前言

ES6有很多新的变化，下面分享的是8个最常用的新特性

1. Let 和 const
2. 箭头函数 =>
3. 默认参数
4. 解构
5. ...
6. object 便捷写法
7. 模板字符 `

掌握这几个基本用法后，看到别人写的ES6代码时就感觉比较清晰了

内容不多，看完大概半小时左右

Let 和 const

ES5中我们使用 `var` 来声明变量，在ES6中 `var` 这个关键字被 `let` 和 `const` 替代了，这两个强大的关键字使开发变得更简单

Let vs var

我们先回顾一下 `var`

```
var me = 'Zell'
console.log(me) // Zell
```

这里声明了一个 `me`，是全局变量，可以在 function 中使用，例如：

```
var me = 'Zell'
function sayMe () {
  console.log(me)
}

sayMe() // Zell
```

反过来是不行的，function 中声明的变量不能在外面使用，例如：

```
function sayMe() {
  var me = 'Zell'
  console.log(me)
}

sayMe() // Zell
console.log(me) // Uncaught ReferenceError: me is not defined
```

我们可以说 `var` 是 function 范围的，在 function 内使用 `var` 创建的变量只存在于 function 内部，如果是在 function 之外创建的，就存在于 function 之外

```
var me = 'Zell' // global scope

function sayMe () {
  var me = 'Sleepy head' // local scope
  console.log(me)
}

sayMe() // Sleepy head
console.log(me) // Zell
```

而 `let` 是块儿范围的，在块儿内使用 `let` 创建的变量只存在于这个块儿内

先看下块儿的概念，例如

```
{
  // new scope block
}

if (true) {
  // new scope block
}

while (true) {
  // new scope block
}

function () {
  // new block scope
}
```

看一个使用 `var` 和 `let` 的对比

```
var me = 'Zell'

if (true) {
  var me = 'Sleepy head'
}

console.log(me) // 'Sleepy head'
```

执行 if 块儿之后，me 被修改了

同样的代码，只是改为 `let` 进行声明

```
let me = 'Zell'

if (true) {
    let me = 'Sleepy head'
}

console.log(me) // 'Zell'
```

块儿外的 `me` 没有受到影响

下面看一个经典的例子

```
for (var i = 1; i < 5; i++) {
  console.log(i)
  setTimeout(function () {
    console.log(i)
  }, 1000)
};
```

执行结果：

```
1
2
3
4
5
5
5
```

```
5
```

这是以前我们常见的情况，我们希望 setTimeout 输出的也是 `1 2 3 4`，但实际却全是 `5`，为了解决这个问题，需要使用闭包的形式：

```
function logLater (i) {
  setTimeout(function () {
    console.log(i)
  })
}

for (var i = 1; i < 5; i++) {
  console.log(i)
  logLater(i)
};
```

如果使用 `let` 就简单了

```
for (let i = 1; i < 5; i++) {
  console.log(i)
  setTimeout(function () {
    console.log(i)
  }, 1000)
};
```

因为 `let` 是块儿范围的，所以ES6中不要使用 `var` 声明变量了， `let` 会使开发变得更简单

Let vs const

像 `let` 一样， `const` 也是块儿范围的，唯一的区别就是：一旦声明，之后不可修改，是一个常量

```
const name = 'Zell'
name = 'Sleepy head' // TypeError: Assignment to constant variable.

let name1 = 'Zell'
name1 = 'Sleepy head'
console.log(name1) // 'Sleepy head'
```

例如声明一个节点变量，由于其是不会改变的，就可以使用 `const` 声明为一个常量

```
const modalLauncher = document.querySelector('.jsModalLauncher'
)
```

> 建议：尽量使用 const，可以防止被意外修改，导致莫名其妙的错误，在明确需要修改时再使用 let

=> 箭头函数

简单理解 `=>` 就是**匿名函数**的简写，例如：

```
let array = [1,7,98,5,4,2]

var moreThan20 = array.filter(function (num) {
  return num > 20
})
```

