ES6/7精选48道面试题（https://github.com/minsion）

面试题 1. ES5、ES6（ES2015）有什么区别?

推荐指数：★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

ES5指的是ECMScript的第五个版本，发布于2009年，是目前最广泛使用的JavaScript版本。
ES6是ECMScript的第六个版本，也成为ES2015，发布于2015年，引入了许多新的语言特性和语法糖。
ES2015是ES6的官方名称，但是由于ES6引入了太多的新特性，因此人们通常使用ES2015来指代ES6。

ES6新增特性：
let 、const定义块级作用域
箭头函数
解构赋值
扩展运算符
常见的数组的方法、伪数组
模板字符串
class类
参数设置默认值
promise
for...of 、for...in
ES6相对于ES5的主要区别包括：
新的语法特性，如箭头函数、类、模板字符串、解构赋值等。
新的数据类型，如Set、Map、Symbol等。
新的迭代器和生成器，使得处理数据集合更加方便。
新的模块化系统，使得代码的组织和管理更加容易。
新的Promise对象，使得异步编程更加简单和可读。
新的默认参数和剩余参数语法，使得函数的定义和调用更加灵活。

总的来说，ES6引入了许多新的特性和语法糖，使得JavaScript的编程体验更加现代化和高效。

面试题 2. 解释babel是什么，有什么作用?

推荐指数：★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

Babel 是一个 JavaScript 编译器，它可以将 ECMAScript 2015+ 版本的代码转换成向后兼容的 JavaScript 代码，以便在现有的浏览器中运行。Babel 可以帮助开发者使用最新的 JavaScript 语言特性，而不用担心浏览器兼容性问题。

Babel 的主要作用是将 ECMAScript 2015+ 的代码转换成低版本的 JavaScript 代码。这些新特性包括箭头函数、解构赋值、模板字符串、let 和 const 等等。Babel 还支持转换 JSX 语法，使得 React 的代码可以在浏览器中运行。

Babel 可以使用插件来扩展其功能，例如，它可以将 TypeScript 代码转换成 JavaScript 代码，或者使用插件来支持新的 ECMAScript 特性。Babel 还可以与许多构建工具（如 Webpack、Rollup 等）集成，以便在构建过程中自动转换代码

面试题 3. 简述ES6 let有什么用，有了var为什么还要用let?

推荐指数： ★★★★★ 试题难度： 高难 试题类型： 原理题

试题回答参考思路：

在ES6之前，声明变量只能用var，var方式声明变量其实是很不合理的，准确的说，是因为ES5里面没有块级作用域是很不合理的，甚至可以说是一个语言层面的bug(这也是很多c++、java开发人员看不懂，也瞧不起JS语言的劣势之一)。没有块级作用域回来带很多难以理解的问题，比如for循环var变量泄露，变量覆盖等问题。let 声明的变量拥有自己的块级作用域，且修复了var声明变量带来的变量提升问题。

在ES6之前，我们都是用var来声明变量，而且JS只有函数作用域和全局作用域，没有块级作用域，所以{}限定不了var声明变量的访问范围.

```
{
var i = 9;
}
console.log(i); // 9
```

ES6新增的let，可以声明块级作用域的变量。

```
{
let i = 9;
}
console.log(i); // Uncaught ReferenceError: i is not defined
```

例如：let在for循环中的使用
let非常适合在for循环中使用,Js中的for循环体比较特殊，每次循环都会创建出一个新的且独立的块级作用域。使用let声明变量传入到for循环体的作用域后，不会改变，不会受到外界的影响。
例如：

```
var a = [];
for(var i=0;i<3;i++){
a[i] = function(){
console.log(i);
}
}
a[0](); // 3
```

```
a[1](); // 3
a[2](); // 3
```
结果并不是我们想象到的0，1，2
这是因为使用var声明的时候，i是全局变量，每次循环，都会修改i的值，相当于给i重新赋值了，因此最后输出后的i是for循环完后的i的值，因此输出结果都是3
如果我们把var改为let后：

```
var a = [];
for(let i=0;i<3;i++){
a[i] = function(){
console.log(i);
}
}
a[0](); // 0
a[1](); // 1
a[2](); // 2
```
这就得到了我们想要的结果
使用let声明时，let作用的是块作用域，for的每次循环就是一个块级作用域，因此每次循环，就相当于声明了一个新的变量i，不会受到外界的干扰。

let没有变量提升
变量提升:用let声明不存在变量提升，必须等let执行完毕之后，变量才可以使用，否则会报错Uncaught ReferenceError;
例如：

```
console.log(a);
let a= '1'; //Uncaught ReferenceError
```
而var则存在变量提升;无论在开头还是结尾使用var声明都可以使用；

```
console.log(a)
var a; // undefined
```
let在同一作用域不可重复声名

```
{
let a = 1;
let a = 2;
}
// 重复声名会报错：Uncaught SyntaxError: Identifier 'a' has already been declared
```
换成var则不会报错

```
{
var a = 1;
var a = 2;
}
```

面试题 4. 简述ES6对String字符串类型做的常用升级优化?

推荐指数：★★★★★ 试题难度：初级 试题类型：原理题 ▸

试题回答参考思路：

1、优化部分：
ES6新增了字符串模板，在拼接大段字符串时，用反斜杠(`)取代以往的字符串相加的形式，能保留所有空格和换行，使得字符串拼接看起来更加直观，更加优雅。

2、升级部分:
ES6在String原型上新增了includes()方法，用于取代传统的只能用indexOf查找包含字符的方法(indexOf返回-1表示没查到不如includes方法返回false更明确，语义更清晰)，此外还新增了startsWith(), endsWith(), padStart(),padEnd(),repeat()等方法，可方便的用于查找，补全字符串

includes()
includes() 方法用于判断字符串是否包含指定的子字符串。如果找到匹配的字符串则返回 true，否则返回 false。注意： includes() 方法区分大小写。

语法：string.includes(searchvalue, start)
searchvalue：必需，要查找的字符串
start：可选，设置从那个位置开始查找，默认为 0。
返回值：Boolean

```
let str = 'hello 树先生'
let n = str.includes('hello')
console.log(n) //true
```

startsWith()
startsWith() 方法用于检测字符串是否以指定的子字符串开始。如果是以指定的子字符串开头返回 true，否则 false。startsWith() 方法对大小写敏感。
语法：string.startsWith(searchvalue, start)
searchvalue：必需，要查找的字符串。
start：可选，设置从那个位置开始查找，默认为 0。
返回值：Boolean

```
let str = 'hello 树先生'
let n = str.startsWith('hello')
console.log(n)//true
```

endsWith()
方法用来判断当前字符串是否是以指定的子字符串结尾的（区分大小写）。
如果传入的子字符串在搜索字符串的末尾则返回 true，否则将返回 false。
语法同startsWith()类似

padStart()方法,padEnd()方法
ES2017 引入了字符串补全长度的功能。如果某个字符串不够指定长度，会在头部或尾部补全。padStart()用于头部补全，padEnd()用于尾部补全。
padStart()和padStart()一共接受两个参数，第一个参数用来指定字符串的最小长度，第二个参数是用来补全的字符串。

```
let str = 'hao'
let b = str.padStart(5,'ni')
let c = str.padStart(3,'ni')
let d = str.padStart(1,'ni')
console.log(b); //nihao
console.log(c); //hao
console.log(d); //hao

let e = str.padEnd(5,'ni')
let f = str.padStart(3,'ni')
console.log(e); //haoni
console.log(f); //hao
```

面试题 5. 简述ES6对Array数组类型做的常用升级优化?

推荐指数: ★★★★★ 试题难度: 中级 试题类型: 原理题

试题回答参考思路:

1、优化部分:

a. 数组解构赋值。ES6可以直接以let [a,b,c] = [1,2,3]形式进行变量赋值，在声明较多变量时，不用再写很多let(var),且映射关系清晰，且支持赋默认值。

b. 扩展运算符。ES6新增的扩展运算符(...)(重要),可以轻松的实现数组和松散序列的相互转化，可以取代arguments对象和apply方法，轻松获取未知参数个数情况下的参数集合。（尤其是在ES5中，arguments并不是一个真正的数组，而是一个类数组的对象，但是扩展运算符的逆运算却可以返回一个真正的数组）。扩展运算符还可以轻松方便的实现数组的复制和解构赋值（let a = [2,3,4]; let b = [...a]）。

2、升级部分:
ES6在Array原型上新增了find()方法，用于取代传统的只能用indexOf查找包含数组项目的方法,且修复了indexOf查找不到NaN的bug([NaN].indexOf(NaN) === -1).此外还新增了copyWithin(), includes(), fill(),flat()等方法，可方便的用于字符串的查找，补全,转换等。

面试题 6. 简述ES6对Number数字类型做的常用升级优化?

