

Раздел

[Промисы, async/await](#)

Навигация по уроку

Неявный try...catch

Пробрасывание ошибок


Необработанные ошибки

Итого

Задачи (1)

Комментарии

Поделиться

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

## Промисы: обработка ошибок

Цепочки промисов отлично подходят для перехвата ошибок. Если промис завершается с ошибкой, то управление переходит в ближайший обработчик ошибок. На практике это очень удобно.

Например, в представленном ниже примере для `fetch` указана неправильная ссылка (сайт не существует), и `.catch` перехватывает ошибку:

```
1 fetch('https://no-such-server.blabla') // ошибка
2   .then(response => response.json())
3   .catch(err => alert(err)) // TypeError: failed to fet
```

Как видно, `.catch` не обязательно должен быть сразу после ошибки, он может быть далее, после одного или даже нескольких `.then`

Или, может быть, с сервером всё в порядке, но в ответе мы получим некорректный JSON. Самый лёгкий путь перехватить все ошибки – это добавить `.catch` в конец цепочки:

```
1 fetch('/article/promise-chaining/user.json')
2   .then(response => response.json())
3   .then(user => fetch(`https://api.github.com/users/${u
4   .then(response => response.json())
5   .then(githubUser => new Promise((resolve, reject) =>
6     let img = document.createElement('img');
7     img.src = githubUser.avatar_url;
8     img.className = "promise-avatar-example";
9     document.body.append(img);
10
11     setTimeout(() => {
12       img.remove();
13       resolve(githubUser);
14     }, 3000);
15   }))
16   .catch(error => alert(error.message));
```

Если все в порядке, то такой `.catch` вообще не выполнится. Но если любой из промисов будет отклонён (проблемы с сетью или некорректная json-строка, или что угодно другое), то ошибка будет перехвачена.

## Неявный try...catch

Вокруг функции промиса и обработчиков находится "невидимый `try...catch`". Если происходит исключение, то оно перехватывается, и промис считается отклонённым с этой ошибкой.

Например, этот код:

```
1 new Promise((resolve, reject) => {
2   throw new Error("Ошибка!");
3 }).catch(alert); // Error: Ошибка!
```

...Работает так же, как и этот:

```
1 new Promise((resolve, reject) => {
2   reject(new Error("Ошибка!"));
3 }).catch(alert); // Error: Ошибка!
```

Раздел

[Промисы, async/await](#)

Навигация по уроку

Неявный try...catch

Пробрасывание ошибок

Необработанные ошибки

Итого

Задачи (1)

Комментарии

Поделиться



Редактировать на GitHub



"Невидимый try...catch" вокруг промиса автоматически перехватывает ошибку и превращает её в отклонённый промис.

Это работает не только в функции промиса, но и в обработчиках. Если мы бросим ошибку (throw) из обработчика (.then), то промис будет считаться отклонённым, и управление перейдёт к ближайшему обработчику ошибок.

Пример:

```
1 new Promise((resolve, reject) => {
2   resolve("ок");
3 }).then((result) => {
4   throw new Error("Ошибка!"); // генерируем ошибку
5 }).catch(alert); // Error: Ошибка!
```

Это происходит для всех ошибок, не только для тех, которые вызваны оператором throw. Например, программная ошибка:

```
1 new Promise((resolve, reject) => {
2   resolve("ок");
3 }).then((result) => {
4   blabla(); // нет такой функции
5 }).catch(alert); // ReferenceError: blabla is not defin
```

Финальный .catch перехватывает как промисы, в которых вызван reject, так и случайные ошибки в обработчиках.

## Пробрасывание ошибок

Как мы уже заметили, .catch ведёт себя как try...catch. Мы можем иметь столько обработчиков .then, сколько мы хотим, и затем использовать один .catch в конце, чтобы перехватить ошибки из всех обработчиков.

В обычном try...catch мы можем проанализировать ошибку и повторно пробросить дальше, если не можем её обработать. То же самое возможно для промисов.

Если мы пробросим (throw) ошибку внутри блока .catch, то управление перейдёт к следующему ближайшему обработчику ошибок. А если мы обработаем ошибку и завершим работу обработчика нормально, то продолжит работу ближайший успешный обработчик .then.

В примере ниже .catch успешно обрабатывает ошибку:

```
1 // the execution: catch -> then
2 new Promise((resolve, reject) => {
3
4   throw new Error("Ошибка!");
5
6 }).catch(function(error) {
7
8   alert("Ошибка обработана, продолжить работу");
9
10 }).then(() => alert("Управление перейдёт в следующий th
```

Здесь блок .catch завершается нормально. Поэтому вызывается следующий успешный обработчик .then.

