


Раздел

[Промисы, async/await](#)

Комментарии

Поделиться

[Редактировать на GitHub](#)[🏠 → Язык программирования JavaScript](#)
[→ Промисы, async/await](#) 8-го августа 2019

Промисификация

Промисификация – это длинное слово для простого преобразования. Мы берём функцию, которая принимает колбэк и меняем её, чтобы она вместо этого возвращала промис.

Такие преобразования часто необходимы в реальной жизни, так как многие функции и библиотеки основаны на колбэках, а использование промисов более удобно, поэтому есть смысл «промисифицировать» их.

Например, у нас есть `loadScript(src, callback)` из главы [Введение: колбэки](#).

```
1 function loadScript(src, callback) {  
2   let script = document.createElement('script');  
3   script.src = src;  
4  
5   script.onload = () => callback(null, script);  
6   script.onerror = () => callback(new Error('Ошибка заг  
7  
8   document.head.append(script);  
9 }  
10  
11 // использование:  
12 // loadScript('path/script.js', (err, script) => {...})
```



Давайте промисифицируем её. Новая функция `loadScriptPromise(src)` будет делать то же самое, но будет принимать только `src` (не `callback`) и возвращать промис.



```
1 let loadScriptPromise = function(src) {  
2   return new Promise((resolve, reject) => {  
3     loadScript(src, (err, script) => {  
4       if (err) reject(err)  
5       else resolve(script);  
6     });  
7   })  
8 }  
9  
10 // использование:  
11 // loadScriptPromise('path/script.js').then(...)
```

Теперь `loadScriptPromise` хорошо вписывается в код, основанный на промисах.

Как видно, она передаёт всю работу исходной функции `loadScript`, предоставляя ей колбэк, по вызову которого происходит `resolve/reject` промиса.

На практике нам, скорее всего, понадобится промисифицировать не одну функцию, поэтому имеет смысл сделать для этого специальную «функцию-помощник».

Мы назовём её `promisify(f)` – она принимает функцию для промисификации `f` и возвращает функцию-обёртку.

Эта функция-обёртка делает то же самое, что и код выше: возвращает промис и передаёт вызов оригинальной `f`, отслеживая результат в своём колбэке:

```
1 function promisify(f) {  
2   return function (...args) { // возвращает функцию-обёртку
```

Раздел

[Промисы, async/await](#)

Комментарии

Поделиться



[Редактировать на GitHub](#)



```
3     return new Promise((resolve, reject) => {
4         function callback(err, result) { // наш специальн
5             if (err) {
6                 return reject(err);
7             } else {
8                 resolve(result);
9             }
10        }
11
12        args.push(callback); // добавляем колбэк в конец
13
14        f.call(this, ...args); // вызываем оригинальную ф
15    });
16 };
17 };
18
19 // использование:
20 let loadScriptPromise = promisify(loadScript);
21 loadScriptPromise(...).then(...);
```

Здесь мы предполагаем, что исходная функция ожидает колбэк с двумя аргументами (`err`, `result`). Это то, с чем мы чаще всего сталкиваемся. Тогда наш колбэк – в правильном формате, и `promisify` отлично работает для такого случая.

Но что, если исходная `f` ожидает колбэк с большим количеством аргументов `callback(err, res1, res2, ...)`?

Ниже описана улучшенная функция `promisify`: при вызове `promisify(f, true)` результатом промиса будет массив результатов `[res1, res2, ...]`:

```
1 // promisify(f, true), чтобы получить массив результато
2 function promisify(f, manyArgs = false) {
3     return function (...args) {
4         return new Promise((resolve, reject) => {
5             function callback(err, ...results) { // наш специ
6                 if (err) {
7                     return reject(err);
8                 } else {
9                     // делаем resolve для всех results колбэка, е
10                    resolve(manyArgs ? results : results[0]);
11                }
12            }
13
14            args.push(callback);
15
16            f.call(this, ...args);
17        });
18    };
19 };
20
21 // использование:
22 f = promisify(f, true);
23 f(...).then(arrayOfResults => ..., err => ...)
```

Для более экзотических форматов колбэка, например без `err`: `callback(result)`, мы можем промисифицировать функции без помощника, «вручную».

Также существуют модули с более гибкой промисификацией, например, [es6-promisify](#) или встроенная функция `util.promisify` в Node.js.

Раздел

[Промисы, `async/await`](#)

Комментарии

Поделиться



[Редактировать на GitHub](#)



На заметку:

Промисификация – это отличный подход, особенно, если вы будете использовать `async/await` (см. следующую главу), но она не является тотальной заменой любых колбэков.

Помните, промис может иметь только один результат, но колбэк технически может вызываться сколько угодно раз.

Поэтому промисификация используется для функций, которые вызывают колбэк только один раз. Последующие вызовы колбэка будут проигнорированы.

Проводим [курсы по JavaScript и фреймворкам](#).



Комментарии

перед тем как писать...