推荐指数: ★★★★★ 试题难度: 中级 试题类型: 原理题

试题回答参考思路:

1、优化部分:
ES6在Number原型上新增了isFinite(), isNaN()方法，用来取代传统的全局isFinite(), isNaN()方法检测数值是否有限、是否是NaN。ES5的isFinite(), isNaN()方法都会先将非数值类型的参数转化为Number类型再做判断，这其实是不合理的，最造成isNaN('NaN') === true的奇怪行为--'NaN'是一个字符串，但是isNaN却说这就是NaN。而

Number.isFinite()和Number.isNaN()则不会有此类问题(Number.isNaN('NaN') === false)。 (isFinite()同上)

2、升级部分:
ES6在Math对象上新增了Math.cbrt(), trunc(), hypot()等等较多的科学计数法运算方法，可以更加全面的进行立方根、求和立方根等等科学计算。

面试题 7. 简述ES6对Object类型做的常用升级优化？（重要）

推荐指数: ★★★★★ 试题难度: 中级 试题类型: 原理题

试题回答参考思路:

1、优化部分:
a. 对象属性变量式声明。ES6可以直接以变量形式声明对象属性或者方法，。比传统的键值对形式声明更加简洁，更加方便，语义更加清晰。
let [apple, orange] = ['red appe', 'yellow orange']; let myFruits = {apple, orange}; // let myFruits = {apple: 'red appe', orange: 'yellow orange'}; 复制代码
尤其在对象解构赋值(见优化部分b.)或者模块输出变量时，这种写法的好处体现的最为明显:

let {keys, values, entries} = Object; let MyOwnMethods = {keys, values, entries}; // let MyOwnMethods = {keys: keys, values: values, entries: entries} 复制代码
可以看到属性变量式声明属性看起来更加简洁明了。方法也可以采用简洁写法:
let es5Fun = { method: function(){} }; let es6Fun = { method(){} } 复制代码
b. 对象的解构赋值。 ES6对象也可以像数组解构赋值那样，进行变量的解构赋值:

let {apple, orange} = {apple: 'red appe', orange: 'yellow orange'}; 复制代码
c. 对象的扩展运算符(...)。 ES6对象的扩展运算符和数组扩展运算符用法本质上差别不大，毕竟数组也就是特殊的对象。对象的扩展运算符一个最最常用也最好用的用处就在于可以轻松的取出一个目标对象内部全部或者部分的可遍历属性，从而进行对象的合并和分解。

let {apple, orange, ...otherFruits} = {apple: 'red apple', orange: 'yellow orange', grape: 'purple grape', peach: 'sweet peach'}; // otherFruits {grape: 'purple grape', peach: 'sweet peach'} // 注意: 对象的扩展运算符用在解构赋值时，扩展运算符只能用在最有一个参数(otherFruits后面不能再跟其他参数) let moreFruits = {watermelon: 'nice watermelon'}; let allFruits = {apple, orange, ...otherFruits, ...moreFruits}; 复制代码
d. super 关键字。ES6在Class类里新增了类似this的关键字super。同this总是指向当前函数所在的对象不同，super关键字总是指向当前函数所在对象的原型对象。

2、升级部分:
a. ES6在Object原型上新增了is()方法，做两个目标对象的相等比较，用来完善'==='方法。'==='方法中NaN === NaN //false其实是不合理的，Object.is修复了这个小bug。(Object.is(NaN, NaN) // true)
b. ES6在Object原型上新增了assign()方法，用于对象新增属性或者多个对象合并。
const target = { a: 1 }; const source1 = { b: 2 }; const source2 = { c: 3 }; Object.assign(target, source1, source2); target // {a:1, b:2, c:3} 复制代码

注意：assign合并的对象target只能合并source1、source2中的自身属性，并不会合并source1、source2中的继承属性，也不会合并不可枚举的属性，且无法正确复制get和set属性（会直接执行get/set函数，取return的值）
c. ES6在Object原型上新增了getOwnPropertyDescriptors()方法，此方法增强了ES5中getOwnPropertyDescriptor()方法，可以获取指定对象所有自身属性的描述对象。结合defineProperties()方法，可以完美复制对象，包括复制get和set属性。
d. ES6在Object原型上新增了getPrototypeOf()和setPrototypeOf()方法，用来获取或设置当前对象的prototype对象。这个方法存在的意义在于，ES5中获取设置prototype对像是通过__proto__属性来实现的，然而__proto__属性并不是ES规范中的明文规定的属性，只是浏览器各大产商“私自”加上去的属性，只不过因为适用范围广而被默认使用了，再非浏览器环境中并不一定就可以使用，所以为了稳妥起见，获取或设置当前对象的prototype对象时，都应该采用ES6新增的标准用法。
d. ES6在Object原型上还新增了Object.keys()，Object.values()，Object.entries()方法，用来获取对象的所有键、所有值和所有键值对数组。

面试题 8. 简述ES6对Function函数类型做的常用升级优化？（重要）

推荐指数：★★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

1、优化部分：
a. 箭头函数(核心)。箭头函数是ES6核心的升级项之一，箭头函数里没有自己的this,这改变了以往JS函数中最让人难以理解的this运行机制。主要优化点:
Ⅰ. 箭头函数内的this指向的是函数定义时所在的对象，而不是函数执行时所在的对象。ES5函数里的this总是指向函数执行时所在的对象，这使得在很多情况下this的指向变得很难理解，尤其是非严格模式情况下，this有时候会指向全局对象，这甚至也可以归结为语言层面的bug之一。ES6的箭头函数优化了这一点，它的内部没有自己的this,这也就导致了this总是指向上一层的this，如果上一层还是箭头函数，则继续向上指，直到指向到有自己this的函数为止，并作为自己的this。
Ⅱ. 箭头函数不能用作构造函数，因为它没有自己的this，无法实例化。
Ⅲ. 也是因为箭头函数没有自己的this,所以箭头函数 内也不存在arguments对象。（可以用扩展运算符代替）
b. 函数默认赋值。ES6之前，函数的形参是无法给默认值得，只能在函数内部通过变通方法实现。ES6以更简洁更明确的方式进行函数默认赋值。
function es6Fuc (x, y = 'default') { console.log(x, y); } es6Fuc(4) // 4, default

2、升级部分:
ES6新增了双冒号运算符，用来取代以往的bind，call,和apply。
foo::bar; // 等同于 bar.bind(foo); foo::bar(...arguments); // 等同于 bar.apply(foo, arguments); 复制代码

面试题 9. 简述ES6 Symbol的作用?

