

Раздел

[Промисы, async/await](#)

Навигация по уроку

Асинхронные функции

Await

Обработка ошибок

Итого

Задачи (3)

Комментарии

Поделиться

[Редактировать на GitHub](#)[🏠 → Язык программирования JavaScript](#)  
[→ Промисы, async/await](#) 12-го сентября 2020

## Async/await

Существует специальный синтаксис для работы с промисами, который называется «async/await». Он удивительно прост для понимания и использования.

### Асинхронные функции

Начнём с ключевого слова `async`. Оно ставится перед функцией, вот так:

```
1 async function f() {  
2   return 1;  
3 }
```

У слова `async` один простой смысл: эта функция всегда возвращает промис. Значения других типов оборачиваются в завершившийся успешно промис автоматически.

Например, эта функция возвратит выполненный промис с результатом `1`:

```
1 async function f() {  
2   return 1;  
3 }  
4  
5 f().then(alert); // 1
```

Можно и явно вернуть промис, результат будет одинаковым:

```
1 async function f() {  
2   return Promise.resolve(1);  
3 }  
4  
5 f().then(alert); // 1
```

Так что ключевое слово `async` перед функцией гарантирует, что эта функция в любом случае вернёт промис. Согласитесь, достаточно просто? Но это ещё не всё. Есть другое ключевое слово – `await`, которое можно использовать только внутри `async`-функций.

### Await

Синтаксис:

```
1 // работает только внутри async-функций  
2 let value = await promise;
```

Ключевое слово `await` заставит интерпретатор JavaScript ждать до тех пор, пока промис справа от `await` не выполнится. После чего оно вернёт его результат, и выполнение кода продолжится.

В этом примере промис успешно выполнится через 1 секунду:

```
1 async function f() {  
2  
3   let promise = new Promise((resolve, reject) => {  
4     setTimeout(() => resolve("готово!"), 1000)  
5   });
```

Раздел

[Промисы, async/await](#)

Навигация по уроку

Асинхронные функции

Await

Обработка ошибок

Итого

Задачи (3)

Комментарии

Поделиться



Редактировать на GitHub



```
6
7   let result = await promise; // будет ждать, пока пром
8
9   alert(result); // "готово!"
10  }
11
12  f();
```

В данном примере выполнение функции остановится на строке (\*) до тех пор, пока промис не выполнится. Это произойдёт через секунду после запуска функции. После чего в переменную `result` будет записан результат выполнения промиса, и браузер отобразит alert-окно «готово!».

Обратите внимание, хотя `await` и заставляет JavaScript дожидаться выполнения промиса, это не отнимает ресурсов процессора. Пока промис не выполнится, JS-движок может заниматься другими задачами: выполнять прочие скрипты, обрабатывать события и т.п.

По сути, это просто «синтаксический сахар» для получения результата промиса, более наглядный, чем `promise.then`.

#### ⚠ `await` нельзя использовать в обычных функциях

Если мы попробуем использовать `await` внутри функции, объявленной без `async`, получим синтаксическую ошибку:

```
1 function f() {
2   let promise = Promise.resolve(1);
3   let result = await promise; // SyntaxError
4 }
```

Ошибки не будет, если мы укажем ключевое слово `async` перед объявлением функции. Как было сказано раньше, `await` можно использовать только внутри `async`-функций.

Давайте перепишем пример `showAvatar()` из раздела [Цепочка промисов](#) с помощью `async/await`:

1. Нам нужно заменить вызовы `.then` на `await`.
2. И добавить ключевое слово `async` перед объявлением функции.

```
1 async function showAvatar() {
2
3   // запрашиваем JSON с данными пользователя
4   let response = await fetch('/article/promise-chaining');
5   let user = await response.json();
6
7   // запрашиваем информацию об этом пользователе из git
8   let githubResponse = await fetch('https://api.github.
9   let githubUser = await githubResponse.json();
10
11  // отображаем аватар пользователя
12  let img = document.createElement('img');
13  img.src = githubUser.avatar_url;
14  img.className = "promise-avatar-example";
15  document.body.append(img);
16
17  // ждём 3 секунды и затем скрываем аватар
18  await new Promise((resolve, reject) => setTimeout(res
19
20  img.remove();
21
22  return githubUser;
23 }
24
25 showAvatar();
```

Раздел

[Промисы, async/await](#)

Навигация по уроку

Асинхронные функции

Await

Обработка ошибок

Итого

Задачи (3)

