JS: Promises

AIT
Hi-tech team

НАШИ ПРАВИЛА

Включенная камера

Вопросы по поднятой руке

Не перебиваем друг друга

Все вопросы, не связанные с тематикой курса (орг-вопросы и т. д.), должны быть направлены куратору

Подготовьте свое рабочее окружение для возможной демонстрации экрана (закройте лишние соцсети и прочие приложения)

ПОИГРАЕМ ;)

- Какие аргументы принимает setTimeout?
- Для чего используется setInterval?
- Что возвращают setTimeout и setInterval?
- Как остановить таймер?
- Расскажите как работает синхронный и асинхронный код
- Что такое callback функция?

ЦЕЛЬ

Изучить новую возможность работы с асинхронным кодом - Promise

ПЛАН ЗАНЯТИЯ

- Определение Promise
- Состояния объекта Promise (fullfilled, rejected, pending)
- Методы resolve, reject
- Методы then, catch, finally

Promise

Определение promise

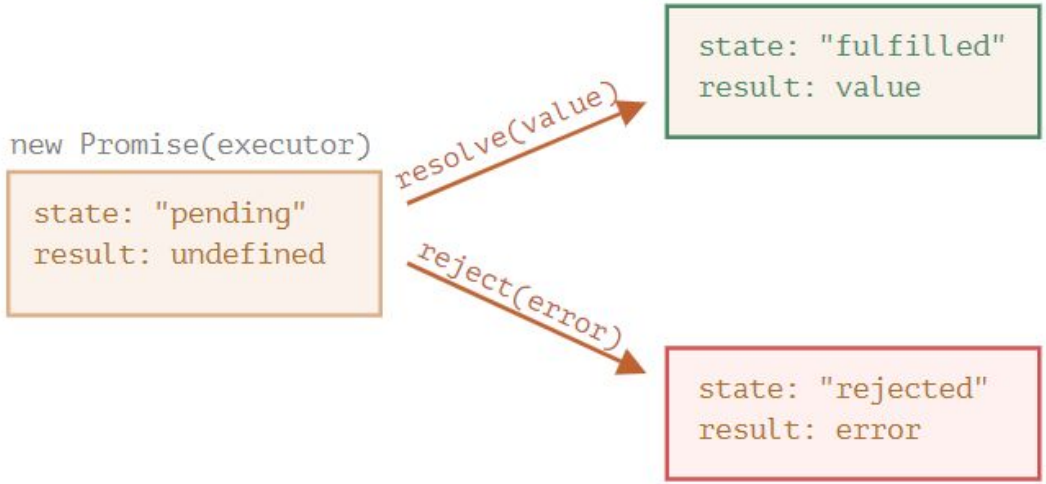
Промис (promise) - это объект, представляющий результат успешного или неудачного завершения асинхронной операции. Асинхронная операция, упрощенно говоря, это некоторое действие, которое выполняется независимо от окружающего ее кода, в котором она вызывается, не блокируя выполнение вызываемого кода.

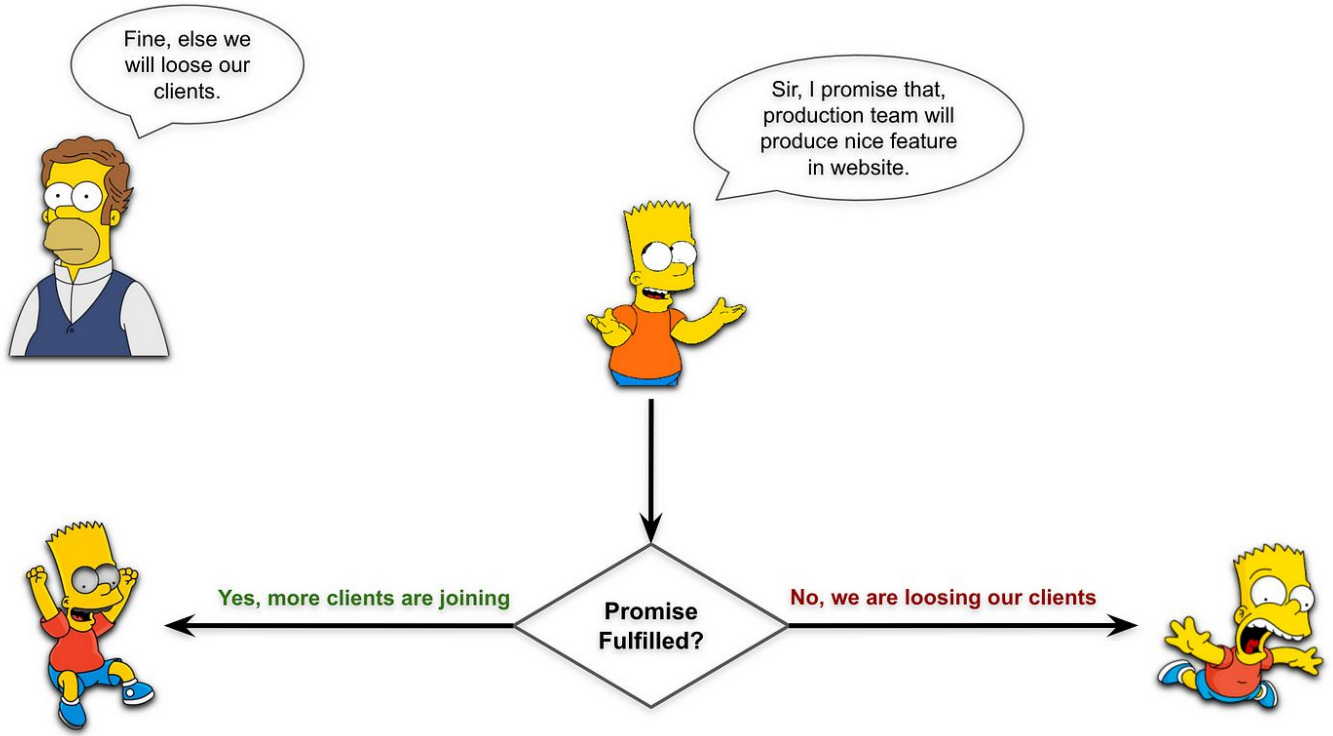
Есть «создающий» код, который делает что-то, что занимает время. Например, загружает данные по сети.