推荐指数：★★★★ 试题难度：高难 试题类型：原理题

试题回答参考思路：

Symbol是ES6引入的第七种原始数据类型（说法不准确，应该是第七种数据类型，Object不是原始数据类型之一，已更正），所有Symbol()生成的值都是独一无二的，可以从根本上解决对象属性太多导致属性名冲突覆盖的问题。对象中Symbol()属性不能被for...in遍历，但是也不是私有属性

特点

1.Symbol 的值是唯一的，用来解决命名冲突的问题

2.Symbol 值不能与其他数据进行运算，也不能自己运算 + - * /

3.Symbol 定义的对象属性不能使用for in循环遍历，但是可以使用 Reflect.ownKeys 来获取对象的所有键名

```
//创建Symbol
let s = Symbol();
// s不可见，在内部实现唯一性
let s2 = Symbol('zhangning187');
// 这里面的字符串只是一个标志，Symbol返回的值都是唯一的
let s3 = Symbol('zhangning187');
console.log(s2 == s3);
// false，确定唯一性
//Symbol.for()方法创建，这是一个对象，这种方式可以得出唯一的Symbol值
let s6 = Symbol.for ('zhangning187');
let s8 = Symbol.for ('zhangning187');
console.log(s6 ==s8);
// true 得到唯一的Symbol值
对象添加Symbol类型的属性
let zn = {
up: function() {
}
,
down: function() {
}
,
name: 'zhangning187',
age: 24
}
// 向对象zn中添加 up down 方法
// zn.up = function(){}// 这个可以添加但是不确定zn中是否存在up方法,可能会覆盖原来的up方法
// 这时候需要考虑通过Symbol添加唯一的方法
// 声明一个对象
let methods = {
up: Symbol(),
down: Symbol()
}
zn[methods.up] = function() {
console.log('我可以爬楼');
}
```

```
zn[methods.down] = function() {
console.log('我可以下楼');
}
console.log(zn);
// 已经添加唯一的方法 up down
let UZI = {
name: '自豪',
// Symbol(): function(){},// 这里不能这样直接使用， Symbol()是一个表达式，是一个动态的
[Symbol('lol')]: function() {
console.log('我会打lol');
}
,
[Symbol('篮球')]: function() {
// Symbol()中还可以添加描述字符串
console.log('我可以打篮球')
}
}
console.log(UZI);
```

面试题 10. 简述ES6 Set的作用?

推荐指数： ★★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

Set是ES6引入的一种类似Array的新的数据结构，Set实例的成员类似于数组item成员，区别是Set实例的成员都是唯一，不重复的。这个特性可以轻松地实现数组去重

Set 对象允许你存储任何类型的唯一值，无论是原始值或者是对象引用。
Set对象创建 我们可以通过构造函数来创建集合；

```
//创建一个空集合
let s = new Set();
//创建一个非空集合
let s1 = new Set([1,2,3,1,2,3]);
```

Set对象属性
属性： size 返回实例的成员总数
案例：

```
let mySet = new Set();
mySet.add(1); // Set(1) {1}
mySet.add(5); // Set(2) {1, 5} s.add(1).add(2).add(2); //链式写法
mySet.add(5); // Set(2) {1, 5} 这里体现了值的唯一性
mySet.add("some text");
// Set(3) {1, 5, "some text"} 这里体现了类型的多样性
var o = {
a: 1,
b: 2
```

```
};
mySet.add(o);
mySet.add({
a: 1,
b: 2
});
// Set(5) {1, 5, "some text", {...}, {...}}
// 这里体现了对象之间引用不同不恒等，即使值相同，Set 也能存储
console.log(mySet.size); //获取数量 5
console.log(mySet.has(3)); //false
console.log(mySet.delete(5)); // false
console.log(mySet.has(2)); //false
for (let v of mySet) {
console.log(v);
}
```

面试题 11. 简述ES6 Map的作用?

推荐指数: ★★★★ 试题难度: 初级 试题类型: 原理题

试题回答参考思路:

JavaScript的对象只能使用字符串当做key，这给它的使用带来了很大的限制。
为了解决这个问题，ES6 提供了 Map 数据结构。它类似于对象，也是键值对的集合。但是“键”的范围不限于字符串，各种类型的值（包括对象）都可以当作键。map结构提供了value与value的对应关系，是一种更完善的hash结构。
Map 也实现了 iterator 接口，所以可以使用『扩展运算符』和『for...of...』进行遍历。
Map（）方法和 Set（） 方法用法类似。
Map 对象保存键值对。任何值(对象或者原始值）都可以作为一个键或一个值。
Map对象属性
属性： size 返回实例的成员总数

案例:

```
//key 是字符串
var myMap = new Map();
var keyString = "a string";
myMap.set(keyString, "和键'a string'关联的值");
// 因为 keyString === 'a string'
console.log(myMap.get(keyString)); // "和键'a string'关联的值"
myMap.get("a string"); // "和键'a string'关联的值"


//key 是对象
var myMap = new Map();
var keyObj = {
classesName: "前端41班"
};
```

```
myMap.set(keyObj, ["张三", "李四", "王五"]);
console.log(myMap.get(keyObj)); // "张三", "李四", "王五"
myMap.get({
classesName: "前端41班"
}); // undefined, 因为 keyObj !== {}

//key 是函数
var myMap = new Map();
var keyFunc = function () {}; // 函数
myMap.set(keyFunc, "和键 keyFunc 关联的值");
console.log(myMap.get(keyFunc)); // "和键 keyFunc 关联的值"
myMap.get(function () {}) // undefined, 因为 keyFunc !== function () {}
//key 是 NaN
var myMap = new Map();
myMap.set(NaN, "not a number");
console.log(myMap.get(NaN)); // "not a number"
var otherNaN = Number("foo");
myMap.get(otherNaN); // "not a number"

//其他方法
var m = new Map();
var o = {
p: "Hello World"
};
m.set(o, "Jimor")
m.get(o) // "Jimor"

console.log(m.has(o)) // true
console.log(m.delete(o)) // true
console.log(m.has(o)) // false
//虽然 NaN 和任何值甚至和自己都不相等(NaN !== NaN 返回true)，NaN作为Map的键来说是没有区别的。
```

面试题 12. 简述ES6 Proxy的作用?

推荐指数：★★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

Proxy是ES6新增的一个构造函数，可以理解为JS语言的一个代理，用来改变JS默认的一些语言行为，包括拦截默认的get/set等底层方法，使得JS的使用自由度更高，可以最大限度的满足开发者的需求。比如通过拦截对象的get/set方法，可以轻松地定制自己想要的key或者value。下面的例子可以看到，随便定义一个myOwnObj的key,都可以变成自己想要的函数。

面试题 13. 简述ES6 Reflect的作用?

推荐指数： ★★★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

Reflect是ES6引入的一个新的对象，他的主要作用有两点，一是将原生的一些零散分布在Object、Function或者全局函数里的方法(如apply、delete、get、set等等)，统一整合到Reflect上，这样可以更加方便更加统一的管理一些原生API。其次就是因为Proxy可以改写默认的原生API，如果一旦原生API别改写可能就找不到了，所以Reflect也可以起到备份原生API的作用，使得即使原生API被改写了之后，也可以在被改写之后的API用上默认的API

面试题 14. 简述ES6 Promise 的作用?

推荐指数： ★★★★ 试题难度： 高难 试题类型： 原理题

试题回答参考思路：

Promise是ES6引入的一个新的对象，他的主要作用是用来解决JS异步机制里，回调机制产生的“回调地狱”。它并不是什么突破性的API，只是封装了异步回调形式，使得异步回调可以写的更加优雅，可读性更高，而且可以链式调用

案例

```
var promise = new Promise(function (resolve, reject) {
// 1、封装异步操作,使用setTimeout模拟异步操作
setTimeout(() => {
// 2、获取成功或失败的结果
// 异步操作成功,那么就使用resolve(成功结果)
// resolve("执行成功");
/* resolve({
name: "张三",
age: 12
}); */
// 异步操作失败,那么就使用reject(失败结果)
reject("数据获取失败");
}, 2000);
});

// 通过promise的then(),then()可以根据promise的状态指向不同的回调
// then()可以传递两个回调,第一个是成功的回调,第二个是失败的回调
// 回调函数里面定义的形参,就是resolve和reject响应出来的数据
// 成功回调的形参一般命名为：value/data; 失败回调的形参一般命名为:reason
promise.then(function (value) {
console.log(value)
}, function (reason) {
console.error(reason)
})
```

面试题 15. 简述ES6 Iterator的作用?（重要）

推荐指数：★★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

Iterator是ES6中一个很重要概念，它并不是对象，也不是任何一种数据类型。因为ES6新增了Set、Map类型，他们和Array、Object类型很像，Array、Object都是可以遍历的，但是Set、Map都不能用for循环遍历，解决这个问题有两种方案，一种是为Set、Map单独新增一个用来遍历的API，另一种是为Set、Map、Array、Object新增一个统一的遍历API，显然，第二种更好，ES6也就顺其自然的需要一种设计标准，来统一所有可遍历类型的遍历方式。Iterator正是这样一种标准。或者说是一种规范理念。

就好像Javascript是ECMAscript标准的一种具体实现一样，Iterator标准的具体实现是Iterator遍历器。Iterator标准规定，所有部署了key值为[Symbol.iterator]，且[Symbol.iterator]的value是标准的Iterator接口函数(标准的Iterator接口函数：该函数必须返回一个对象，且对象中包含next方法，且执行next()能返回包含value/done属性的Iterator对象)的对象，都称之为可遍历对象，next()后返回的Iterator对象也就是Iterator遍历器。

ES6给Set、Map、Array、String都加上了[Symbol.iterator]方法，且[Symbol.iterator]方法函数也符合标准的Iterator接口规范，所以Set、Map、Array、String默认都是可以遍历的。

案例：

```
// 声明一个数组
const xiyou = ['唐僧', '孙悟空', '猪八戒', '沙僧'];
console.log(xiyou);

// 使用for in遍历数组
for (let v in xiyou) {
// console.log(v);
console.log(xiyou[v]);
}

// 使用 for...of 遍历数组
for (let v of xiyou) {
console.log(v);
}

let iterator = xiyou[Symbol.iterator]();

//调用对象的next方法
console.log(iterator.next());
console.log(iterator.next());
console.log(iterator.next());
```

```
console.log(iterator.next());
console.log(iterator.next());
```

面试题 16. 简述ES6规定for...in 和for...of有什么区别?

