未整理前原文见素材es6（提案都不整理）

0、前言

1、ECMAScript 6简介

（6）babel转码器

babel可以将es6转为es5。

项目根目录下有配置文件.babelrc，基本格式如下：

{

"presets": [], 用来设置转码规则

"plugins": [] 用来设置插件

}

babel-cli：命令行转码

$ babel 待转文件 --out-file或-o 转后文件

$ babel 待转目录 --out-dir或-d 转后目录

-s参数生成source map文件

$ npm install --global babel-cli 环境依赖

$ npm install --save-dev babel-cli 项目依赖

项目依赖可以改写package.json文件：

{

// ...

"devDependencies": {

"babel-cli": "^6.0.0"

},

"scripts": {

"build": "babel src -d lib"

},

}

执行$ npm run build进行转码

babel-node：babel-cli自带babel-node命令，支持ES6的REPL环境，可以直接运行ES6代码。

$ babel-node es6代码或文件

babel-register：为require命令加上钩子。此后，每当使用require加载.js、.jsx、.es和.es6后缀名的文件，就会先用Babel进行转码。

$ npm install --save-dev babel-register

当前文件（当前文件不会被转码。实时转码。只适合开发环境使用）：

require("babel-register");

require("需要被转码文件");

babel-core：代码调用 babel 的 api 进行转码

babel-polyfill：babel转换语法，babel-polyfill转换API

2、let 和 const 命令

for循环控制是父作用域，循环体是子作用域

es5有全局作用域和函数作用域，es6有全局作用域和函数作用域和块级作用域

var的环境最低保存在函数作用域function内，let的环境最低保存在块级作用域{}内

var可不声明，可先用再声明，可多次声明，let必须先声明再用，必须单次声明

const保证变量指向的内存地址不得改变。const声明的变量赋值后不能再改变，const声明的变量若为对象或数组，则对象或数组内的属性可以改变

顶层对象：浏览器环境指的是window对象，Node环境指的是global对象

3、变量的解构赋值

解构赋值允许指定默认值。===undefined时，默认值生效

（1）数组的解构赋值

等号两侧关于数组结构满足模式匹配即可，[数组第一个值的别名，[数组第二个值的别名],数组第三个值的别名， ，数组第五个值的别名,…数组剩下值的别名]

等号右侧是可遍历结构（实现了iterator接口）即可

（2）对象的解构赋值:次序可不相同

等号两侧关于对象结构满足模式匹配即可

若变量与属性同名，解构为{属性名}

若变量与属性不同名，解构为{属性名：变量名}

let node = {

a: {

b: {

c: 1,

d: 5

}

}

};

let { a, a: { b }, a: { b: { c }} } = node;

数组本质是特殊的对象：

let arr = [1, 2, 3];

let {0 : first, [arr.length - 1] : last} = arr;

first // 1

last // 3

（3）字符串的解构赋值

let [a, b] = 'hi'; //字符串被转换成类似数组的对象

a // "h"

b // "i"

let {length} = 'hello'; //类似数组的对象都有一个length属性

length // 5

4、字符串的扩展

（1）字符的unicode表示法

"\a" === "a" // true

"\141" === "a" // true 八进制

"\x61" === "a" // true 十六进制

"\u0061" === "a" // true unicode的es5表示法

"\u{61}" === "a" // true unicode的es6表示法

若字符超过\u0000~\uFFFF则：

"\uD842\uDFB7" === 特殊字符 // true

"\u{20BB7}" === 特殊字符 // true

（2）codePointAt()

javascript内部，字符以UTF-16的格式储存，字符的unicode在\u0000~\uFFFF之内，占2个字节。

若字符的unicode大于0xFFFF，占4个字节。javascript会认为它们是两个字符。

let b = '特殊字符a';

for (let c of b) {

console.log(c.codePointAt(0).toString(16));

}

// 20bb7

// 61

（3）String.fromCodePoint()

String.fromCodePoint(0x20BB7)

// "特殊字符"

