全局变量和局部变量

变量作用域

- JavaScript是函数级作用域编程语言：变量只在其定义时所在的function内部有意义。

```
function fun() {
    var a = 10;
}
fun();
console.log(a); // 报错
```

变量a是在fun函数中被定义的，所以变量a只在fun函数内部有定义，fun函数就是a的“作用域”，变量a被称为局部变量。

全局变量

- 如果不将变量定义在任何函数的内部，此时这个变量就是全局变量，它在任何函数内都可以被访问和更改。

```
var a = 10;

function fun() {
    a++;
    console.log(a); //  输出11
}
fun();
console.log(a);     // 输出11
```

变量a没有定义在任何函数内部，它是“全局变量”

遮蔽效应

- 如果函数中也定义了和全局同名的变量，则函数内的变量会将全局的变量“遮蔽”

```
var a = 10;

function fun() {
    var a = 5;
    a++;
    console.log(a); //  输出6
}
fun();
console.log(a);     // 输出10
```

局部变量a将全局变量a“遮蔽”了

改变的是局部变量a的值，不影响全局变量

注意考虑变量声明提升的情况

- 这个程序的运行结果是什么呢？

```
var a = 10;

function fun() {
    a++;
    var a = 5;
    console.log(a);  //  输出5
}
fun();
console.log(a);      //  输出10
```

局部变量a被自增1，a此时是undefined，自增1结果是NaN

重新将a赋值为5

形参也是局部变量

- 这个程序的运行结果是什么呢？

```
var a = 10;

function fun(a) {
    a++;
    console.log(a);  //  输出8
}
fun(7);
console.log(a);      //  输出10
```

形参a也是函数内部的局部变量

作用域链

- 先来认识函数的嵌套：一个函数内部也可以定义一个函数。和局部变量类似，定义在一个函数内部的函数是局部函数。

```
function fun() {
    function inner() {
        console.log('你好');
    }
    inner();        //  调用内部函数
}

fun();              //  调用外部函数
```

局部函数

作用域链

- 在函数嵌套中，变量会从内到外逐层寻找它的定义。

```
var a = 10;
var b = 20;
function fun() {
    var c = 30;
    function inner() {
        var a = 40;
        var d = 50;
        console.log(a, b, c, d);
    }
    inner();
}
fun();
```

使用变量时，JS会从当前层开始，逐层向上寻找定义

不加var将定义全局变量

- 在初次给变量赋值时，如果没有加var，则将定义全局变量。

```
function fun() {
    a = 3;
}
fun();
console.log(a); // 3
```

闭包

从一个题目开始

- 这个程序的运行结果是什么呢？

```
function fun() {
    var name = '慕课网';

    function innerFun() {
        alert(name);
    }
    return innerFun;
}

var inn = fun();
inn();   // 弹出慕课网
```

返回了内部函数

内部函数被移动到了**外部**执行

什么是闭包

- JavaScript中函数会产生闭包（closure）。闭包是函数本身和该函数声明时所处的环境状态的组合。

什么是闭包

- 函数能够“记忆住”其定义时所处的环境，即使函数不在其定义的环境中被调用，也能访问定义时所处环境的变量。

```
function fun() {
    var name = '慕课网';

    function innerFun() {
        alert(name);
    }
    return innerFun;
}

var inn = fun();
inn();
```