推荐指数: ★★★ 试题难度: 初级 试题类型: 原理题

试题回答参考思路:

ES6规定，有所部署了载了Iterator接口的对象(可遍历对象)都可以通过for...of去遍历，而for..in仅仅可以遍历对象。
这也就意味着，数组也可以用for...of遍历，这极大地方便了数组的取值，且避免了很多程序用for..in去遍历数组的恶习。
上面提到的扩展运算符本质上也就是for..of循环的一种实现。

面试题 17. 简述ES6 Generator函数的作用?

推荐指数: ★★★★ 试题难度: 初级 试题类型: 原理题

试题回答参考思路:

如果说Javascript是ECMAscript标准的一种具体实现、Iterator遍历器是Iterator的具体实现，那么Generator函数可以说是Iterator接口的具体实现方式。

执行Generator函数会返回一个遍历器对象，每一次Generator函数里面的yield都相当一次遍历器对象的next()方法，并且可以通过next(value)方法传入自定义的value,来改变Generator函数的行为。

Generator函数可以通过配合Thunk 函数更轻松更优雅的实现异步编程和控制流管理

面试题 18. 简述ES6 async函数的?

推荐指数: ★★★★ 试题难度: 中级 试题类型: 原理题

试题回答参考思路:

async函数可以理解为内置自动执行器的Generator函数语法糖，它配合ES6的Promise近乎完美的实现了异步编程解决方案。

面试题 19. 简述ES6 Class、extends是什么，有什么作用?

推荐指数: ★★★★ 试题难度: 中级 试题类型: 原理题

试题回答参考思路:

ES6 的class可以看作只是一个ES5生成实例对象的构造函数的语法糖。它参考了java语言，定义了一个类的概念，让对象原型写法更加清晰，对象实例化更像是一种面向对象编程。Class类可以通过extends实现继承。它和ES5构造函数的不同点:

a. 类的内部定义的所有方法，都是不可枚举的。
b.ES6的class类必须用new命令操作，而ES5的构造函数不用new也可以执行。

c.ES6的class类不存在变量提升，必须先定义class之后才能实例化，不像ES5中可以将构造函数写在实例化之后。

d.ES5 的继承，实质是先创造子类的实例对象this，然后再将父类的方法添加到this上面。ES6 的继承机制完全不同，实质是先将父类实例对象的属性和方法，加到this上面（所以必须先调用super方法），然后再用子类的构造函数修改this。

面试题 20. ES6简述module、export、import的作用？

推荐指数：★★★★★ 试题难度：高难 试题类型：原理题

试题回答参考思路：

module、export、import是ES6用来统一前端模块化方案的设计思路和实现方案。export、import的出现统一了前端模块化的实现方案，整合规范了浏览器/服务端的模块化方法，用来取代传统的AMD/CMD、requireJS、seaJS、commondJS等等一系列前端模块不同的实现方案，使前端模块化更加统一规范，JS也能更加能实现大型的应用程序开发。
import引入的模块是静态加载（编译阶段加载）而不是动态加载（运行时加载）。
import引入export导出的接口值是动态绑定关系，即通过该接口，可以取到模块内部实时的值

面试题 21. 简述开发过程中有哪些值得用ES6去改进的编程优化或者规范?

推荐指数：★★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

1、常用箭头函数来取代var self = this;的做法。
2、常用let取代var命令。
3、常用数组/对象的结构赋值来命名变量，结构更清晰，语义更明确，可读性更好。
4、在长字符串多变量组合场合，用模板字符串来取代字符串累加，能取得更好地效果和阅读体验。
5、用Class类取代传统的构造函数，来生成实例化对象。
6、在大型应用开发中，要保持module模块化开发思维，分清模块之间的关系，常用import、export方法。

面试题 22. 详细阐述ES6 箭头函数？

推荐指数：★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

特点：①.简化代码；②改变this的指向，谁创建函数，this就指向谁。

1. 简单的定义：
胖箭头函数 Fat arrow functions，又称箭头函数，是一个来自ECMAScript 2015（又称ES6）的全新特性。有传闻说，箭头函数的语法=>，是受到了CoffeeScript 的影响，并且它与CoffeeScript中的=>语法一样，共享this上下文。
箭头函数的产生，主要由两个目的：更简洁的语法和与父作用域共享关键字this。接下来，让我们来看几个详细的例子
当需要编写一个简单的单一参数函数时，可以采用箭头函数来书写，标识名 => 表达式。
这样就可以省却 function 和 return 的输入，还有括号，分号等。箭头函数是ES6新增加的一个特性。
let f = v => v;
最直接的感觉就是简便，当然不可能就是这么一点好处，下面就一起来探讨一下。
几个小细节 ：
如果箭头函数的代码块多余一条语句，就必须要使用大括号将其括起来，并且使用return 语句返回。
由于大括号会被解释位为代码块，所以如果箭头函数直接返回一个对象，必须在外面加上括号。
let f = id => ({ age: 22, name: Alice })
箭头函数还可以和解构赋值 Destructuring 联合使用.
const f = ({ first, last }) => first + '' + last;
可以简化回调函数，大大简化和缩短代码行数。
2. 箭头函数和普通函数的区别(this的指向)
普通函数与箭头函数有个微小的不同点。 箭头函数没有自己的this值 ，其this值是通过继承其它传入对象而获得的,通常来说是上一级外部函数的 this 的指向。
function f() {
setTimeout(() => {
console. log("id:", this. id);
}, 100);
}

f. call({ id: 42 }); //id: 42;
这个例子中， setTimeout 的参数是一个箭头函数， 每隔100毫秒运行一次，如果是普通函数，执行的 this 应该指向全局对象， 但是箭头函数会让 this 总是指向函数所在的对象
箭头函数里面嵌套箭头函数会有多少个this呢?
看一个简单的例子
function f() {
return () => {
return () => {
return () => {
console. log("id:", this. id);
};
};
};
}

f(). call({ id: 42 })()()(); //id: 42
上面的代码中只有一个 this， 就是函数f的this 。这是因为所有的内层函数都是箭头函数都没有自己的this，都是最外层f函数的this。
注意：还有三个变量在箭头函数中也是不存在的arguments ，super, new.target所以顺理成章，箭头函数也就不能再用这些方法call(),apply(),bind(),因为这是一些改变this指向的方法，箭头函数并没有this啊。

```
var adder = {
base: 1,
add : function ( a) {
var f = v => v + this. base; return f( a);
},
addThruCall : function ( a) {
var f = v => v + this. base;
var b = { base: 2 };
return f. call( b, a);
}
};
console. log( adder. add( 1));
```

3. 怎么处理好箭头函数的使用问题呢?
使用非箭头函数来处理由object.method()语法调用的方法。因为它们会接收到来自调用者的有意义的this值。
在其它场合都使用箭头函数。
4. 使用箭头函数的注意点
箭头函数在参数和箭头之间不能换行。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。

```
'use strict';
var obj = { a: 10 };
Object. defineProperty( obj, "b", {
get : () => {
console. log( this. a, typeof this. a, this);
return this. a + 10;
// represents global object 'Window', therefore 'this.a' returns 'undefined'
}
});
```

不可以当作构造函数，简单说就是不能再使用new命令了，不然会报错。

```
var Foo = () => { };
var foo = new Foo();
// TypeError: Foo is not a constructor
```

不可以使用arguments 对象，该对象在函数体内不存在，如果实在要用可以用rest代替。
不可以使用yield命令，箭头函数不可用作Generator函数。
值得注意的一点就是this对象的指向是可变的，但在箭头函数内是固定的。
5. 总结
箭头函数是我最喜欢的ES6特性之一。使用=>来代替function是非常便捷的。但我也曾见过只使用
=>来声明函数的代码，我并不认为这是好的做法，因为=>也提供了它区别于传统function，其所
独有的特性。我个人推荐，仅在你需要使用它提供的新特性时，才使用它。

当只有一条声明语句(statement)时， 隐式 return。
需要使用到父作用域中的this。

面试题 23. 解释ES6 includes(), startsWith(), endsWith()?

