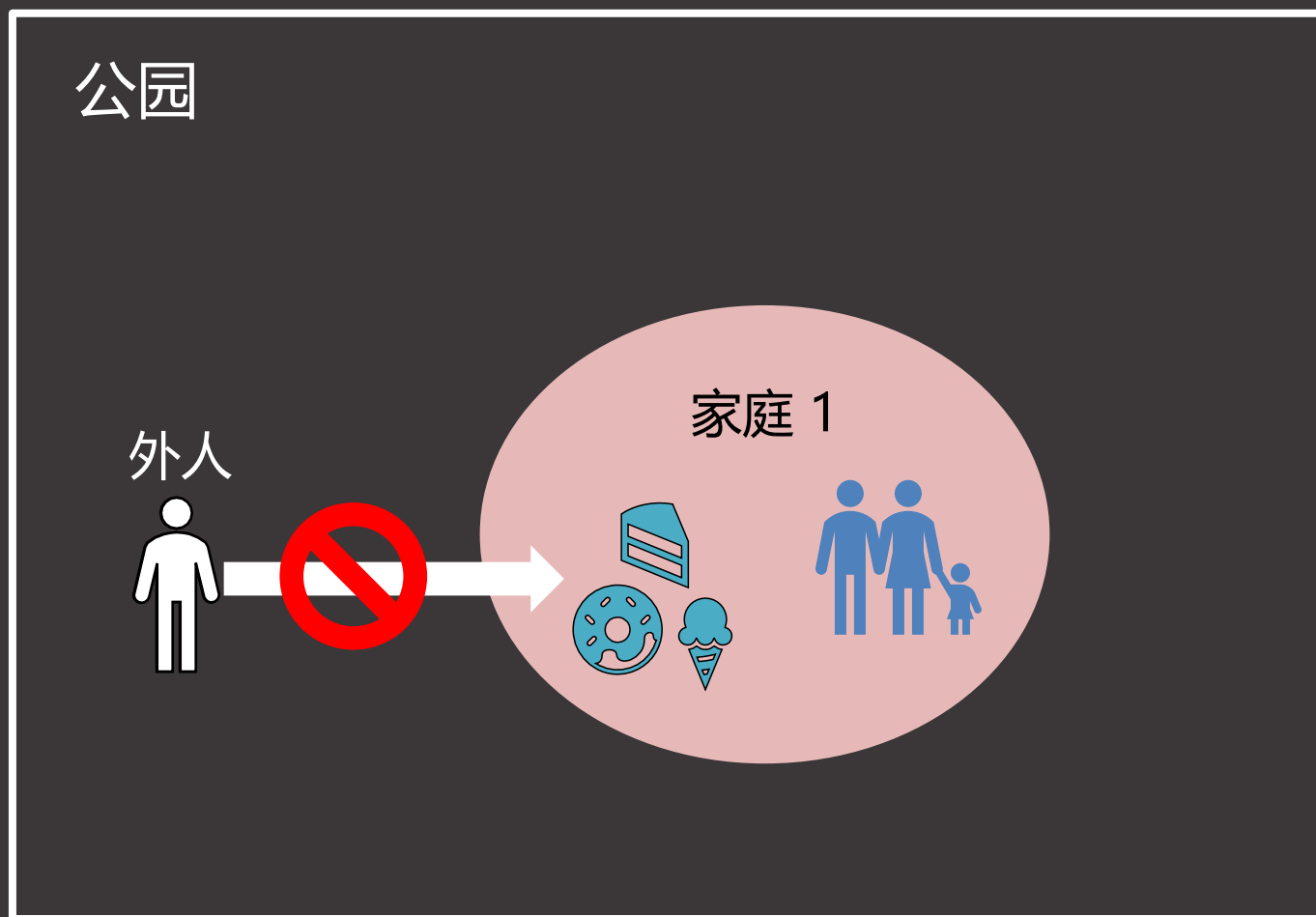
作用域

- 现实中：孩子既可以吃自家带的东西，又可以去公园的商店里买东西吃



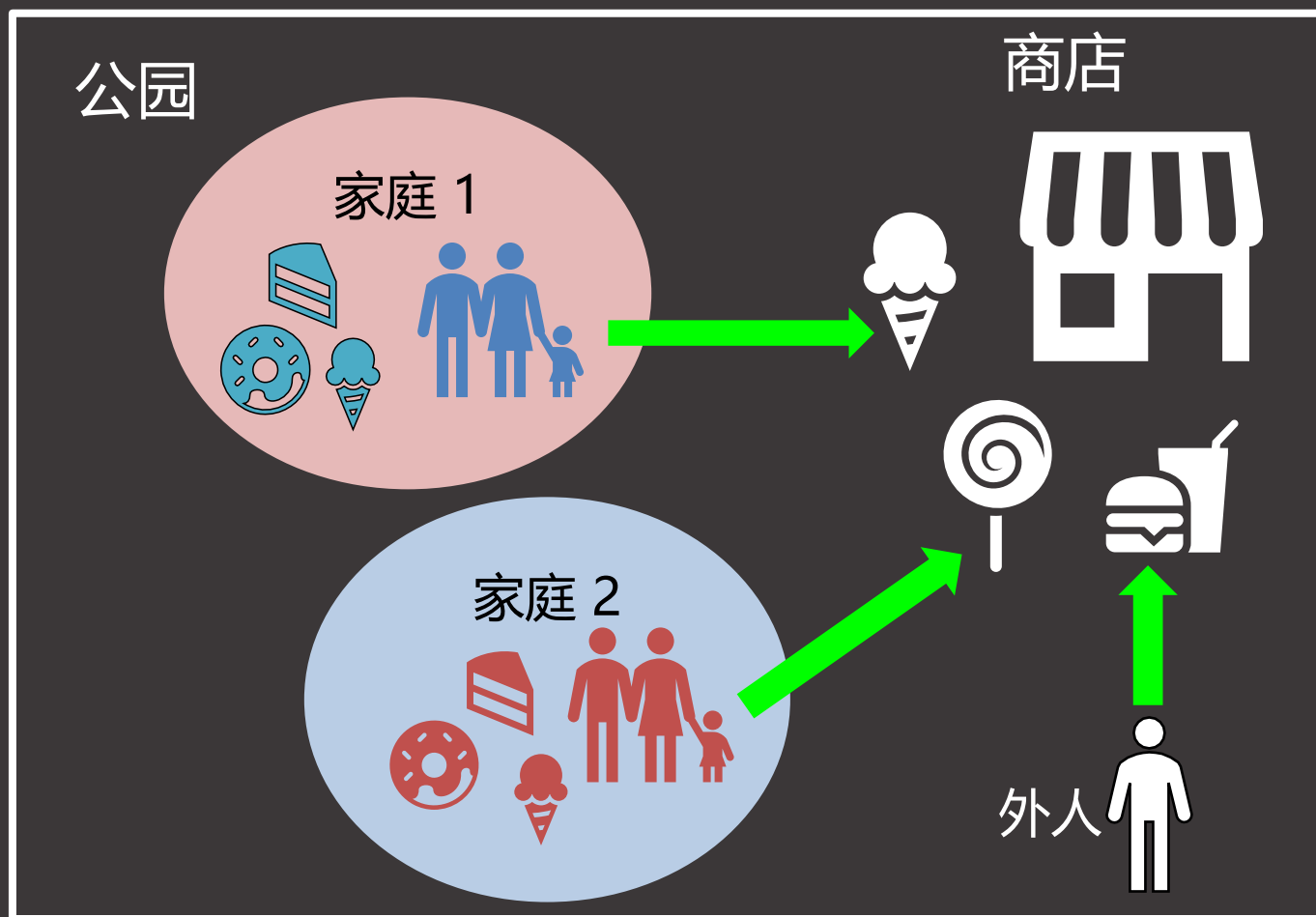
作用域

- 现实中：外人不能随意去别人家拿东西



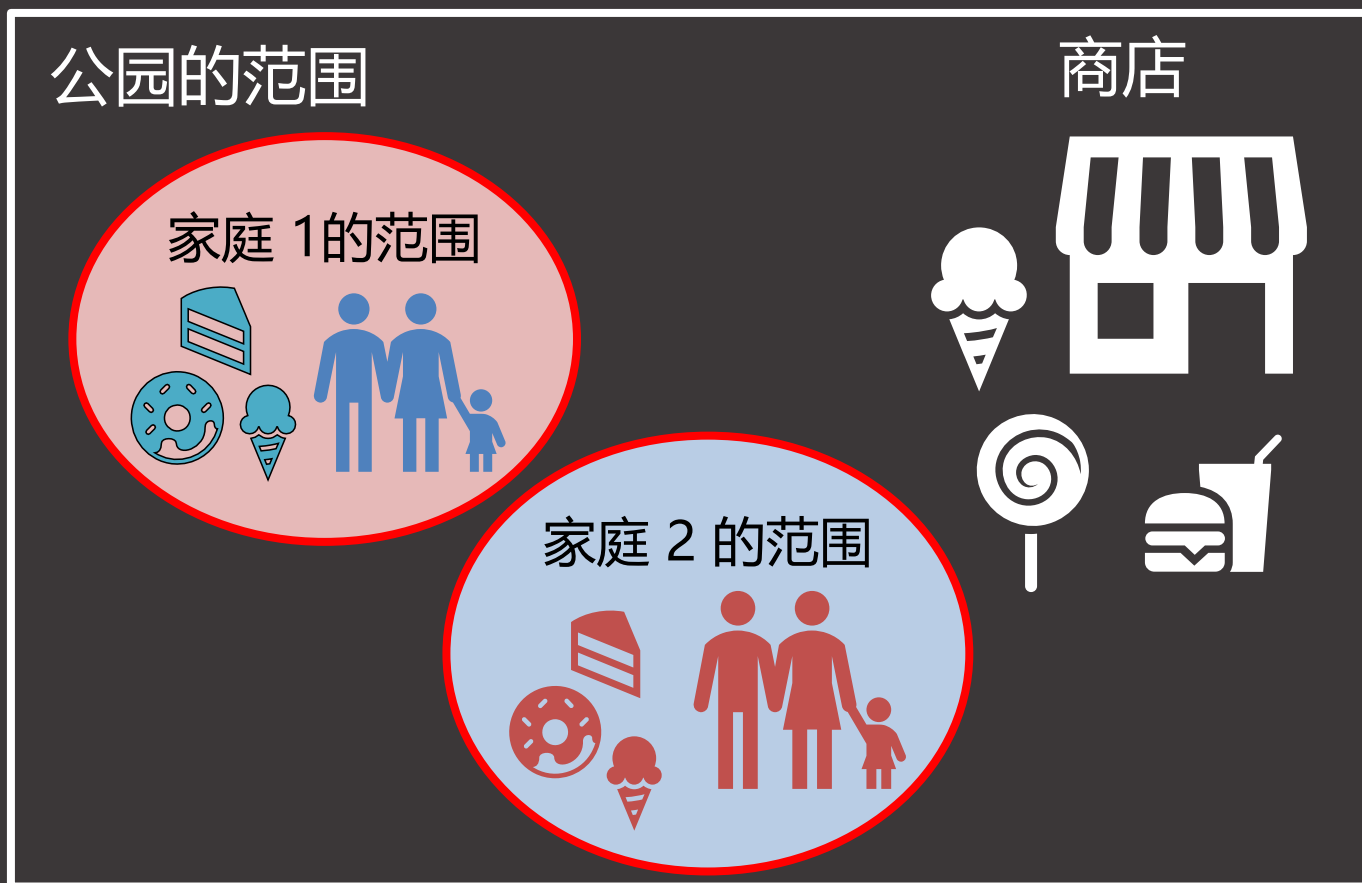
作用域

- 现实中：而公园里的商店，所有人都可以去买东西



