Venkat.R Follow

Oct 20, 2016 · 6 min read · Listen

JS Promise (Part 1 - Basics)

One of the good Features in ES6 is **Promises Object** and their useful **methods** and they are called software abstraction helps to works smoothly with asynchronous operations. **Promise** API followed **Promises/A+** Specification prior to this, there was **Promise/A.**

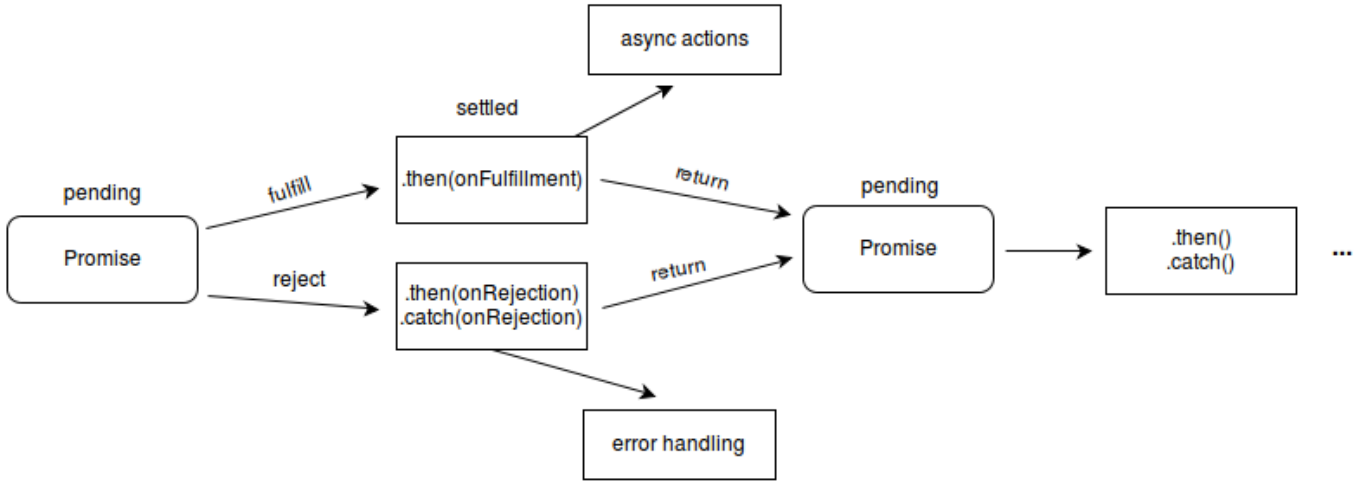
What is Promises/A and Promises/A+ ?

1. Both are specification for open standard but Javascript currently uses **Promise/A+**.
2. The main three reason for using **Promise/A+** is **Three** different states, **Value** for Fulfilment and for Rejection and **thenable** object. for more detail see Differences from Promises/A
3. These **Promise/A+** organisation occasionally revise and address the corner

Dif

1. Rejection / Fulfilment method returns **thenable** object.
2. Rejection triggers catch method, if errors thrown.

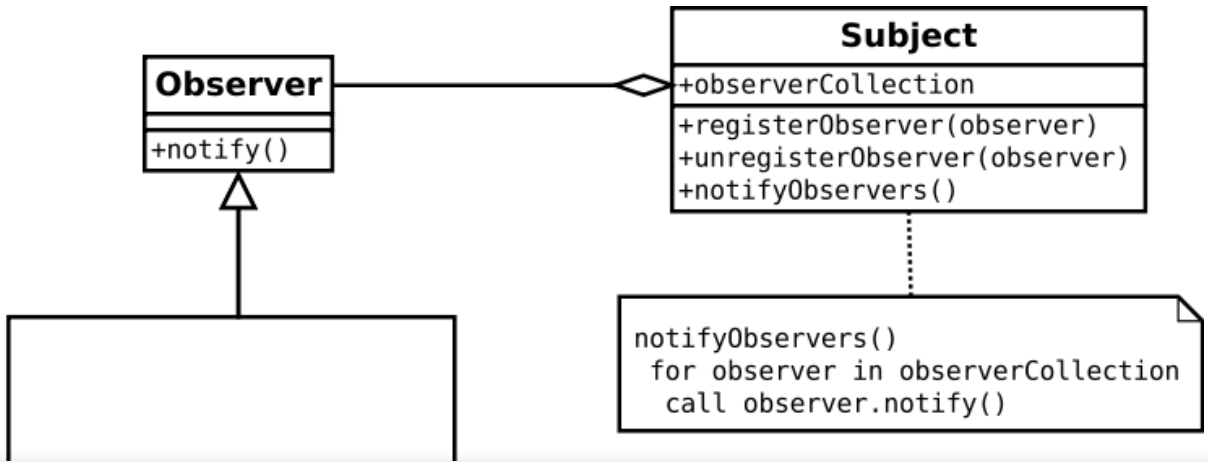
In short, running continuation-passing style. Let me shoot few bullets with simple diagram on **Promises** below.



Promise API Flow (Image Credits to MDN — **Mozilla Developer Network)**

1. It’s a placeholder for the eventual results of a deferred (and possibly

3. t an actual value.
4. Allows you to add handler to asynchronous of success or failure conditional actions.
5. Return a promise instead of final value.
6. It implements observer pattern (maintains a dependents and notifies them automatically of any state changes but can cause memory leaks, known as the lapsed listener problem) and its chain-able.
7. It comes with integrated error handling and handles automatically fulfilled with value or reject with reason.