（4）字符串的遍历器接口

ES6为字符串实现了iterator接口，使得字符串可以被for...of循环遍历

（7）includes(), startsWith(), endsWith()

includes()：返回布尔值，表示是否找到了参数字符串。

startsWith()：返回布尔值，表示参数字符串是否在源字符串的头部。

endsWith()：返回布尔值，表示参数字符串是否在源字符串的尾部。

（8）repeat()

repeat()方法返回一个新字符串，表示将原字符串重复n次。

（9）padStart()，padEnd()

padStart()用于头部补全，padEnd()用于尾部补全。

（10）模板字符串

模板字符串被反引号**`**包围。可以定义多行文本和内嵌表达式，

多行文本的空格和换行都被保留

内嵌表达式被${}包围

（12）标签模板

方法名**`**a${b}c${d}e${f}**`** 等同于

方法名（['a','c','e'],b,d,f）

方法名接受的第一个数组参数有一个raw属性

标签模板可以过滤html字符串，防止用户恶意输入

（13）String.raw（）：可以对斜杠进行转义

String.raw({ raw: 'abc' }, 1, 2); 等同于

String.raw({ raw: ['a','b','c'}, 1, 2);

// 'a1b2c'

5、正则的扩展

（1）RegExp 构造函数

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

//修饰符ig会被i覆盖

（2）字符串的正则方法：共有4个

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]

（3）u 修饰符：

unicode修饰符，正确处理若字符的unicode大于0xFFFF，占4个字节。使javascript认为它们是一个字符。

（4）y 修饰符

粘连修饰符，后一次匹配从上一次匹配成功后剩余的第一个位置开始

let s = 'aaa\_aa\_a';

let r1 = /a+/g;

let r2 = /a+/y;

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

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

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

r2.exec(s) // null

（5）sticky 属性

表示是否设置了y修饰符

let a = /b/y;

a.sticky // true

（6）flags 属性

返回正则表达式的修饰符

/abc/g.flags

// "g"

/abc/g.source

// "abc"

6、数值的扩展

（1）二进制和八进制表示法

二进制数值写法：0b（或0B）

八进制数值写法：0o（或0O）

将0b和0o前缀的字符串数值转为十进制

Number('0b10') // 2

Number('0o10') // 8

（2）Number.isFinite(), Number.isNaN()

Number.isFinite()一个数值是否为有限的。非数值一律返回false

Number.isNaN()一个数值是NaN返回true，不是NaN返回false。

传统的全局方法isFinite()和isNaN()先调用Number()将非数值转为数值，再进行判断。这两个新方法只对数值有效

（3）Number.parseInt(), Number.parseFloat()

将全局方法parseInt()和parseFloat()，移植到Number对象上面，行为不变。目的是逐步减少全局性方法，使语言逐步模块化。

（4）Number.isInteger()

判断一个值是否为整数。在 JavaScript 内部，整数和浮点数是同样的储存方法，所以1和1.0被视为同一个值。

（5）Number.EPSILON

在Number对象上面，新增一个极小的常量Number.EPSILON。

（6）安全整数和Number.isSafeInteger()

javascript能够准确表示的整数范围在-2^53到2^53之间（不含两个端点）

ES6引入Number.MAX\_SAFE\_INTEGER和Number.MIN\_SAFE\_INTEGER两个常量，用来表示这个范围的上下限。

Number.isSafeInteger()用来判断一个整数是否落在这个范围之内。

（7）Math对象的扩展

Math.trunc() 去除一个数的小数部分，返回整数部分

Math.sign() 判断一个数是正数、负数、还是零

Math.cbrt() 计算一个数的立方根。

Math.clz32()一个数的32位无符号整数形式有多少个前导0。

Math.imul()返回两个数以32位带符号整数形式相乘的结果，返回的也是一个32位的带符号整数

Math.fround()返回一个数的单精度浮点数形式。

Math.hypot() 返回所有参数的平方和的平方根。

（9）指数运算符

ES2016 新增了一个指数运算符\*\*

2 \*\* 3 // 8

（10）Integer 数据类型

javascript所有数字都保存成64位浮点数

7、函数的扩展

（1）函数参数的默认值

只有参数的值是undefined才会等于默认值

指定了默认值后，函数的length属性将返回没有指定默认值的参数个数

特例：

(function(...rest) {}).length // 0

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

（2）rest 参数

...变量名，是一个数组，将剩余的参数放入数组中。这样就不需要使用arguments对象了。

（3）严格模式

只要函数参数使用了默认值、解构赋值或者扩展运算符，那么函数内部就不能显式设定为严格模式，否则会报错。

两种方法可以规避这种限制：第一种是设定全局性的严格模式，第二种是把函数包在一个无参数的立即执行函数里面。

（4）name属性

返回该函数的函数名。

function a() {}

a.name // "a"

var a = function () {};

a.name // "a"

const b = function a() {};

b.name // "a"

