# ES6

1. **Let、Var、Const的区别**
2. Var 声明变量绑定到window 上，而其它两个不会
3. Let和Const会有临时性死区，在声明之前访问变量报错（不存在变量提升？）
4. Let 和 Const 只在代码块内有效，并且不能重复声明
5. **Symbol**
6. 不是构造函数，而是直接调用Symbol()，返回的变量值永远不同
7. Symbol.for()全局搜索被登记的 Symbol 中是否有该字符串参数作为名称的 Symbol 值，如果有即返回该 Symbol 值，若没有则新建并返回一个以该字符串参数为名称的 Symbol 值，并登记在全局环境中供搜索
8. Symbol.keyFor(参数为Symbol变量) 返回一个已登记的 Symbol 类型值的 key ，用来检测该字符串参数作为名称的 Symbol 值是否已被登记。
9. **Map和 Set**

Map是一个映射数据结构，Set是一个集合数据结构（自带去重效果）

**Map：**

1. 记住插入的顺序，Object的间遍历时无序的
2. NaN认为相等，+0和-0认为相等
3. Get、Set、Delete、Has(键)、Clear、Size、可以forEach遍历

**Set：**

1. 有序
2. Add、Has、Delete、Clear
3. NaN、undefined、+0，-0
4. **对象和Class**

**对象：**

1. 写法上：属性的简洁写法、属性名表达式
2. 对象展开运算符…
3. API：Object.assign，Object.is （处理-0、NaN问题）

**Class：**

Constructor初始化函数，super调用父类初始化函数，箭头函数的绑定特殊性，

本质还是一个构造函数，静态属性static，extend继承

1. **函数**
2. 默认参数，参数临死性死区 ，不定参数符
3. 箭头函数
4. **数组**
5. Array.of 参数中所有值作为元素形成数组
6. Array.from 将类数组对象或可迭代对象转化为数组
7. Find、findIndex、fill、includes、flat、flatMap
8. **迭代器**

**迭代过程：**

1. Symbol.iterator 返回可迭代对象，并且指向当前数据结构的起始位置
2. 可迭代对象有next方法，调用方法返回当前位置的对象包含值value和是否迭代完成done
3. 当done为true时，迭代完毕

**可迭代的数据结构：**

String、Array、Map、Set、类数组、DOM

目的就是为了迭代有序的数据结构，方便访问和使用for…of

1. **Promise**

定义：是一种异步编程的解决方案，能够将异步操作封装，并将其返回的结果或者错误原因同Promise实例的then方法传入的处理函数联系起来

**New Promise：**

1. 传入构造器函数，两个参数，都是敲定函数，第一个成功的敲定，第二个失败的敲定，构造器同步执行，但是可以异步的执行敲定函数
2. 构造器中调用对应的敲定函数，返回的promise实例的对象状态就是那种，当然中间能会抛出异常throw，那么…状态是拒绝
3. 状态的变化只有两种 pending => fulfilled，pending => rejected，一旦发生变化就不会再改变

**Promise.then返回的Promise的状态：**

1. 非Promise对象，直接包裹成为Promise（Promise.resolve）
2. Promise对象，状态跟随，值跟随 ，通过这一点可以串联Promise（构造器函数的成功敲定函数也是这样的功能）
3. 报错返回拒绝的Promise，错误对象是Promise的错误原因

**异常穿透：**

Catch在最后进行异常的捕获

**中断Promise链：**

处理函数返回等待状态的Promise

**实现Promise.all方法：**

new Promise找准时机调用敲定函数

1. **Generator、Yield 函数**

Generator函数返回一个可迭代对象，迭代的顺序就是在生成器函数使用yield关键字分段的顺序，每调用一次迭代对象的next，函数内部代码执行一段

1. **Async、Await**

**和Promise，Generator 有很大关联的：**

Async函数返回一个Promise，await可以等待Promise实例状态敲定，自己进行迭代器的next方法

1. **模块化**

模块化开发是一种管理方式，是一种生产方式，一种解决问题的方案，一个模块就是实现特定功能的文件，有了模块，我们就可以更方便地使用别人的代码，想要什么功能，就加载什么模块，但是模块开发需要遵循一定的规范，否则就都乱套了，因此，才有了后来大家熟悉的AMD规范，CMD规范

**CommonJS：**

一个单独的文件就是一个模块，主要运行与服务器端，同步加载模块。require输入其他模块提供的功能 module.exports规范模块对外接口，输出一个值的拷贝。 输出之后不能改变，会缓存起来

**AMD：**

异步加载，一个单独文件一个模块，主要运行于浏览器端，模块和模块的依赖可以被异步加载。define定义模块 require用于输入其他模块提供的功能.规范模块对外接口。 define.amd是一个对象，表明函数遵守AMD规范。AMD的运行逻辑是，提前加载，提前执行，申明依赖模块的时候，会第一时间加载并执行模块内的代码，使后面的回调函数能在所需的环境中运行

**CMD：**

通用模块。一个文件一个模块。主要在浏览器中运行，define全局函数，定义模块，通过exports向外 提供接口，用require获取接口，使用某个组件时用use()调用。通过require引入的模块，只有当程序运行到这里时候才会加载执行

**UMD：**

通用模块。解决commonJS和AMD不能通用的问题。define.amd存在，执行AMD规范； module.exports,执行CommonJS规范；都没有，原始代码规范。

**ES6 Module：**  
 ES6模块功能主要由两个命令构成： import 和 export 。 import 命令用于输入其他模块提供的功能。 export 命令用于规范模块的对外接口