Javascript ES6

Lesson 17
Asynchronous Programming in ES6

Capgemini

Lesson Objectives

At the end of this module you will be able to:

- Create Promises and explain how it works
- Understand the different states of a Promise
- Perform operation on various methods of the Promise object.

ES6 – Promise Constructor

ES6 introduces a new native pattern for writing the asynchronous code called as Promise pattern.

The Promise constructor is used to create new Promise instances. A Promise (or a Promise object) represents an asynchronous operation

Promise constructor takes a callback(executor), which executes the asynchronous operation.

The executor should take two parameters, i.e. the resolve and reject callbacks

- The resolve callback should be executed if the asynchronous operation was successful, and the reject callback should be executed if the operation was unsuccessful.
- If the asynchronous operation was successful and has a result, then result can be passed the resolve callback. If the asynchronous operation was unsuccessful, then failure reason can be passed to the reject callback.

Promise states

Fulfilled

- If the resolve callback is invoked with a non-Promise object as argument or no argument

Rejected

- If the reject callback is invoked or an exception occurs in the executor scope

Pending

- If the resolve or reject callback is yet to be invoked

Settled

- A Promise is said to be settled if it's either fulfilled or rejected, but not pending

Demo

promise-pattern

Promise.resolve(value) & Promise.reject(value)

resolve() method of the Promise object takes a value and returns a promise object that resolves the passed value.

resolve() method is basically used to convert a value to an promise object.

resolve() method is useful when you find yourself with a value that may or may not be a Promise, but you want to use it as a Promise

Using the resolve() method jQuery Promises can be converted into ES6 Promises.

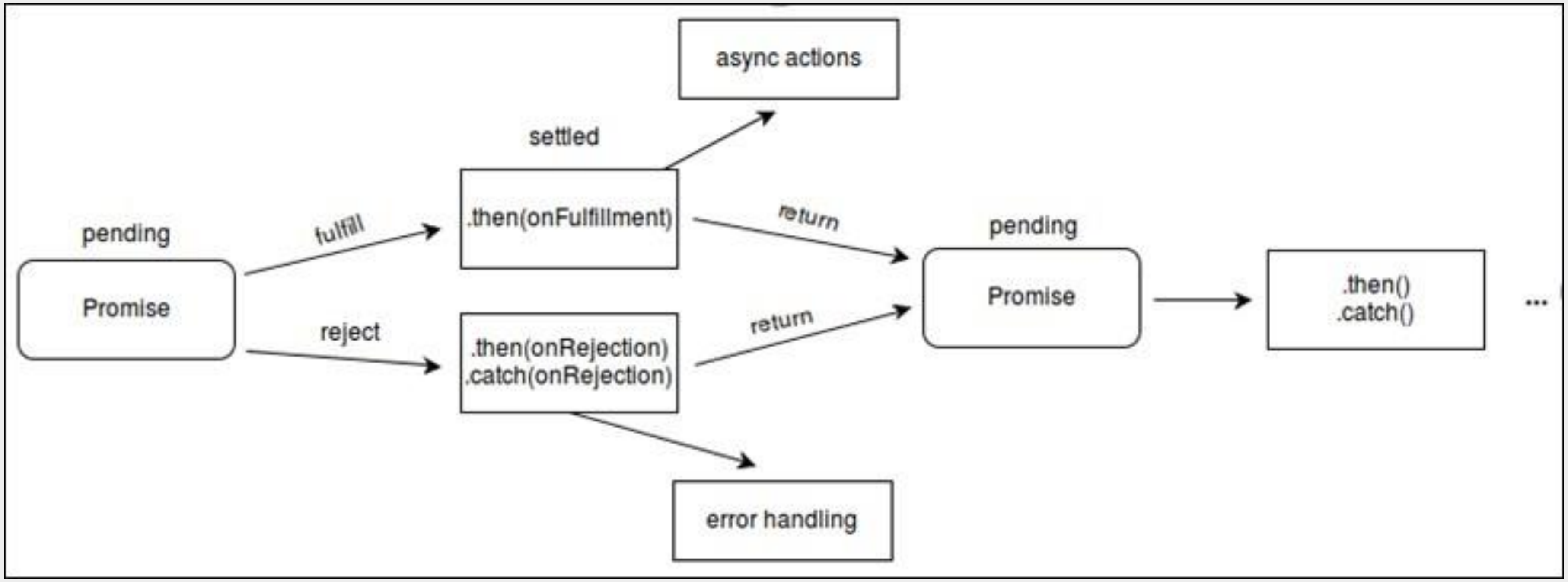
reject() method of the Promise object takes a value and returns a rejected promise object with the passed value as the reason.

Unlike Promise.resolve() method, the reject() method is used for debugging purposes and not for converting values into Promises.

Demo

promise-functions

Execution of multiple chained promises



Demo

promise-resolving-another

Promise.all(iterable)

all() method of the Promise object takes an iterable object as an argument and returns a Promise that fulfills when all of the Promises in the iterable object have been fulfilled.

It is useful for executing some task after some asynchronous operations have finished.

If the iterable object contains a value that is not a promise object, then it's converted to the Promise object using the Promise.resolve() method.

In case aby of the passed Promises get rejected, then the Promise.all() method immediately returns a new rejected Promise.

Demo

promise-all

Promise.race(iterable)

race() method of the Promise object takes an iterable object as the argument and returns a Promise that fulfills or rejects as soon as one of the Promises in the iterable object is fulfilled or rejected, with the fulfillment value or reason from that Promise.

race() method is used to race between Promises and see which one finishes first.

Demo

promise-race

Summary

Promise pattern in ES6, makes it easier to read and write the asynchronous code.

A Promise (or a Promise object) represents an asynchronous operation.

Once a Promise is fulfilled or rejected, it cannot be transitioned back

Promise.all() is useful for executing some task after some asynchronous operations have finished

Promise.race() method is used to race between Promises and see which one finishes first.