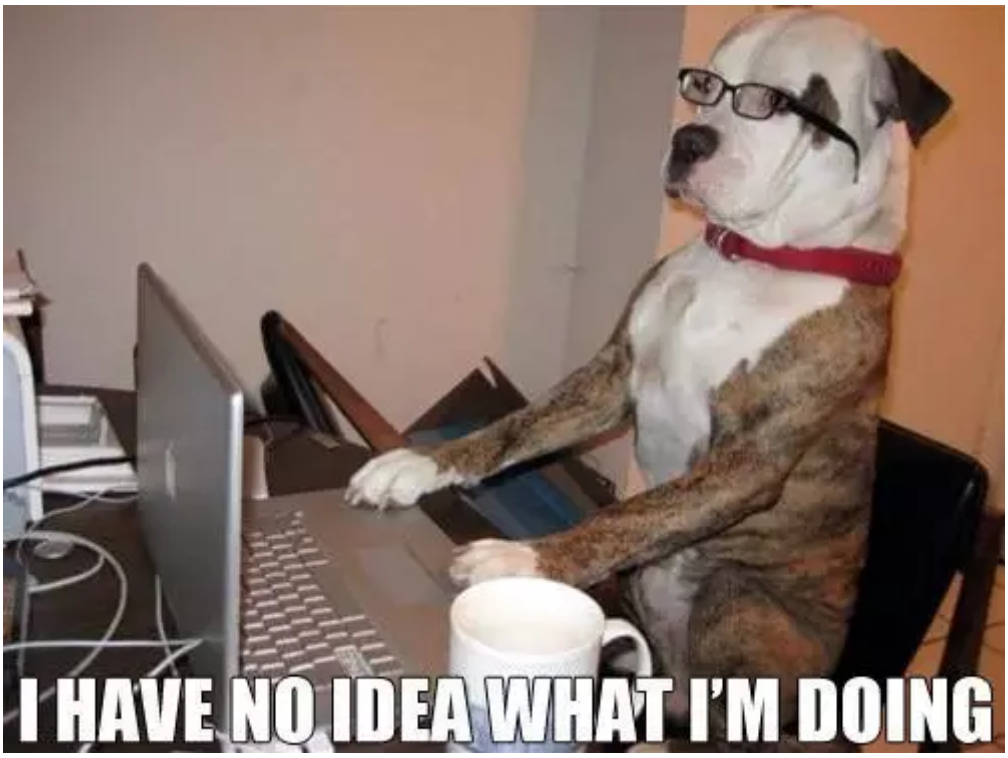
谁说 JavaScript 简单的

2017-06-19 前端大全

（点击上方公众号，可快速关注）

> 英文：Aurélien Hervé 译文：众成翻译/msmailcode
> zcfy.cc/article/who-said-javascript-was-easy-hacker-noon-3026.html
> 如有好文章投稿，请点击 → 这里了解详情

这里有一些 Javascript初学者应该知道的技巧和陷阱。如果你已经是专家了，顺便温习一下。



Javascript也只不过是一种编程语言。怎么可能出错嘛?

1. 你有没有尝试给一组数字排序?

Javascript 的sort()函数在默认情况下使用字母数字（字符串Unicode码点）排序。

所以[1,2,5,10].sort() 会输出 [1, 10, 2, 5].

要正确的排序一个数组, 你可以用 [1,2,5,10].sort((a, b) => a—b)

很简单的解决方案, 前提是你得知道有这么个坑

2. new Date() 很棒

new Date() 可以接受：

- 没有参数： 返回当前时间
- 一个参数 x： 返回1970年1月1日 + x 毫秒。 了解 Unix 的人知道为什么。
- new Date(1, 1, 1) 返回 1901, 二月 , 1号。因为，第一个参数表示1900年加1年，第二个参数表示这一年的第二个月（因此是二月）—脑回路正常的人会从1开始索引—，第三个参数很明显是这个月的第一天，所以1—有时候索引确实从1开始—。
- new Date(2016, 1, 1) 不会给1900年加上2016。它仅代表2016年。

3. Replace 并不“替代”

```
let s = "bob"
const replaced = s.replace('b', 'l')
replaced === "lob"
s === "bob"
```

我觉得这是一件好事，因为我不喜欢函数改变它们的输入。 你还应该知道 replace 只会替换第一个匹配的字符串:

如果你想替换所有匹配的字符串，你可以使用带/g标志的正则表达式：

```
"bob".replace(/b/g, 'l') === 'lol' // 替换所有匹配的字符串
```

4. 比较的时候要注意

```
// These are ok
'abc' === 'abc' // true
1 === 1         // true
// These are not
[1,2,3] === [1,2,3] // false
{a: 1} === {a: 1}   // false
{} === {}           // false
```

原因：[1,2,3]和[1,2,3]是两个独立的数组。它们只是恰好包含相同的值。它们具有不同的引用，无法用===相比较。

5. 数组不是原始数据类型

```
typeof {} === 'object'  // true
typeof 'a' === 'string' // true
```

```
typeof 1 === number    // true
// But....
typeof [] === 'object'  // true
```

如果你想知道你的变量是不是数组，你仍然可以用Array.isArray(myVar)

6. 闭包

这是一个很有名的面试题：

```
const Greeters = []
for (var i = 0; i < 10; i++) {
  Greeters.push(function () {
    return console.log(i)
  })
}
Greeters[0]() // 10
Greeters[1]() // 10
Greeters[2]() // 10
```

你是不是认为它会输出 0, 1, 2... ？ 你知道它为什么不是这样输出的吗? 你会怎样修改让它输出 0, 1, 2... ?

这里有两种可能的解决方法：

用 let 替代 var. Boom. 解决了.

> let和var的不同在于作用域。var的作用域是最近的函数块，let的作用域是最近的封闭块，封闭块可以小于函数块（如果不在任何块中，则let和var都是全局的）。

替代方法: 用 bind:

```
Greeters.push(console.log.bind(null, i))
```

还有很多其他方法。这只是我的两个首选

7. 谈到 bind

你认为这个会输出什么?

```
class Foo {
```

```
  constructor (name) {
    this.name = name
  }
  greet () {
    console.log('hello, this is ', this.name)
  }
  someThingAsync () {
    return Promise.resolve()
  }
  asyncGreet () {
    this.someThingAsync()
    .then(this.greet)
  }
}
new Foo('dog').asyncGreet()
```

如果你认为这个程序会崩溃提示 Cannot read property 'name' of undefined，给你一分。

原因： greet 没有在正确的上下文中运行。同样，这个问题依然有很多解决方案。

我个人喜欢

```
asyncGreet() {
    this.someThingAsync()
        .then(this.greet.bind(this))
}
```

这样可以确保类的实例作为上下文调用greet。

如果你认为greet 不应该在实例上下文之外运行, 你可以在类的constructor中绑定它:

```
class Foo {
    constructor(name) {
        this.name = name
        this.greet = this.greet.bind(this)
    }
}
```

你还应该知道箭头函数（ => ）可以用来保留上下文。这个方法也可以：

```
asyncGreet() {
```

```
    this.someThingAsync()
      .then(() = > {
      this.greet()
    })
}
```

尽管我认为最后一种方法并不优雅。

我很高兴我们解决了这个问题。

总结

祝贺你，你现在可以放心地把你的程序放在互联网上了。甚至运行起来可能都不会出岔子（但是通常会）Cheers o/

如果还有什么我应该提到的，请告诉我!