> 作用域、变量提升的知识点，面试时会经常遇到。

作用域（Scope）的概念

- **概念**：通俗来讲，作用域是一个变量或函数的作用范围。作用域在**函数定义**时，就已经确定了。
- **目的**：为了提高程序的可靠性，同时减少命名冲突。

作用域的分类

在 JS 中，一共有两种作用域：（ES6 之前）

- 全局作用域：作用于整个 script 标签内部，或者作用域一个独立的 JS 文件。
- 函数作用域（局部作用域）：作用于函数内的代码环境。

作用域的访问关系

在内部作用域中可以访问到外部作用域的变量，在外部作用域中无法访问到内部作用域的变量。

代码举例：

```
var a = 'aaa';
function foo() {
    var b = 'bbb';
    console.log(a); // 打印结果：aaa。说明 内层作用域 可以访问 外层作用域 里的变量
}

foo();
console.log(b); // 报错：Uncaught ReferenceError: b is not defined。说明 外层作用域 无法访问 内层作用域 里的变量
```

变量的作用域

根据作用域的不同，变量可以分为两类：全局变量、布局变量。

全局变量：

- 在全局作用域下声明的变量，叫「全局变量」。在全局作用域的任何一地方，都可以访问这个变量。
- 在全局作用域下，使用 var 声明的变量是全局变量。
- 特殊情况：在函数内不使用 var 声明的变量也是全局变量（不建议这么用）。

局部变量：

- 定义在函数作用域的变量，叫「局部变量」。
- 在函数内部，使用 var 声明的变量是局部变量。
- 函数的**形参**也是属于局部变量。

从执行效率来看全局变量和局部变量：

- 全局变量：只有浏览器关闭时才会被销毁，比较占内存。
- 局部变量：当其所在的代码块运行结束后，就会被销毁，比较节约内存空间。

作用域的上下级关系

当在函数作用域操作一个变量时，它会先在自身作用域中寻找，如果有就直接使用（**就近原则**）。如果没有则向上一级作用域中寻找，直到找到全局作用域；如果全局作用域中依然没有找到，则会报错ReferenceError。

在函数中要访问全局变量可以使用window对象。（比如说，全局作用域和函数作用域都定义了变量a，如果想访问全局变量，可以使用`window.a`）

全局作用域

直接编写在script标签中的JS代码，都在全局作用域。

- 全局作用域在页面打开时创建，在页面关闭时销毁。
- 在全局作用域中有一个全局对象window，它代表的是一个浏览器的窗口，由浏览器创建，我们可以直接使用。

在全局作用域中：

- 创建的**变量**都会作为window对象的属性保存。比如在全局作用域内写`var a = 100`，这里的`a`等价于`window.a`。
- 创建的**函数**都会作为window对象的方法保存。

变量的声明提前（变量提升）

使用var关键字声明的变量（比如`var a = 1`），**会在所有的代码执行之前被声明**（但是不会赋值），但是如果声明变量时不是用var关键字（比如直接写`a = 1`），则变量不会被声明提前。

举例1：

```
    console.log(a);
    var a = 123;
```

打印结果：undefined。注意，打印结果并没有报错，而是 undefined，说明变量 a 被提前声明了，只是尚未被赋值。

举例2：

```
    console.log(a);
    a = 123;   //此时a相当于window.a
```

程序会报错：`Uncaught ReferenceError: a is not defined`。

举例3：

```
    a = 123;   //此时a相当于window.a
    console.log(a);
```

打印结果：123。

举例4：

```
foo();

function foo() {
    if (false) {
        var i = 123;
    }
    console.log(i);
}
```

