**Promise & async/await**

**Async/await** is a mechanism in programming that facilitates working with **asynchronous operations**.

What is an asynchronous operation?

Imagine you're ordering coffee at a café:

**Synchronously**: You stand at the counter and wait until the coffee is ready before doing anything else. If making the coffee takes 5 minutes, you stand there for the entire 5 minutes, blocking the queue.

**Asynchronously**: You order coffee, get an order number, and sit down. You can read a book, talk to someone, etc., during this time. When the coffee is ready, the barista calls your number.

How do async and await work?

**async**: This keyword indicates that a function **can operate asynchronously**. When a function is marked as async, it can use await inside.

**await**: This keyword tells the program to **"wait" for an asynchronous operation to complete**, but it doesn't block the entire program. The program can do other things in the meantime. 'await' means 'expect' in English.

Imagine you have a function that simulates making coffee:

1. // Definition of an asynchronous function that simulates making coffee
2. async function makeCoffee() {
3. console.log("Starting to make coffee..."); // Logging the start of the process
5. // Using 'await' with a new 'promise' that the coffee will come (or not\*)
6. await new Promise(resolve => {
7. // setTimeout is a function that executes the passed callback after a specified time
8. setTimeout(() => {
9. // After 2000 milliseconds (2 seconds), the Promise is resolved
10. resolve();
11. }, 2000);
12. });
14. console.log("Coffee ready!"); // Logging that the coffee is ready
15. return "Your coffee"; // Returning the function result
16. }
18. // Definition of an asynchronous function that simulates a morning
19. async function morning() {
20. console.log("Good morning!"); // Logging the start of the morning
22. // Using 'await' to wait for the makeCoffee function to complete
23. let coffee = await makeCoffee();
25. console.log(coffee); // Logging the result, which is "Your coffee"
26. console.log("Time for breakfast."); // Logging the transition to the next activity
27. }
29. // Calling the morning function
30. morning();
32. // The console.log below will be called immediately after calling morning(), without waiting for the asynchronous operations inside morning() to complete
33. console.log("In the meantime, I can check my messages.");

**Explanation**:

* **makeCoffee** is an **async function** that uses **await** to wait for 2 seconds (simulating preparation time).
* **morning** is also **async** and uses **await** to wait for makeCoffee.
* Even though **await pauses execution inside the morning function**, it **doesn't block the entire program**. That's why "In the meantime, I can check my messages." will appear before "Coffee ready!".

**Summary**:

* **Async** means that a function can contain operations that **don't have to finish immediately**.
* **Await** allows us to "wait" for the result of an operation inside an async function **without blocking the rest of the program**. It's like waiting for coffee while you can do other things before you hear that your coffee is ready.

Alright, but what does "**Promise**" mean and why is it in this code? :-)

Imagine that a Promise is like **an order number you get at a café**:

**Order (Promise)**: When you order coffee, you get a number. This number is your "promise" that the coffee will be prepared.

**States that a "promise" can take**:

1. **Pending**: Your order has been accepted, but the coffee isn't ready yet. It's like you're waiting for your number to be called.
2. **Fulfilled**: Your coffee is ready! The barista called your number, and you got your coffee. In a Promise, this is when the asynchronous operation has successfully completed.
3. **Rejected**: Something went wrong, e.g., they ran out of coffee. In the café, they might tell you they can't make coffee. In code, this means the operation failed.
4. let coffeePromise = new Promise((resolve, reject) => {
5. // Simulating the coffee brewing process
6. let brewingTime = 2000; // 2 seconds
7. setTimeout(() => {
8. if (brewingTime < 3000) { // If the time is less than 3 seconds, the coffee is ready
9. resolve("Coffee ready!"); // Success
10. } else {
11. reject("Sorry, the coffee got burned."); // Error
12. }
13. }, brewingTime);
14. });

How to use the above code?

1. coffeePromise
2. .then(coffee => {
3. console.log(coffee); // Called when the coffee is ready
4. })
5. .catch(error => {
6. console.log(error); // Called when there's a problem with the coffee
7. });

**Explanation**:

* **new Promise((resolve, reject) => {...})**: Creates a new promise. **resolve** is a function you call when the operation succeeds, and **reject** when it fails.
* **.then()**: It's like approaching the counter when you hear your number. It executes when the promise is **fulfilled**.
* **.catch()**: It's like hearing that something went wrong with your order. It handles the situation when the promise is **rejected**.

When you use **async/await**, await essentially waits for the Promise to resolve:

1. async function orderCoffee() {
2. try {
3. let result = await coffeePromise;
4. console.log(result);
5. } catch (error) {
6. console.log(error);
7. }
8. }

**await**: Waits for coffeePromise to be resolved (either through resolve or reject). If everything goes well, you'll get your coffee (result); if not, you'll catch an error.

In summary, **Promise** is a way to handle operations that may complete in the future, whether successfully or with an error, allowing for more readable and manageable asynchronous code.

And now a small glossary of terms:

**Promise**

* **Definition**: In programming, a Promise is an object that represents an operation that hasn't finished yet but will finish in the future.
* **Origin**: The term "promise" is used because the code "promises" to deliver a result (success or failure) in the future.

**Resolve**

* **Definition**: In the context of a Promise, resolve means that the operation has ended successfully.
* **Origin**: "Resolve" comes from solving a problem. When a promise is "resolved", it means the operation has completed successfully.

**Reject**

* **Definition**: In a Promise, reject is used when the operation cannot be completed due to an error or other obstacle.
* **Origin**: "Reject" means refusing to accept the result of an operation because something went wrong.

**Pending**

* **Definition**: This is a state where something is in progress but hasn't been completed yet.
* **Origin**: "Pending" literally means "waiting" or "in progress", which reflects the state of an operation that hasn't finished yet.

**Fulfilled**

* **Definition**: When a promise is fulfilled, it means that what was promised has been delivered or executed.
* **Origin**: "Fulfilled" comes from the full execution (fulfillment) of a promise, meaning the operation has ended successfully.

**.then()**

* **Definition**: In the context of a Promise, .then() is a method that executes when the promise is fulfilled.
* **Origin**: The use of "then" suggests that something will happen after the promise is resolved.

**.catch()**

* **Definition**: In programming, "catch" refers to catching (handling) an exception or error.
* **Origin**: "Catch" is used because this part of the code "catches" any errors that might cause the promise to be rejected.

**Async/Await**

* **Async**:
  + **Definition**: "Async" refers to operations that can work independently of the main program flow.
  + **Origin**: "Asynchronous" means not simultaneous, which in programming means that operations can be performed in the background.
* **Await**:
  + **Definition**: Waiting for the result of an asynchronous operation.
  + **Origin**: "Await" comes from "awaiting", which means waiting for something. In code, await means waiting for a Promise result without blocking the rest of the program.

In conclusion, the terms used in the context of Promises and asynchronous programming in JavaScript are chosen to reflect their function in a more natural, understandable way, referring to everyday concepts of promises, solutions, waiting, and reactions to various outcomes.

Course content

AI Assistant

**Beta**