Promise, async/await

2021年11月27日 星期六　下午 8:29

1. callback回调

回调示例

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

loadScript('https://cdnjs.cloudflare.com/ajax/libs/lodash.js/3.2.0/lod
ash.js'
  ,script=>{
  alert(`Cool, the script ${script.src} is loaded`);
  alert(_ );// 所加载的脚本中声明的函数
});
```

回调缺陷：如果要依次加载脚本，就需要在回调中加载，也就是在回调中回调，也就是**回调地狱**

2. 有ERROR处理的回调

```
function loadScript(src,callback) {
  let script = document.createElement('script');
  script.src = src;

  script.onload=()=>callback(null,script);
  script.onerror=()=>callback(newError(`Script load error for
${src}`)); //一旦出现 ERROR，callback(err) 就会被调用。
//1. callback 的第一个参数是为 error 而保留的。
loadScript('/my/script.js', function(error, script) {
  if (error) {
    // 处理 error
  } else {
    // 脚本加载成功
  }
});
//2. 第二个参数（和下一个参数，如果需要的话）用于成功的结果。此
时 callback(null, result1, result2…) 就会被调用。

  document.head.append(script);
}
```

Promises	Callbacks
Promises 允许我们按照自然顺序进行编码。首先，我们运行 `loadScript` 和 `.then` 来处理结果。	在调用 `loadScript(script, callback)` 时，在我们处理的地方（disposal）必须有一个 `callback` 函数。换句话说，在调用 `loadScript` **之前**，我们必须知道如何处理结果。
我们可以根据需要，在 promise 上多次调用 `.then`。每次调用，我们都会在"订阅列表"中添加一个新的"分析"，一个新的订阅函数。在下一章将对此内容进行详细介绍：Promise 链。	只能有一个回调。

3. Promise & .then\.catch\.finally

Promise(传入两个回调函数 => executor代码运行结束后，由status决定使用哪个函数回调)

e.g.: ajax加载数据 -> 获得这些数据去做其他操作 / throw ERROR

```
let promise = new Promise(function(resolve, reject) {
  // executor
});
```

当 executor 获得了结果，无论是早还是晚都没关系，它应该调用以下回调之一：

- resolve(value) — 如果任务成功完成并带有结果 value。
- reject(error) — 如果出现了 error，error 即为 error 对象。

Promise 对象的 state 和 result 属性都是内部的。我们无法直接访问它们。但我们可以对它们使用 .then/.catch/.finally 方法

- .then method

```
promise.then(
  result => alert(result), // 1 秒后显示 "done!"
  error => alert(error) // 不运行 对error不感兴趣可以不传入reject
);
```

- .catch method

如果我们只对 error 感兴趣，那么我们可以使用 null 作为第一个参数：.then(null, errorHandlingFunction)。或者我们也可以使用 .catch(errorHandlingFunction)，其实是一样的.

- .finally method

.finally(f) 调用与 .then(f, f) 类似，在某种意义上，f 总是在 promise 被 settled 时运行：即 promise 被 resolve 或 reject。

finally 是执行清理（cleanup）的很好的处理程序（handler）。

这非常方便，因为 finally 并不是意味着要处理 promise 的结果。所以它将结果传递了下去。

promise加载script实例：

```
function loadScript(src) {
  return new Promise(function(resolve, reject) {
    let script = document.createElement('script');
    script.src = src;

    script.onload = () => resolve(script);
    script.onerror = () => reject(new Error(`Script load error for
${src}`));

    document.head.append(script);
  });
}

let promise =
loadScript("https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.17.11/l
odash.js");

promise.then(
  script => alert(`${script.src} is loaded!`),
```

这非常方便，因为 finally 并不是意味着要处理 promise 的结果。所以它将结果传递了下去。

4.Promise 链

pormise then 通过result来给handler传递参数

```
New Promise(  )
.then( (resolve) => {
    …
    return result;
} )
```

promise.then 的调用会返回了一个 promise，所以我们可以在其之上调用下一个 .then。

当处理程序（handler）返回一个值时，它将成为该 promise 的 result，所以将使用它调用下一个 .then。

5. 使用 promise 进行错误处理

Promise 链在错误（error）处理中十分强大。当一个 promise 被 reject 时，控制权将移交至最近的 rejection 处理程序（handler）。

将 .catch 附加到链的末尾：任意一个 promise 被 reject（网络问题或者无效的 json 或其他），.catch 就会捕获它。这在实际开发中非常方便。

也可以.then() .catch .then .catch 每个catch对应一块区域的handler

如果没有捕捉error，那么此error会出现在window作用域中。

6. Promise API

promise的五种静态方法

- promise.all([…promises…]) 执行多个promise，结束后返回一个新的 promise，可以继续执行then 操作。如果其中一个reject，则后续的都失效，返回reject
- promise.allSettled([…promises…]) 类似all，但是不管reject，内部 promise settled就行
- Promise.race
  与 Promise.all 类似，但只等待第一个 settled 的 promise 并获取其结果（或 error）。
- Promise.resolve/reject
  在现代的代码中，很少需要使用 Promise.resolve 和 Promise.reject 方法，因为 async/await 语法（我们会在 稍后 讲到）使它们变得有些过时了。

7. Promisify

https://zh.javascript.info/promisify

将函数变成 return promise

类似于包装，工厂模式最后的promise可以顺利进行then catch操作

8. 微任务队列（Microtask queue）

异步任务需要适当的管理。为此，ECMA 标准规定了一个内部队列 PromiseJobs，通常被称为“微任务队列（microtask queue）” （ES6 术语）。

先进先出

9. Async/await

- 本质：promise的语法糖，promisify别的函数

Async ：

```
async function f() {
  return 1;
}
f( ).then(alert);
```

函数前面的关键字 async 有两个作用：

- 让这个函数总是返回一个 promise。
- 允许在该函数内使用 await。

Await ：

await关键词，只在 async 函数内工作、

Promise 前的关键字 await 使 JavaScript 引擎等待该 promise settle，然后：

如果有 error，就会抛出异常 — 就像那里调用了 throw error 一样。

否则，就返回结果。

async/await示例：

```
async function loadJson(url) { // (1)
  let response = await fetch(url); // (2)

  if (response.status == 200) {
    let json = await response.json(); // (3)
    return json;
  }

  throw new Error(response.status);
}

loadJson('no-such-user.json')
  .catch(alert); // Error: 404 (4)
```

```
loadScript("https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.17.11/lodash.js");

promise.then(
  script => alert(`${script.src} is loaded!`),
  error => alert(`Error: ${error.message}`)
);

promise.then(script => alert('Another handler...'));
```

```
loadScript("/article/promise-chaining/one.js")
 .then(script => loadScript("/article/promise-chaining/two.js"))
 .then(script => loadScript("/article/promise-chaining/three.js"))
 .then(script => {
  // 脚本加载完成，我们可以在这儿使用脚本中声明的函数
  one();
  two();
  three();
 });
```

Mixin

2021年11月29日 星期一　　下午 2:25

```
Object.assign(Menu.prototype, eventMixin);
```

Mixin – 是一个通用的面向对象编程术语：一个包含其他类的方法的类。
用得到的时候，用Object.assign(xx.prototype, Mixin)复制到需要Mixin封装的方法的class原型里。