ES6文档

[搭建es6环境 2](#_Toc23013)

[es6语法 3](#_Toc12178)

[Let命令： 3](#_Toc25262)

[Const命令:只能定义一个常量不能改，声明的时候必须赋值。 4](#_Toc28316)

[数组结构赋值： 5](#_Toc22013)

[对象结构赋值： 7](#_Toc19853)

[正则方法： 8](#_Toc12344)

[1.修饰符： 8](#_Toc29912)

[2.字符串的正则方法 8](#_Toc18181)

[3. u修饰符（含义为“Unicode 模式”） 8](#_Toc32388)

[1. 点字符 8](#_Toc25443)

[上面代码表示，如果不添加u修饰符，正则表达式就会认为字符串为两个字符，从而匹配失败。 9](#_Toc4705)

[2. Unicode 字符表示法 9](#_Toc11654)

[3. 量词 9](#_Toc24814)

[4. 预定义模式 9](#_Toc160)

[5. i 修饰符 9](#_Toc23162)

[4. y修饰符 10](#_Toc11782)

[5. Stricky属性 10](#_Toc11336)

[6. Flags属性 10](#_Toc12869)

[7. s修饰符 10](#_Toc7090)

[字符串的拓展 12](#_Toc13356)

[1. 字符串的unicode表示法 12](#_Toc22445)

[2. codePointAt方法 13](#_Toc9427)

[3. String.fromCharCode 13](#_Toc23316)

[数值的拓展 13](#_Toc20772)

[1. 二进制与八进制的表示法 13](#_Toc3751)

[2. Number.isFinite()和Number.isNaN() 13](#_Toc30842)

[函数的拓展 14](#_Toc3999)

[默认值 14](#_Toc22853)

[rest参数 15](#_Toc26693)

[箭头函数（函数名+函数参数+函数返回值） 16](#_Toc27615)

[伪调用（） 18](#_Toc12790)

[函数的length属性： 18](#_Toc1265)

[作用域 19](#_Toc5229)

[数组的拓展 20](#_Toc9244)

[复制数组 20](#_Toc29784)

[对象拓展 20](#_Toc6333)

# 搭建es6环境

1. npm install gulp gulp-if gulp-concat webpack webpack-stream vinyl-named gulp-livereload gulp-plumber gulp-rename gulp-uglify gulp-util yargs --save-dev
2. Npm install gulp-live-server del gulp-util --save-dev
3. Npm install babel-loader babel-core babel-preset-env webpack --save-dev
4. Npm install require-dir --save-dev
5. Npm install babel-preset-es2015 --save-dev
6. gulp-sequence
7. Gulp --watch

安装补丁处理兼容：

Npm install babel-polyfill --save-dev

# es6语法

# Let命令：

1.Let声明的变量只在它所在的代码块有效，例如：,for循环

for (let i = 0; i < 10; i++) {

// ...}

console.log(i)

Function user(){

For(let i =0; i<3; i++){

Consloe.log(i)

}

Console.log(i);//会报错，let声明的i只在{}内有效

}

1. 不存在变量提升:

Console.log(A);

Let A = 1;//这样会报错的，因为let定义的不会变量提升，打印的时候没有定义。

3.let变量不能重复定义变量，例如

Let a = 1;

Let b=2；//这样会报错

# Const命令:只能定义一个常量不能改，声明的时候必须赋值。

但是const定义的是一个数组的时候能够添加，例如：

Const k = { a:1 };

K.b= 3;

Console.log(K);//K指向的是{}里面的东西所以他是能变的

# 数组结构赋值：

let [a, b, c] = [1, 2, 3];

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, 4];

x // 1

y // 4

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

head // 1

tail // [2, 3, 4]

let [x, y, ...z] = ['a'];

x // "a"

y // undefined

z // []

let a,b;

({a,b}={a:1,b:2})

A//1

B//2

Let a,b,c;

[a,b,c=3]=[1,2];

a//1

B//2

C//3

‘...’可以截取后面的数组。 ’,’ 代表一个值。 ，和...是可以混合使用的。

默认值(一个数组成员不严格等于undefined，默认值是不会生效的。只有等于undefined的时候默认值才会生效)。

如果解构不成功，变量的值就等于undefined。例如：

let [foo] = [];//undefined

let [bar, foo] = [1];//undefined

另一种情况是不完全解构，即等号左边的模式，只匹配一部分的等号右边的数组。这种情况下，解构依然可以成功。

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

x // 1

y // 2

let [a, [b], d] = [1, [2, 3], 4];

a // 1

b // 2

d // 4

对于 Set 结构，也可以使用数组的解构赋值。

let [x, y, z] = new Set(['a', 'b', 'c']);

x // "a"

# 对象结构赋值：

let o = { p:42 , q:true };

Let { p , q } = o;

p // 42;

q // true;

