# 第九课 闭包

## 学习目录

- **匿名函数**
- **什么是闭包**
- **内存泄漏与垃圾回收机制**

## 一．匿名函数

匿名函数就是没有名字的函数，匿名函数既可以配合其他语法使用，也可以单独使用。

```
var btn = document.getElementById('btn');
btn.onclick = function(){}
```

**匿名函数自执行**

匿名函数自执行的调用方式是使用()将匿名函数包括起来，调用的话在后面加上一对圆括号，这个圆括号里也可以传递在匿名函数中要使用的参数。

```
(function(a,b){
    console.log(a + b);
    console.log('匿名函数自执行');
})(100,200);
```

**模仿块级作用域**

很多时候我们做逻辑封装的时候，既希望有些变量可以被使用者访问，有些变量不想被

访问只想作为私有变量来在封装逻辑的内部使用 ,这就可以用到匿名函数自执行的特点来做到，这也是模仿块级作用域的方法。

1. 通过一个变量来接收匿名函数自执行的结果。
2. 访问这个匿名函数返回的允许被外部使用的变量和方法。

```
var addNum = (function(num1,num2){
    var num3 = 3;
    function _addFn(){
        return num1 + num2;
    }
    function addMore(){
        return _addFn() * num3;
    }
    return {
        num3:num3,
        getResult:addMore
    };

})(100,200);

console.log(addNum.num3);
console.log(addNum.getResult());
```

## 二．什么是闭包

闭包的官方定义是指函数和对其周围状态的引用捆绑在一起构成闭包。

实际上闭包最通俗的解释可以理解成一个可以访问另一个函数内部变量的函数。

从开发角度也就是说，闭包可以让你从内部函数访问外部函数作用域，就是在一个函数内部创建另一个函数，通过另一个函数访问这个函数的局部变量。

**函数作为返回值**

```
function fn1(){
    var a = 100;
    return function(){
        var b = 200;
        console.log(a + b);//300
    }
}
var f1 = fn1();
f1();
```

**函数作为参数**

```
function fn1(a){
    var a = 300;
    var b = 200;
    console.log(a + b);//500
```

```
}

(function(fn){
    var a = 100;
    fn(a);
})(fn1);
```

执行 fn1()时，返回的是一个函数，这个返回的函数也可以创建一个独立的局部作用域，另一方面是，在返回的这个函数内部中，还有变量 a，这个变量 a 是引用自 fn1 作用域下的 a，所以为了保证这个变量能够继续被返回的新函数继续引用，因此，这个 a 不会被销毁，一旦销毁 a 的引用就会报错了。因此，调用 fn1()形成执行环境不会被销毁，还依然存在与执行栈中，也就是说变量 a 会一直在内存中保留。

因此闭包的第一个问题是会增加内存开销，不过闭包的优点也显而易见，因为它允许将函数与其所操作的外部环境数据关联起来，这种关联关系有利于变量的持久化缓存，可以加快有些变量的访问速度，同时这种关联性也类似于面向对象编程。在面向对象编程中，对象允许我们将这个对象的某些属性和方法与一个或者多个方法相关联。

## 谢谢观看！

我是星星课堂老师：周小周