



黑马程序员™
www.itheima.com

传智播客旗下
高端IT教育品牌



ES6

目录 Contents

- ◆ ES6 简介
- ◆ ES6 的新增语法
- ◆ ES6 的内置对象扩展

什么是 ES6 ?

ES 的全称是 ECMAScript , 它是由 ECMA 国际标准化组织,制定的一项脚本语言的标准化规范。

年份	版本
2015年6月	ES2015
2016年6月	ES2016
2017年6月	ES2017
2018年6月	ES2018
...	...

ES6 实际上是一个泛指, 泛指 ES2015 及后续的版本。

为什么使用 ES6 ?

每一次标准的诞生都意味着语言的完善，功能的加强。JavaScript语言本身也有一些令人不满意的地方。

- 变量提升特性增加了程序运行时的不可预测性
- 语法过于松散，实现相同的功能，不同的人可能会写出不同的代码

目录 Contents

- ◆ ES6 简介
- ◆ ES6 的新增语法
- ◆ ES6 的内置对象扩展

let

ES6中新增的用于声明变量的关键字。

- let声明的变量只在所处于的块级有效

```
if (true) {  
    let a = 10;  
}  
  
console.log(a) // a is not defined
```

注意：使用let关键字声明的变量才具有块级作用域，使用var声明的变量不具备块级作用域特性。

let

ES6中新增的用于声明变量的关键字。

- 不存在变量提升

```
console.log(a); // a is not defined  
let a = 20;
```

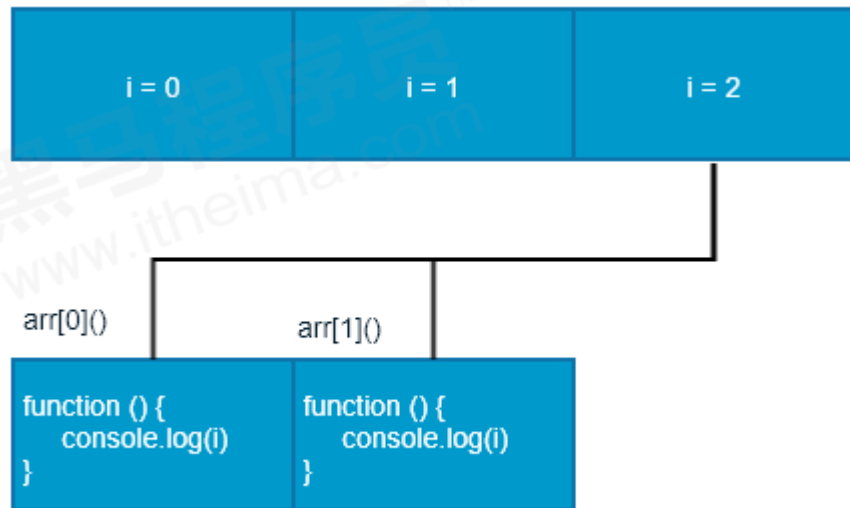
- 暂时性死区

```
var tmp = 123;  
if (true) {  
    tmp = 'abc';  
    let tmp;  
}
```

let

经典面试题

```
var arr = [];  
for (var i = 0; i < 2; i++) {  
  arr[i] = function () {  
    console.log(i);  
  }  
}  
arr[0]();  
arr[1]();
```

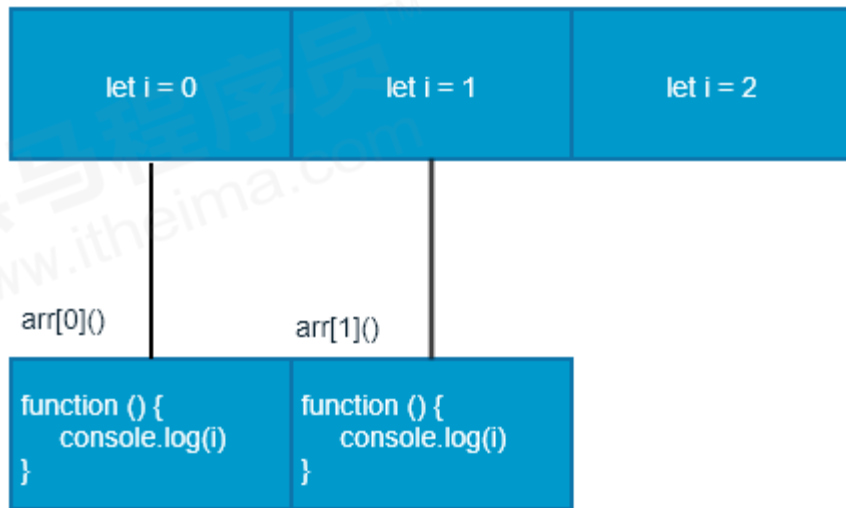


经典面试题图解：此题的关键点在于变量*i*是全局的，函数执行时输出的都是全局作用域下的*i*值。

let

经典面试题

```
let arr = [];  
for (let i = 0; i < 2; i++) {  
  arr[i] = function () {  
    console.log(i);  
  }  
}  
arr[0]();  
arr[1]();
```