可以使用 `=>` 进行简写为：

```
let moreThan20 = array.filter(num => num > 20)
```

=> 的本质

创建函数可以使用这种方式：

```
function namedFunction() {
  // ...
}

// 调用
namedFunction()
```

也可以使用第二种方式，创建一个匿名函数付给一个变量

```
var namedFunction = function() {
```

```
  // ...
}
```

还有第三种方式，直接把 function 作为其他函数的参数

```
// 使用一个匿名函数作为回调
button.addEventListener('click', function() {
  // ...
})
```

对于第二种和第三种的匿名函数形式，可以替换为 `=>`

```
// 正常写法
const namedFunction = function (arg1, arg2) {
  // ...
}

// =>
const namedFunction2 = (arg1, arg2) => {
  // ...
}

// 正常写法
button.addEventListener('click', function () {
  // ...
})

// =>
button.addEventListener('click', () => {
  // ...
})
```

通过这几个例子已经可以大概看出箭头函数的用法，就是把 `function` 去掉，在参数与函数体中间加上一个 `=>`

但箭头函数不只这么简单，还有其他的便利之处

（1）简化参数

当只有一个参数时，可以把包裹参数的括号去掉，如果不需要参数，可以只用一个 `_` 代替

下面几种写法都是合法的

```
const zeroArgs = () => {
  // ...
}
// 没有参数时使用 _ 代替
const zeroWithUnderscore = _ => {
  // ...
}
// 一个参数时可以去掉括号
const oneArg = arg1 => {
  // ...
}
const oneArgWithParenthesis = (arg1) => {
  // ...
}
const manyArgs = (arg1, arg2) => {
  // ...
}
```

（2）隐式返回

当函数体只有一行时，可以不用 `return`，因为箭头函数会自动创建一个 `return` 关键字，并且可以省略方法体外面的 `{ }`，例如

```
const sum1 = (num1, num2) => num1 + num2
const sum2 = (num1, num2) => { return num1 + num2 }
```

对比示例

```
let array = [1,7,98,5,4,2]

// ES5
var moreThan20 = array.filter(function (num) {
  return num > 20
})

// ES6
let moreThan20 = array.filter(num => num > 20)
```

this 关键字在箭头函数中的应用

先回顾下没有箭头函数时 `this` 的用法

```
console.log(this) // 输出 Window
```

在简单函数调用时，`this` 也是被设置为全局对象

```
function hello () {
  console.log(this)
}

hello() // Window
```

在简单方法调用中，JavaScript 总是把 `this` 设置为 `Window` 对象，这就是为什么在 `setTimeout` 这类函数中 `this` 总是为 `Window`

当调用对象方法时，`this` 被设为对象本身

```
let o = {
  sayThis: function() {
    console.log(this)
  }
}

o.sayThis() // o
```

在创建一个新对象时，`this` 指向对象本身

```
function Person (age) {
  this.age = age
}

let greg = new Person(22)
let thomas = new Person(24)

console.log(greg) // this.age = 22
console.log(thomas) // this.age = 24
```

在使用事件监听器时，`this` 被设置为触发事件的那个节点

```
let button = document.querySelector('button')

button.addEventListener('click', function() {
```

```
  console.log(this) // button
})
```

以上就是 ES5 中 `this` 的主要几种场景，在箭头函数中，`this` 有了新的行为方式，不能简单的等同于 function 形式

需要记住：在箭头函数中，`this` 永远不会被绑定一个新值，`this` 的值是外层的 `this` 值

通俗理解，在箭头函数中看到 this 时，就向外找，遇到了第一个 this 的值就是箭头函数中 this 的值

看个例子

```
let o = {
  // 箭头函数
  notThis: () => {
    console.log(this) // Window
    this.objectThis() // Uncaught TypeError: this.objectThis is
 not a function
  },
  // 普通函数
  objectThis: function () {
    console.log(this) // o
  }
}
```

