JS

Promise

Andrey Belov | 2025

Promise

Промис (по англ. promise - обещание) – это специальный объект в JavaScript, который связывает код «обещающий» выполнить какую-то операцию и код «потребляющий» результаты.

Promise

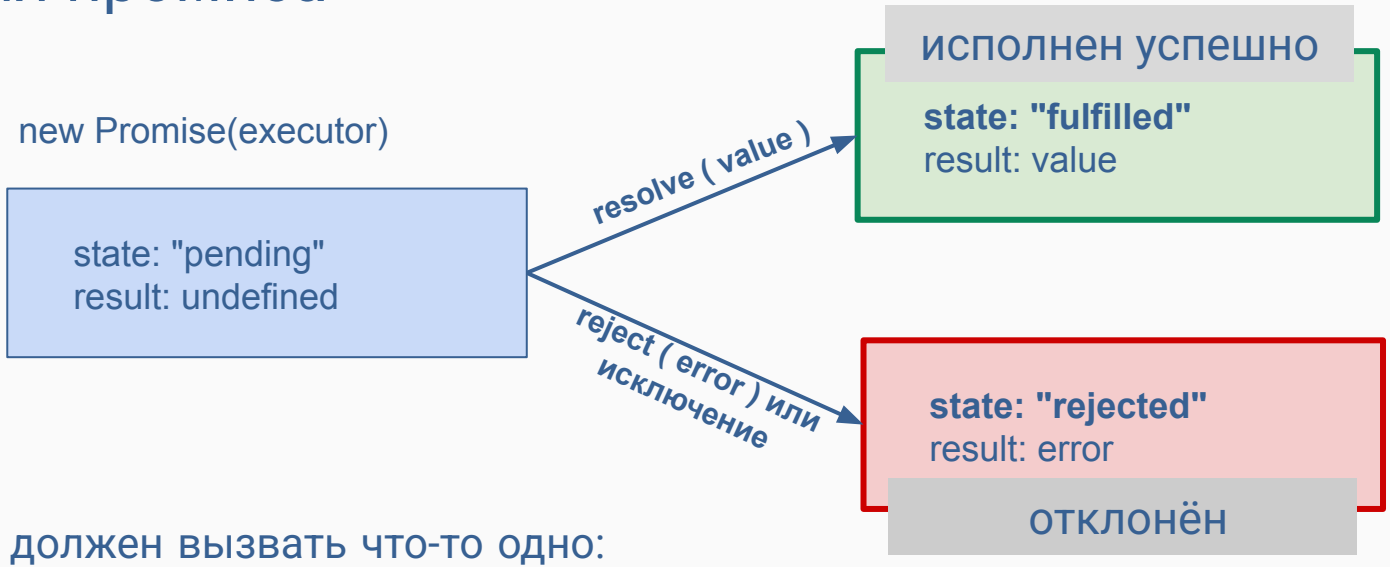
```
let promise = new Promise(function(resolve, reject) {
  // функция-исполнитель (executor)
});
```

Функция исполнитель (**executor)** — описывает выполнение какой-то асинхронной работы, вызывается при создание promise, а по завершении должна вызвать одну из функций-параметров resolve или reject.

- **resolve(value)** — если работа завершилась успешно, с результатом value
- **reject(error)** — если произошла ошибка.

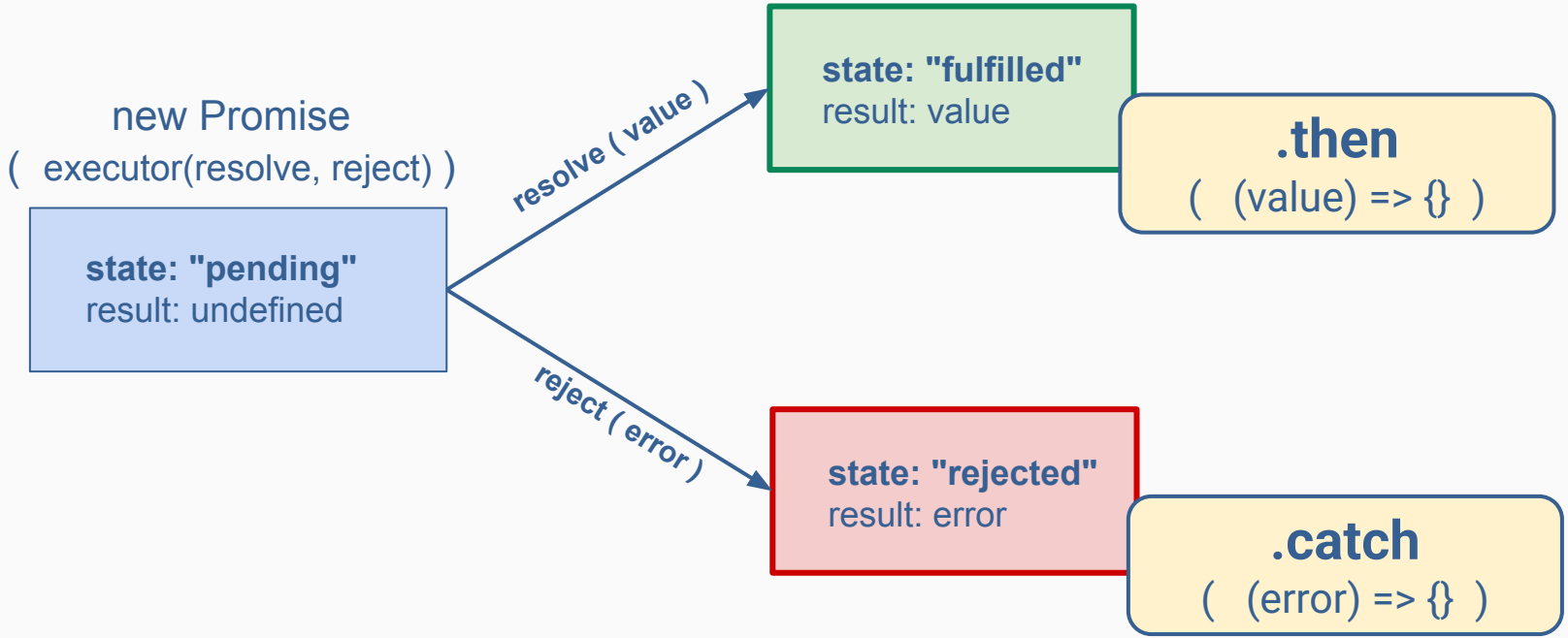
resolve и **reject** – это колбеки, которые предоставляет JavaScript