(new Function).name // "anonymous"

(function(){}).bind({}).name // "bound "

function foo() {};

foo.bind({}).name // "bound foo"

（5）箭头函数

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

箭头函数没有自己的new.target和super和this和arguments。

使用注意点：

（a）箭头函数体内的this对象，就是定义时所在的对象，而不是使用时所在的对象。

（b）不可以当作构造函数，即不可以使用new命令，否则会抛出一个错误。

（c）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。

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

（6）绑定this

（7）尾调用调优

尾调用：指某个函数的最后一步是调用另一个函数

尾调用优化：用内层函数的调用帧，取代外层函数的调用帧，不存在调用栈。可用于优化递归

尾调用优化只在严格模式下开启，正常模式是无效的

尾递归优化实现，原理用循环替换递归

（a）蹦床函数

function trampoline(f) {

while (f && f instanceof Function) {

f = f();

}

return f;

}

（b）

（8）函数参数的尾逗号

允许函数的最后一个参数有尾逗号

8、数组的扩展

（1）扩展运算符

...任何实现了iterator接口的对象（如数组或字符串），将该对象转为用逗号分隔的参数序列。

扩展运算符后面还可以放置表达式。

...(a> 0 ? ['b'] : [])

能够正确识别32位的Unicode字符

'x\uD83D\uDE80y'.length // 4

[...'x\uD83D\uDE80y'].length // 3

（2）Array.from()

将类似数组的对象和实现了iterable接口的对象转为数组

类似数组的对象：

let arrayLike = {

'0': 'a',

'1': 'b',

length: 2

};

常见类似数组的对象：dom操作返回的nodeList集合和函数内部的arguments对象

（3）Array.of()

将一组值，转换为数组。

（4）数组实例的 copyWithin()

在当前数组内部，用指定位置的成员覆盖其他位置，然后返回当前数组

（5）数组实例的 find() 和 findIndex()

find()方法，返回第一个符合条件的数组成员

findIndex()，返回第一个符合条件的数组成员的位置

（6）数组实例的fill()

使用给定值，填充一个数组。

（7）数组实例的 entries()，keys() 和 values()

entries()，keys()和values()用于遍历数组，返回一个遍历器对象，可以用for...of进行循环遍历，keys()是对键名的遍历、values()是对键值的遍历，entries()是对键值对的遍历。

（8）数组实例的 includes()

与字符串的includes方法类似。表示某个数组是否包含给定的值。

（9）数组的空位

ES6明确将空位转为undefined

9、对象的扩展

（1）属性的简洁表示法

ES6 允许在对象中，直接写变量。这时，属性名为变量名, 属性值为变量值。

简洁表示法的属性名总是字符串

var obj = {

class () {}

};

// 等同于

var obj = {

'class': function() {}

};

（2）属性名表达式

属性名用表达式定义时，左右两侧加[]

属性名表达式如果是一个对象，默认将对象转为字符串[object Object]

（3）方法的 name 属性

如果对象的方法使用了取值函数（getter）和存值函数（setter），则name属性不是在该方法上面，而是该方法的属性的描述对象的get和set属性上面，返回值是方法名前加上get和set。

const a = {

get b() {},

set b(x) {}

};

a.b.name

// TypeError: Cannot read property 'name' of undefined

const descriptor = Object.getOwnPropertyDescriptor(a, 'b');

descriptor.get.name // "get b"

descriptor.set.name // "set b"

如果对象的方法是一个 Symbol 值，那么name属性返回的是这个 Symbol 值的描述。

const a = Symbol('b');

const c = Symbol();

let obj = {

[a]() {},

[c]() {},

};

obj[a].name // "[b]"

obj[c].name // ""

（4）Object.is()

==，自动转换数据类型。===，NaN不等于自身，+0等于-0。

Object.is()在===基础上，NaN等于自身，+0不等于-0。即“Same-value equality”算法

（5）Object.assign()

