JavaScript 运行三部曲

- 语法分析
- 预编译
- 解释执行

预编译前奏

> 在讲预编译前，我们先来普及下面两个规律。

两个规律

规律1：任何变量，如果未经声明就赋值，此变量是属于 window 的属性，而且不会做变量提升。（注意，无论在哪个作用域内赋值）

比如说，如果我们直接在代码里写 `console.log(a)`，这肯定会报错的，提示找不到 `a`。但如果我直接写 `a = 100`，这就不会报错，此时，这个 `a` 就是 `window.a`。

规律2：一切声明的全局变量，全是window的属性。（注意，我说的是在全局作用域内声明的全局变量，不是说局部变量）

比如说，当我定义 `var a = 200` 时，这此时这个 `a` 就是 `window.a`。

由此，我们可以看出：**window 代表了全局作用域**（是说「代表」，没说「等于」）。

举例

掌握了上面两句话之后，我们再来看看下面的例子。

```
function foo() {
    var a = b = 100; // 连续赋值
}

foo();

console.log(window.b); // 在全局范围内访问 b
console.log(b); // 在全局范围内访问 b，但是前面没有加 window 这个关键字

console.log(window.a); // 在全局范围内访问 a
console.log(a); // 在全局范围内访问 a，但是前面没有加 window 这个关键字

```

上方代码的打印结果：

```
100
100
undefined
（会报错，提示 Uncaught ReferenceError: a is not defined）

```

解释：

当执行了`foo()`函数之后，`var a = b = 100`这行**连续赋值**的代码等价于`var a = (b = 100)`，其执行顺序是：

（1）先把 100 赋值给 b；

（2）再声明变量 a；

（3）再把 b 的值赋值给 a。

我们可以看到，b 是未经声明的变量就被赋值了，此时，根据规律1，这个 b 是属于`window.b`；而 a 的作用域仅限于 foo() 函数内部，不属于 window。所以也就有了这样的打印结果。

预编译

函数预编译的步骤

> 函数预编译，发生在函数执行的前一刻。

（1）创建AO对象。AO即 Activation Object 活跃对象，其实就是「执行期上下文」。

（2）找形参和变量声明，将形参名和变量作为 AO 的属性名，值为undefined。

（3）将实参值和形参统一，实参的值赋给形参。

（4）查找函数声明，函数名作为 AO 对象的属性名，值为整个函数体。

这个地方比较难理解。但只有了解了函数的预编译，才能理解明白函数的执行顺序。

代码举例：

```
function fn(a) {
    console.log(a);
    var a = 666;
    console.log(a);
    function a() {}
    console.log(a);
    var b = function() {};
    console.log(b);
    function c() {}
}

fn(1);
```

打印结果：

```
ƒ a() {}
666
666
ƒ () {}
```

参考链接

- JavaScript预编译原理分析：https://blog.csdn.net/q1056843325/article/details/52951114
- https://segmentfault.com/a/1190000018001871
- 预编译及变量提升：https://juejin.im/post/5aa6693df265da23884cb571
- https://juejin.im/post/5adaf8215188256712781830
- https://www.qqzmly.com/archives/1521

- 宏任务&微任务相关：https://segmentfault.com/a/1190000018134157