作用域

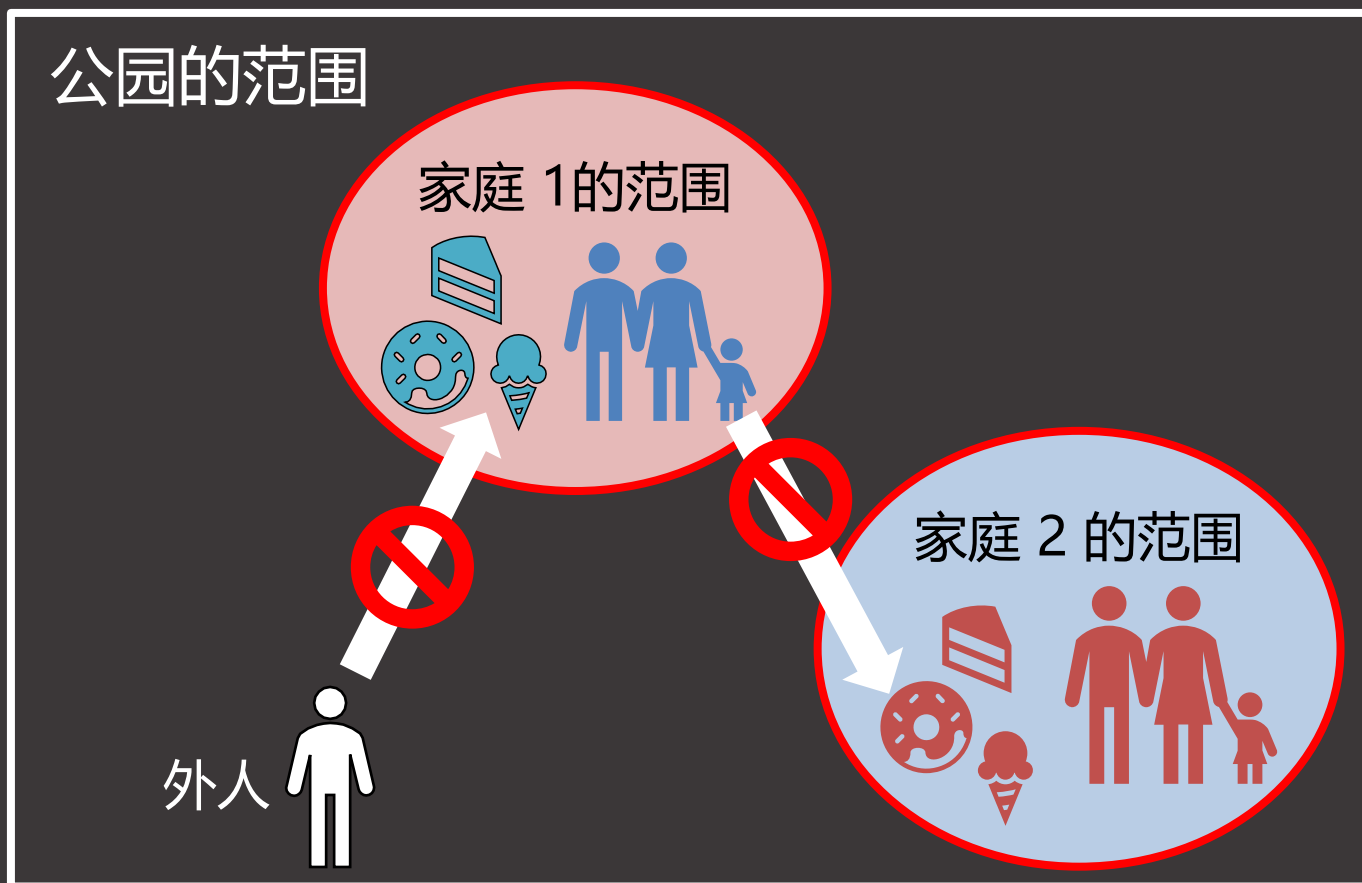
- 范围：
 - 自家带的吃的，仅限于家庭范围内享用
 - 公园提供的吃的，公园范围内的所有人都可以去买来吃



作用域

- 作用域英文原文为scope，本意就是“范围”的意思
- 作用域是指一个数据的可用范围。
- 定义作用域是为了避免不同范围内的数据之间互相干扰

知识讲解



作用域

- JavaScript程序中，包含两级作用域：
 - 1. 全局作用域，是指始终存在于内存中的，任何函数都可访问的公共区域。在浏览器中，这个区域称为window
 - 2. 函数作用域，是指仅限于每个函数内可访问的，每个函数私有的区域

全局作用域（全局范围） window

```
function 函数(){  
    这里是 函数作用域（函数范围）  
}
```



作用域

- 作用域本质是一块保存多个变量的存储空间。
- 全局变量，是指保存在全局作用域中的变量。
- 全局变量可在程序中的任何位置都能访问。
- 比如：下方程序中的变量“商店”

全局范围 window

var 商店 = “公园的商店”

function 函数(){ 函数范围
 去 商店 买东西
}

去 商店 买东西



使用全局变量

- 定义全局变量，并在函数内外，同时访问全局变量




作用域

- 局部变量，是指保存在某一个函数内的变量。
- 局部变量只能在函数内使用。
- 比如：下方程序中，函数范围内的变量“烤肠”