这里箭头函数中 this 是 Window，因为往外找时，最近的就是 Window

通过这个例子需要记住，不要使用箭头函数声明对象方法，因为这样就不能通过 this 引用到当前对象

还有，不能使用箭头函数创建事件监听器，因为 this 不会被绑到监听的那个节点元素上，如果就喜欢用箭头函数，那么就不能通过 this 去获取触发元素，要用 `event.currentTarget`

示例代码

```
button.addEventListener('click', function () {
  console.log(this) // button
})
```

```
button.addEventListener('click', e => {
  console.log(this) // Window
  console.log(event.currentTarget) // button
})
```

下面看一个箭头函数带来的好处

```
let o = {
  // 老方法
  oldDoSthAfterThree: function () {
    let that = this
    setTimeout(function () {
      console.log(this) // Window
      console.log(that) // o
    })
  },
  // 箭头函数方式
  doSthAfterThree: function () {
    setTimeout(() => {
      console.log(this) // o
    }, 3000)
  }
}
```

老方法中 `setTimeout` 的 this 是 Window，如果想使用对象方法的 this，就要使用中间变量，麻烦，容易出错

而在 setTimeout 中箭头函数时，其中的 this 不会被绑定值，而是去外层寻找，就会找到外层 function 的 this，这就是我们预期的了

最后看个例子作为总结

```
function test() {
      return () => {
         return () => {
            console.log("id:", this.id);
         };
      };
}

test.call( { id: 1 } )()(); // 输出 1
```

箭头函数不管被调用几次，this 都不会被绑定值，只是简单向外层寻找 this，此

处最近的外层 this 就是 test 函数的，所以输出1

默认参数

例如一个函数需要3个参数

```
function announcePlayer (firstName, lastName, teamName) {
  console.log(firstName + ' ' + lastName + ', ' + teamName)
}

announcePlayer('Stephen', 'Curry', 'Golden State Warriors')
```

如果最后一个参数不是必须，可能会不传，就会出问题

```
announcePlayer('Zell', 'Liew') // Zell Liew, undefined
```

我们就需要添加预先处理的代码

```
function announcePlayer (firstName, lastName, teamName) {
  if (!teamName) {
    teamName = 'self'
  }
  console.log(firstName + ' ' + lastName + ', ' + teamName)
}

announcePlayer('Zell', 'Liew')
// Zell Liew, self
```

如果需要这样处理的参数较多就很麻烦了，ES6中可以直接给参数一个默认值

```
const announcePlayer = (firstName, lastName, teamName = 'self')
 => {
  console.log(firstName + ' ' + lastName + ', ' + teamName)
}

announcePlayer('Zell', 'Liew') // Zell Liew, self
```

这样就方便了很多，而且如果传入的是 `undefined` 未定义，那么也会使用默认值

```
announcePlayer('Zell', 'Liew', undefined)
// Zell Liew, self
```

解构

解构可以帮助我们方便的从数组和对象中取值

解构对象

```
const Zell = {
  firstName: 'Zell',
  lastName: 'Liew'
}
```

想取得其中的 `firstName` 和 `lastName`，可以创建两个变量，然后分别赋值

```
let firstName = Zell.firstName // Zell
let lastName = Zell.lastName // Liew
```

使用解构的话一行代码就可以解决，声明变量和赋值一次性解决

```
let { firstName, lastName } = Zell

console.log(firstName) // Zell
console.log(lastName) // Liew
```

使用 `{}` 就是告诉 js 要创建这两个变量，并把 Zell 中对应的值赋给他们

```
let { firstName, lastName } = Zell

ES6 自动执行：

let firstName = Zell.firstName
let lastName = Zell.lastName
```

如果对象中的属性名已经被之前的变量占用了怎么办？ 例如

```
let name = 'Zell Liew'
let course = {
  name: 'JS Fundamentals for Frontend Developers'
  // ... other properties
}

let { name } = course // Uncaught SyntaxError: Identifier 'name
' has already been declared
```