经典面试题图解：此题的关键点在于每次循环都会产生一个块级作用域，每个块级作用域中的变量都是不同的，函数执行时输出的是自己上一级（循环产生的块级作用域）作用域下的*i*值。

const

作用：声明常量，常量就是值（内存地址）不能变化的量。

- 具有块级作用域

```
if (true) {  
    const a = 10;  
}  
  
console.log(a) // a is not defined
```

- 声明常量时必须赋值

```
const PI; // Missing initializer in const declaration
```

const

作用：声明常量，常量就是值（内存地址）不能变化的量。

- 常量赋值后，值不能修改。

```
const PI = 3.14;
```

```
PI = 100; // Assignment to constant variable.
```

```
const ary = [100, 200];
```

```
ary[0] = 'a';
```

```
ary[1] = 'b';
```

```
console.log(ary); // ['a', 'b'];
```

```
ary = ['a', 'b']; // Assignment to constant variable.
```

let、const、var 的区别

1. 使用 **var** 声明的变量，其作用域为**该语句所在的函数内**，且存在**变量提升现象**。
2. 使用 **let** 声明的变量，其作用域为**该语句所在的代码块内**，不存在**变量提升**。
3. 使用 **const** 声明的是常量，在后面出现的代码中**不能再修改该常量的值**。

var	let	const
函数级作用域	块级作用域	块级作用域
变量提升	不存在变量提升	不存在变量提升
值可更改	值可更改	值不可更改

解构赋值

ES6中允许从数组中提取值，按照对应位置，对变量赋值。对象也可以实现解构。

数组解构

```
let [a, b, c] = [1, 2, 3];  
console.log(a)  
console.log(b)  
console.log(c)
```

如果解构不成功，变量的值为undefined。

```
let [foo] = [];  
let [bar, foo] = [1];
```

解构赋值

按照一定模式，从数组中或对象中提取值，将提取出来的值赋值给另外的变量。

对象解构

```
let person = { name: 'zhangsan', age: 20 };  
let { name, age } = person;  
console.log(name); // 'zhangsan'  
console.log(age); // 20
```

```
let {name: myName, age: myAge} = person; // myName myAge 属于别名  
console.log(myName); // 'zhangsan'  
console.log(myAge); // 20
```



箭头函数

ES6中新增的定义函数的方式。

```
() => {}  
const fn = () => {}
```

函数体中只有一句代码，且代码的执行结果就是返回值，可以省略大括号

```
function sum(num1, num2) {  
    return num1 + num2;  
}  
const sum = (num1, num2) => num1 + num2;
```

箭头函数

如果形参只有一个，可以省略小括号

```
function fn (v) {  
    return v;  
}  
  
const fn = v => v;
```


箭头函数

箭头函数不绑定this关键字，箭头函数中的this，指向的是函数定义位置的上下文this。

```
const obj = { name: '张三' }  
  
function fn () {  
  console.log(this);  
  return () => {  
    console.log(this)  
  }  
}  
  
const resFn = fn.call(obj);  
resFn();
```

剩余参数

剩余参数语法允许我们将一个不定数量的参数表示为一个数组。

```
function sum (first, ...args) {  
    console.log(first); // 10  
    console.log(args); // [20, 30]  
}  
sum(10, 20, 30)
```

剩余参数

剩余参数和解构配合使用

```
let students = ['wangwu', 'zhangsan', 'lisi'];  
let [s1, ...s2] = students;  
console.log(s1); // 'wangwu'  
console.log(s2); // ['zhangsan', 'lisi']
```

目录 Contents

- ◆ ES6 简介
- ◆ ES6 的新增语法
- ◆ ES6 的内置对象扩展

Array 的扩展方法

扩展运算符（展开语法）

扩展运算符可以将数组或者对象转为用逗号分隔的参数序列。

```
let ary = [1, 2, 3];  
...ary // 1, 2, 3  
console.log(...ary); // 1 2 3  
console.log(1, 2, 3)
```

Array 的扩展方法

扩展运算符（展开语法）

扩展运算符可以应用于合并数组。

// 方法一

```
let ary1 = [1, 2, 3];  
let ary2 = [3, 4, 5];  
let ary3 = [...ary1, ...ary2];
```