全局范围 window

```
function 函数(){ 函数范围  
    var 烤肠= “函数家的烤肠”  
    吃 烤肠  
}
```



吃 烤肠



使用局部变量

- 定义局部变量，并在函数内外，同时访问全局变量



作用域

- 现实中：游客不能住在公园里不走了！都是变化的。
- 应该是：
 - 游客进入公园，找地方野餐
 - 野餐完，游客离开时，带走本次野餐的垃圾
 - 下次来了，再重新找地方野餐。

公园

家庭 1



家庭 1



作用域

- 函数作用域不是一直都有的，其实也是动态变化的
 - 调用函数时，创建函数作用域，保存函数内声明的局部变量
 - 调用函数后，函数作用域的存储空间，被垃圾回收器释放

全局范围 window

//只是定义函数，但不执行，不创建变量“烤肠”

```
function 函数(){  
    var 烤肠= “自家带的烤肠”  
    吃 烤肠  
}
```

函数() //调用函数执行时

{
 烤肠= “自家带的烤肠”
}

函数作用域

烤肠= “自家带的烤肠”

实实在在的变量

作用域

- 因此：
 - 1. 函数调用后，在函数定义以外的地方访问不到局部变量
 - 2. 函数作用域内的局部变量不可反复使用，每次都重新创建新变量；

全局范围 window

```
function 函数() { //定义函数
    var 烤肠 = "自家带的烤肠"
    吃 烤肠
}
```

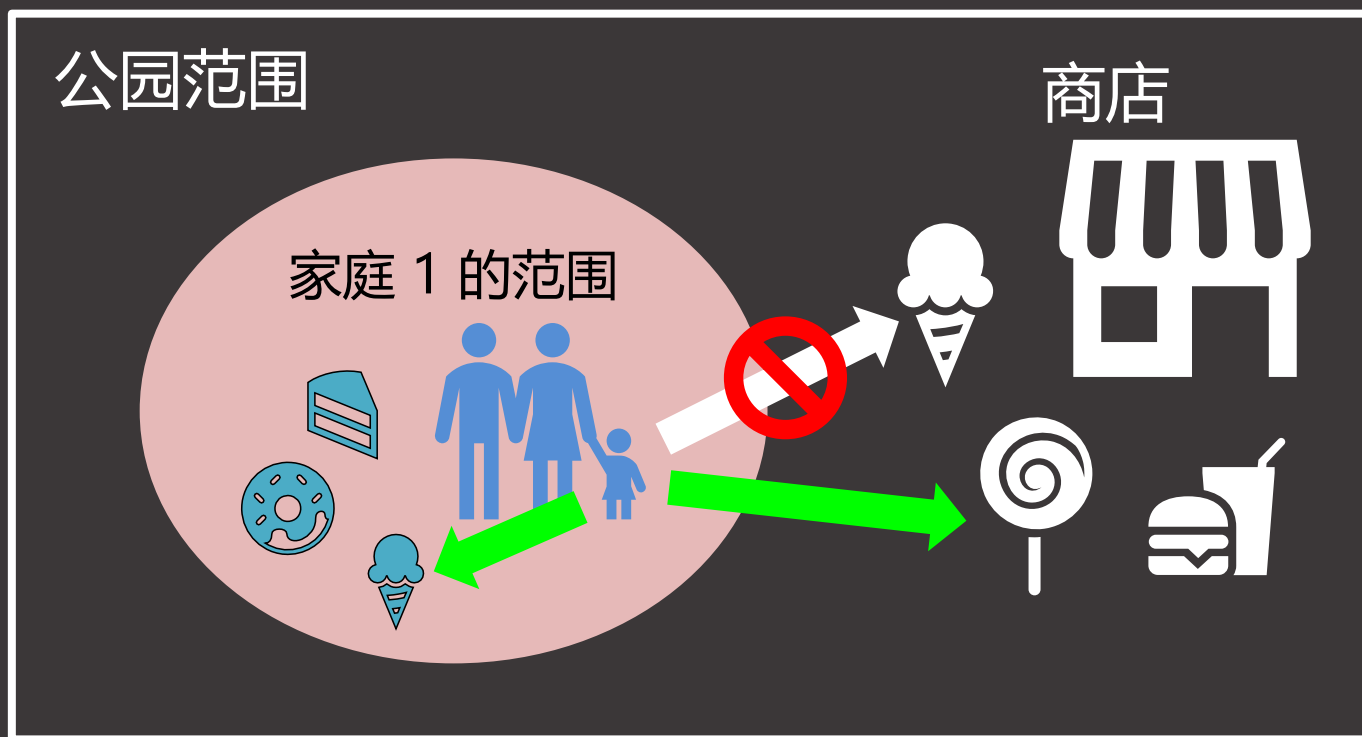
函数() //报错! "烤肠" 未定义

```
函数() //调用函数执行时
{
    烤肠 = "自家带的烤肠"
}
烤肠 = "自家带的烤肠"
```

