WSA | Forward looking IT finishing school

Promises

Asynchronous Javascript

Understanding Promises

“Imagine you are a kid. Your mom promises you that she'll get you a new phone next week.”

Understanding Promises...

- You don't know if you will get that phone until next week.
- Your mom can either really buy you a brand new phone, or
- Reject the phone if she is not happy :(.

Promise States

That is a promise. A promise has 3 states. They are:

1. Promise is **pending**: You don't know if you will get that phone until next week.
2. Promise is **resolved**: Your mom really buys you a brand new phone.
3. Promise is **rejected**: You don't get a new phone because your mom is not happy.

The constructor syntax for a promise object

```
let promise = new Promise(function(resolve, reject) {
// executor (the producing code, "request to mom")
});
```

- The function passed to new Promise is called the **executor**.
- When the promise is created, this executor function runs automatically. It contains the producing code, that should eventually produce a result.

The resulting promise object has internal properties

- state — initially “pending”, then changes to either “fulfilled” or “rejected”
- result — an arbitrary value of your choosing, initially undefined.

Promise outcome

- When the executor finishes the job, it should call one of the functions that it gets as arguments:
- resolve(value) — to indicate that the job finished successfully:
- sets state to "fulfilled",
- sets result to value.

Promise outcome

- reject(error) — to indicate that an error occurred:
- sets state to "rejected"
- sets result to error.

Sample Code (Definition)

```
let promise = new Promise(function(resolve, reject) {

// the function is executed automatically when the promise is constructed

// after 1 second signal that the job is done with the result "done!"
  setTimeout(() => resolve("done!"), 2000);
});
```

On Creation

- We can see two things by running the code above:
- The executor is called automatically and immediately (by the new Promise).
- The executor receives two arguments: resolve and reject — these functions are pre-defined by the JavaScript engine. So we don’t need to create them. Instead, we should write the executor to call them when ready.

On Execution

- After one second of “processing” the executor calls resolve("done") to produce the result:

- That was an example of a successful job completion, a “fulfilled promise”.

Rejecting the promise with an error

```
let promise = new Promise(function(resolve, reject) {
  // after 1 second signal that the job is finished with an error
  setTimeout(() => reject(new Error("Whoops!")), 1000);
});
```


new Promise(executor)
state: "pending"
result: undefined
reject(error)
state: "rejected"
result: error

Consumers: “then” and “catch”

```
promise.then(
  function(result) { /* handle a successful result
*/ },
  function(error) { /* handle an error */ }
);
```

Web Stack Academy (P) Ltd

#83, Farah Towers,

1st floor,MG Road,

Bangalore – 560001

M: +91-80-4128 9576

T: +91-98862 69112

E: info@www.webstackacademy.com