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