第一次调用函数时
创建的新烤肠
第二次调用函数时
创建的新烤肠

作用域链 (scope chain)

- 现实中：
 - 孩子既可以吃自家带的东西，又可以去公园商店里买东西
 - 但通常只要自己家里有，就不去外部买。总是优先吃自己家带的。



作用域链 (scope chain)

- 作用域链，就是由多级作用域组成的链式结构
- 当调用函数时，函数会将自己能用到的所有作用域都用“锁链”串联（引用）起来。
- 作用域链：
 - 1. 串连着当前函数可使用的所有作用域范围，保存着当前函数可用的所有变量
 - 2. 控制着变量的使用顺序：先局部，后全局；只要局部有，就不去全局找。

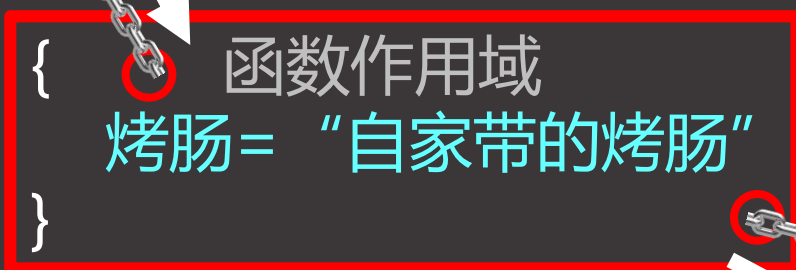


作用域链 (scope chain)

- 比如:

```
function 函数(){  
    var 烤肠= "自家带的烤肠"  
    吃 烤肠           //输出 "自家带的烤肠"  
    去 商店 买东西 //输出 "公园的商店"  
}
```

函数() //调用函数时，用锁链串联（引用）两级作用域



验证作用域链

- 同时定义全局变量和局部变量，其中，全局变量和局部变量中有部分变量重名
- 定义函数使用这些变量，观察使用顺序



作用域链

- 现实中：出游之前，就想好哪些东西需要自己带，哪些东西可以在公园买到

知识讲解



作用域链

- 定义函数时，函数非常悲观，怕以后找不到上级作用域，就已经引用了自己所处的上级作用域。

```
function 函数(){  
    var 烤肠= "自家带的烤肠"  
    吃 $烤肠  
    去 $商店 买东西  
} [[scopes]]
```



全局范围 window
\$商店= "公园的商店"
\$烤肠= "公园的烤肠"



作用域链

- 输出每个函数对象，就可看到[[scopes]]属性记录着函数可用的上级作用域：console.log(函数名)

```
▼ f 家庭1() ⓘ  
  arguments: null  
  ▼ [[Scopes]]: Scopes[1]  
    ▼ 0: Global  
      $商店: "公园的商店"  
      $烤肠: "公园的烤肠"
```

- 调用时
 - 先创建函数自有的函数作用域对象，保存局部变量
 - 再用刚创建的函数作用域对象，继续引用上级作用域对象，形成作用域链



闭包

- 全局变量和局部变量都有不可兼得的优缺点：
 - 全局变量：优点：可重用；缺点：易被篡改
 - 局部变量：缺点：不可重用；优点：不会被篡改
- 比如：定义一个函数，实现取号机效果：

```
var i=1; ?  
function 取号机(){  
    var i=1; ?  
    console.log( i++ )  
}
```