将源对象的所有可枚举的自身属性（不包括继承属性），浅拷贝到目标对象。不会拷贝属性的赋值方法或取值方法

浅拷贝：如果源对象某个属性的值是对象，那么目标对象拷贝得到的是这个对象的引用。

如果目标对象与源对象有同名属性，或多个源对象有同名属性，则后面的属性会覆盖前面的属性。

字符串的包装对象，会产生可枚举属性。

（6）属性的可枚举性

对象的每个属性都有一个描述对象（descriptor），用来控制该属性的行为。Object.getOwnPropertyDescriptor方法可以获取该属性的描述对象。

let a= { b: 123 };

Object.getOwnPropertyDescriptor(a, 'b')

// {

// value: 123,

// writable: true,

// enumerable: true,

// configurable: true

// }

以下四个操作会忽略enumerable为false的属性。

Object.assign()

for...in循环：遍历对象自身的和继承的可枚举的属性（不含 Symbol 属性）

Object.keys()：返回对象自身的可枚举的属性

JSON.stringify()：只串行化对象自身的可枚举的属性

常见enumerable为false的属性

对象原型(Object.prototype)的toString()方法

数组的length属性

所有Class的原型的方法

（7）属性的遍历

ES6 共有5种方法可以遍历对象的属性。

（a）for...in

遍历对象自身的和继承的可枚举的属性（不含 Symbol 属性）

（b）Object.keys(obj)

返回对象自身的可枚举的属性（不含 Symbol 属性）。

（c）Object.getOwnPropertyNames(obj)

返回对象自身的所有属性（不含 Symbol 属性）。

（d）Object.getOwnPropertySymbols(obj)

返回对象自身的Symbol属性。

（e）Reflect.ownKeys(obj)

返回对象自身的所有属性（含 Symbol 属性）

以上的5种方法遍历对象的属性，都遵守同样的属性遍历的次序规则。

首先遍历所有属性名为数值的属性，按照数字排序。

其次遍历所有属性名为字符串的属性，按照生成时间排序。

最后遍历所有属性名为 Symbol 值的属性，按照生成时间排序。

（8）\_\_proto\_\_属性，Object.setPrototypeOf()，Object.getPrototypeOf()

\_\_proto\_\_属性，用来设置或读取当前对象的prototype对象。其功能最好用以下方法代替：

Object.setPrototypeOf()，设置一个对象的prototype对象

Object.getPrototypeOf()，读取一个对象的prototype对象。

（9）Object.keys()，Object.values()，Object.entries()

Object.keys(obj)

返回对象自身的可枚举的属性的键名（不含 Symbol 属性）。

Object.values(obj)

返回对象自身的可枚举的属性的键值（不含 Symbol 属性）。

Object.entries(obj)

返回对象自身的可枚举的属性的键名和键值（不含 Symbol 属性）。

（10）对象的扩展运算符

解构赋值：必须是最后一个参数，是浅拷贝，只拷贝对象自身的属性

let { a, b, ...c } = { a: 1, b: 2, d: 3, e: 4 };

a // 1

b // 2

c // { d: 3, e: 4 }

扩展运算符：

let a = { b: 1, c: 2 };

let d = { ...a }; //等同于使用Object.assign()方法

d // { b: 1, c: 2 }

自定义的属性，放在扩展运算符后面，则扩展运算符内部的同名属性会被覆盖掉。

对象的扩展运算符后面可以跟表达式。

const a = {

...(b > 1 ? {c: 1} : {}),

};

（11）Object.getOwnPropertyDescriptors()

Object.getOwnPropertyDescriptor()方法，返回指定对象指定自身属性的描述对象。

Object.getOwnPropertyDescriptors()方法，返回指定对象所有自身属性的描述对象。

10、Symbol

javascript语言的第七种数据类型：undefined、null、Boolean、String、Number、Object、Symbol 。

对象的属性名有两种类型，一种是字符串，另一种是Symbol

Symbol 值作为属性名时，该属性还是公开属性，不是私有属性。

Symbol是不可改变且唯一的

let s1 = Symbol(”key”);

let s2 = Symbol(”key”);

s1 === s2 //错

可用作对象的属性名

let s1 = Symbol();

let obj = {

[s1]:”value”

