

Раздел

[Промисы, async/await](#)

Навигация по уроку


Очередь микрозадач

Необработанные ошибки

Итого

Комментарии

Поделиться

[Редактировать на GitHub](#)[🏠 → Язык программирования JavaScript](#)  
[→ Промисы, async/await](#) 30-го ноября 2019

## Микрозадачи

Обработчики промисов `.then / .catch / .finally` всегда асинхронны.

Даже когда промис сразу же выполнен, код в строках *ниже* `.then / .catch / .finally` будет запущен до этих обработчиков.

Вот демо:

```
1 let promise = Promise.resolve();
2
3 promise.then(() => alert("промис выполнен"));
4
5 alert("код выполнен"); // этот alert показывается первым
```

Если вы запустите его, сначала вы увидите код выполнен, а потом промис выполнен.

Это странно, потому что промис определённо был выполнен с самого начала.

Почему `.then` срабатывает позже? Что происходит?

## Очередь микрозадач

Асинхронные задачи требуют правильного управления. Для этого стандарт предусматривает внутреннюю очередь `PromiseJobs`, более известную как «очередь микрозадач (microtask queue)» (термин V8).

Как сказано в [спецификации](#):


- Очередь определяется как первым-пришёл-первым-ушёл (FIFO): задачи, попавшие в очередь первыми, выполняются тоже первыми.
- Выполнение задачи происходит только в том случае, если ничего больше не запущено.


Или, проще говоря, когда промис выполнен, его обработчики `.then/catch/finally` попадают в очередь. Они пока не выполняются. Движок JavaScript берёт задачу из очереди и выполняет её, когда он освободится от выполнения текущего кода.

Вот почему сообщение «код выполнен» в примере выше будет показано первым.

```
promise.then(handler);
...
alert("code finished");
```

-----  
выполнение скрипта завершено

 **handler в очереди**



**запускается handler из очереди**

Обработчики промисов всегда проходят через эту внутреннюю очередь.

Если есть цепочка с несколькими `.then/catch/finally`, то каждый из них выполняется асинхронно. То есть сначала ставится в очередь, а потом выполняется, когда выполнение текущего кода завершено и добавленные ранее в очередь обработчики выполнены.

**Но что если порядок имеет значение для нас? Как мы можем вывести код выполнен после промис выполнен?**

Легко, используя `.then`:

Раздел

[Промисы, async/await](#)

Навигация по уроку

Очередь микрозадач

Необработанные ошибки

Итого

Комментарии

Поделиться



[Редактировать на GitHub](#)

```
1 Promise.resolve()  
2   .then(() => alert("промис выполнен!"))  
3   .then(() => alert("код выполнен"));
```



Теперь порядок стал таким, как было задумано.



## Необработанные ошибки

Помните «необработанные ошибки» из главы [Промисы: обработка ошибок?](#)

Теперь мы можем описать, как именно JavaScript понимает, что ошибка не обработана.

**"Необработанная ошибка" возникает в случае, если ошибка промиса не обрабатывается в конце очереди микрозадач.**

Обычно, если мы ожидаем ошибку, мы добавляем `.catch` в конец цепочки промисов, чтобы обработать её:

```
1 let promise = Promise.reject(new Error("Ошибка в промисе!"));  
2 promise.catch(err => alert('поймана!'));  
3  
4 // не запустится, ошибка обработана  
5 window.addEventListener('unhandledrejection', event =>  
6   alert(event.reason);  
7 });
```



...Но если мы забудем добавить `.catch`, то, когда очередь микрозадач опустеет, движок сгенерирует событие:

```
1 let promise = Promise.reject(new Error("Ошибка в промисе!"));  
2  
3 // Ошибка в промисе!  
4 window.addEventListener('unhandledrejection', event => {
```



А что, если мы поймем ошибку, но позже? Вот так:

```
1 let promise = Promise.reject(new Error("Ошибка в промисе!"));  
2  
3 setTimeout(() => promise.catch(err => alert('поймана')),  
4  
5 // Ошибка в промисе!  
6 window.addEventListener('unhandledrejection', event => {
```



Теперь, при запуске, мы сначала увидим «Ошибка в промисе!», а затем «поймана».

Если бы мы не знали про очередь микрозадач, то могли бы удивиться: «Почему сработал обработчик `unhandledrejection`? Мы же поймали ошибку!».

Но теперь мы понимаем, что событие `unhandledrejection` возникает, когда очередь микрозадач завершена: движок проверяет все промисы и, если какой-либо из них в состоянии «rejected», то генерируется это событие.

В примере выше `.catch`, добавленный в `setTimeout`, также сработает, но позже, уже после возникновения `unhandledrejection`, так что это ни на что не влияет.

## Итого

Обработка промисов всегда асинхронная, т.к. все действия промисов проходят через внутреннюю очередь «promise jobs», так называемую «очередь микрозадач (microtask queue)» (термин v8).

Таким образом, обработчики `.then/catch/finally` вызываются после выполнения текущего кода.

Раздел

[Промисы, async/await](#)

Навигация по уроку

Очередь микрозадач

Необработанные ошибки

Итого

Комментарии

Поделиться



[Редактировать на GitHub](#)



Если нам нужно гарантировать выполнение какого-то кода после `.then/catch/finally`, то лучше всего добавить его вызов в цепочку `.then`.

В большинстве движков JavaScript, включая браузеры и Node.js, микрозадачи тесно связаны с так называемым «событийным циклом» и «макрозадачами». Так как они не связаны напрямую с промисами, то рассматриваются в другой части учебника, в главе [Событийный цикл: микрозадачи и макрозадачи](#).

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

## Комментарии

перед тем как писать...