取号机();//1 取号机();//2 取号机();//3 取号机();//4



实现取号机函数

- 使用局部变量尝试实现取号机程序
- 使用全局变量实现取号机程序，并尝试中途篡改取号机程序



闭包

- 比如: 定义一个函数, 实现取号机效果:

```
var i=1; //极易被篡改  
function 取号机(){
```

```
    var i=1; //不可重用, 始终返回1  
    console.log( i++ )
```

```
}
```

取号机(); //1 取号机(); //2

i=1; //篡改全局变量, 后续都受影响

取号机(); //1 取号机(); //2

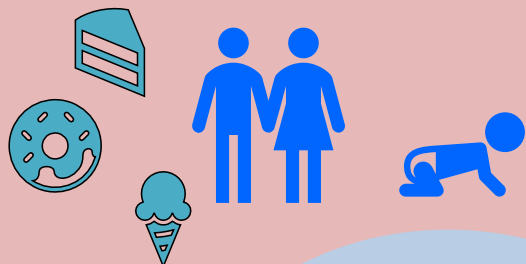


闭包

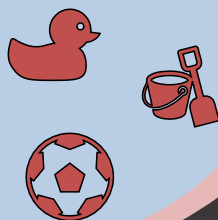
- 现实中：孩子要自己去公园

知识讲解

家里给准备的



孩子自己准备的
东西



公园已经有的商店

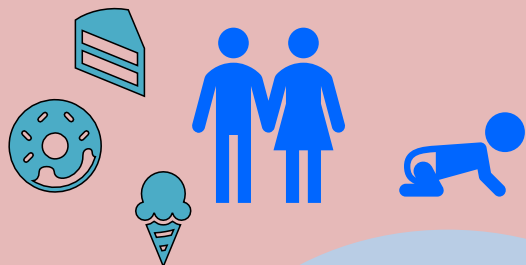


闭包

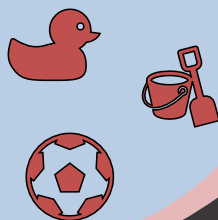
- 现实中：外人用不了家里给孩子准备的东西

知识讲解

家里给准备的



孩子自己准备的
东西



公园已经有的商店



外人



闭包

- 用外层函数包裹内层函数，可为内层函数准备专属的变量——也就是外层函数的局部变量

全局范围 window

商店= “公园的商店”
冰淇淋= “公园的冰淇淋”

```
function 家里(){  
    var 甜甜圈= “家里给的甜甜圈”  
    var 冰淇淋= “家里给的冰淇淋”
```

```
    function 孩子(){  
        var 玩具= “孩子自己的玩具”  
    }  
}
```



闭包

- 问题：外部要的是调用“孩子()”执行任务，所以爸妈应该把孩子送到外部

全局范围 window

```
商店 = “公园的商店”  
冰淇淋 = “公园的冰淇淋”
```

```
function 家里(){  
    var 甜甜圈 = “家里给的甜甜圈”  
    var 冰淇淋 = “家里给的冰淇淋”  
  
    return function 孩子(){  
        var 玩具 = “孩子自己的玩具”  
    }  
}
```

