Theory: "then", "catch" and "finally" methods

25 minutes 0 / 5 problems solved

Previously, we have learned the basics of promises; now, it's time to interact with them. In this topic, we will consider three promise methods that programmers use to work with a settled promise: `.then`, `.catch`, and `.finally`.

§1. .then

`.then` method is used to handle positive or negative promise results.

Let's look at an example: say, we're working on a program that helps busy students keep track of their exams dates. We create a promise to let the user know if they missed their exam based on the current date. If the event did not take place yet, we resolve the "`You should prepare for the exam`" value; otherwise, we reject with the error text "`Oops! You have missed your exam!`".

```
const examDate = new Date(2020, 7, 5);
const promise = new Promise(function(resolve, reject) {
  const currentDate = new Date();
  if (currentDate < examDate) {
    resolve("You should prepare for the exam");
  } else {
    reject("Oops! You have missed your exam!");
  }
});
```

Now, let's make the output show the response message via console in case of success, or an alert in case of failure. To do that, we use `.then` function with two parameters: a function to handle a positive result, and another to handle an exception. If the promise was resolved, we call the `successStatus` function with the promised result and display the response in the console. Otherwise, we execute the `failStatus` function with the promise outcome and print the error message in an alert.

```
promise.then(
  function successStatus(response) {
    console.log(response);
    return response;
    },
  function failStatus(error) {
    console.log(error);
    return error;
  }

);
```

So, `.then` method is used to work the promised result and launch certain actions based on it. Both arguments of .`then` are optional.

§2. .catch

Let's say we need to handle only errors. In this case, you can use either `.then` without the first argument `.then(null, function failStatus(error) { ... })`, or `.catch`:

```
promise.catch(function failStatus(error) {
  console.log(error);
  return error;
});
```

In the `.catch` method, we told the program what we want to do in case of failure. In our situation, we show an alert with an error message.

Current topic:

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

Topic depends on:

✕ Introduction to Promises

Topic is required for:

Async/await

Anonymous function

Table of contents:

↑ "then", "catch" and "finally" methods

§1. .then

§2. .catch

§3. .finally

§4. Promise chaining

§5. Conclusion

Feedback & Comments

It is possible to use `.then` and `.catch` together for the same promise:

```
promise
  .then(function successStatus(response) {
    console.log(response);
    return response;
  })
  .catch(function failStatus(error) {
    console.log(error);
    return error;
  })
```

§3. .finally

The method `.finally` is used when we want to invoke a function after the promise was settled, regardless of the promise state. Let's look at the syntax:

```
promise
  .then(function successStatus(response) {
        console.log(response);
        return response;
  })
  .catch(function failStatus(error) {
    console.log(error);
    return error;
  })

.finally(function stopLoader() {

  console.log("The loader has stopped");

});
```

The text "`The loader has stopped`" will be shown after the promise has been settled. The user will see it no matter if the promise was resolved or rejected. It's useful when you have some required actions that don't depend on the result of the promise, for instance, for stopping loaders or showing the default greeting text.

§4. Promise chaining

Suppose you have some scripts depending on other scripts, and you need to load them sequentially. For example, first you have to load the user's role. Second, you should load the user info, and third, a personal banner according to their preferences. In such cases, promise chaining is a great feature that allows to call the request only after the previous has been resolved:

```
loadData("https://mywebsite.com/loadRoles")
  .then(function() {
    return loadData("https://mywebsite.com/loadUserInfo");
  })
  .then(function(user) {
    return loadData(`https://mywebsite.com/loadBanner_${user.id}`);
  })
  .catch(function(error) {
    console.log("Oops! An error occured!")

  });
```

The `.catch` method will be executed for errors in any step.

> Keep it in mind that to make the code asynchronous, you should return the promise at each step.

§5. Conclusion

We have covered three methods that help us work with the promise results. Let's recap:

- `.then` is used to take some actions depending on the output;
- `.catch` is used for handling errors;
- `.finally` is used to launch a function as soon as the promise was settled, ignoring the promise state.

All of them are useful in real programming tasks. Speaking of real tasks: shall we?

Report a typo

39 users liked this theory. **2** didn't like it. **What about you?**

Start practicing

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