推荐指数： ★★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

传统上，JavaScript 只有indexOf方法，可以用来确定一个字符串是否包含在另一个字符串中。ES6 又提供了三种新方法。

1.includes()：返回布尔值，表示是否找到了参数字符串。
2.startsWith()：返回布尔值，表示参数字符串是否在原字符串的头部。
3.endsWith()：返回布尔值，表示参数字符串是否在原字符串的尾部。

```
let s = 'Hello world!';
s. startsWith( 'Hello') // true
s. endsWith( '!') // true
s. includes( 'o') // true
```

这三个方法都支持第二个参数，表示开始搜索的位置。

```
let s = 'Hello world!';
s. startsWith( 'world', 6) // true
s. endsWith( 'Hello', 5) // true
s. includes( 'Hello', 6) // false
```

上面代码表示，使用第二个参数n时， endsWith的行为与其他两个方法有所不同。它针对前n个字符，而其他两个方法针对从第n个位置直到字符串结束。

面试题 24. 简述ES中什么是padStart()，padEnd()？

推荐指数： ★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

ES2017 引入了字符串补全长度的功能。如果某个字符串不够指定长度，会在头部或尾部补全。padStart()用于头部补全，padEnd()用于尾部补全。

```
'x'. padStart( 5, 'ab') // 'ababx'
'x'. padStart( 4, 'ab') // 'abax'
'x'. padEnd( 5, 'ab') // 'xabab'
'x'. padEnd( 4, 'ab') // 'xaba'
```

上面代码中，padStart和padEnd一共接受两个参数，第一个参数用来指定字符串的最小长度，第二个参数是用来补全的字符串。
如果原字符串的长度，等于或大于指定的最小长度，则返回原字符串。

```
'xxx'. padStart( 2, 'ab') // 'xxx'
'xxx'. padEnd( 2, 'ab') // 'xxx'
```

如果用来补全的字符串与原字符串，两者的长度之和超过了指定的最小长度，则会截去超出位数的补全字符串。

'abc'. padStart(10, '0123456789')
// '0123456abc'
如果省略第二个参数，默认使用空格补全长度。
'x'. padStart(4) // ' x'
'x'. padEnd(4) // 'x '
padStart的常见用途是为数值补全指定位数。下面代码生成 10 位的数值字符串。
'1'. padStart(10, '0') // "0000000001"
'12'. padStart(10, '0') // "0000000012"
'123456'. padStart(10, '0') // "0000123456"
另一个用途是提示字符串格式。

'12'. padStart(10, 'YYYY-MM-DD') // "YYYY-MM-12"
'09-12'. padStart(10, 'YYYY-MM-DD') // "YYYY-09-12"

面试题 25. 简述ES var、let、const之间的区别?

推荐指数： ★★★★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

var：
var声明的变量既是全局变量，也是顶层变量（顶层对象，在浏览器环境下指的是window对象，在node中指的是global对象）
var存在变量提升，可以对一个变量进行多次声明，后面声明的变量会覆盖前面的变量声明
在函数中使用var声明变量，变量是局部的，如果在函数内不使用var，那么这个变量就是全局的
let:
let是es6新增的命令，用来声明变量，他和var类似，但是let只在代码块中有效
不存在变量提升，不允许重复声明变量
let存在暂时性死区，是个块级作用域，
只要在let块级作用域内存在let命令，这个区域就不会受外部影响
在使用let声明变量之前，该变量都不可用，也就是我们常说的暂时性死区
const：
const声明一个只读的常量，一旦声明，常量的值就不能进行改变
const 一旦声明变量，必须立即初始化，不能留到以后赋值
const实际上保证的并不是变量的值不能改动，而是变量指向的内存地址所保存的数据不能改动
对于简单类型的数据，值就保存在变量指向的那个内存地址，因此等同于常量
区别：
可以根据以下5点展开
变量提升：var存在变量提升，let和const不存在变量提升
暂时性死区：var不存在暂时性死区，let和const存在暂时性死区
块级作用域：var不存在块级作用域，let和const存在块级作用域
修改声明的变量：var允许重复声明变量，let和const一律不允许重复声明变量
使用：能使用const尽量使用const，其他情况使用let，避免使用var

面试题 26. 简述汇总ES6中数组新增了哪些扩展?

推荐指数: ★★ 试题难度: 高难 试题类型: 原理题

试题回答参考思路:

扩展运算符: (...) 好比rest参数的逆运算，将一个数组转化为用逗号分隔的参数序列
可以实现简单的数组复制，数组的合并也更加简洁
通过扩展运算符实现浅拷贝，修改了引用指向的值，会同步反映到新的数组
构造函数新增的方法:
Array.from（）:将两类对象转为真正的数组，类似数组的对象和可遍历的对象
Array.of() 用于将一组值，转换为数组，没有参数时，返回一个空数组，当参数有一个时，是指定数组的长度
实例对象新增的方法:
copyWithin()将指定位置的成员复制到其他位置（会覆盖原有成员）
target（必需）：从该位置开始替换数据。如果为负值，表示倒数。
start（可选）：从该位置开始读取数据，默认为 0。如果为负值，表示从末尾开始计算。
end（可选）：到该位置前停止读取数据，默认等于数组长度。如果为负值，表示从末尾开始计算
find()、findIndex() 用于找出第一个符合条件的数组成员，参数是一个回调函数，接受三个参数依次为当前的值、当前的位置和原数组
fill(): 使用给定值，填充一个数组
三个参数：分别是填充的值，填充的开始位置，填充的结束位置
entries()，keys()，values(): 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
includes(): 用于判断数组是否包含给定的值
方法的第二个参数表示搜索的起始位置，默认为0
参数为负数则表示倒数的位置
flat()，flatMap(): 将数组扁平化处理，返回一个新数组，对原数据没有影响
flat()默认只会“拉平”一层，如果想要“拉平”多层的嵌套数组，可以将flat()方法的参数写成一个整数，表示想要拉平的层数，默认为1
flatMap()方法对原数组的每个成员执行一个函数相当于执行Array.prototype.map()，然后对返回值组成的数组执行flat()方法。该方法返回一个新数组，不改变原数组
flatMap()方法还可以有第二个参数，用来绑定遍历函数里面的this
数组的空位
空位指的是某一个位置没有任何值
ES6 则是明确将空位转为undefined，包括Array.from、扩展运算符、copyWithin()、fill()、entries()、keys()、values()、find()和findIndex()

面试题 27. 简述汇总ES7对象新增了哪些扩展?