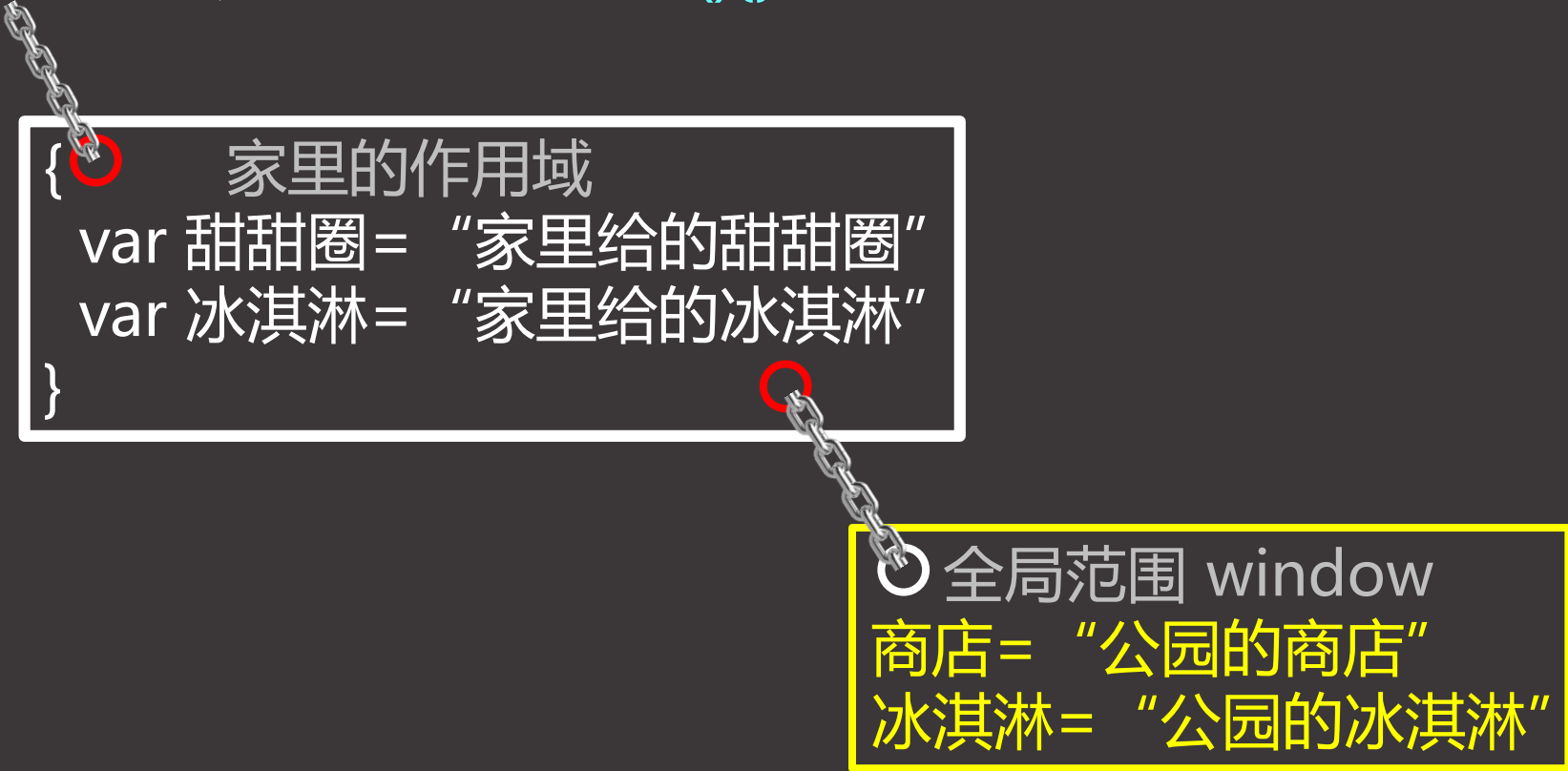
```
孩子 = 家里(); //孩子=function 孩子()
```



闭包

- 结果：内层函数还未调用，就保存了两级父级作用域

孩子 //等效于function 孩子(){}



```
{  
  家里的作用域  
  var 甜甜圈= "家里给的甜甜圈"  
  var 冰淇淋= "家里给的冰淇淋"  
}
```

```
全局范围 window  
商店= "公园的商店"  
冰淇淋= "公园的冰淇淋"
```



查看闭包内层函数的作用域

课堂练习




闭包

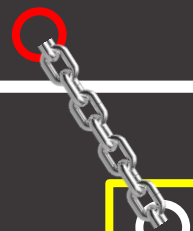
- 调用“孩子()”时，其实有三级作用域里的东西可用

孩子()

```
{ 孩子自己的作用域  
  var 玩具= “孩子自己的玩具”  
}
```



```
{ 家里的作用域  
  var 甜甜圈= “家里给的甜甜圈”  
  var 冰淇淋= “家里给的冰淇淋”  
}
```



```
○ 全局范围 window  
商店= “公园的商店”  
冰淇淋= “公园的冰淇淋”
```



闭包

- 闭包，是既重用变量又保证变量不被篡改的一种编程方法
- 今后，只要希望重用一個变量，又保证变量不会被篡改时，就要用闭包保护变量。
- 如何：3步
 - 1. 用外层函数包裹受保护的变量和内层函数
 - 2. 外层函数将内层函数返回到外部
 - 3. 调用者将外层函数返回的内层函数保存在全局变量中



使用闭包实现取号机函数

课堂练习



总结和答疑