Комментарии

Поделиться



Редактировать на GitHub



### **i** await нельзя использовать на верхнем уровне вложенности

Программисты, узнав об `await`, часто пытаются использовать эту возможность на верхнем уровне вложенности (вне тела функции). Но из-за того, что `await` работает только внутри `асинх`-функций, так сделать не получится:

```
1 // SyntaxError на верхнем уровне вложенности
2 let response = await fetch('/article/promise-chair')
3 let user = await response.json();
```

Можно обернуть этот код в анонимную `асинх`-функцию, тогда всё заработает:

```
1 (async () => {
2   let response = await fetch('/article/promise-chair')
3   let user = await response.json();
4   ...
5 })();
```

### **i** await работает с «thenable»-объектами

Как и `promise.then`, `await` позволяет работать с промис-совместимыми объектами. Идея в том, что если у объекта можно вызвать метод `then`, этого достаточно, чтобы использовать его с `await`.

В примере ниже, экземпляры класса `Thenable` будут работать вместе с `await`:

```
1 class Thenable {
2   constructor(num) {
3     this.num = num;
4   }
5   then(resolve, reject) {
6     alert(resolve);
7     // выполнить resolve со значением this.num * 2
8     setTimeout(() => resolve(this.num * 2), 1000);
9   }
10 };
11
12 async function f() {
13   // код будет ждать 1 секунду,
14   // после чего значение result станет равным 2
15   let result = await new Thenable(1);
16   alert(result);
17 }
18
19 f();
```

Когда `await` получает объект с `.then`, не являющийся промисом, JavaScript автоматически запускает этот метод, передавая ему аргументы – встроенные функции `resolve` и `reject`. Затем `await` приостановит дальнейшее выполнение кода, пока любая из этих функций не будет вызвана (в примере это строка `(*)`). После чего выполнение кода продолжится с результатом `resolve` или `reject` соответственно.

Раздел

[Промисы, async/await](#)

Навигация по уроку

Асинхронные функции

Await

Обработка ошибок

Итого

Задачи (3)

Комментарии

Поделиться



[Редактировать на GitHub](#)



### Асинхронные методы классов

Для объявления асинхронного метода достаточно написать `async` перед именем:

```
1 class Waiter {
2   async wait() {
3     return await Promise.resolve(1);
4   }
5 }
6
7 new Waiter()
8   .wait()
9   .then(alert); // 1
```



Как и в случае с асинхронными функциями, такой метод гарантированно возвращает промис, и в его теле можно использовать `await`.

## Обработка ошибок

Когда промис завершается успешно, `await promise` возвращает результат. Когда завершается с ошибкой – будет выброшено исключение. Как если бы на этом месте находилось выражение `throw`.

Такой код:

```
1 async function f() {
2   await Promise.reject(new Error("Упс!"));
3 }
```



Делает то же самое, что и такой:



```
1 async function f() {
2   throw new Error("Упс!");
3 }
```

Но есть отличие: на практике промис может завершиться с ошибкой не сразу, а через некоторое время. В этом случае будет задержка, а затем `await` выбросит исключение.

Такие ошибки можно ловить, используя `try...catch`, как с обычным `throw`:

```
1 async function f() {
2
3   try {
4     let response = await fetch('http://no-such-url');
5   } catch(err) {
6     alert(err); // TypeError: failed to fetch
7   }
8 }
9
10 f();
```



В случае ошибки выполнение `try` прерывается и управление прыгает в начало блока `catch`. Блоком `try` можно обернуть несколько строк:

```
1 async function f() {
2
3   try {
4     let response = await fetch('/no-user-here');
5     let user = await response.json();
```



Раздел

[Промисы, async/await](#)

Навигация по уроку

Асинхронные функции

Await

Обработка ошибок

Итого

Задачи (3)

Комментарии

Поделиться



[Редактировать на GitHub](#)



```
6     } catch(err) {
7         // перехватит любую ошибку в блоке try: и в fetch,
8         alert(err);
9     }
10 }
11
12 f();
```

Если у нас нет `try..catch`, асинхронная функция будет возвращать завершившийся с ошибкой промис (в состоянии `rejected`). В этом случае мы можем использовать метод `.catch` промиса, чтобы обработать ошибку:

```
1 async function f() {
2     let response = await fetch('http://no-such-url');
3 }
4
5 // f() вернёт промис в состоянии rejected
6 f().catch(alert); // TypeError: failed to fetch // (*)
```

Если забыть добавить `.catch`, то будет сгенерирована ошибка «Uncaught promise error» и информация об этом будет выведена в консоль. Такие ошибки можно поймать глобальным обработчиком, о чём подробно написано в разделе [Промисы: обработка ошибок](#).

