JavaScript 解析机制

1.预解析概念

在当前作用域下,js 运行之前，会把带有 var 和 function 关键字的事先声明，并在内存中安排好。（这个过程也可以理解为变量提升）然后再从上到下执行 js 语句。预解析只会发生在通过 var 定义的变量和 function 上。

2.var

使用 var 定义的变量预解析 :告诉解析器知道有这个名字的存在并默认将该变量赋值 undefined ，如下：

```
console.log(x) //undefined
var x=15;
```

变量 x 虽然是在 console.log 后面定义的 ,但是使用 var 申明的 x 会提前保存在内存中，并赋值 undefined ，然后再从上往下执行 js 语句 。它的执行顺序类似于下面的结构：

```
var x;
console.log(x);//undefined
x=15;
```

先声明了 x ,x 没有定义赋值为 undefined ，输出的结果自然为 undefined 。然后再给 x 赋值为 15。

需要注意的是，如果变量声明没有使用 var ,不存在变量提升的。如下：

```
console.log(x) //error:x is not defined
x=15;
```

x 没有使用 var 声明，所以报错找不到 x。

3.function

使用 function 定义函数的预解析：先告诉解析器这个函数名的存在，然后在告诉解析器这个函数名的函数体是什么 。如下：

```
console.log(xx)
function xx(){
    return "整个函数都会提升到最前面"
}
```

声明函数会把整个函数都提升到最前面 ,所以结果会输出整个函数 ,结果如下：

```
f xx(){
        return "整个函数都会提升到最前面"
    }
```

如果在一个函数作用域中声明一个变量 ，那么它也会提升到函数作用域的最上面，如下：

```
var a=1;
function xx(){
    console.log(a); //undefined
    var a=2;
}
xx();
```

以上虽然全局作用域声明了一个变量 a ,但是函数里面也声明了一个变量 a ,所以会先查找函数里面是否有变量 a，如果有的话就不会再全局下查找了。函数里面

的变量 a 会被提升到函数作用域的最前面 ，并且赋值为 undefined，所以输出结果为 undefined ，类似于如下结构：

```
var a=1;
function xx(){
    var a;
    console.log(a); //undefined
    a=2;
}
xx();
```

函数的参数也可以理解为函数作用域的变量 ，如下：

```
var a=1;
function xx(a){
    console.log(a); //undefined
}
xx();
console.log(a); //1
```

为函数 xx 传递一个形参 a ,由于函数在调用时没有传递实参 （也就是说变量 a 没有赋值） ,所以为 undefined 。而在全局下输出 a 自然在全局下查找变量 a ，结果为 1。

4.变量或函数覆盖

如果在同一个作用域下声明两个相同的变量或者函数 ,那么后一个会覆盖前一个。如下：

```
var a=1;
var a=2;
console.log(a) //2
```

```
function x(){
  console.log("xx")
}

function x(){
  console.log("xxx")
}

x() //xxx
```

如果声明的函数与变量名字相同 ，那又会怎么覆盖呢？可以看如下例子：

```
var m=1;
function m(){
    console.log("11")
}
m(); //error: m is not a function
```

JavaScript 中 ，函数的预解析优先级是要高于变量的预解析的。无论函数在什么位置声明 ，都优选把整个函数提升到最前面。所以上面的例子中 ，虽然函数 m 是在变量 m 下面定义的 ,但是在预解析时先解析函数 m ,然后再解析变量 m , 后面的变量 m 会把前面的函数 m 覆盖，最后 m 为 1 为数值类型 ，所以调用 m 时报错 ，m 不是一个函数。

需要注意的是 ，如果变量 m 定义后没有赋值 ，那么函数就不会被覆盖了，如下：

```
var m;
function m(){
    console.log("11")
}
m(); //11
```

掌握以上知识，就不难理解课程中的讲的的例子了 ，如下：

```
console.log(a); //function a(){console.log(4);}
var a=1;
console.log(a); //1
function a(){
    console.log(2)
}
console.log(a); //1
var a=3;
console.log(a); //3
function a(){
    console.log(4);
}
console.log(a); //3
a();  //error:  a is not a function
```

以上例子，两个函数 a 优先提升 ，所以第二个函数 a 覆盖第一个函数 a。然后两个变量 a 提升，由于变量 a 提升后为 undefined，所以第二个函数没有被覆盖 ，第一个输出 a 结果为第二个函数 function a(){console.log(4);}。随后 a 被赋值为 1 ，所以第二个输出 a 结果为 1。因为第一个函数 a 已经被提升到前面去了，所以第三个输出 a 结果还是 1。随后为 a 赋值为 3，所以第四，第五输出 a 结果为 3。最后调用 a,a 因为是数值类型，所以会报错 a 不是一个函数。