我们都知道，js的作用域分两种，全局和局部，基于我们所熟悉的作用域链相关知识，我们知道在js作用域环境中访问变量的权利是由内向外的，内部作用域可以获得当前作用域下的变量并且可以获得当前包含当前作用域的外层作用域下的变量，反之则不能，也就是说在外层作用域下无法获取内层作用域下的变量，同样在不同的函数作用域中也是不能相互访问彼此变量的，**那么我们想在一个函数内部也有限权访问另一个函数内部的变量该怎么办呢？闭包就是用来解决这一需求的，闭包的本质就是在一个函数内部创建另一个函数。**

我们首先知道闭包有3个特性：

①函数嵌套函数

②函数内部可以引用函数外部的参数和变量

③参数和变量不会被垃圾回收机制回收

函数作为返回值:

```
function a() {
    var name = 'dov';
    return function() {
        return name;
    }
}

var b = a();
b(); // dov
```

闭包经典例子：

```
function fn() {
    var num = 3;
    return function () {
        var n = 0;
        console.log(++n);
        console.log(++num);
    }
}
var fn1 = fn();
fn1(); 1 4
fn1(); 1 5
```

定时器与闭包

```
for(var i = 0; i < 5; i++) {
    setTimeout(function() {
        console.log(i + '');
    },1000);
}
5 //打印5次
```

两种解法：

1、将var改为let

2、引入闭包

```
for(var i = 0; i < 5; i++) {
    (function() {
        setTimeout({
            console.log(i);
        },i*100)
    })(i);
}
```

在这段代码中，相当于同时启动3个定时器，i*100是为4个定时器分别设置了不同的时间，同时启动，但是执行时间不同，每个定时器间隔都是100毫秒，实现了每隔100毫秒就执行一次打印的效果。

```
var num = 15;
function fn1(){
    if(x > num) {
        console.log(x);
    }
}
(function(fn2) {
    var num = 100;
    fn2(30)
})(fn2)
30
```

在这段代码中，函数fn1作为参数传入立即执行函数中，在执行到fn2(30)的时候，30作为参数传入fn1中，这时候if(x>num)中的num取的并不是立即执行函数中的num，而是取创建函数的作用域中的num这里函数创建的作用域是全局作用域下，所以num取的是全局作用域中的值15，即30>15，打印30

好处

①保护函数内的变量安全 ，实现封装，防止变量流入其他环境发生命名冲突
②在内存中维持一个变量，可以做缓存（但使用多了同时也是一项缺点，消耗内存）
③匿名自执行函数可以减少内存消耗

坏处

①其中一点上面已经有体现了，就是被引用的私有变量不能被销毁，增大了内存消耗，造成内存泄漏，解决方法是可以在使用完变量后手动为它赋值为null；
②其次由于闭包涉及跨域访问，所以会导致性能损失，我们可以通过把跨作用域变量存储在局部变量中，然后直接访问局部变量，来减轻对执行速度的影响