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 声明的变量和 function 声明的函数上。

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 声明的 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如下：

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

先声明了 x ,x 没有赋值，默认赋值为 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，然后再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，如果有的话就不会在全局下查找了。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：

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

x() //xx
```

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

如上中，先声明变量 m ，然后声明函数 m。函数 m 会把变量 m 覆盖，所以调用 m 输出“11”。

继续看如下例子：

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

如上代码中，也是先声明变量 m ，然后声明函数 m， 函数 m 会把变量 m 覆盖。需要注意的是，变量赋值不会进行预解析。代码执行阶段，函数 m 重新赋值为 1 。即调用的时候 m 已经不是一个函数了，会报错。代码相当于如下：

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

m=1;

m();// error:m is not a function
```

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
```

如上代码中,变量 a 与函数 a 依次预解析。function a() {console.log(4)} 是最后声明的 ,所以会把前面都覆盖 ,第一次输出的 function a() {console.log(4)}。后面给函数 a 重新赋值为 1 ，第二次输出 1 。第二次输出后面是一个函数，函数已经提前声明了，这里不会执行它，则第三次输出 a 还是 1 。紧跟着后面重新赋值为 3，所以第四次输出为 3。 后面又是一个函数，也不会执行。所以第五次输出还是 3。 最后调用 a，由于 a 是一个数字，调用它会报错“a 不是一个函数”。