从本质上理解JavaScript中的变量提升

> 链接：https://juejin.cn/post/6844903895341219854#comment

JavaScript中奇怪的一点是你可以在变量和函数声明之前使用它们。就好像是变量声明和函数声明被**提升**了代码的顶部一样。

```
sayHi() // Hi there!

function sayHi() {
    console.log('Hi there!')
}

name = 'John Doe'
console.log(name)   // John Doe
var name
复制代码
```

然而JavaScript并不会移动你的代码，所以JavaScript中"变量提升"并不是真正意义上的"提升"。

JavaScript是单线程语言，所以执行肯定是按顺序执行。但是并不是逐行的分析和执行，而是一段一段地分析执行，会先进行编译阶段然后才是执行阶段。

在编译阶段阶段，代码真正执行前的几毫秒，会检测到所有的变量和函数声明，所有这些函数和变量声明都被添加到名为Lexical Environment的JavaScript数据结构内的内存中。所以这些变量和函数能在它们真正被声明之前使用。

函数提升

```
sayHi() // Hi there!

function sayHi() {
    console.log('Hi there!')
}
复制代码
```

因为函数声明在编译阶段会被添加到词法环境（Lexical Environment）中，当JavaScript引擎遇到`sayHi()`函数时，它会从词法环境中找到这个函数并执行它。

```
lexicalEnvironment = {
  sayHi: < func >
}
复制代码
```

var变量提升

```
console.log(name)   // 'undefined'
var name = 'John Doe'
console.log(name)   // John Doe
复制代码
```

上面的代码实际上分为两个部分：

- `var name`表示声明变量`name`
- `= 'John Doe'`表示的是为变量`name`赋值为'John Doe'。

```
var name    // 声明变量
name = 'John Doe' // 赋值操作
复制代码
```

只有声明操作`var name`会被提升，而赋值这个操作并不会被提升，但是为什么变量`name`的值会是`undefined`呢?

原因是当JavaScript在编译阶段会找到`var`关键字声明的变量会添加到词法环境中，并初始化一个值`undefined`，在之后执行代码到赋值语句时，会把值赋值到这个变量。

```
// 编译阶段
lexicalEnvironment = {
  name: undefined
}

// 执行阶段
lexicalEnvironment = {
  name: 'John Doe'
}
复制代码
```

所以函数表达式也不会被"提升"。`helloWorld`是一个默认值是`undefined`的变量，而不是一个`function`。

```
helloWorld();  // TypeError: helloWorld is not a function

var helloWorld = function(){
  console.log('Hello World!');
}
复制代码
```

let & const提升

```
console.log(a)  // ReferenceError: a is not defined
let a = 3
复制代码
```

为什么会报一个`ReferenceError`错误，难道`let`和`const`声明的变量没有被"提升"吗?

事实上所有的声明（function, var, let, const, class）都会被“提升”。但是只有使用`var`关键字声明的变量才会被初始化`undefined`值，而`let`和`const`声明的变量则不会被初始化值。

只有在执行阶段JavaScript引擎在遇到他们的词法绑定(赋值)时，他们才会被初始化。这意味着在JavaScript引擎在声明变量之前，无法访问该变量。这就是我们所说的**Temporal Dead Zone**，即变量创建和初始化之间的时间跨度，它们无法访问。

如果JavaScript引擎在`let`和`const`变量被声明的地方还找不到值的话，就会被赋值为`undefined`或者返回一个错误(`const`的情况下)。

举例：

```
let a
console.log(a)  // undefined
a = 5
```
复制代码

在编译阶段，JavaScript引擎遇到变量`a`并将它存到词法环境中，但因为使用`let`关键字声明的，JavaScript引擎并不会为它初始化值，所以在编译阶段，此刻的词法环境像这样:

```
lexicalEnvironment = {
  a: <uninitialized>
}
```
复制代码

如果我们要在变量声明之前使用变量，JavaScript引擎会从词法环境中获取变量的值，但是变量此时还是`uninitialized`状态，所以会返回一个错误`ReferenceError`。

在执行阶段，当JavaScript引擎执行到变量被声明的时候，如果声明了变量并赋值，会更新词法环境中的值，如果只是声明了变量没有被赋值，那么JavaScript引擎会给变量赋值为`undefined`。

tips： 我们可以在`let`和`const`声明之前使用他们，只要代码不是在变量声明之前执行:

```
function foo() {
    console.log(name)
}

let name = 'John Doe'

foo()   // 'John Doe'
```
复制代码

Class提升

同`let`和`const`一样，`class`在JavaScript中也是会被“提升”的，在被真正赋值之前都不会被初始化值, 同样受**Temporal Dead Zone**的影响。

```
let peter = new Person('Peter', 25) // ReferenceError: Person is not defined

class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
}

let John = new Person('John', 25);
console.log(John) // Person { name: 'John', age: 25 }
```