观察闭包现象

- 在JavaScript中，每次创建函数时都会创建闭包。

- 但是，闭包特性往往需要将函数“换一个地方”执行，才能被观察出来。

闭包非常实用

- 闭包很有用，因为它允许我们将数据与操作该数据的函数关联起来。这与“面向对象编程”有少许相似之处。

- 闭包的功能：记忆性、模拟私有变量。

闭包用途1 - 记忆性

- 当闭包产生时，函数所处环境的状态会始终保持在内存中，不会在外层函数调用后被自动清除。这就是闭包的记忆性。

闭包的记忆性举例

- 创建体温检测函数checkTemp(n)，可以检查体温n是否正常，函数会返回布尔值。

- 但是，不同的小区有不同的体温检测标准，比如A小区体温合格线是37.1℃，而B小区体温合格线是37.3℃，应该怎么编程呢？

闭包用途2 - 模拟私有变量

- 题目：请定义一个变量a，要求是能保证这个a只能被进行指定操作（如加1、乘2），而不能进行其他操作，应该怎么编程呢？

- 在Java、C++等语言中，有私有属性的概念，但是JavaScript中只能用闭包来模拟。

使用闭包的注意点

- 不能滥用闭包，否则会造成网页的性能问题，严重时可能导致内存泄露。所谓内存泄漏是指程序中己动态分配的内存由于某种原因未释放或无法释放。

- 所谓内存泄漏是指程序中己动态分配的内存由于某种原因未释放或无法释放。

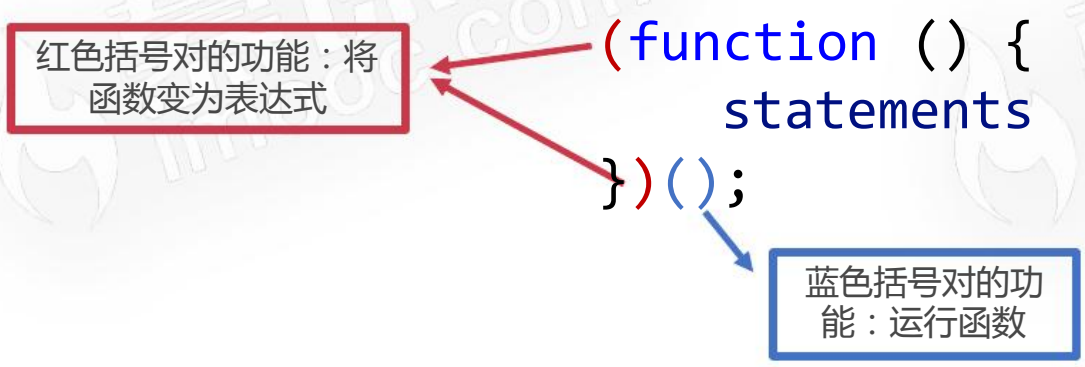
闭包的一道面试题

```
function addCount() {
    var count = 0;
    return function () {
        count = count + 1;
        console.log(count);
    };
}
var fun1 = addCount();
var fun2 = addCount();
fun1();
fun2();
fun2();
fun1();
```

立即执行函数IIFE

什么是IIFE

- IIFE（Immediately Invoked Function Expression，立即调用函数表达式）是一种特殊的JavaScript函数写法，一旦被定义，就立即被调用。



形成IIFE的方法

- 函数不能直接加圆括号被调用。

```
function() {
    alert(1)
}();
```

- 函数必须转为“函数表达式”才能被调用。

```
(function() {
    alert(1)
})();
```

```
+function() {
    alert(1)
}();
```

```
-function() {
    alert(1)
}();
```

IIFE的作用1 - 为变量赋值

- 为变量赋值：当给变量赋值需要一些较为复杂的计算时（如if语句），使用IIFE显得语法更紧凑。

IIFE的作用1 - 为变量赋值

```
var age = 12;
var sex = '男';
var title = (function () {
    if (age < 18) {
        return '小朋友'
    } else {
        if (sex == '男') {
            return '先生';
        } else {
            return '女士';
        }
    }
})();
```

IIFE

IIFE的作用2

- 先来看一个题目：

```
var arr = [];
for (var i = 0; i < 5; i++) {
    arr.push(function () {
        alert(i);
    });
}
arr[2]();   // 弹出5
```

变量i是全局变量，所有函数都共享内存中的同一个变量i

IIFE的作用 - 将全局变量变为局部变量

- IIFE可以在一些场合（如for循环中）将全局变量变为局部变量，语法显得紧凑。

```
var arr = [];
for (var i = 0; i < 5; i++) {
    (function (i) {
        arr.push(function () {
            alert(i);
        });
    })(i);
}
arr[2]();   // 弹出2
```

谢谢