Promise in JavaScript



Just like in real life, a **promise**is an object that may produce a single value some time in the future.

A promise can be:

* **resolved**- promise succeeded
* **rejected** - promise failed
* **pending** - Hasn't fulfilled or rejected yet

Promise can be created using  **new Promise().**Lets understand it with an example.

1. function getTrip(){
2. return new Promise(function(resolve){
3. setTimeout(function() {
4. resolve("Lets go to Trip");
5. }, 2000);
6. });
7. };

In the above code, we are not returning the actual data. We are returning a promise of the actual data.

Line2 says we are returning a promise. We can see that while creating a promise, we have to pass a function as a parameter. The code that will be executed in future is placed inside this function.

What is resolve in line4? Resolve is the data what we are promising. The above code says we are promising a string data in future, if everything goes well.

Even in real life, we promise about things that will happen in future. So when we create a promise in JS, we are in effect creating a data that will be available in the future, if everything goes well.

So how can we access this data in future? Well, a promise can be handled using **then**method. The general syntax is:

1. promiseObject.then(
2. function(futureData){
3. ....
4. })

For example:

1. getTrip().then(
2. function(futureData){
3. console.log(futureData);
4. }
5. );

The above example will invoke the function inside **then** after 2000 milliseconds as getTrip() resolves a promise only after 2000 ms.

Resolving a promise is valid only if nothing goes wrong. What if something goes wrong and we are not able to keep the promise? Then we need to reject the promise instead of resolving the promise. So instead of returning the promised data, we can return an error. In any new Promise we can define resolve and reject both.

**Syntax:**

1. new Promise (resolve, reject) { }

Let's understand it with example.

1. function getTrip(location) {
2. return new Promise(function(resolve, reject) {
3. if (location == "ooty") {
4. resolve("Trip to " + location);
5. } else {
6. reject(Error("Some error occured"));
7. }
8. });
9. }

Now, let's call the above function. Here, promise will be **resolved**.

1. getTrip("ooty").then(
2. function(data) {
3. console.log(data);
4. },
5. function(error) {
6. console.log(error);
7. }
8. );
9. *// Trip to ooty*

In this call, condition will fail and promise will be **rejected**.

1. getTrip("coorg").then(
2. function(data) {
3. console.log(data);
4. },
5. function(error) {
6. console.log(error.message);
7. }
8. );
9. *// Error: Some error occurred*

Promise chaining



Sometimes in real life, we make multiple promises which are interdependent on each other. Only if we fulfill the first promise, the second promise even makes sense. If we fail to keep the first promise, then the second promise cannot be fulfilled.

The same is true in JS also. For example, I have two promises: one to book flight and the other to book hotel. Each promise is resolving the amount of money spent for the booking

1. function bookFlight(){
2. return new Promise(function(resolve){
3. setTimeout(resolve(5600),2000);
4. })
5. }
6. function bookHotel(flightPrice){
7. return new Promise(function(resolve){
8. setTimeout(resolve(7000+flightPrice),1000);
9. })
10. }

We want the hotelPromise to be resolved only after the flightPromise is resolved and calculate the total price. How can we do this?

To ensure that the second promise is resolved only after the first promise is resolved, we can chain these promises together as shown:

1. function bookFlight() {
2. return new Promise(function (resolve) {
3. setTimeout(resolve(5600), 2000);
4. })
5. }
6. function bookHotel(flightPrice) {
7. return new Promise(function (resolve) {
8. setTimeout(resolve(7000 + flightPrice), 1000);
9. })
10. }
11. bookFlight()
12. .then(function (flightData) { return bookHotel(flightData) })
13. .then(function (cumulativeData) { console.log(" Total is " + cumulativeData) })

The promise returned by the first **then** is chained and resolved in the next **then**.

What will happen if any of promises fail in the chained promises?

Promises after the failed promise will not be resolved. It will throw an error and promises in chain will not be executed after it.

1. function bookFlight(airline) {
2. return new Promise(function (resolve, reject) {
3. if (airline == "AirIndia") {
4. setTimeout(resolve(5600), 2000);
5. } else {
6. reject(Error("Flight can not be booked"))
7. }
8. })
9. }
10. function bookHotel(flightPrice) {
11. return new Promise(function (resolve) {
12. setTimeout(resolve(7000 + flightPrice), 1000);
13. })
14. }
15. bookFlight("indigo")
16. .then(function (flightData) { return bookHotel(flightData) })
17. .then(function (cumulativeData) { console.log(" Total is " + cumulativeData) })
18. .catch(e => console.log(e.message))

The promise returned by the first **then** will throw error.