2-44 Promise

```
function loadScript (src) {
  let script = document.createElement('script')
  script.src = src
  document.head.append(script)
}
```

这个函数的作用是加载一个新的脚本。当使用 `<script src="…">` 将其添加到文档中时，浏览器就会对它进行加载和执行。

我们可以像这样使用：

```
// 加载并执行脚本
loadScript('/my/script.js');
```

函数是异步调用的，因为操作不是此刻（加载脚本）完成的，而是之后。

调用初始化脚本加载，然后继续执行。当脚本正在被加载时，下面的代码可能已经完成了执行，如果加载需要时间，那么同一时间，其他脚本可能会被运行。

```
loadScript('/my/script.js');
// 下面的代码在加载脚本时，不会等待脚本被加载完成
// ...
```

现在，我们假设想在新脚本被加载完成时，被立即使用。它可能声明了新函数，因此我们想要运行它们。

但如果我们在 loadScript(…) 调用后，立即那么做，就会导致操作失败。

```
loadScript('/my/script.js'); // 脚本含有 "function newFunction() {…}"

newFunction(); // 没有这个函数！
```

很明显，浏览器没有时间去加载脚本。因此，对新函数的立即调用失败了。loadScript 函数并没有提供追踪加载完成时方法。脚本加载然后最终执行，仅此而已。但我们希望了解脚本何时加载完成，以使用其中的新函数和新变量。

我们将 callback 函数作为第二个参数添加至 loadScript 中，函数在脚本加载时被执行：

```
function loadScript (src, callback) {
  let script = document.createElement('script')
  script.src = src
  script.onload = () => callback(script)
  document.head.append(script)
}
```

如果现在你想从脚本中调用新函数，我们应该在回调函数中那么写：

```
loadScript('/my/script.js', function() {
  // 在脚本被加载后，回调才会被运行
  newFunction(); // 现在起作用了
  ...
});
```

这个过程大家并不陌生，可是如果在回调之后再回调呢？

```
loadScript('/my/script.js', function (script) {
  alert(`Cool, the ${script.src} is loaded, let's load one more`)

  loadScript('/my/script2.js', function (script) {
    alert(`Cool, the second script is loaded`)
  })
})
```

或者