В примере ниже мы видим другую ситуацию с блоком .catch. Обработчик (\*) перехватывает ошибку и не может обработать её (например, он знает как обработать только URIError), поэтому ошибка пробрасывается далее:

```
1 // the execution: catch -> catch -> then
2 new Promise((resolve, reject) => {
3
```

Раздел

[Промисы, async/await](#)

Навигация по уроку

Неявный try...catch

Пробрасывание ошибок

Необработанные ошибки

Итого

Задачи (1)

Комментарии

Поделиться



[Редактировать на GitHub](#)



```
4   throw new Error("Ошибка!");
5
6 }).catch(function(error) { // (*)
7
8   if (error instanceof URIError) {
9     // обрабатываем ошибку
10  } else {
11    alert("Не могу обработать ошибку");
12
13    throw error; // пробрасывает эту или другую ошибку
14  }
15
16 }).then(function() {
17   /* не выполнится */
18 }).catch(error => { // (**)
19
20   alert(`Неизвестная ошибка: ${error}`);
21   // ничего не возвращаем => выполнение продолжается в
22
23 });
```

Управление переходит от первого блока `.catch (*)` к следующему `(**)`, вниз по цепочке.

## Необработанные ошибки

Что произойдёт, если ошибка не будет обработана? Например, мы просто забыли добавить `.catch` в конец цепочки, как здесь:

```
1 new Promise(function() {
2   noSuchFunction(); // Ошибка (нет такой функции)
3 })
4   .then(() => {
5     // обработчики .then, один или более
6   }); // без .catch в самом конце!
```

В случае ошибки выполнение должно перейти к ближайшему обработчику ошибок. Но в примере выше нет никакого обработчика. Поэтому ошибка как бы «застревает», её некому обработать.

На практике, как и при обычных необработанных ошибках в коде, это означает, что что-то пошло сильно не так.

Что происходит, когда обычная ошибка не перехвачена `try...catch`? Скрипт умирает с сообщением в консоли. Похожее происходит и в случае необработанной ошибки промиса.

JavaScript-движок отслеживает такие ситуации и генерирует в этом случае глобальную ошибку. Вы можете увидеть её в консоли, если запустите пример выше.

В браузере мы можем поймать такие ошибки, используя событие `unhandledrejection`:

```
1 window.addEventListener('unhandledrejection', function(
2   // объект события имеет два специальных свойства:
3   alert(event.promise); // [Object Promise] - промис, к
4   alert(event.reason); // Error: Ошибка! - объект ошибок
5 });
6
7 new Promise(function() {
8   throw new Error("Ошибка!");
9 }); // нет обработчика ошибок
```

Это событие является частью [стандарта HTML](#).

Если происходит ошибка, и отсутствует её обработчик, то генерируется событие `unhandledrejection`, и соответствующий объект `event` содержит информацию об ошибке.

Раздел

[Промисы, async/await](#)

Навигация по уроку

Неявный try...catch

Пробрасывание ошибок

Необработанные ошибки

Итого

Задачи (1)

Комментарии

Поделиться



[Редактировать на GitHub](#)



Обычно такие ошибки неустранимы, поэтому лучше всего – информировать пользователя о проблеме и, возможно, отправить информацию об ошибке на сервер.

В не-браузерных средах, таких как Node.js, есть другие способы отслеживания необработанных ошибок.

## Итого

- `.catch` перехватывает все виды ошибок в промисах: будь то вызов `reject()` или ошибка, брошенная в обработчике при помощи `throw`.
- Необходимо размещать `.catch` там, где мы хотим обработать ошибки и знаем, как это сделать. Обработчик может проанализировать ошибку (могут быть полезны пользовательские классы ошибок) и пробросить её, если ничего не знает о ней (возможно, это программная ошибка).
- Можно и совсем не использовать `.catch`, если нет нормального способа восстановиться после ошибки.
- В любом случае нам следует использовать обработчик события `unhandledrejection` (для браузеров и аналог для других окружений), чтобы отслеживать необработанные ошибки и информировать о них пользователя (и, возможно, наш сервер), благодаря чему наше приложение никогда не будет «просто умирать».

## ✓ Задачи

### Ошибка в `setTimeout`

Что вы думаете? Выполнится ли `.catch`? Поясните свой ответ.

```
1 new Promise(function(resolve, reject) {
2   setTimeout(() => {
3     throw new Error("Whoops!");
4   }, 1000);
5 }).catch(alert);
```

решение

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



## 💬 Комментарии

перед тем как писать...