// 方法二

```
ary1.push(...ary2);
```

Array 的扩展方法

扩展运算符（展开语法）

将类数组或可遍历对象转换为真正的数组

```
let oDivs = document.getElementsByTagName('div');  
oDivs = [...oDivs];
```

Array 的扩展方法

构造函数方法：Array.from()

将类数组或可遍历对象转换为真正的数组

```
let arrayLike = {  
  '0': 'a',  
  '1': 'b',  
  '2': 'c',  
  length: 3  
};  
  
let arr2 = Array.from(arrayLike); // ['a', 'b', 'c']
```


Array 的扩展方法

构造函数方法：Array.from()

方法还可以接受第二个参数，作用类似于数组的map方法，用来对每个元素进行处理，将处理后的值放入返回的数组。

```
let arrayLike = {
  "0": 1,
  "1": 2,
  "length": 2
}

let newArray = Array.from(arrayLike, item => item * 2)
```

Array 的扩展方法

实例方法：find()

用于找出第一个符合条件的数组成员，如果没有找到返回undefined

```
let ary = [{
  id: 1,
  name: '张三'
}, {
  id: 2,
  name: '李四'
}];

let target = ary.find((item, index) => item.id == 2);
```

Array 的扩展方法

实例方法：findIndex()

用于找出第一个符合条件的数组成员的位置，如果没有找到返回-1

```
let ary = [1, 5, 10, 15];  
let index = ary.findIndex((value, index) => value > 9);  
console.log(index); // 2
```

Array 的扩展方法

实例方法：includes()

表示某个数组是否包含给定的值，返回布尔值。

```
[1, 2, 3].includes(2) // true  
[1, 2, 3].includes(4) // false
```

String 的扩展方法

模板字符串

ES6新增的创建字符串的方式，使用反引号定义。

```
let name = `zhangsan`;
```

String 的扩展方法

模板字符串

ES6新增的创建字符串的方式，使用反引号定义。

```
let name = `zhangsan`;
```

String 的扩展方法

模板字符串

模板字符串中可以解析变量。

```
let name = '张三';  
let sayHello = `hello, my name is ${name}`; // hello, my name is zhangsan
```

String 的扩展方法

模板字符串

模板字符串中可以换行

```
let result = {  
  name: 'zhangsan',  
  age: 20,  
  sex: '男'  
}  
  
let html = `

<span>${result.name}</span>  
  <span>${result.age}</span>  
  <span>${result.sex}</span>  
</div>`;


```


String 的扩展方法

模板字符串

在模板字符串中可以调用函数。

```
const sayHello = function () {  
    return '哈哈哈哈 追不到我吧 我就是这么强大';  
};  
  
let greet = `${sayHello()} 哈哈哈哈`;  
console.log(greet); // 哈哈哈哈 追不到我吧 我就是这么强大 哈哈哈哈
```

String 的扩展方法

实例方法：startsWith() 和 endsWith()

- **startsWith()**：表示参数字符串是否在原字符串的头部，返回布尔值
- **endsWith()**：表示参数字符串是否在原字符串的尾部，返回布尔值

```
let str = 'Hello world!';  
str.startsWith('Hello') // true  
str.endsWith('!')       // true
```

String 的扩展方法

实例方法：repeat()

repeat方法表示将原字符串重复n次，返回一个新字符串。

```
'x'.repeat(3)      // "xxx"  
'hello'.repeat(2)  // "hellohello"
```

Set 数据结构

ES6 提供了新的数据结构 Set。它类似于数组，但是成员的值都是唯一的，没有重复的值。

Set本身是一个构造函数，用来生成 Set 数据结构。

```
const s = new Set();
```

Set函数可以接受一个数组作为参数，用来初始化。

```
const set = new Set([1, 2, 3, 4, 4]);
```

Set 数据结构

实例方法

- `add(value)`：添加某个值，返回 Set 结构本身
- `delete(value)`：删除某个值，返回一个布尔值，表示删除是否成功
- `has(value)`：返回一个布尔值，表示该值是否为 Set 的成员
- `clear()`：清除所有成员，没有返回值

```
const s = new Set();  
s.add(1).add(2).add(3); // 向 set 结构中添加值  
s.delete(2)             // 删除 set 结构中的2值  
s.has(1)                 // 表示 set 结构中是否有1这个值 返回布尔值  
s.clear()                // 清除 set 结构中的所有值
```

Set 数据结构

遍历

Set 结构的实例与数组一样，也拥有forEach方法，用于对每个成员执行某种操作，没有返回值。

```
s.forEach(value => console.log(value))
```



黑马程序员

www.itheima.com

传智播客旗下高端IT教育品牌