ECMAScript 6（以下简称ES6）是JavaScript语言的下一代标准。因为当前版本的ES6是在2015年发布的，所以又称ECMAScript 2015。

Babel是一个广泛使用的ES6转码器，可以将ES6代码转为ES5代码，从而在现有环境执行。

let：*let*实际上为JavaScript新增了块级作用域。用它所声明的变量，只在let命令所在的代码块（花括号）内有效；与var的区别是，var用来做循环的计数变量，会泄露成全局变量，在外部调用的值是循环完成后的值。

const：声明变量，但声明的是常量，一旦声明，常量的值不能改变。

class：定义一个类，可定义构造方法在其中，构造方式*this*关键字指向实例。*constructor*内定义的方法和属性是实例对象自己的，而*constructor*外定义的方法和属性则是所有实例对象可以共享的。

extends：*class*之间可用*extends*关键字实现继承

super：指代父类的实例（即父类的this对象）。子类必须在*constructor*中调用*super*方法，否则新建实例就会报错，因为子类没有自己的*this*对象，而是继承父类的*this*对象，然后对其进行加工。

ES6的继承机制，实质是先创造父类的实例对象this（所以必须先调用super方法），然后再用子类的构造函数修改this。

arrow function：箭头函数 (i)=> i+1
箭头函数体内的this对象指向定义时所在的对象（箭头函数内无自己的this，其this继承外面的作用域）

template string：用反引号（`）来标识起始，用*${}*来引用变量，所有的空格和缩进都会被保留在输出之中

destructuring:从数组和对象中提取值，对变量进行赋值，这被称为解构（Destructuring）。

```
//  解构
let cat = 'ken'
let dog = 'lili'
let zoo = {cat, dog}
console.log(zoo)   //Object {cat: "ken", dog: "lili"}

// 赋值
let dog = {type: 'animal', many: 2}
let { type, many} = dog
console.log(type, many)   //animal 2
```

default：变量未赋值时给该变量一个默认值

```
function animal(type = 'cat') {
   console.log(type)
}
animal()   //cat
```

rest：过滤变量

```
function animals(once, ...types){
   console.log(types)
}
animals('cat', 'dog', 'fish')     //[ "dog", "fish"]
```

es6的模块功能， import export

之前的js无模块化体系，无法拆分把庞大的js工程拆分为多个功能相对独立但相互依赖的小工程，再将这些小工程连接在一起，解决方案　　：CommonJS，AMD。

require.js

```
/* 定义模块    */
//content.js
define('content.js', function(){
```

```
      return 'A cat';
})

/* 载入模块 */
//index.js
require(['./content.js'], function(animal){
      console.log(animal); //A cat
})
```

CommonJS

```
//index.js
var animal = require('./content.js')

//content.js
module.exports = 'A cat'
```

ES6

```
//index.js
import animal from './content'

//content.js
export default 'A cat'
```

ES6 module的其他高级用法

输出类型：export命令除了输出变量，还可以输出函数，甚至是类（react的模块基本都是输出类）

```
//content.js

export default 'A cat'
export function say(){
   return 'Hello!'
}
export const type = 'dog';

//index.js

import { say, type } from './content'
let says = say()
```

```
console.log(`The ${type} says ${says}`) //The dog says Hello
```

变量名：大括号里面的变量名，必须与被导入模块（content.js）对外接口的名称相同，当然可能会出现重名的情况，es6中可以用as实现一键换名。

```
//index.js

import animal, { say, type as animalType } from './content'
let says = say()
console.log(`The ${animalType} says ${says} to ${animal}`)
//The dog says Hello to A cat
```

模块的整体加载：除了指定加载某个输出值，还可以使用整体加载，即用星号（*）指定一个对象，所有输出值都加载在这个对象上面。 通常星号*结合as一起使用比较合适。

```
//index.js

import animal, * as content from './content'
let says = content.say()
console.log(`The ${content.type} says ${says} to ${animal}`)
//The dog says Hello to A cat
```

部分使用： 考虑下面的场景：上面的content.js一共输出了三个变量（default, say, type）,假如我们的实际项目当中只需要用到type这一个变量，其余两个我们暂时不需要。我们可以只输入一个变量：

```
import { type } from './content'
```

由于其他两个变量没有被使用，我们希望代码打包的时候也忽略它们，抛弃它们，这样在大项目中可以显著减少文件的体积，ES6实现了改功能。不过webpack、browserify都还不支持这一功能...

Promise/A+ 规范