WI

1. Promises mainly to solve **Callback Hell** (heavy nested callback code) and Pyramid of doom Problem**.**
2. Promises helps to group your asynchronous in a efficient way.
3. Javascript is not a new candidate for adapting Promises/A+ Specification. its been already included by other famous programming languages like Java (sample), Scala (sample), Python (sample) and Clojure (sample).

Promise States

There are three states which are below.

1. **pending**: Initial Case where promise instantiated.
2. **fulfilled**: Success Case which means promise resolved.
3. **rejected**: Failure Case which means promise rejected.

Promise Methods

2.

3. Promise.reject(value);

4. Promise.resolve(value);

5. Promise.catch(onRejection);

6. Promise.then(onFulFillment, onRejection);

Promise Iterable Methods

Below are the two Methods which accepts array of promises objects.

```
'use strict';

// All method
// Trigger `catch` Method, if any one promise rejected.
// Trigger `then` Method, once every promise fulfilled.

Promise.all([promise1, promise2]);

// Race method
// Trigger `catch` Method, if any one promise rejected.
```

Wh

1. Executor Functions are Parameter for Promise Constructor which holds Resolve and Reject Callbacks.
2. It is executed immediately by the Promise implementation which provides the resolve and reject functions.
3. It's Triggered before the Promise constructor even returns the created object.
4. The Resolve and Reject functions are bound to the promise to fulfill or reject.
5. It's expected to initiate some asynchronous work and then, once that completes, call either the resolve or reject.

What is the value of Promise.length?

It will be always one (1) due to number of constructor arguments.

. . .

Are You Tired with above Theories?

Ho

Below is the Code Snippet for Simple Promise Instantiation.

```
'use strict';

// ES5
var myPromise = new Promise(function (resolve, reject) {
    // Your Code which you are unsure about execution time duration.

    // Call resolve() method at the end of successful execution.

    // Call reject() method for your failure case.

    // Catch method will be called automatically, if any error occurs.
});

// ES6, Example with Arrow Functions

var myPromise = new Promise((resolve, reject) => { ... })
```

How Promise methods can be overridden ?

Ho

There are two possible ways to handle errors in **Promise**. Defining **then** method with second parameter which is onRejection callback or defining **catch** method.

```
'use strict';

// First Approach

yourPromise.catch(function (error) {
 // Your Error Callback
});

// Second Approach

yourPromise.then(undefined, function (error) {
 // Your Error Callback
});
```

How to detect whether Promise rejected ?

Promise triggers **.catch** or onRejection callback of **.then** method depends upon

```
'use strict';

Promise.catch(onRejected);
Promise.then(onFulfilled, onRejected);
```

How to combine multiple Promises in to one ?

Use `**Promise.all**` Method which combines multiple promise in to one.

```
'use strict';

// Usage

Promise.all([Promise1, Promise2, …]).then(onFulfilled, onRejected);

// Example

var promiseCall = function (waitSecond, returnData) {
    return function (resolve, reject) {
        setTimeout(resolve, waitSecond, returnData);
    };
};
```



```
});

// Calling Promise 1 - 4 in Promise.all()
Promise.all([p1, p2, p3, p4]).then(function (value) {
    console.log(value);
}, function (reason) {
    // Not Called
    console.log(reason);
});


// Expected Output: ["one", "two", "three", "four"]


// Calling Promise 1 - 5 in Promise.all()
Promise.all([p1, p2, p3, p4, p5]).then(function (value) {
    // Not Called
    console.log(value);
}, function (reason) {
    console.log(reason);
});


// Trigger Rejection (Second) Callback if any one is rejected.
// Expected Output: 5th Promise Rejected
```

What is Promise Racing ?

```
var p1 = new Promise(function(resolve, reject) {
    setTimeout(resolve, 500, "one");
});
var p2 = new Promise(function(resolve, reject) {
    setTimeout(resolve, 100, "two");
});

Promise.race([p1, p2]).then(function(value) {
  console.log(value); // "two"
  // Both resolve, but p2 is faster
});
```

What Promise.all() return for invalid parameter ?

Promise.**all** method accept array of objects which is promises in nature. Let's see How promise.all behaves for various invalid parameters.

Promise.all() Example for Invalid Parameters

Why not Promise.race() with invalid parameter ?

Promise.race method also similar to all method accept array of objects. Let's see How promise.race behaves for various invalid parameters.

Promise.race() Example for Invalid Parameters

What's the main difference between callback-based functions and promise-based functions ?

1. You don't have to write error conditions `*if (err) return callback(err)*`.
2. Callback need to be invoked the right callback immediately whereas Promise can be return as promise object and invoke later

4. t error there.
5. In Callback, We must call the passed callback to pass that error.

In short, Promise is better than callback for future asynchronous operation.

How many ways to resolve the Promise ?

There are two ways, with and without instantiation.

```
var responseValue = 'Success';

// First Approach
var directPromiseValue = Promise.resolve(responseValue);

// Second Approach
var instantiatePromise = new Promise(function (resolve) {
 resolve(value);
});
```

So... ...t into a true Promise. **Promise.resolve()** Call should be right choice.

```
var directPromiseValue = Promise.resolve('Success');
```

How to reject with hard-coded value to Promise?

Same here for error. Use **Promise.reject()** Call.

```
var directPromiseValue = Promise.reject('Custom Error Object');
```

Yes, You are done and you are in the last lines of this Article but this is not end of Promise Story and there is few more parts soon, stay tuned !

Wait is over, Have a look at next Article, Part 2 — Q.js, When.js and RSVP.js