# 第四章、变量、作用域与内存

1、原始数据类型两种初始化的区别

let name1 = "Nicholas";  
 let name2 = new String("Matt");  
 name1.age = 27;  
 name2.age = 26;  
 console.log(name1.age); // undefined  
 console.log(name2.age); // 26  
 console.log(typeof name1); // string  
 console.log(typeof name2); // object

2、

var num =1;  
num instanceof Object //false

3、全局上下文就是我们常说的 window 对象

var在全局作用域下定义的变量和函数都会成为window对象的属性和方法

而let const定义的变量不会在window中，但在作用域链解析上效果是一样的



4、作用域链增强

try/catch语句的catch块和with语句，这些语句会在**作用域链**前端临时添加一个上下文，这个上下文在代码执 行后会被删除。

function buildUrl() {  
 let qs = "?debug=true";  
 with(location){  
 let url = href + qs;  
 }  
return url;  
}  
buildUrl() // Uncaught ReferenceError: url is not defined，  
 // 这里let url = href + qs改为var url = href + qs则正常。

对于上面with，执行时会在作用域链前端添加一个location的变量对象，href实际上就是location.href。

而在 with 语句中使用 var 声明的变量 url 会成为函数上下文的一部分。可这里let声明的变量url被限制在块级作用域中，所以return url会报错。

5、变量未经声明就被初始化了(没有声明直接赋值)， 会自动被添加到全局上下文

在严格模式下，未经声明就初始化变量会报错。

6、Object.freeze()让整个对象都不能修改,再给属性赋值时虽然不会报错，但会静默失败。

let obj = Object.freeze({name:'wdnmd'});  
obj.age = 66;  
console.log(obj.age) //undefined

7、由于 const 声明暗示变量的值是单一类型且不可修改，JavaScript 运行时编译器可以将其所有实例 8 都替换成实际的值，而不会通过查询表进行变量查找。谷歌的 V8 引擎就执行这种优化。

8、解除对一个值的引用并不会自动导致相关内存被回收。解除引用的关键在于确保相关的值已经不在上下文里了，因此它在下次垃圾回收时会被回收。

9、隐藏类和删除操作（V8引擎）

10、内存泄漏的情况

①变量没有声明直接赋值，在非严格模式下相当于window.name = 'Jake'，只要 window 本身不被清理就不会消失，导致内存泄漏。

function setName() {  
 name = 'Jake';  
 }

②定时器一直运行，里面引用的外部变量就会一直占用内存

let name = 'Jake';  
setInterval(() => {  
 console.log(name);  
}, 100);

定时器的回调通过闭包引用了外部变量，只要定时器一直运行，回调函数中引用的 name 就会一直占用内存。垃圾回收程序当然知道这一点，因而就不会清理外部变量。

③JavaScript闭包导致内存泄漏

let outer = function() {  
 let name = 'Jake';  
 return function() {  
 return name;  
 };  
};  
let foo = outer();  
let name = 'zhangsan';  
console.log(foo());

调用outer(),就会导致分配给name的内存被泄漏。

以上代码执行后创建了一个内部闭包，只要返回的函数存在就不能清理 name，因为闭包一直在引用着它。假如 name 的内容很大(不止是一个小字符 串)，那可能就是个大问题了。