Promises in JavaScript [1] [2] [3]

A **Promise** in JavaScript is an object that represents the eventual completion (or failure) of an asynchronous operation and its resulting value. It helps manage asynchronous code more effectively, avoiding the infamous "callback hell" and making code cleaner and easier to read.

Key Features of Promises

Promises have three states:

- **Pending**: The initial state, where the operation has not yet completed.
- **Fulfilled**: The operation completed successfully, and the promise now holds a resolved value.
- **Rejected**: The operation failed, and the promise holds a reason for the failure.

Once a promise is either fulfilled or rejected, it becomes **settled**, and its state cannot change further.

Creating a Promise

A promise is created using the `Promise` constructor, which takes a function with two parameters: `resolve` (to mark the promise as fulfilled) and `reject` (to mark it as rejected).

```
const promise = new Promise((resolve, reject) => {
  const success = true; // Simulate success or failure
  if (success) {
    resolve("Operation successful!");
  } else {
    reject("Operation failed!");
  }
});
```

Consuming a Promise

Promises are consumed using `.then()`, `.catch()`, and `.finally()` methods.

- `.then()`: Handles the resolved value of a promise.
- `.catch()`: Handles errors or rejections.
- `.finally()`: Executes code after the promise is settled, regardless of its outcome.

Example:

```
promise
  .then((result) => {
    console.log(result); // Logs "Operation successful!"
  })
  .catch((error) => {
    console.error(error); // Logs "Operation failed!" if rejected
  })
  .finally(() => {
    console.log("Promise settled.");
  });
```

Promise Chaining

Promises can be chained to handle sequential asynchronous operations. Each `.then()` returns a new promise, allowing further chaining.

```
fetchData()
  .then((data) => {
    console.log(data);
    return processData(data);
  })
  .then((processedData) => {
    console.log(processedData);
  })
  .catch((error) => {
    console.error("Error:", error);
  });
```

Utility Methods

- `Promise.all()`: Resolves when all promises in an array are fulfilled or rejects if any promise is rejected.
- `Promise.race()`: Resolves or rejects as soon as one promise in the array settles.
- `Promise.allSettled()`: Resolves when all promises settle, regardless of their outcome.
- `Promise.any()`: Resolves as soon as any promise is fulfilled or rejects if all are rejected.

Example:

```
const promise1 = Promise.resolve("First");
const promise2 = Promise.reject("Second failed");
const promise3 = Promise.resolve("Third");

Promise.all([promise1, promise3])
  .then((results) => console.log(results)) // Logs ["First", "Third"]
  .catch((error) => console.error(error));

Promise.race([promise1, promise2])
  .then((result) => console.log(result)) // Logs "First"
  .catch((error) => console.error(error));
```

Conclusion

Promises simplify asynchronous programming by providing a structured way to handle success and failure. They are a cornerstone of modern JavaScript, especially when combined with `async/await` for even cleaner syntax.

Learn more: **1 -** viblo.asia **2 -** w3schools.com **3 -** developer.mozilla.or