推荐指数: ★★★★★ 试题难度: 高难 试题类型: 原理题

试题回答参考思路:

属性的简写:
当我们键名和对应值名相等时，可以进行简写，方法也可以进行简写

注意：简写的对象方法不能用作构造函数，否则会报错

属性名表达式
es6允许字面量定义对象，将表达式放在括号中，表达式还可以用于定义方法名

super关键字
this关键字总是指向函数所在的当前对象，super关键字指向当前对象的原型对象

扩展运算符
在结构赋值中，未被读取的可遍历的属性，分配到指定的对象上

解构赋值必须是最后一个参数，否则会报错

解构赋值是浅拷贝

属性遍历
es6一共有5种方法遍历对象的属性

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

Object.keys(obj)：返回一个数组，包括对象自身的（不含继承的）所有可枚举属性（不含 Symbol 属性）的键名

Object.getOwnPropertyNames(obj)：回一个数组，包含对象自身的所有属性（不含 Symbol 属性，但是包括不可枚举属性）的键名

Object.getOwnPropertySymbols(obj)：返回一个数组，包含对象自身的所有 Symbol 属性的键名

Reflect.ownKeys(obj)：返回一个数组，包含对象自身的（不含继承的）所有键名，不管键名是 Symbol 或字符串，也不管是否可枚举

参数
ES6允许为函数的参数设置默认值，函数的形参是默认声明的，不能使用let或const再次声明

参数的默认值可以与解构赋值的默认值结合起来使用

属性
length： 函数的length属性，length将返回没有指定默认值的参数的个数

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

name：返回函数的函数名

作用域
一旦设置了参数的默认值，函数进行声明初始化时，参数就会形成一个单独的作用域

等初始化结束之后，这个作用域也会消失

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，那么函数的内部就不能显式设定为严格模式，否则会报错

箭头函数
=>定义函数，如果箭头函数不需要参数或需要多个参数，就使用一个圆括号代表参数部分

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

注意点：

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

面试题 28. 简述你对ES6中新增的set，map两种数据结构的理解?

推荐指数：★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

set是ES6新增的数据结构，类似于数组，但是成员的值都是唯一的，没有重复的值，我们一般称为集合
set本身是一个构造函数，用来生成set数据结构
增删改查
add()添加某个值，返回 Set 结构本身
delete()删除某个值，返回一个布尔值，表示删除是否成功
has()返回一个布尔值，判断该值是否为Set的成员
clear()清除所有成员，没有返回值
遍历
keys()：返回键名的遍历器
values()：返回键值的遍历器
entries()：返回键值对的遍历器
forEach()：使用回调函数遍历每个成员
扩展运算符和Set 结构相结合实现数组或字符串去重
map
map类型是键值对的有序列表，而键和值都可以是任意类型
map本身是一个构造函数，用来生成map数据结构
增删改查
size 属性返回 Map 结构的成员总数
set()设置键名key对应的键值为value，然后返回整个 Map 结构
如果key已经有值，则键值会被更新，否则就新生成该键

同时返回的是当前Map对象，可采用链式写法
get()get方法读取key对应的键值，如果找不到key，返回undefined
has()has方法返回一个布尔值，表示某个键是否在当前 Map 对象之中
delete()delete方法删除某个键，返回true。如果删除失败，返回false
clear()clear方法清除所有成员，没有返回值
遍历
eys()：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
forEach()：遍历 Map 的所有成员

面试题 29. 如何怎么理解ES6中的Promise?

推荐指数： ★★★★★ 试题难度： 高难 试题类型： 原理题

试题回答参考思路：

promise（承诺），他是异步编程的一种解决方案，比传统的解决方案更加合理和更加强大
promise解决异步操作的优点：
链式操作降低了编码难度
代码的可读性明显增强
状态：
promise有三种状态
pending：进行中
fulfilled：已成功
rejected：已失败
特点：
对象的状态不受外界影响，只有异步操作的结果，可以决定当前是哪一种状态
一旦状态改变（从pending变为fulfilled和从pending变为rejected），就不会再变，任何时候都可以得到这个结果
用法
promise对象是一个构造函数，用来生成promise实例，promise构造函数接收一个函数作为参数，该函数的两个参数分别是resolve和reject
resolve函数的作用是将对象的状态从未完成变为成功
reject函数的作用是将对象的状态从未完成变为失败
实例方法
then
then是实例状态发生改变时的回调函数，第一个参数是resolved状态的回调函数，第二个参数是rejected状态的回调函数
then方法返回的是一个新的Promise实例，也就是promise能链式书写的原因
catch
catch()方法是.then(null, rejection)或.then(undefined, rejection)的别名，用于指定发生错误时的回调函数
Promise对象的错误具有“冒泡”性质，会一直向后传递，直到被捕获为止
finally
finally()方法用于指定不管 Promise 对象最后状态如何，都会执行的操作
all

Promise.all()方法用于将多个 Promise实例，包装成一个新的 Promise实例
race
Promise.race()方法同样是将多个 Promise 实例，包装成一个新的 Promise 实例
使用场景
将图片加载写成一个promise，一旦加载完成，promise的状态就发生变化

面试题 30. 如何理解ES6中 Generator的？使用场景？

推荐指数：★★★★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

Generator 函数是 ES6 提供的一种异步编程解决方案，语法行为与传统函数完全不同
回顾下上文提到的解决异步的手段：
回调函数
promise
Generator函数
执行 Generator 函数会返回一个遍历器对象，可以依次遍历 Generator 函数内部的每一个状态
形式上，Generator函数是一个普通函数，但是有两个特征：
function关键字与函数名之间有一个星号
函数体内部使用yield表达式，定义不同的内部状态
使用
Generator 函数会返回一个遍历器对象，即具有Symbol.iterator属性，并且返回给自己
通过yield关键字可以暂停generator函数返回的遍历器对象的状态
异步解决的方案
回调函数：回调函数，就是把任务代码单独写在一个函数里，等到重新执行这个任务时，再调用这个函数
Promise 对象：就是为了解决回调地狱而产生的
generator 函数：yield表达式可以暂停函数执行，next方法用于恢复函数执行，这使得Generator函数非常适合将异步任务同步化
async/await：代码更加简介，语义化更强
区别?
promise和async/await是专门用于处理异步操作的
Generator并不是为异步而设计出来的，它还有其他功能（对象迭代、控制输出、部署Interator接口...）
promise编写代码相比Generator、async更为复杂化，且可读性也稍差
Generator、async需要与promise对象搭配处理异步情况
async实质是Generator的语法糖，相当于会自动执行Generator函数
async使用上更为简洁，将异步代码以同步的形式进行编写，是处理异步编程的最终方案
使用场景
Generator是异步解决的一种方案，最大特点则是将异步操作同步化表达出来

面试题 31. 如何理解ES6中Proxy的？使用场景？

推荐指数：★★★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

用来定义基本操作中的自定义行为
本质就是修改程序的默认行为，形同与在编程语言层面上做一些修改，属于元编程
元编程（Metaprogramming，又译超编程，是指某类计算机程序的编写，这类计算机程序编写或者操纵其它程序（或者自身）作为它们的数据，或者在运行时完成部分本应在编译时完成的工作
元编程的特点：与手工编写全部代码相比，程序员可以获得更高的工作效率，或者给与程序更大的灵活度去处理新的情形而无需重新编译
Proxy 亦是如此，用于创建一个对象的代理，从而实现基本操作的拦截和自定义（如属性查找、赋值、枚举、函数调用等）
用法：
Proxy为 构造函数，用来生成 Proxy实例
参数：target（表示要拦截的目标对象，任何类型的对象包括原生数组，函数，甚至是另一个代理）
handler（通常以函数作为属性的对象，个属性中的函数分别定义了在执行各种操作时代理p的行为）
var proxy = new Proxy(target, handler)
取消代理
Proxy.revocable(target, handler);
使用场景
拦截和监视外部对对象的访问
降低函数或类的复杂度
在复杂操作前对操作进行校验或对所需资源进行管理

面试题 32. 如何理解ES6中Module的？使用场景？

推荐指数：★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

模块，是能够单独命名并独立的完成一定功能的程序语句的集合（程序代码和数据结构的集合体）
两个基本的特征：外部特征和内部特征
外部特征是指模块跟外部环境联系的接口（即其他模块或程序调用该模块的方式，包括有输入输出参数、引用的全局变量）和模块的功能
内部特征是指模块的内部环境具有的特点（即该模块的局部数据和程序代码）
为什么需要模块化？
代码抽象
代码封装
代码复用
依赖管理
如果我们没有模块化，我们的代码会变得不容易维护，容易污染全局作用域
加载资源的方式通过script标签从上到下
依赖的环境主观逻辑偏重，代码较多就会比较复杂
大型项目资源难以维护，特别是多人合作的情况下，资源会让人崩溃
使用

模块功能主要由两个命令构成
export：用于规定模块的对外接口
一个模块就是一个独立的文件，该文件内部的所有变量，外部无法获取，如果希望外部能够读取内部的某个变量，就必须通过export关键字输出该变量
import： 用于输入其他模块提供的功能
使用export命令定义了模块的对外接口以后，其他的js文件就可以通过import加载这个命令
我们可以通过as 关键字配置别名
当我们需要加载整个模块，使用*
输入的变量都是只读的，不允许修改，但是如果是对象，允许修改属性
动态加载
允许仅在需要时动态加载模块，不必预先加载所有模块，性能优势明显
这个功能允许我们将import作为函数调用，将其作为参数传递给模块的路径
使用场景
如今，ES6模块化已经深入我们日常项目开发中，像vue、react项目搭建项目，组件化开发处处可见，其也是依赖模块化实现

面试题 33. 如何理解ES6中 Decorator 的？使用场景?