};

console.log(obj[s1]);

let getClassNameSymbol = Symbol();

class c{

[getClassNameSymbol](){

}

}

let c = new c();

c[getClassNameSymbol]();

（5）Symbol.for()，Symbol.keyFor()

Symbol.for()方法接受一个字符串作为参数，然后搜索有没有以该参数作为名称的Symbol值。如果有，就返回这个Symbol值，否则就新建并返回一个以该字符串为名称的Symbol值。Symbol.for()为Symbol值登记的名字，是全局环境的

var a= Symbol.for('b');

var c = Symbol.for('b');

a === c // true

Symbol.keyFor()方法返回一个已登记的 Symbol 类型值的key。

var s1 = Symbol.for("foo");

Symbol.keyFor(s1) // "foo"

（7）内置的Symbol值

11个内置的Symbol值，指向语言内部使用的方法。

Symbol.hasInstance 执行instanceof时调用

Symbol.isConcatSpreadable 执行Array.prototype.concat()时，表示是否可以展开

数组的默认可以展开。类似数组的对象默认不可以展开，必须手动展开。对于一个类来说，Symbol.isConcatSpreadable属性必须写成实例的属性。

Symbol.species 执行构造函数时调用

Symbol.match 执行str.match(myObject)时调用

Symbol.replace 执行String.prototype.replace时调用

Symbol.search 执行String.prototype.search时调用

Symbol.split 执行String.prototype.split时调用

Symbol.iterator 执行遍历器方法时调用

Symbol.toPrimitive 当对象被转为原始类型的值时调用

Symbol.toStringTag 执行Object.prototype.toString时调用

Symbol.unscopables 执行with时，哪些属性会被排除。

11、Set和Map数据结构

（1）Set

类似数组，但是成员没有重复

实现了iterable 接口的数据结构可以作为参数

判断两个值是否相同，使用“Same-value equality”算法

两个对象总是不相等的

Set 实例有以下属性。

Set.prototype.constructor：构造函数

Set.prototype.size：返回成员数。

Set 实例的操作方法

add(value)：添加某个值，返回Set本身。

delete(value)：删除某个值，返回删除是否成功。

has(value)：返回某个值是否为Set的成员。

clear()：清除所有成员

Set 实例的遍历方法：Set的遍历顺序就是插入顺序

keys()：返回键名的遍历器

values()：返回键值的遍历器,Set.prototype[Symbol.iterator] === Set.prototype.values

entries()：返回键值对的遍历器

forEach()：使用回调函数遍历每个成员，没有返回值。

（2）WeakSet

类似Set。但是WeakSet成员只能是对象，且WeakSet 中的对象都是弱引用，即垃圾回收机制不考虑 WeakSet 对该对象的引用。

实现了iterable 接口的对象可以作为参数

WeakSet 不可遍历

WeakSet实例的操作方法

add(value)：添加某个值

delete(value)：删除某个值

has(value)：返回某个值是否为WeakSet的成员。

（3）Map

Object 结构提供了“字符串—值”的对应，Map结构提供了“值—值”的对应

实现了Iterator 接口、且每个成员都是一个双元素的数组的数据结构可以作为参数

Map 的键是跟内存地址绑定的，只要内存地址不一样，就视为两个键。

size属性返回 Map 结构的成员总数。

set(key, value)设置键名key对应的键值为value

get(key)读取key对应的键值

has(key)某个键是否在当前 Map 对象之中。

delete(key)删除某个键

clear()清除所有成员

遍历方法：遍历顺序就是插入顺序

keys()：返回键名的遍历器。

values()：返回键值的遍历器。

entries()：返回所有成员的遍历器，map[Symbol.iterator] === map.entries

forEach()：遍历 Map 的所有成员。

（4）WeakMap

类似WeakMap。WeakMap键名只能是对象，且WeakMap 键名中的对象都是弱引用，即垃圾回收机制不考虑WeakMap键名对该对象的引用。

WeakMap不可遍历

set(key, value)设置键名key对应的键值为value

get(key)读取key对应的键值

has(key)某个键是否在当前 Map 对象之中。

delete(key)删除某个键

12、Proxy

（1）概述

let proxy = new Proxy(target, handler);

