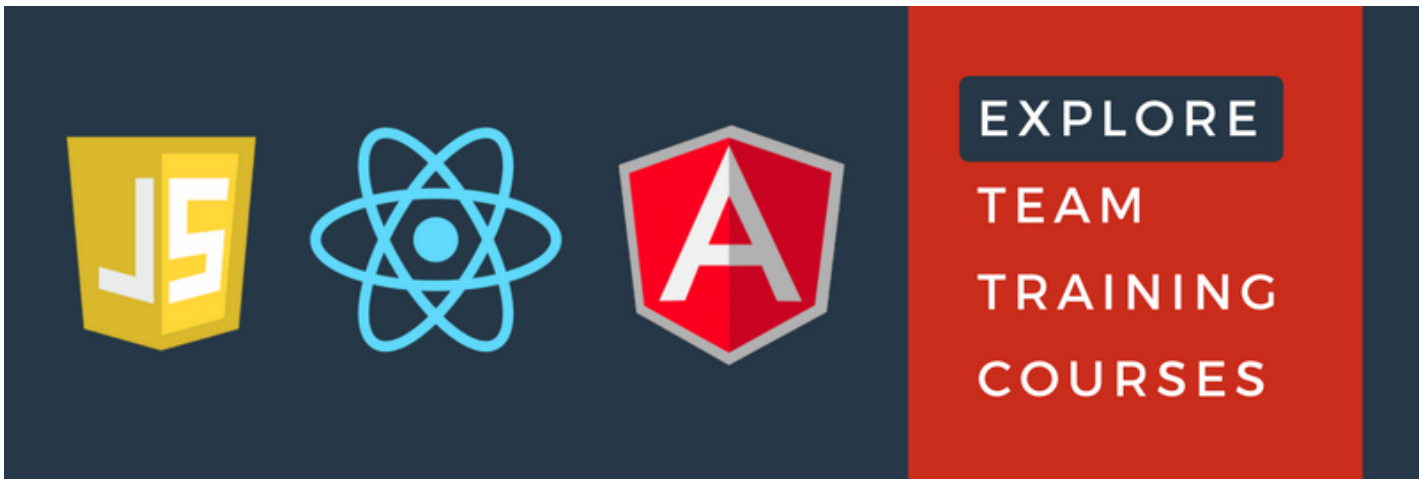
JavaScript Promises Primer



In JavaScript, a promise is simply a placeholder for a future value. It's a piece of code that *promises* to return a value at some point in time. The beauty of promises is that they provide an easy and elegant way to handle asynchronous operations (e.g., `setTimeout`, AJAX calls, etc.). In the following article, I'll show you the basic setup for JavaScript promises. I'll also show a couple of examples of their typical operation. You'll find the information useful, I promise!

Basic Setup

A promise looks like this:

JS Result EDIT ON

```
let promise = new Promise((resolve, reject) => {
  //asynchronous code goes here

  if (`check for good`) {
    resolve(`good`);
  } else {
    reject(`bad`);
  }
});

promise
  .then(resolution => console.log(`All good: ${resolution}`))
  .catch(error => console.log(`Not good: ${error}`));
```

Things to note:

1. A promise is instantiated with a call to the `Promise` constructor
2. Here, the promise is assigned to a `promise` variable
3. A callback function is passed to the promise with the parameters `resolve` and `reject`
4. Asynchronous code is executed within the promise's callback function
5. Good results are passed to the promise's `then` method by calling `resolve`
6. Bad results (and errors) are passed to the promise's `catch` method by calling `reject`
7. A callback is used in `then` to handle the good data
8. A callback is used in `catch` to handle the bad data

Examples

Here's an example of a promise in action:

HTML CSS JS Result EDIT ON

```
//Testing a promise >>>>>

ction testPromise() {
 promise = new Promise((resolve,
t) => {
output.textContent = `promise is
pending...`;
    //asynchronous code goes here
    let i;
    setTimeout(() => {
      i = Math.random().toFixed(2);
      if (i < .5) {
        resolve(i);
      } else {
        reject(i);
      }
    }, 2000);
  });

  promise
    .then(i => output.innerHTML = `<span
style="color: limegreen; font-weight:
bold;">Fulfilled:</span> ${i} is less than
0.50.`)
    .catch(i => output.innerHTML = `<span
style="color: red; font-weight:
```

Test Promise

Need Help?

Notice how the asynchronous `setTimeout` code is wrapped in the promise and it's callback function, and how it returns a value 2 seconds after the test function is called. The conditional logic within the promise/callback is used to determine whether `resolve` or `reject` gets called. The value resulting from the asynchronous code is passed to either `then` or `catch` through `resolve` or `reject`. If an error were to occur, it would also be passed to `reject`.

Additionally, worth noting is the common terminology used with promises:

1. Before a promise receives a value, it is said to be *pending*
2. Once a promise receives a value, if the value meets acceptance criteria, it's *fulfilled*
3. Once a value is received, if it doesn't meet acceptance criteria, it's *rejected*

Here's one more example using AJAX:

Notice how the AJAX code is wrapped in an instance of a promise. If everything goes well with the AJAX request, the returned value is resolved and sent to the promise's then method for handling. If something goes wrong and the response comes back with something other than is intended or if there is an error, then that data will be picked up by the promises catch method, with the error data eventually being logged to the console. Promise, then, catch; it's as easy as that.

Hopefully you found this information on JavaScript promises useful. Remember that promises offer a clean way to work with asynchronous operations. They are constructed with a call to the Promise constructor. They have then and catch methods that are used to handle results; the then method handles good values once they've been fulfilled/resolved, and the catch method handles bad data or errors.

Share this:

Chase Allen

Chase is a self-taught HTML, CSS, and JavaScript enthusiast. He enjoys creating web pages and web applications. His interests include art, music, movies, food, tech, and just general musings.

Need Help?

uary 6, 2017 by Chase Allen in

javascript promise

AJAX, Articles, es6, JavaScript, JavaScript, learning, Patterns

Comments

Comment

Name

Email

Add Comment