**1.**

**ES6 引入了块级作用域，明确允许在块级作用域之中声明函数。**

**2.ES6 的块级作用域必须有大括号，如果没有大括号，JavaScript 引擎就认为不存在块级作用域。**

**3.Promise**

**当某个任务的完成时间不确定或太长时，我们可以创建一个 promise 。例如 —— 根据连接速度的不同，一个网络请求可能需要 10 ms 甚至需要 200 ms 这么久。我们不想等待这个数据获取的过程。对你而言，200 ms 可能看起来很少，但对于计算机来说是一段非常漫长的时间。promise 的目的就是让这种异步（asynchrony）变得简单而轻松。**

JavaScript 代码：

1. const myPromise = new Promise((resolve, reject) => {
2. if (Math.random() \* 100 < = 90) {
3. resolve('Hello, Promises!');
4. }
5. reject(new Error('In 10% of the cases, I fail. Miserably.'));
6. }
7. );
8. ----------------------------------------------------------------------
9. // 两个函数
10. const onResolved = (resolvedValue) => console.log(resolvedValue);
11. const onRejected = (error) => console.log(error);
12. 通过.then()**获取** resolve **或** reject **函数传递过来的值,** 第一个回调在 promise 被 **解决**时调用。第二个回调在 promise 被 **拒绝**时调用。
13. myPromise.then(onResolved, onRejected);
15. // 效果同上，代码更加简明扼要
16. myPromise.then((resolvedValue) => {
17. console.log(resolvedValue);
18. }, (error) => {
19. console.log(error);
20. });
22. // 有 90% 的概率输出下面语句
24. // resolving the promise ...
25. // Hello, Promises!
26. // Hello, Promises!

这个构造函数就可以发现其接收一个带有两个参数的函数，这个函数被称为 **执行器**函数，并且它 **描述了需要完成的计算**。执行器函数的参数通常被称为 **resolve（解决）** 和 **reject（拒绝）**，分别标记执行器函数的成功和不成功的最终完成结果。

resolve 和 reject 本身也是函数，它们用于将返回值返回给 promise 对象。当计算成功或未来值准备好时，我们使用 resolve 函数将值返回。 **这时我们说这个 promise 已经被成功解决（resolve）了**。

如果计算失败或遇到错误，我们通过在 reject 函数中传递错误对象告知 promise 对象。 **这时我们说这个 promise 已经被拒绝（reject）了**。 reject 可以接收任何类型的值。但是，建议传递一个 Error 对象，因为它可以通过查看堆栈跟踪来帮助调试。

**使用 Promise**

我们创建了一个 promise 并将其存储在 myPromise 中。 **那我们如何才能获取通过** resolve **或** reject **函数传递过来的值呢**？所有的 Promise 都有一个 .then() 方法。.then() 接收两个回调函数。第一个回调在 promise 被 **解决**时调用。第二个回调在 promise 被 **拒绝**时调用。

一些需要注意的 **重要**事项

* 一个 promise 只能成功（resolved）或失败（reject）一次。它不能成功或失败两次，也不能从成功切换到失败，反之亦然。
* 如果一个 promise 在你添加成功/失败回调（即 .then）之前就已经成功或者失败，则 promise 还是会正确地调用回调函数，即使事件发生地比添加回调函数要早。