Theory: Introduction to Promises

18 minutes 0 / 5 problems solved Skip this topic Start practicing

449 users solved this topic. Latest completion was 2 days ago.

Imagine your friend promised to hire you as a frontend developer if her company opens a Junior position, and you started learning all JS topics. You will spend the next two months memorizing frontend features, waiting to get your friend's response. If the result is positive, you will make a big party to celebrate your first real programmer job; otherwise, you will continue studying JS. In this case, "**promise**" means that you have only an agreement for now, and your friend will contact you later when there is an actual result.

The same concept exists in programming: a function is waiting for a result, and when it gets a response it performs some action. Its behavior depends on whether the outcome is positive or negative. Let's take a closer look at how it works and consider specific examples.

Current topic:

Introduction to Promises ···

Topic depends on:

✕ Conditional operators ···

Topic is required for:

"then", "catch" and "finally" methods ···

§1. Promise syntax

Take a look at the general schema of the Promise syntax:

```
let promise = new Promise(function(resolve, reject) {
 // code to be executed
});
```

Inside a new Promise, there is a function called **executor** that immediately launches the created promise. You don't know the result right away: the code only shows that you will get a value later. The most significant benefit of promises is that the program continues running while the promised value is being calculated. As soon as the executed function has finished, you will see the result.

For example, as a user, you can see the whole website without a loaded avatar, so you can interact with the site instead of staring at a blank page for 30 seconds while the browser is loading the personal image. Once the avatar has been loaded, you will see it where it's supposed to be.

There are two arguments inside the executor: `resolve` and `reject`. These pre-defined functions are called depending on whether the promise was fulfilled successfully or unsuccessfully:

- `resolve(value)` in case of success with the result `value`;
- `reject(error)` in case of failure with the error object as `error`.

Remember that the executor calls either the resolve or reject function: it cannot invoke both of them.

Table of contents:

§2. Examples

Let's go back to your hypothetical programming job. Here is a respective promise with data:

```
const myFriendHasApprovedMyPosition = true;

let promise = new Promise(function(resolve, reject) {
  if (myFriendHasApprovedMyPosition){
    resolve("Hooray! Now I'm a true programmer!");
  } else {
    reject(new Error("Whoops! Have to study more =("));
 }
});
```

In this case, we created a promise depending on the `myFriendHasApprovedMyPosition` constant. If the value is true, we call the resolve function with `"Hooray! Now I'm a true programmer!"`; if the value is false, we use the reject function with the Error object with `"Whoops! Have to study more =("`. As you can see, the value is true, so the resolve function is executed.

In the example, we used the resolve function immediately because the value was known. In reality, programmers mostly use promises in cases when executing a function takes some time. Let's take a look at an example with `setTimeout`:

```
let promise = new Promise(function(resolve, reject) {
  setTimeout(() => resolve("I have completed"), 5000);
});
```

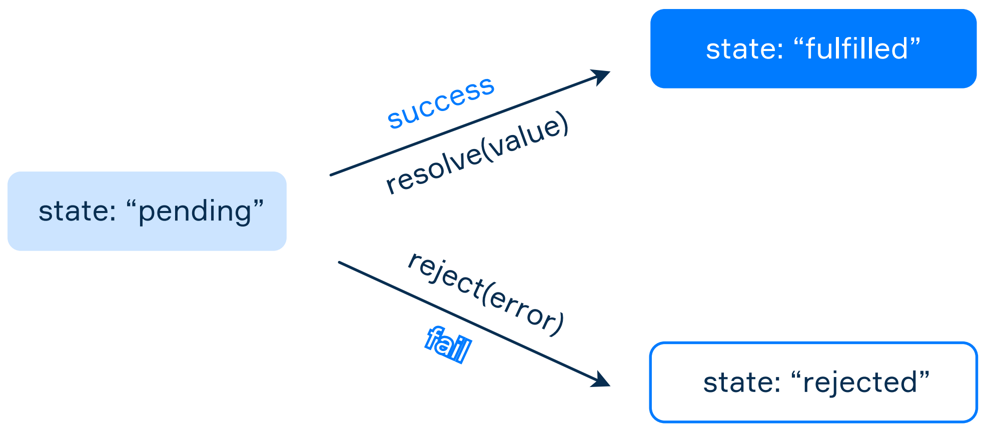
Now, in this situation, the browser has launched the promise function immediately, but the resolve function will be called only in 5 seconds. This is particularly helpful when we need to load data from a remote source.

§3. Promise states

A promise is an **object** that has a **state** property. At any time, Promise can be in one of these states:

- **pending**: the initial state when a promise has launched but hasn't settled;
- **fulfilled**: the promise has completed successfully, the resolve function is called;
- **rejected**: the promise has failed, the reject function is called.

To understand the process, take a look at the diagram:



As you can see, at first the promise has the *"pending"* state. In case of success, the executor calls resolve(value) and changes the promise state to *"fulfilled"*. If the promise has failed, it calls reject(error), which sets the promise state to *"rejected"*. The promised state can be changed only once.

§4. Conclusion

Promises are a convenient feature that allows executing a function that takes some time to finish without pausing other processes. Promises let you start the function immediately but set a result only after the process is completed. It's especially useful in case of loading large data when you want to show the loader to a user while the response is pending. In this topic, we have covered the main idea of promises, and later you will learn how to use the result of promises in your code.

Report a typo

41 users liked this theory. 1 didn't like it. **What about you?**

😍 🙂 😐 🙁 😡

Start practicing

Comments (1) Hints (0) Useful links (1) Show discussion