MASTERING JAVASCRIPT PROMISES

Yasith Wimukthi

Understanding Promises

- Promises are objects that represent the eventual completion (or failure) of an asynchronous operation.
- They are cleaner and more readable than using callbacks, which can lead to "callback hell."
- A Promise is in one of these states:
  - pending: initial state, neither fulfilled nor rejected.
  - fulfilled: meaning that the operation was completed successfully.
  - rejected: meaning that the operation failed.

Creating a Promise

- A promise can be created using the Promise constructor, which takes a function with resolve and reject parameters.
- Inside this function, you write the asynchronous code. When the operation completes successfully, you call resolve with the result; if there's an error, you call reject with the error.

```
const fetchData = new Promise((resolve, reject) => {
  setTimeout(() => {
    const data = { id: 1, name: 'John Doe' };
    resolve(data);
  }, 2000);
});

fetchData.then(data => {
  console.log('Data:', data);
}).catch(error => {
  console.error('Error:', error);
});
```

Chained Promises

- You can chain promises using then() to perform sequential operations.
- Each then() returns a new promise, allowing you to chain multiple asynchronous operations.

```
fetchData.then(data => {
  console.log('Data:', data);
  return anotherPromise();
}).then(result => {
  console.log('Another result:', result);
}).catch(error => {
  console.error('Error:', error);
});
```

Promise Concurrency

- JavaScript is inherently single-threaded, but promises can manage concurrency.
- Promise.all() waits for all promises to resolve before continuing.
- Promise.race() resolves or rejects with the first promise to resolve or reject.

```
const promise1 = fetchData();
const promise2 = fetchData();

Promise.all([promise1, promise2]).then(([data1, data2]) => {
  console.log('Data 1:', data1);
  console.log('Data 2:', data2);
}).catch(error => {
  console.error('Error:', error);
});
```

Static Promise Methods

- Promise.resolve(): Returns a promise that resolves with a given value.
- Promise.reject(): Returns a promise that rejects with a given reason.
- Promise.allSettled():Returns a promise that resolves after all of the given promises have either resolved or rejected.
- Promise.any():Returns a promise that resolves as soon as any of the given promises resolve, or rejects if all of the promises reject.
- Promise.all(): Waits for all promises in an iterable to resolve or reject. Returns a promise that resolves when all promises have resolved or rejects when one rejects.

Promise Instance Properties

These properties are defined on Promise.prototype and shared by all Promise instances.

- Promise.prototype.constructor : The constructor function that created the instance object. For Promise instances, the initial value is the Promise constructor.
- Promise.prototype[@@toStringTag] : The initial value of the @@toStringTag property is the string "Promise". This property is used in Object.prototype.toString().

Promise Instance Methods

- then(): Handles successful completion of a promise. Takes onFulfilled and onRejected arguments. If fulfilled, onFulfilled is called with the result; if rejected, onRejected is called with the error.
- catch():Handles promise rejections. Takes a function called with the error causing the rejection.
- finally(): Executes a callback regardless of fulfillment or rejection.

Conclusion

- Promises are powerful tools for asynchronous operations.
- Chaining promises enables sequential processing.
- By mastering promises and their methods, developers can create more efficient and reliable web applications.

Yasith Wimukthi