推荐指数： ★★★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

Decorator，即装饰器，简单来说，装饰着模式就是一种在不改变原类和使用继承的情况下，动态的扩展对象功能的设计理论
本质也是一个普通的函数，用于扩展类属性和类方法
用法：
类的装饰：对类本身进行装饰，能够接收一个参数
类属性的装饰：
类的原型对象
需要装饰的属性名
装饰属性名的描述对象
首先定义yigereadonly装饰器，使用readonly装饰类的name方法，如果一个方法有对各装饰器，就像洋葱一样，从外到内，再由内到外执行
注意：装饰器不能用于装饰函数，因为函数存在变量声明情况

面试题 34. 简述ECMAScript 和 JavaScript 的关系 ?

推荐指数： ★★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

1996 年 11 月，JavaScript 的创造者 Netscape 公司，决定将 JavaScript 提交给标准化组织 ECMA，希望这种语言能够成为国际标准。次年，ECMA 发布 262 号标准文件（ECMA-262）的第一版，规定了浏览器脚本语言的标准，并将这种语言称为 ECMAScript，这个版本就是 1.0 版。

该标准从一开始就是针对 JavaScript 语言制定的，但是之所以不叫 JavaScript，有两个

原因。一是商标，Java 是 Sun 公司的商标，根据授权协议，只有 Netscape 公司可以合法地使用 JavaScript 这个名字，且 JavaScript 本身也已经被 Netscape 公司注册为商标。二是想体现这门语言的制定者是 ECMA，不是 Netscape，这样有利于保证这门语言的开放性和中立性。

因此，ECMAScript 和 JavaScript 的关系是，前者是后者的规范，后者是前者的一种实现（另外的 ECMAScript 方言还有 JScript 和 ActionScript）。日常场合，这两个词是可以互换的。

面试题 35. 详细描述ES6 与 ECMAScript 2015 的关系？

推荐指数：★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

ECMAScript 2015（简称 ES2015）这个词，也是经常可以看到的。它与 ES6 是什么关系呢？

2011 年，ECMAScript 5.1 版发布后，就开始制定 6.0 版了。因此，ES6 这个词的原意，就是指 JavaScript 语言的下一个版本。

但是，因为这个版本引入的语法功能太多，而且制定过程当中，还有很多组织和个人不断提交新功能。事情很快就变得清楚了，不可能在一个版本里面包括所有将要引入的功能。常规的做法是先发布 6.0 版，过一段时间再发 6.1 版，然后是 6.2 版、6.3 版等等。

但是，标准的制定者不想这样做。他们想让标准的升级成为常规流程：任何人在任何时候，都可以向标准委员会提交新语法的提案，然后标准委员会每个月开一次会，评估这些提案是否可以接受，需要哪些改进。如果经过多次会议以后，一个提案足够成熟了，就可以正式进入标准了。这就是说，标准的版本升级成为了一个不断滚动的流程，每个月都会有变动。

标准委员会最终决定，标准在每年的 6 月份正式发布一次，作为当年的正式版本。接下来的时间，就在这个版本的基础上做改动，直到下一年的 6 月份，草案就自然变成了新一年的版本。这样一来，就不需要以前的版本号了，只要用年份标记就可以了。

ES6 的第一个版本，就这样在 2015 年 6 月发布了，正式名称就是《ECMAScript 2015 标准》（简称 ES2015）。2016 年 6 月，小幅修订的《ECMAScript 2016 标准》（简称 ES2016）如期发布，这个版本可以看作是 ES6.1 版，因为两者的差异非常小（只新增了数组实例的includes方法和指数运算符），基本上是同一个标准。根据计划，2017 年 6 月发布 ES2017 标准。

因此，ES6 既是一个历史名词，也是一个泛指，含义是 5.1 版以后的 JavaScript 的下一代标准，涵盖了 ES2015、ES2016、ES2017 等等，而 ES2015 则是正式名称，特指该年发布的正式版本的语言标准。本书中提到 ES6 的地方，一般是指 ES2015 标准，但有时也是泛指“下一代 JavaScript 语言”。

面试题 36. 详细简述ES6的数值扩展？

推荐指数：★★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

Number.EPSILON
Number.EPSILON 是 JavaScript 表示的最小精度

EPSILON 属性的值接近于 2.2204460492503130808472633361816E-16

```
function equal(a, b) {
if (Math.abs(a - b) < Number.EPSILON) {
return true;
} else {
return false;
}
}
console.log(0.1 + 0.2 === 0.3);
console.log(equal(0.1 + 0.2, 0.3))
```

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

```
//ES6 之前没有二进制表示法
//ES 不同版本八进制表示法区别很大
let b = 0b1010;
let o = 0o777;
let d = 100;
let x = 0xff;
console.log(x);
```

Number.isFinite()
Number.isFinite() 用来检查一个数值是否为有限的

```
// Number.isFinite 检测一个数值是否为有限数
console.log(Number.isFinite(100));
console.log(Number.isFinite(100 / 0));
console.log(Number.isFinite(Infinity));
```

Number.isNaN()
Number.isNaN() 用来检查一个值是否为 NaN

```
// Number.isNaN 检测一个数值是否为 NaN
console.log(Number.isNaN(123));
```

Number.parseInt() / Number.parseFloat()
ES6 将全局方法 parseInt 和 parseFloat，移植到 Number 对象上面，使用不变。

```
//Number.parseInt Number.parseFloat字符串转整数
console.log(Number.parseInt('5211314love'));
console.log(Number.parseFloat('3.1415926神奇'));

Number.isInteger()
// Number.isInteger 判断一个数是否为整数
console.log(Number.isInteger(5));
console.log(Number.isInteger(2.5));

Math.trunc()
用于去除一个数的小数部分，返回整数部分。

//Math.trunc 将数字的小数部分抹掉
console.log(Math.trunc(3.5));

Math.sign()
//Math.sign 判断一个数到底为正数 负数 还是零
console.log(Math.sign(100));
console.log(Math.sign(0));
console.log(Math.sign(-20000));
```

面试题 37. 简述ES6的对象方法扩展？

推荐指数：★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

ES6 新增了一些 Object 对象的方法
Object.is 比较两个值是否严格相等，与『===』行为基本一致（+0 与 NaN）
Object.assign 对象的合并，将源对象的所有可枚举属性，复制到目标对象
__proto__、setPrototypeOf、 setPrototypeOf 可以直接设置对象的原型

```
//1. Object.is 判断两个值是否完全相等
console.log(Object.is(120, 120)); // true
console.log(Object.is(NaN, NaN)); // true
console.log(NaN === NaN); // false

//2. Object.assign 对象的合并
// 用来合并配置项非常有用
const config1 = {
host: 'localhost',
port: 3306,
name: 'root',
pass: 'root',
test: 'test'
};
const config2 = {
```

```
host: 'http://www.newcapec.com.cn/',
port: 33060,
name: 'newcapec',
pass: 'root',
test2: 'test2'
}
console.log(Object.assign(config1, config2));

//3. Object.setPrototypeOf 设置原型对象 Object.getPrototypeof
// 不推荐,还是用原来的原型方式
const school = {
name: '新开普'
}
const cities = {
xiaoqu: ['北京', '上海', '深圳']
}
Object.setPrototypeOf(school, cities);
console.log(Object.getPrototypeOf(school));
console.log(school);
```