Состояния промиса



Исполнитель должен вызвать что-то одно: resolve или reject. Состояние промиса может быть изменено только один раз.

Обработка результата



Promise

Когда мы вызываем .then, .catch или .finally, мы регистрируем обработчики, которые будут вызваны позже, когда промис перейдёт в конечное состояние.

.then

```
promise
.then(
    function(result) { /* обработает успешное выполнение */ },
    function(error) { /* обработает ошибку */ }
);
```

- Первый аргумент метода .then – функция, которая выполняется, когда промис переходит в состояние «fulfilled», и получает результат.
- Второй аргумент .then (необязательный) – функция, которая выполняется, когда промис переходит в состояние «rejected», и получает ошибку.

Promise

.catch

```
promise
  .catch( function(error) { /* обработает ошибку */ });
```

catch позволяет только обработать ошибку. Вызов .catch(f) – это сокращённый, «укороченный» вариант .then(null, f)

.finally

функция завершения операции вне зависимости от результата

Цепочки promise: последовательные операции

.then всегда возвращает новый promise

```
promise
   .then(....)
   .then(....)
   .then(....)
   .catch(....);
```

Когда обработчик в .then возвращает:

- обычное значение → оно «оборачивается» в Promise.resolve(value).
- promise → следующий .then будет ждать его.
- ошибку (throw) → промис становится rejected

.catch ловит ошибки из всей цепочки выше.

.finally не меняет значение/ошибку, просто «проходит мимо»

Promise.all

```
Promise.all([
  fetch('/api/a').then(r => r.json()),
  fetch('/api/b').then(r => r.json())
]).then(([a, b]) => {
  console.log('Оба готовы:', a, b);
}).catch(err => {
  console.error('Упала хотя бы одна:', err);
});
```

Promise.all (массив промисов) — ждёт выполнение всех промисов, возвращает новый промис.

Если все промисы выполнятся успешно, то вернётся массив их результатов в том же порядке, в котором они были переданы.

Если хотя бы один промис отклонится (reject), то весь Promise.all сразу отклонится с этой ошибкой.

Promise.any

Promise.any (массив промисов) — считается выполненным, когда один из промисов выполнен успешно.

Если все промисы завершаться с ошибкой, то Promise.any отклоняется с объектом AggregateError (специальный тип ошибки, который содержит массив всех ошибок).

```
Promise.any([tryMirror1(), tryMirror2(), tryMirror3()])
  .then(value => console.log('Самый быстрый успех:', value))
  .catch(err => console.error('Все упали:', err.errors));
```

Promise.allSettled

Promise.allSettled (массив промисов) — всегда успешно завершается, когда все промисы завершились — не важно, с результатом (resolve) или с ошибкой (reject).

Результатом будет массив объектов вида:

```
[
  { status: "fulfilled", value: ... }, // если промис выполнился
  { status: "rejected", reason: ... }  // если промис отклонился
]
```

Promise.race

Promise.race (массив промисов) — завершается или отклоняется с тем значением, с которым завершится самый первый промис из переданных.

Кто первый:

```
const p1 = new Promise(resolve => setTimeout(() => resolve("Успех от p1"), 500));
const p2 = new Promise(resolve => setTimeout(() => resolve("Успех от p2"), 100));

Promise.race([p1, p2])
  .then(result => console.log("Результат:", result))
  .catch(error => console.error("Ошибка:", error));
```