这时可以使用新的名字

```
let { name: courseName } = course

console.log(courseName) // JS Fundamentals for Frontend Develop
ers

// ES6 这样做:
let courseName = course.name
```

如果解构时对象中没有这个属性，会自动返回 `undefined`

```
let course = {
  name: 'JS Fundamentals for Frontend Developers'
}

let { package } = course

console.log(package) // undefined
```

如果不想要 `undefined` 可以指定默认值

```
let course = {
  name: 'JS Fundamentals for Frontend Developers'
}

let { package = 'full course' } = course

console.log(package) // full course
```

改名和默认值可以一起使用

```
let course = {
  name: 'JS Fundamentals for Frontend Developers'
}

let { package: packageName = 'full course' } = course

console.log(packageName) // full course
```

解构数组

解构对象是使用 `{ }` 解构数组是使用 `[ ]`

```
let [one, two] = [1, 2, 3, 4, 5]
console.log(one) // 1
console.log(two) // 2
```

第一个变量对应数组的第一个元素，以此类推

如果解构时变量的数量超出了数组元素的数量，多出的变量就是 `undefined`

```
let [one, two, three] = [1, 2]
console.log(one) // 1
console.log(two) // 2
console.log(three) // undefined
```

解构数组时，我们通常只要我们需要的，对应其他数组元素，可以使用 rest 操作符放到一个数组中

```
let scores = ['98', '95', '93', '90', '87', '85']
let [first, second, third, ...rest] = scores

console.log(first) // 98
console.log(second) // 95
console.log(third) // 93
console.log(rest) // [90, 87, 85]
```

小技巧：使用解构来交换变量

有两个变量

```
let a = 2
let b = 3
```

想把他们对换，变成 `a=3,b=2`，ES5 中需要这样写：

```
let a = 2
let b = 3
let temp

// swapping
temp = a // temp is now 2
a = b // a is now 3
b = temp // b is now 2
```

使用 ES6 的解构就方便多了

```
let a = 2;
let b = 3;

[a, b] = [b, a]

console.log(a) // 3
console.log(b) // 2
```

函数中应用解构

```
function topThree (scores) {
  let [first, second, third] = scores
  return {
    first: first,
    second: second,
    third: third
  }
}
```

可以简化为：

```
function topThree ([first, second, third]) {
  return {
    first: first,
```

```
    second: second,
    third: third
  }
}
```

代码更简洁，而且可以方便的看出这个函数的参数应该是一个数组

看下面的代码是什么含义

```
function sayMyName ({
  firstName = 'Zell',
  lastName = 'Liew'
} = {}) {
 console.log(firstName + ' ' + lastName)
}
```

首先，这个函数的参数时一个对象，有一个默认值 `{}`

然后，对传入的对象进行解构，声明了两个变量 `firstName` 和 `lastName`，如果对象中没有对应的属性，就使用它们的默认值

调用示例：

```
sayMyName() // Zell Liew
sayMyName({firstName: 'Zell'}) // Zell Liew
sayMyName({firstName: 'Vincy', lastName: 'Zhang'}) // Vincy Zha
ng
```

▪▪▪

`...` 有两个含义：1. 剩余参数，2. 展开操作

剩余参数

例如有一个函数，把他的所有参数加起来

```
sum(1, 2, 3, 4, 5, 6, 7, 8, 9, 10) // 55
```

ES5 中需要使用 `arguments` 变量来处理参数数量未知的情况，如

```
function sum () {
  console.log(arguments)
}

sum(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

处理累加操作：

```
function sum () {
  let argsArray = Array.prototype.slice.call(arguments)
  return argsArray.reduce(function(sum, current) {
    return sum + current
  }, 0)
}
```

ES6 中使用**剩余参数**可以把所有逗号分隔的变量打包到一个数组中

```
const sum = (...args) => args.reduce((sum, current) => sum + cu
rrent, 0)
```

如果箭头函数太多不好理解，可以写成这样

```
function sum (...args) {
  return args.reduce((sum, current) => sum + current, 0)
}
```

展开操作

与剩余参数的操作相反，会把一个数组展开成一个逗号分隔的列表