target参数表示所要拦截的目标对象，handler参数也是一个对象，用来定制拦截行为

如果属性不可配置（configurable）和不可写（writable），则该属性不能被代理

（2）Proxy 实例的方法

（a）get(target, propKey, receiver)拦截属性的读取

（b）set(target, propKey, value, receiver)拦截属性的赋值

（c）has(target, propKey)拦截HasProperty操作

（d）deleteProperty(target, propKey)拦截delete操作

（e）ownKeys(target)拦截对象自身属性的读取操作

（f）getOwnPropertyDescriptor(target, propKey)拦截Object.getOwnPropertyDescriptor

（g）defineProperty(target, propKey, propDesc)拦截Object.defineProperty、Object.defineProperties操作

（h）preventExtensions(target)拦截Object.preventExtensions()

（i）getPrototypeOf(target)拦截获取对象原型

（j）isExtensible(target)拦截Object.isExtensible

（k）setPrototypeOf(target, proto)拦截Object.setPrototypeOf操作

（l）apply(target, object, args) 拦截函数的调用、call和apply操作

（m）construct(target, args)拦截new命令

（3）Proxy.revocable()

let {proxy, revoke} = Proxy.revocable(target, handler);

revoke(); //取消Proxy实例

（4）this 问题

在 Proxy 代理的情况下，目标对象内部的this关键字会指向 Proxy 代理。

13、Reflect

（1）概述

Reflect对象的设计目的：

（a）将Object对象属于语言内部的方法放到Reflect对象上。

（b）修改某些Object方法的返回结果，使其更合理。

（c）让Object操作都变成函数行为。

（d）Reflect对象的方法与Proxy对象的方法一一对应。

（2）静态方法

Reflect.get(target,name,receiver)

Reflect.set(target,name,value,receiver)

Reflect.has(target,name)

Reflect.deleteProperty(target,name)

Reflect.ownKeys(target)

Reflect.getOwnPropertyDescriptor(target, name)

Reflect.defineProperty(target,name,desc)

Reflect.preventExtensions(target)

Reflect.getPrototypeOf(target)

Reflect.isExtensible(target)

Reflect.setPrototypeOf(target, prototype)

Reflect.apply(target,thisArg,args)

Reflect.construct(target,args)

14、Promise 对象

（1）Promise 的含义

异步操作从Pending变为Resolved或从Pending变为Rejected。状态一旦改变就不会再变，任何时候都可以得到这个结果。

一旦新建就会立即执行，无法中途取消。

如果不设置回调函数，Promise内部抛出的错误，不会反应到外部。

当处于Pending状态时，无法得知目前进展到哪一个阶段

（2）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);

}

});

promise.then(function(value) {

// success

}, function(error) {

// failure

});

（3）Promise.prototype.then()

then方法返回的是一个新的Promise实例。因此then方法后面再调用另一个then方法……

（4）Promise.prototype.catch()

是.then(null, rejection)的别名

Promise 对象的错误会一直向后传递，直到被捕获为止

（5）Promise.all()

Promise.all方法的参数必须实现了Iterator 接口，且返回的每个成员都是 Promise 实例，如果不是，就先调用Promise.resolve方法，将该成员转为 Promise 实例

Promise.all方法的参数成员全为Resolved，才返回Resolved。

（6）Promise.race()

Promise.race方法的参数必须实现了Iterator 接口，且返回的每个成员都是 Promise 实例，如果不是，就先调用Promise.resolve方法，将该成员转为 Promise 实例

Promise.race的返回状态由参数中率先改变状态的Promise 实例决定

（7）Promise.resolve()

将现有对象转为Promise对象

参数分成四种情况：

（a）参数是Promise实例。直接返回这个实例。

（b）参数是thenable对象（具有then方法的对象）。将这个对象转为Promise对象，然后立即执行thenable对象的then方法。

（c）参数不是thenable对象，或根本就不是对象。返回新的Promise对象，状态为Resolved。

（d）不带有任何参数。返回新的Promise对象，状态为Resolved。

立即resolve的Promise对象，是在本轮“事件循环”（event loop）的结束时，而不是在下一轮“事件循环”（event loop）的开始时。

（8）Promise.reject()

