ES6-下一代Javascript标准

1.简介

ECMAScript6 就是ECMAScript 6.0（以下简称 ES6）是 JavaScript 语言的下一代标准，已经在2015年6月正式发布了,不管你是写nodejs,还是angular,react,只要是js代码,你都可以看到es6的身影,它已经变成了前端必会的技能之一。

2.let和const

- 局部作用域

```
{
  let a = 10;
  var b = 1;
}

a // ReferenceError: a is not defined.
b // 1
```

- 不存在变量提升

```
// var 的情况
console.log(a); // 输出undefined
var a = 2;

// let 的情况
console.log(b); // 报错ReferenceError
let b = 2;
```

- 之前var 不合理的场景

```
var a = [];
for (var i = 0; i < 10; i++) {
  a[i] = function () {
    console.log(i);
  };
}
a[6](); // 10

var a = [];
for (let i = 0; i < 10; i++) {
  a[i] = function () {
    console.log(i);
  };
}
a[6](); // 6
```

- 暂时性死区

  ES6明确规定，如果区块中存在`let`和`const`命令，这个区块对这些命令声明的变量，从一开始就形成了封闭作用域。凡是在声明之前就使用这些变量，就会报错。

  总之，在代码块内，使用`let`命令声明变量之前，该变量都是不可用的。这在语法上，称为“暂时性死区”（temporal dead zone，简称 TDZ）。

例如：

```
if (true) {
  // TDZ开始
  tmp = 'abc'; // ReferenceError
  console.log(tmp); // ReferenceError

  let tmp; // TDZ结束
  console.log(tmp); // undefined

  tmp = 123;
  console.log(tmp); // 123
}
```

- 不允许重复声明

3.模板字符串

- 传统3种写法
- es6的模板字符串

4.函数的扩展

- 函数参数的默认值

```
function log(x, y = 'World') {
  console.log(x, y);
}

log('Hello') // Hello World
log('Hello', 'China') // Hello China
log('Hello', '') // Hello
```

- 箭头函数

```
var f = v => v;
```

上面的箭头函数等同于：

```
var f = function(v) {
  return v;
};
```

如果箭头函数的代码块部分多于一条语句，就要使用大括号将它们括起来，并且使用`return`语句返回。

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

由于大括号被解释为代码块，所以如果箭头函数直接返回一个对象，必须在对象外面加上括号。

```
var getTempItem = id => ({ id: id, name: "Temp" });
```

箭头函数可以与变量解构结合使用。

```
const full = ({ first, last }) => first + ' ' + last;

// 等同于
function full(person) {
  return person.first + ' ' + person.last;
}
```

- 箭头函数中的this

函数体内的`this`对象，就是定义时所在的对象，而不是使用时所在的对象。

- 箭头的缩写

5.对象的扩展

- 对象的简写
- Object.keys()
- Object.assgin()

6.class类

- 传统的方式怎么写
- 使用class创建类
- constructor构造函数
- es6的继承

```
class ColorPoint extends Point {
  constructor(x, y, color) {
    super(x, y); // 调用父类的constructor(x, y)
    this.color = color;
  }

  toString() {
    return this.color + ' ' + super.toString(); // 调用父类的toString()
  }
}
```

大多数浏览器的ES5实现之中，每一个对象都有`__proto__`属性，指向对应的构造函数的prototype属性。Class作为构造函数的语法糖，同时有prototype属性和`__proto__`属性，因此同时存在两条继承链。

（1）子类的`__proto__`属性，表示构造函数的继承，总是指向父类。

（2）子类`prototype`属性的`__proto__`属性，表示方法的继承，总是指向父类的`prototype`属性。

```
class A {
}

class B extends A {
}

B.__proto__ === A // true
B.prototype.__proto__ === A.prototype // true
```

- 静态方法

```
class Foo {
  static classMethod() {
    return 'hello';
  }
}

Foo.classMethod() // 'hello'

var foo = new Foo();
foo.classMethod()
```

父类的静态方法，可以被子类继承。

```
class Foo {
  static classMethod() {
    return 'hello';
  }
}

class Bar extends Foo {
}

Bar.classMethod(); // 'hello'
```

7.解构赋值

- 数组解构

```
let [foo, [[bar], baz]] = [1, [[2], 3]];
foo // 1
bar // 2
baz // 3

let [ , , third] = ["foo", "bar", "baz"];
third // "baz"

let [x, , y] = [1, 2, 3];
x // 1
y // 3

let [head, ...tail] = [1, 2, 3, 4];
head // 1
tail // [2, 3, 4]

let [x, y, ...z] = ['a'];
x // "a"
y // undefined
z // []
```

- 对象解构

```
var obj = { foo: function(){return 'aaa'}, bar: "bbb" };
let {foo,bar} = obj
foo();//"aaa"
```

- 字符串解构

```
const [a, b, c, d, e] = 'hello';
a // "h"
b // "e"
c // "l"
d // "l"
e // "o"
```

- 函数参数解构

函数的参数也可以使用解构赋值。

```
function add([x, y]){
  return x + y;
}
add([1, 2]); //
```

8.编译环境搭建

- babel
- vue-cli

9.Module 模块的语法

- export
- import

```
// circle.js

export function area(radius) {
  return Math.PI * radius * radius;
}

export function circumference(radius) {
  return 2 * Math.PI * radius;
}
```

现在，加载这个模块。

```
// main.js

import { area, circumference } from './circle';

console.log('圆面积：' + area(4));
console.log('圆周长：' + circumference(14));
```

上面写法是逐一指定要加载的方法，整体加载的写法如下。

```
import * as circle from './circle';

console.log('圆面积：' + circle.area(4));
console.log('圆周长：' + circle.circumference(14));
```

```
// 正确
export var a = 1;

// 正确
var a = 1;
export default a;

// 错误
export default var a = 1;
```

10.Promise对象

- 基本用法

```
var promise = new Promise(function(resolve, reject) {
  // ... some code

  if (/* 异步操作成功 */){
    resolve(value);
  } else {
    reject(error);
  }
});
```

下面是异步加载图片的例子。

```
function loadImageAsync(url) {
  return new Promise(function(resolve, reject) {
    var image = new Image();

    image.onload = function() {
      resolve(image);
    };

    image.onerror = function() {
      reject(new Error('Could not load image at ' + url));
    };

    image.src = url;
  });
}
```

下面是一个用Promise对象实现的Ajax操作的例子。

```
var getJSON = function(url) {
  var promise = new Promise(function(resolve, reject){
    var client = new XMLHttpRequest();
    client.open("GET", url);
    client.onreadystatechange = handler;
    client.responseType = "json";
    client.setRequestHeader("Accept", "application/json");
    client.send();

    function handler() {
      if (this.readyState !== 4) {
        return;
      }
      if (this.status === 200) {
        resolve(this.response);
      } else {
        reject(new Error(this.statusText));
      }
    };
  });

  return promise;
};

getJSON("/posts.json").then(function(json) {
  console.log('Contents: ' + json);
}, function(error) {
  console.error('出错了', error);
});•
```

11.案例：躁动的小球

- 传统的写法
- es6的写法