```
let array = ['one', 'two', 'three']


console.log(...array) // one two three
console.log('one', 'two', 'three') // one two three
```

展开操作可以方便的完成数组的联合，例如有3个数组

```
let array1 = ['one', 'two']
let array2 = ['three', 'four']
let array3 = ['five', 'six']
```

ES5 需要使用 `Array.concat`

```
let combinedArray = array1.concat(array2).concat(array3)
console.log(combinedArray)
// ['one', 'two', 'three', 'four', 'five', 'six']
```

使用展开操作：

```
let combinedArray = [...array1, ...array2, ...array3]
console.log(combinedArray)
// ['one', 'two', 'three', 'four', 'five', 'six']
```

提升对象写法

对象是我们非常熟悉的，例如：

```
const anObject = {
  property1: 'value1',
  property2: 'value2',
  property3: 'value3',
}
```

ES6 中带来了3个改变：
1. 属性值简写
2. 方法简写
3. 使用动态属性名

1） 属性值简写

下面的情况比较常见：

```
const fullName = 'Zell Liew'
```

```
const Zell = {
  fullName: fullName
}
```

对象中名字和值是同名的，ES6 中可以简写：

```
const fullName = 'Zell Liew'

const Zell = {
  fullName
}

// ES6 自动去做：
const Zell = {
  fullName: fullName
}
```

2）方法简写

例如在对象中定义个方法：

```
const anObject = {
  aMethod: function () { console.log("I'm a method!~~")}
}
```

ES6 中可以移除 `: function`

```
const anObject = {
  aShorthandMethod (arg1, arg2) {}
}
```

3）使用动态属性名

创建对象时有时需要一个动态属性名，ES5 中需要这样做：

```
const newPropertyName = 'smile'

const anObject = { aProperty: 'a value' }
```

```
anObject[newPropertyName] = ':D'

anObject['bigger ' + newPropertyName] = 'XD'
```

ES6 中不再需要这么繁琐了，可以直接使用动态属性名

```
const newPropertyName = 'smile'

const anObject = {
  aProperty: 'a value',


}
```

字符串模板

例如把几个变量拼接成一个字符串

```
function announcePlayer (firstName, lastName, teamName) {
  console.log(firstName + ' ' + lastName + ', ' + teamName)
}
```

ES6 中可以使用倒引号 `` ` `` 包裹一个字符串，其中能够放入变量

```
const firstName = 'Zell'
const lastName = 'Liew'
const teamName = 'unaffiliated'

const theString = `${firstName} ${lastName}, ${teamName}`

console.log(theString)
// Zell Liew, unaffiliated
```

还支持多行

```
const container = document.createElement('div')
const aListOfItems =
  `<ul>
    <li>Point number one</li>
    <li>Point number two</li>
```

```
    <li>Point number three</li>
    <li>Point number four</li>
  </ul>`

container.innerHTML = aListOfItems

document.body.append(container)
```

标签函数

用来处理模板字符串，例如

```
var person = 'Mike';
var age = 28;

function myTag(strings, personExp, ageExp) {

  var str0 = strings[0]; // "that "
  var str1 = strings[1]; // " is a "
  var str1 = strings[2]; // " !"

  var ageStr;
  if (ageExp > 99){
    ageStr = 'centenarian';
  } else {
    ageStr = 'youngster';
  }

  return str0 + personExp + str1 + ageStr;

}

var output = myTag`that ${ person } is a ${ age } !`;

console.log(output);

// 输出: that Mike is a youngster
```

myTag 中的3个参数：
1. 模板字符串中静态文字
2. 动态部分 - `${ person }`
3. 动态部分 - `${ age }`

myTag 对 `age` 做了判断，根据结果返回了不同的字符串

小结

以上是ES6特性中最为常用的几个，非常值得花一点时间学习一下

如果对本教程有建议或者意见，欢迎到公众号（性能与架构）中给我留言反馈