面试题 38. 简述ECMASript 7 新特性？

推荐指数：★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

新增数组的includes属性
引入了 ** 运算符 （指数运算符） 示例2 **3 //8
Array.prototype.includes
定义和用法
includes() 方法用来判断一个数组是否包含一个指定的值，如果是返回 true，否则false。

语法
arr.includes(searchElement)
arr.includes(searchElement, fromIndex)

面试题 39. 简述ECMASript 8 新特性？

推荐指数：★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

加async await使得异步改同步成为可能，避免代码书写的来回嵌套

增加Object.values() Object.entries()

增加String padding：String.prototype.padStart、String.prototype.padEnd

允许函数参数列表结尾存在逗号

添加Object.getOwnPropertyDescriptors(): 获取一个对象的所有自身属性的描述符,如果没有任何自身属性，则返回空对象

新增SharedArrayBuffer 对象：用来表示一个通用的，固定长度的原始二进制数据缓冲区

新增Atomics 对象：提供了一组静态方法用来对 SharedArrayBuffer 对象进行原子操作

面试题 40. 简述ECMASript 9 新特性？

推荐指数： ★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

允许异步迭代：await可以和for...of循环一起使用，以串行的方式运行异步操作
添加Promise.finally()

面试题 41. 简述ECMASript 10 新特性？

推荐指数： ★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

修改了try catch 的使用，catch不必再由入参

增加数组的flat flatMap方法
增加字符串的trimStart, trimEnd方法，分别是去掉首尾空格
增加Object.fromEntries方法，可以把对应数组转成对象
增加Function.prototype.toString()方便看到函数对应的内部代码
增加Symbol.prototype.description方法
示例 Symbol('test'). description === 'test' // true
对JSON对象的优化 JSON.superset 、JSON.stringify

面试题 42. 简述ECMASript 11 新特性？

推荐指数： ★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

增加Bigint用于大数计算
增加可选链 简化书写判断
增加 ?? 运算，如果左侧不为null或者undefined则返回 ??左侧内容
解决了 let num = number || 1 这种计算方式的bug
增加Promise.allSettled方法
支持import()函数用于异步加载

面试题 43. 简述ECMASript 12 新特性？

推荐指数：★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

增加字符串的replactAll方法，之前要实现替换全部，需要使用正则表达式
增加Promise.any方法
新增了逻辑赋值操作符 ??=、&&=、 ||=
增加下划线（_）分隔符：使用 _ 分隔数字字面量以方便阅读 1_0000 === 10000 //true
以及WeakRefs、Intl.ListFormat、Intl.DateTimeFormat

面试题 44. 简述怎样通过ES5及ES6声明一个类？

推荐指数：★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

1、传统的构造函数，声明一个类
```
function Animal() {
this.name = 'name';
}
```
2、es6中的class声明
```
class Animal2{
constructor() {
this.name = name;
}
}
```

面试题 45. 简述ES6 之前使用 prototype 实现继承？

推荐指数：★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

Object.create() 会创建一个“新”对象，然后将此对象内部的 [[Prototype]] 关联到你指定的对象（Foo.prototype）。Object.create(null) 创建一个空 [[Prototype]] 链接的对象，这个对象无法进行委托。

```
function Foo(name) {
this.name = name;
}
Foo.prototype.myName = function () {
return this.name;
}
// 继承属性，通过借用构造函数调用
function Bar(name, label) {
Foo.call(this, name);
this.label = label;
}
// 继承方法，创建备份
Bar.prototype = Object.create(Foo.prototype);
// 必须设置回正确的构造函数，要不然在会发生判断类型出错
Bar.prototype.constructor = Bar;
// 必须在上一步之后
Bar.prototype.myLabel = function () {
return this.label;
}
var a = new Bar("a", "obj a");
a.myName(); // "a"
a.myLabel(); // "obj a"
```

面试题 46. 简述ES5/ES6 的继承除了写法以外还有什么区别？

推荐指数：★★ 试题难度：高难 试题类型：原理题

试题回答参考思路：

1. ES5 的继承实质上是先创建子类的实例对象，然后再将父类的方法添加到 this 上（Parent.apply(this)）. 2. ES6 的继承机制完全不同，实质上是先创建父类的实例对象 this（所以必须先调用父类的 super()方法），然后再用子类的构造函数修改 this。
3. ES5 的继承时通过原型或构造函数机制来实现。
4. ES6 通过 class 关键字定义类，里面有构造方法，类之间通过 extends 关键字实现继承。
5. 子类必须在 constructor 方法中调用 super 方法，否则新建实例报错。因为子类没有自己的 this 对象，而是继承了父类的 this 对象，然后对其进行加工。如果不调用 super 方法，子类得不到 this 对象。
6. 注意 super 关键字指代父类的实例，即父类的 this 对象。
7. 注意：在子类构造函数中，调用 super 后，才可使用 this 关键字，否则报错。
function 声明会提升，但不会初始化赋值。Foo 进入暂时性死区，类似于 let、const 声明变量。

```
const bar = new Bar();
// it's ok
```

```
function Bar() {
this.bar = 42;
}const foo = new Foo();
// ReferenceError: Foo is not definedclass Foo {
constructor() {
this.foo = 42;
}
}
```

class 声明内部会启用严格模式。

```
// 引用一个未声明的变量 function Bar() {
baz = 42;
// it's ok}const bar = new Bar();
class Foo {
constructor() {
fol = 42;
// ReferenceError: fol is not defined
}
}const foo = new Foo();
```

class 的所有方法（包括静态方法和实例方法）都是不可枚举的。

```
// 引用一个未声明的变量 function Bar() {
this.bar = 42;}Bar.answer = function()
{ return 42;
};
Bar.prototype.print = function() {
console.log(this.bar);
};
const barKeys = Object.keys(Bar);
// ['answer']const barProtoKeys = Object.keys(Bar.prototype);
// ['print']class Foo {
constructor() {
this.foo = 42;
}
static answer() {
return 42;
}
print() {
console.log(this.foo);
}}const fooKeys = Object.keys(Foo);
// []const fooProtoKeys = Object.keys(Foo.prototype);
// []
```

class 的所有方法（包括静态方法和实例方法）都没有原型对象 prototype，所以也没有[[construct]]，不能使用 new 来调用。

```
function Bar() {
this.bar = 42;
}Bar.prototype.print = function() {
console.log(this.bar);
};
```

```
const bar = new Bar();
const barPrint = new bar.print();
// it's okclass Foo {
constructor() {
this.foo = 42;
} print() {
console.log(this.foo);
}}const foo = new Foo();
const fooPrint = new foo.print();
// TypeError: foo.print is not a constructor
必须使用 new 调用 class。
function Bar() {
this.bar = 42;
}const bar = Bar();
// it's okclass Foo {
constructor() {
this.foo = 42;
}}const foo = Foo();
// TypeError: Class constructor Foo cannot be invoked without 'new'
class 内部无法重写类名。
function Bar() {
Bar = 'Baz';
// it's ok this.bar = 42;
}const bar = new Bar();
// Bar: 'Baz'
// bar: Bar {bar: 42}
class Foo {
constructor() {
this.foo = 42;
Foo = 'Fol';
// TypeError: Assignment to constant variable
}}const foo = new Foo();
Foo = 'Fol';
// it's ok
```

面试题 47. 简述异步笔试题请写出下面代码的运行结果？

```
async function async1() {
console.log('async1 start')
await async2()
console.log('async1 end')}async function async2()
{
console.log('async2')}console.log('script
start')setTimeout(function()
{
console.log('setTimeout')},
0) async1()new Promise(function(resolve)
1) {
console.log('promise1')
```

```
resolve()}).then(function()
{
console.log('promise2')})console.log('script end'
```

推荐指数：★★★★★ 试题难度：初级 试题类型：编程题

试题回答参考思路：

```
//输出
//script start
//async1 start
//async2
//promise1
//script end
//async1 end
//promise2
//setTimeout
```

面试题 48. 简述ES6 代码转成 ES5 代码的实现思路是什么？

推荐指数：★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

ES6 转 ES5 目前行业标配是用 Babel，转换的大致流程如下：
1. 解析：解析代码字符串，生成 AST；
2. 转换：按一定的规则转换、修改 AST；
3. 生成：将修改后的 AST 转换成普通代码。
如果不用工具，纯人工的话，就是使用或自己写各种 polyfill 了