Let {a:10,b:5} = {a:3}

a //3

b //5

Let metaDate = {

Title:’abc’,

Test:[{

Title:’test’,

Desc:’dshlsf’

}]

}

Let { title:destitle , Test:[{retitle:Title}] } = metaDate;

destitle //abc

Retitle //test

# 正则方法：

## 1.修饰符：

如果RegExp构造函数第一个参数是一个正则对象，那么可以使用第二个参数指定修饰符

new RegExp(/abc/ig, 'i').flags

// "i"

原有正则对象的修饰符是ig，它会被第二个参数i覆盖。

## **2.字符串的正则方法**

String.prototype.match调用RegExp.prototype[Symbol.match]

String.prototype.replace调用RegExp.prototype[Symbol.replace]

String.prototype.search调用RegExp.prototype[Symbol.search]

String.prototype.split调用RegExp.prototype[Symbol.split]

## u修饰符（含义为“Unicode 模式”）

/^\uD83D/u.test('\uD83D\uDC2A') // false/^\uD83D/.test('\uD83D\uDC2A') // true

一旦加上u修饰符号，就会修改下面这些正则表达式的行为。

### 点字符

var s = '吉';

/^.$/.test(s)

// false

/^.$/u.test(s)

// true

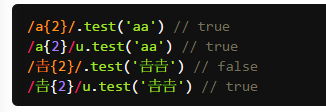
### 上面代码表示，如果不添加u修饰符，正则表达式就会认为字符串为两个字符，从而匹配失败。

### **Unicode 字符表示法**

/\u{61}/.test('a') // false/\u{61}/u.test('a') // true/\u{20BB7}/u.test('馉') // true

上面代码表示，如果不加u修饰符，正则表达式无法识别\u{61}这种表示法，只会认为这匹配 61 个连续的u。

### 量词



使用u修饰符后，所有量词都会正确识别码点大于0xFFFF的 Unicode 字符。

### 预定义模式

/^\S$/.test('馉') // false

/^\S$/u.test('馉') // true

### i 修饰符

/[a-z]/i.test('\u212A') // false

/[a-z]/iu.test('\u212A') // true

上面代码中，不加u修饰符，就无法识别非规范的K字符。

## y修饰符

y修饰符的作用与g修饰符类似，也是全局匹配，后一次匹配都从上一次匹配成功的下一个位置开始。不同之处在于，g修饰符只要剩余位置中存在匹配就可，而y修饰符确保匹配必须从剩余的第一个位置开始，这也就是“粘连”的涵义。

var s = 'aaa\_aa\_a';

var r1 = /a+/g;

var r2 = /a+/y;

r1.exec(s) // ["aaa"]

r2.exec(s) // ["aaa"]

r1.exec(s) // ["aa"]

r2.exec(s) // null

## Stricky属性

表示是否设置了y修饰符。

## Flags属性

返回正则表达式的修饰符。

## s修饰符

正则新增特性:

G：在上一次匹配的之后的寻找相同字符开始匹配（直接跳过\_找到ss）

Y：匹配的第一个必须是匹配的下一个字符开始的(sss之后有个\_不能跳过)

{

Let s=”sss\_ss\_s”;

Let a1 = /b+/g;

Let a2 = /b+/y;

Console.log(‘one’,a1.exec(s),a2.exec(s));//sss sss

Console.log(‘two,a1.exec(s),a2.exec(s));//ss null

Let regex = new RegExp(‘xyz’,’’i’);

Let regex = new RegExp(‘xyz’,’’i’);

Console.log(a1.sticky,a2.sticky);//false true(是否开启y修饰符)

}

u修饰符：

{

console.log(‘u-1’,/^\uD83D/.test(‘\uD83D\uDC2A));

}

字符串：

Include

startsWITH

EndsWith

{

Let name = ‘list’;

Let info = ‘hello world’;

Let m = `i am ${name},${info}`;//1左右按键

}

补日期的，7-8——07-08

Console.log(‘1’.padStart(2,’0’));

Console.log(‘1’.padEnd(2,’0’));

# 字符串的拓展

## 字符串的unicode表示法

共有 6 种方法可以表示一个字符

'\z' === 'z' // true

'\172' === 'z' // true

'\x7A' === 'z' // true

'\u007A' === 'z' // true

'\u{7A}' === 'z' // true

## codePointAt方法

codePointAt方法会正确返回 32 位的 UTF-16 字符的码点。对于那些两个字节储存的常规字符，它的返回结果与charCodeAt方法相同。

codePointAt方法返回的是码点的十进制值，如果想要十六进制的值，可以使用toString方法转换一下。

let s = '馉a';

s.codePointAt(0).toString(16) // "20bb7"

s.codePointAt(2).toString(16) // "61"