#### **i** async/await и promise.then/catch

При работе с `async/await`, `.then` используется нечасто, так как `await` автоматически ожидает завершения выполнения промиса. В этом случае обычно (но не всегда) гораздо удобнее перехватывать ошибки, используя `try..catch`, нежели чем `.catch`.

Но на верхнем уровне вложенности (вне `async`-функций) `await` использовать нельзя, поэтому `.then/catch` для обработки финального результата или ошибок – обычная практика.

Так сделано в строке `(*)` в примере выше.

#### **i** async/await отлично работает с Promise.all

Когда необходимо подождать несколько промисов одновременно, можно обернуть их в `Promise.all`, и затем `await`:

```
1 // await будет ждать массив с результатами выполне
2 let results = await Promise.all([
3     fetch(url1),
4     fetch(url2),
5     ...
6 ]);
```

В случае ошибки она будет передаваться как обычно: от завершившегося с ошибкой промиса к `Promise.all`. А после будет сгенерировано исключение, которое можно отловить, обернув выражение в `try..catch`.

## Итого

Ключевое слово `async` перед объявлением функции:

1. Обязывает её всегда возвращать промис.
2. Позволяет использовать `await` в теле этой функции.

Ключевое слово `await` перед промисом заставит JavaScript дожидаться его выполнения, после чего:

Раздел

[Промисы, async/await](#)

Навигация по уроку

Асинхронные функции

Await

Обработка ошибок

Итого

Задачи (3)

Комментарии

Поделиться



[Редактировать на GitHub](#)



1. Если промис завершается с ошибкой, будет сгенерировано исключение, как если бы на этом месте находилось `throw`.
2. Иначе вернётся результат промиса.

Вместе они предоставляют отличный каркас для написания асинхронного кода. Такой код легко и писать, и читать.

Хотя при работе с `async/await` можно обходиться без `promise.then/catch`, иногда всё-таки приходится использовать эти методы (на верхнем уровне вложенности, например). Также `await` отлично работает в сочетании с `Promise.all`, если необходимо выполнить несколько задач параллельно.

## ✓ Задачи

### Перепишите, используя `async/await` [↗](#)

Перепишите один из примеров раздела [Цепочка промисов](#), используя `async/await` вместо `.then/catch`:

```
1 function loadJson(url) {
2   return fetch(url)
3     .then(response => {
4       if (response.status == 200) {
5         return response.json();
6       } else {
7         throw new Error(response.status);
8       }
9     })
10 }
11
12 loadJson('no-such-user.json') // (3)
13   .catch(alert); // Error: 404
```

решение

### Перепишите, используя `async/await` [↗](#)

Ниже пример из раздела [Цепочка промисов](#), перепишите его, используя `async/await` вместо `.then/catch`.

В функции `demoGithubUser` замените рекурсию на цикл: используя `async/await`, сделать это будет просто.

```
1 class HttpError extends Error {
2   constructor(response) {
3     super(`${response.status} for ${response.url}`);
4     this.name = 'HttpError';
5     this.response = response;
6   }
7 }
8
9 function loadJson(url) {
10  return fetch(url)
11    .then(response => {
12      if (response.status == 200) {
13        return response.json();
14      } else {
15        throw new HttpError(response);
16      }
17    })
18 }
19
20 // Запрашивать логин, пока github не вернёт существующе
21 function demoGithubUser() {
22   let name = prompt("Введите логин?", "iliakan");
23 }
```

Раздел

[Промисы, async/await](#)

Навигация по уроку

Асинхронные функции

Await

Обработка ошибок

Итого

Задачи (3)

Комментарии

Поделиться



[Редактировать на GitHub](#)



```
24     return loadJson(`https://api.github.com/users/${name}`
25     .then(user => {
26         alert(`Полное имя: ${user.name}.`);
27         return user;
28     })
29     .catch(err => {
30         if (err instanceof HttpError && err.response.stat
31             alert("Такого пользователя не существует, пожа
32             return demoGithubUser();
33         } else {
34             throw err;
35         }
36     }));
37 }
38
39 demoGithubUser();
```

решение

### Вызовите async-функцию из "обычной"

Есть «обычная» функция. Как можно внутри неё получить результат выполнения async-функции?

```
1  async function wait() {
2      await new Promise(resolve => setTimeout(resolve, 1000
3
4      return 10;
5  }
6
7  function f() {
8      // ...что здесь написать?
9      // чтобы вызвать wait() и дождаться результата "10" о
10     // не забывайте, здесь нельзя использовать "await"
11 }
```

P.S. Технически задача очень простая, но этот вопрос часто задают разработчики, недавно познакомившиеся с async/await.

решение

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

### Комментарии

перед тем как писать...