Есть «потребляющий» код, который хочет получить результат «создающего» кода, когда он будет готов. Он может быть необходим более чем одной функции.

Промис может находиться в одном из трёх состояний:

- **pending** — стартовое состояние, операция стартовала;
- **fulfilled** — получен результат;
- **rejected** —получена ошибка;




Fine, else we will loose our clients.
Sir, I promise that, production team will produce nice feature in website.
Yes, more clients are joining
Promise Fulfilled?
No, we are loosing our clients

Ниже пример конструктора Promise и простого исполнителя с кодом, дающим успешный результат с задержкой (через setTimeout):

```
let promise = new Promise(function(resolve, reject) {
  // эта функция выполнится автоматически, при вызове new Promise

  // через 1 секунду сигнализировать, что задача выполнена с результатом "done"
  setTimeout(() => resolve("done"), 1000);
});
```

Ниже пример конструктора Promise и простого исполнителя с кодом, дающим результат с ошибкой с задержкой (через setTimeout):

```
let promise = new Promise(function(resolve, reject) {
  // спустя одну секунду будет сообщено, что задача выполнена с ошибкой
  setTimeout(() => reject(new Error("Whoops!")), 1000);
});
```

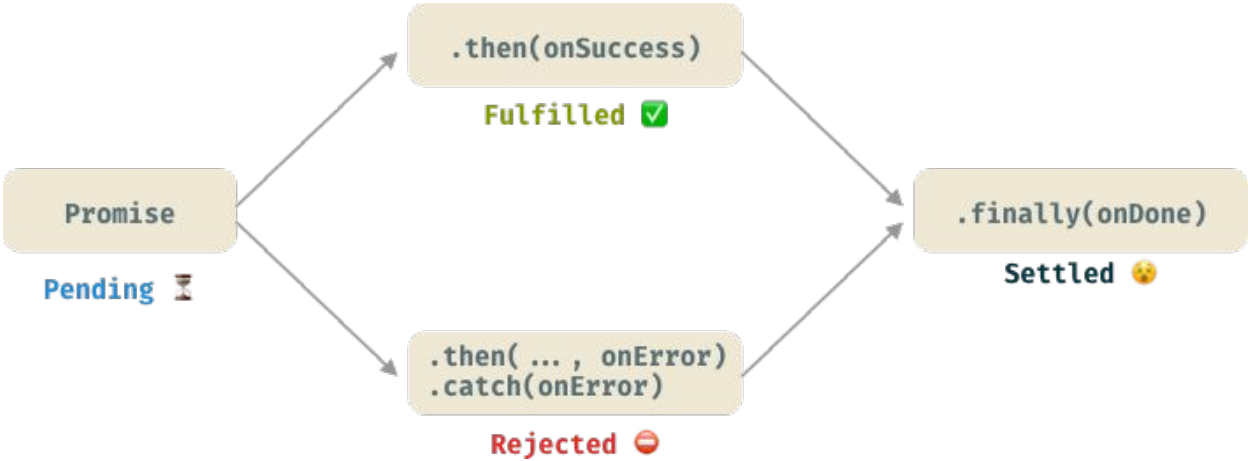
Состояние промиса может быть изменено только один раз. Все последующие вызовы resolve и reject будут проигнорированы:

```
let promise = new Promise(function(resolve, reject) {
  resolve("done");

  reject(new Error("…")); // игнорируется
  setTimeout(() => resolve("…")); // игнорируется
});
```

Существует три метода, которые позволяют работать с результатом выполнения вычисления внутри промиса:

- then()
- catch()
- finally()



Метод then()

Первый аргумент метода .then – функция, которая выполняется, когда промис переходит в состояние «выполнен успешно», и получает результат.

Второй аргумент .then – функция, которая выполняется, когда промис переходит в состояние «выполнен с ошибкой», и получает ошибку.

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

Метод catch()

Если мы хотели бы только обработать ошибку, то можно использовать null в качестве первого аргумента: .then(null, errorHandlingFunction). Или можно воспользоваться методом .catch(errorHandlingFunction), который сделает то же самое:

```
let promise = new Promise((resolve, reject) => {
  setTimeout(() => reject(new Error("Ошибка!")), 1000);
});

// .catch(f) это то же самое, что promise.then(null, f)
promise.catch(alert); // выведет "Error: Ошибка!" спустя одну секунду
```

Метод finally()

Вызов .finally(f) похож на .then(f, f), в том смысле, что f выполнится в любом случае, когда промис завершится: успешно или с ошибкой.

```
new Promise((resolve, reject) => {
  /* сделать что-то, что займёт время, и после вызвать resolve или может reject */
})
  // выполнится, когда промис завершится, независимо от того, успешно или нет
  .finally(() => остановить индикатор загрузки)
  // таким образом, индикатор загрузки всегда останавливается, прежде чем мы продолжим
  .then(result => показать результат, err => показать ошибку)
```

Ваша новая IT-профессия – Ваш новый уровень жизни

Программирование с нуля в
немецкой школе AIT TR GmbH

AIT
Hi-tech team