返回一个新的 Promise 实例，该实例的状态为rejected。

15、Iterator 和 for...of 循环

（1）Iterator（遍历器）的概念

javascript数据集合Array、Map、Set和Object。

Iterator 的作用有三个：一是为各种数据结构，提供统一访问接口；二是使数据结构的成员能够按某种次序排列；三是ES6创造了一种新的遍历命令for...of循环，Iterator接口主要供for...of执行

每一次调用next方法，都会返回一个包含value和done两个属性的对象。其中，value属性是当前成员的值，done属性是一个布尔值，表示遍历是否结束。done: false和value: undefined属性都是可以省略的

（2）默认 Iterator 接口

一个数据结构只要具有Symbol.iterator属性，就可以认为是“可遍历的”（iterable）。Symbol.iterator属性本身是一个函数，即当前数据结构默认的遍历器生成函数

原生具有Iterator接口的数据结构有：

Array、Map、Set

String、TypedArray、函数的 arguments 对象、DOM NodeList 对象、Generator 对象

类似数组的对象（存在数值键名和length属性），部署 Iterator 接口的简便方法是该对象的Symbol.iterator方法直接引用数组的Symbol.iterator方法。

（3）调用 Iterator 接口的场合

（a）解构赋值

（b）扩展运算符

（c）yield\*

（d）任何可以接受数组作为参数的场合

Array.from()

Map(), Set(), WeakMap(), WeakSet()

Promise.all()

Promise.race()

for...of

（4）字符串的 Iterator 接口

（5）Iterator接口与Generator函数

（6）遍历器对象的return()，throw()

如果for...of循环提前退出（出错或者break或continue），就会调用return方法。可以用于清理或释放资源。return方法必须返回一个对象

throw方法主要配合Generator函数使用，一般的遍历器对象用不到这个方法

（7）for...of循环

会正确识别32位 UTF-16 字符。

与其他遍历语法的比较

forEach：break和return失效

for...of：break、return和continue有效

16、Generator 函数的语法

（1）简介

Generator 函数是 ES6 提供的一种异步编程解决方案

两个特征。一是，function关键字与函数名之间有一个星号。二是，函数体内部使用yield表达式，定义不同的内部状态

调用 Generator 函数，返回一个遍历器对象，代表 Generator 函数的内部指针。以后，每次调用遍历器对象的next方法，就会返回一个有着value和done两个属性的对象。value属性表示当前的内部状态的值，是yield（或return）表达式后面那个表达式的值；done属性是一个布尔值，表示是否遍历结束。

yield表达式如果用在另一个表达式之中，必须加括号。

（2）next 方法的参数

next方法可以带一个参数，该参数被当作上一个yield表达式的返回值。

（3）for...of 循环

一旦next方法的返回对象的done属性为true，for...of循环就会中止，且不包含该返回对象

（4）Generator.prototype.throw()

Generator 函数返回的遍历器对象，都有一个throw方法，可以在Generator 函数体外抛出错误。Generator 函数体内捕获错误。

一旦 Generator 执行过程中抛出错误，若被内部捕获，会附带执行一次next方法。

一旦 Generator 执行过程中抛出错误，若没有被内部捕获，就不会再执行下去了。如果此后还调用next方法，将返回一个value属性等于undefined、done属性等于true的对象

（5）Generator.prototype.return()

{ value: return方法参数值, done: true }

如果 Generator 函数内部有try...finally代码块，那么return方法会推迟到finally代码块执行完再执行。

（6）yield\* 表达式

yield\*表达式，用来在一个 Generator 函数里面执行另一个 Generator 函数

如果被代理的 Generator 函数有return语句，那么就可以向代理它的 Generator 函数返回数据。

（7）作为对象属性的Generator函数

（8）Generator 函数的this

Generator 返回的遍历器是 Generator 函数的实例，也继承了 Generator 函数的prototype对象上的方法。

function\* gen() {

this.a = 1;

yield this.b = 2;

yield this.c = 3;

}

function F() {

return gen.call(gen.prototype);

}

var f = new F();

f.next(); // Object {value: 2, done: false}

f.next(); // Object {value: 3, done: false}

f.next(); // Object {value: undefined, done: true}

f.a // 1

f.b // 2

f.c // 3

17、Generator 函数的异步应用