## String.fromCharCode

# 数值的拓展

## 二进制与八进制的表示法

ES6 提供了二进制和八进制数值的新的写法，分别用前缀0b（或0B）和0o（或0O）表示。

0b111110111 === 503 // true

0o767 === 503 // true

## **Number.isFinite()和Number.isNaN()**

# 函数的拓展

## 默认值

默认值后面不能再有没有默认值的写法。例如：

这个y有默认值，后面若在给一个没有默认值的z就会出错，给一个z=”htl”就不会出错。

{

function test( x, y = ’hello’ ){

Console.log(x,y)

}

}

{

Let x = ‘test’;

function test( x , y=x ){

Console.log(x,y)

}

test(‘kill’);//kill kill

test();//undefined undefined

}

{

Let x = ‘test’;

function test( c, y = x ){

Console.log(c,y)

}

test();//test test 默认去外面x的值

}

如果传入undefined，将触发该参数等于默认值，null则没有这个效果。

function foo(x = 5, y = 6) {

console.log(x, y);}

foo(undefined, null)

// 5 null

上面代码中，x参数对应undefined，结果触发了默认值，y参数等于null，就没有触发默认值。

## rest参数

rest作用：不确定数组中有多少值的时候，把一系列的参数转换成了数组。是1开始的。

rest后面不能再有别的参数。

{

function test( ...arg ){

for(let v of arg){

Console.log(‘reset’,v);

}

test(1,2,3,4,’a’);

}

test();//reset 1 reset 2 reset 3 reset 4 reset a

}

{

Console.log(...[1,2,4]);//1 2 4

Console.log(‘a’,...[1,2,4]);//a 1 2 4

}

## 箭头函数（函数名+函数参数+函数返回值）

{

Let srrow = v => v\*2 //function srrow(v){ return v\*2 }

Let arrow2 = () => 5

console.log(srrow(3)) // 6

console.log(arrow2()) // 5

}

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，否则会报错。

// 报错let getTempItem = id => { id: id, name: "Temp" };

// 不报错let getTempItem = id => ({ id: id, name: "Temp" });

箭头函数有几个使用注意点。

1. 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。（尤其值得注意。this对象的指向是可变的，但是在箭头函数中，它是固定的。）

箭头函数导致this总是指向函数定义生效时所在的对象（本例是{id: 42}），所以输出的是42

function foo() {

setTimeout(() => {

console.log('id:', this.id);

}, 100);}

var id = 21;

foo.call({ id: 42 });

// id: 42

（2）不可以当作构造函数，也就是说，不可以使用new命令，否则会抛出一个错误。

（3）不可以使用arguments对象，该对象在函数体内不存在。如果要用，可以用 rest 参数代替。

（4）不可以使用yield命令，因此箭头函数不能用作 Generator 函数。

## 伪调用（）

{

function tail(x){

Console.log(‘tail’,x);

}

function fx(x){

return tail(x)

}

fx(123) //tail 123

}

## 函数的length属性：

指定了默认值后，length属性将失真。

(function (a) {}).length // 1

(function (a = 5) {}).length // 0

(function (a, b, c = 5) {}).length // 2

length属性的返回值，等于函数的参数个数减去指定了默认值的参数个数。

如果设置了默认值的参数不是尾参数，那么length属性也不再计入后面的参数了。

(function (a = 0, b, c) {}).length // 0

(function (a, b = 1, c) {}).length // 1

## 作用域

在这个作用域里面，默认值变量x指向第一个参数x，而不是全局变量x，所以输出是2。

var x = 1;

function f(x, y = x) {

console.log(y);}

f(2) // 2

参数y = x形成一个单独的作用域。这个作用域里面，变量x本身没有定义，所以指向外层的全局变量x。函数调用时，函数体内部的局部变量x影响不到默认值变量x。

如果此时，全局变量x不存在，就会报错。

let x = 1;

function f(y = x) {

let x = 2;

console.log(y);}

f() // 1

# 数组的拓展

扩展运算符（spread）是三个点（...）

## 复制数组

a2都是a1的克隆。

const a1 = [1, 2];

const a2 = [...a1];// 写法一

const [...a2] = a1;// 写法二

合并数组

// ES5[1, 2].concat(more)

// ES6[1, 2, ...more]

var arr1 = ['a', 'b'];var arr2 = ['c'];var arr3 = ['d', 'e'];

// ES5的合并数组arr1.concat(arr2, arr3);

// [ 'a', 'b', 'c', 'd', 'e' ]

// ES6的合并数组[...arr1, ...arr2, ...arr3]

// [ 'a', 'b', 'c', 'd', 'e' ]

# 对象拓展

Let a = ‘b’; [a] 就是b的意思。

## 函数的name属性

const person = {

sayName() {

console.log('hello!');

},};

person.sayName.name // "sayName"

方法的name属性返回函数名（即方法名）

浅拷贝：object.assign , 只有自身对象的属性，如果又继承不会拷贝。