打印结果：undefined。注意，打印结果并没有报错，而是 undefined。这个例子，再次说明了：变量 i 在函数执行前，就被提前声明了，只是尚未被赋值。

函数的声明提前

函数声明：

使用`函数声明`的形式创建的函数`function foo(){}`，**会被声明提前。**

也就是说，整个函数会在所有的代码执行之前就被**创建完成。**所以，在代码顺序里，我们可以先调用函数，再定义函数。

代码举例：

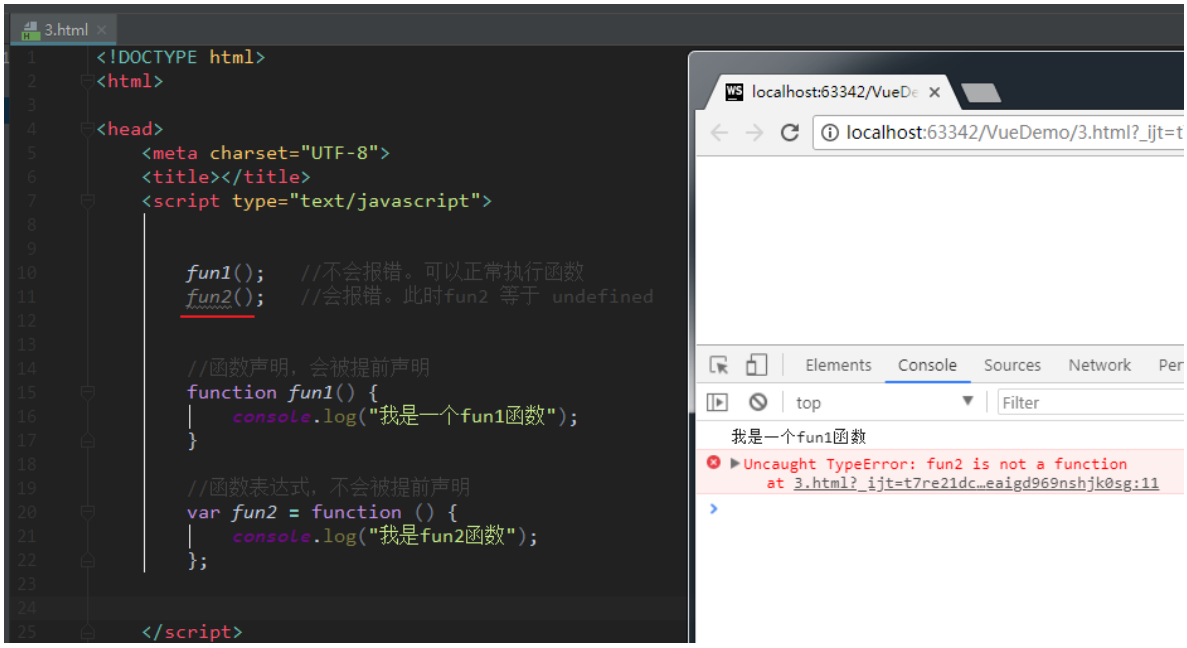
```
    fn1();  // 虽然 函数 fn1 的定义是在后面，但是因为被提前声明了， 所以此处可以调用函数

    function fn1() {
        console.log('我是函数 fn1');
    }

```

函数表达式：

使用`函数表达式`创建的函数`var foo = function(){}`，**不会被声明提前**，所以不能在声明前调用。

很好理解，因为此时foo被声明了（这里只是变量声明），且为undefined，并没有把`function(){}`赋值给 foo。

所以说，下面的例子，会报错：



函数作用域

提醒1：在函数作用域中，也有声明提前的特性：

- 函数中，使用var关键字声明的变量，会在函数中所有的代码执行之前被声明。
- 函数中，没有var声明的变量都是**全局变量**，而且并不会提前声明。

举例：

```
    var a = 1;

    function foo() {
        console.log(a);
        a = 2;      // 此处的a相当于window.a
    }

    foo();
    console.log(a);    //打印结果是2

```

上方代码中，执行foo()后，函数里面的打印结果是`1`。如果去掉第一行代码，执行foo()后，函数里面的打印结果是`Uncaught ReferenceError: a is not defined`。

提醒2：定义形参就相当于在函数作用域中声明了变量。

```
    function fun6(e) { // 这个函数中，因为有了形参 e，此时就相当于在函数内部的第一行代码里，写了 var e;
        console.log(e);
    }

    fun6();  //打印结果为 undefined
    fun6(123);//打印结果为123
```

JavaScript 没有块级作用域（ES6之前）

在其他编程语言中（如 Java、C#等），存在块级作用域，由`{}`包括起来。比如在 Java 语言中，if 语句里创建的变量，只能在if语句内部使用：

```
if(true){
    int num = 123;
    system.out.print(num); // 123
}
system.out.print(num); // 报错
```

但是，在 JS 中没有块级作用域（ES6之前）。举例如下：

```
if(true){
var num = 123;
    console.log(123); //123
}

console.log(123); //123（可以正常打印）

```

作用域链

引入：

- 只要是代码，就至少有一个作用域
- 写在函数内部的局部作用域
- 如果函数中还有函数，那么在这个作用域中就又可以诞生一个作用域

基于上面几条内容，我们可以得出作用域链的概念。

作用域链：内部函数访问外部函数的变量，采用的是链式查找的方式来决定取哪个值，这种结构称之为作用域链。查找时，采用的是**就近原则**。

代码举例：

```
var num = 10;

function fn() {
    // 外部函数
    var num = 20;

    function fun() {
        // 内部函数
        console.log(num);
    }
    fun();
}
fn();

```

打印结果：20。