JAVASCRIPT

Promise

Sreekanth M. E.

Freelance Trainer & Consultant

http://www.SreekanthME.com

Promises are an alternative to callbacks for delivering the results of an asynchronous computation.

A promise will return a value in future.

A Promise is an object representing the eventual completion or failure of an asynchronous operation.

STATES OF A PROMISE

Pending: Initial state, neither fulfilled nor rejected.

Fulfilled (Resolved): Operation completed successfully.

Rejected: Operation failed.

A promise is said to be **settled** if it is either **fulfilled** or **rejected**, but not pending.

A Promise object is created using the new keyword and its constructor.

The promise constructor takes as its argument a function, called the executor function.

Executor function takes a resolve function and a reject function as parameters.

Resolve function must be called, inside executor, when the async task of the promise succeeds. The success value must be passed to it as a parameter.

Reject function is called, inside executor, when the async task of the promise fails. The reason for failure (typically an error object) must be passed to it as a parameter.

The executor function must perform the async operation and call resolve() in case of success and reject() in case of failure.

A promise is typically returned by a function.

The wrapping function now behaves like an async function.

```
function oneThing() {
    return (new Promise((resolve, reject) => {
        // do something async which eventually calls either:
        // resolve(someValue); // Fulfilled
        // or
        // reject("failure reason"); // Rejected
    }));
}
```

Promise is a thenable or an object that supplies a standard-compliant .then() method.

```
testPromise.then(onFulfilled[, onRejected]);
```

```
doOneThing()
    .then(result => {
      console.log(`Got final result: ${result}`);
    })
    .catch(error=>{
        console.log(`Got error: ${error}`);
    });
```

THEN() METHOD

Parameters:

onFulfilled: A Function called if the Promise is fulfilled. It has one argument, the resolve value.

onRejected: A Function called if the Promise is rejected. It has one argument, the reject error. It is an optional parameter.

Return value:

A Promise in the pending status.

CATCH() METHOD

testPromise.catch(onRejected);

Parameters:

onRejected: A Function called when the Promise is rejected. This function has one argument, the reject error.

This method acts as catch block for the promise block.

BENEFITS OF PROMISE

Promise has three benefits over traditional callbacks:

- Chaining
- Readable / Maintainable Code
- Better Error Handling

CHAINING

The then() method returns a Promise which allows for promise chaining.

This operation is called composition.

```
doOneThing()
    .then(result => doSecondThing(result))
    .then(newResult => doThirdThing(newResult))
    .then(finalResult => {
      console.log(`Got final result: ${finalResult}`);
    })
    .catch(error => {
        console.log(`Got error: ${error}`);
    });
```

A promise in a chain, passes its resolve value to the next promise in the chain and its reject error to the catch handler at the end.

A promise chain stops if there is an exception in any promise of the chain and looks for catch handler instead.

A trailing .then() block acts as a finally.

Traditional callbacks could not be chained.
They had to be written in a nested format.

This would lead to the classic callback pyramid of doom (or callback hell).

CALLBACK APPROACH

```
doOneThing(function(result) {
  doSecondThing(result, function(newResult) {
    doThirdThing(newResult, function(finalResult) {
      console.log(`Got final result: ${finalResult}`);
    }, failureCallback);
  }, failureCallback);
}, failureCallback);
```

Callback hell means really confusing and difficult-to-read code.

It leads to code that is not maintainable.

BETTER ERROR HANDLING

In case of callbacks, try/catch block is needed inside every one of the nested callbacks.

Moreover, there is no mechanism to communicate an error in a callback to its outer callback.

Promises solve a fundamental flaw with the callback hell by catching all errors, even thrown exceptions, in one place.

END OF CHAPTER

APPENDIX