Lesson Monday

Intermediate JavaScript (/intermediate-javascript) / Asynchrony and APIs (/intermediate-javascript/asynchrony-and-apis) / Promises with API Calls

Text

Now that we've learned about promises, we're ready to update our weather API code to use one. We will be using the same code that we used during the weekend homework — we'll only need to make changes in one file to incorporate promises in our project. Click the following link to get the code up to this point. Take note that this links to a branch called `1_xhr_api_call`, the default branch of this repo:

Example GitHub Repo for Project (https://github.com/epicodus-lessons/section-6-js-api-call-with-webpack)

Using a Promise with an API Call

The only file we'll need to update is `index.js`. We'll start by providing the code, and then we'll walk through the changes.

src/index.js

```
import 'bootstrap';
import 'bootstrap/dist/css/bootstrap.min.css';
import './css/styles.css';

// Business Logic

function getWeather(city) {
  let promise = new Promise(function(resolve, reject) {
    let request = new XMLHttpRequest();
    const url = `http://api.openweathermap.org/data/2.5/wea
ther?q=${city}&appid=${process.env.API_KEY}`;
    request.addEventListener("loadend", function() {
      const response = JSON.parse(this.responseText);
      if (this.status === 200) {
        resolve([response, city]);
      } else {
        reject([this, response, city]);
      }
    });
    request.open("GET", url, true);
    request.send();
  });

  promise.then(function(response) {
    printElements(response);
  }, function(errorMessage) {
    printError(errorMessage);
  });
}

// UI Logic

function printElements(results) {
  document.querySelector('#showResponse').innerText = `The
humidity in ${results[1]} is ${results[0].main.humidity}%.
  The temperature in Kelvins is ${results[0].main.temp} deg
rees.`;
}

function printError(error) {
  document.querySelector('#showResponse').innerText = `Ther
```

```
e was an error accessing the weather data for ${error[2]}:
${error[0].status} ${error[0].statusText}: ${error[1].messa
ge}`;
}

function handleFormSubmission(event) {
  event.preventDefault();
  const city = document.querySelector('#location').value;
  document.querySelector('#location').value = null;
  getWeather(city);
}

window.addEventListener("load", function() {
  document.querySelector('form').addEventListener("submit",
handleFormSubmission);
});
```

Now let's break out the new code:

src/index.js

```
function getWeather(city) {
  let promise = new Promise(function(resolve, reject) {
    let request = new XMLHttpRequest();
    const url = `http://api.openweathermap.org/data/2.5/wea
ther?q=${city}&appid=${process.env.API_KEY}`;
    request.addEventListener("loadend", function() {
      const response = JSON.parse(this.responseText);
      if (this.status === 200) {
        resolve([response, city]);
      } else {
        reject([this, response, city]);
      }
    });
    request.open("GET", url, true);
    request.send();
  });

  promise.then(function(response) {
    printElements(response);
  }, function(errorMessage) {
    printError(errorMessage);
  });
}
```

In the `getWeather` function, we now take our previous code that creates and sends an `XMLHttpRequest` object and wrap it in a promise:

src/index.js

```
let promise = new Promise(function(resolve, reject) {
  let request = new XMLHttpRequest();
  const url = `http://api.openweathermap.org/data/2.5/weath
er?q=${city}&appid=${process.env.API_KEY}`;
  request.addEventListener("loadend", function() {
    const response = JSON.parse(this.responseText);
    if (this.status === 200) {
      resolve([response, city]);
    } else {
      reject([this, response, city]);
    }
  });
  request.open("GET", url, true);
  request.send();
});
```

As we learned in the last lesson, we need to determine when a promise is resolved or rejected:

```
// We use resolve() and reject() to determine whether a pro
mise should be resolved or rejected.

let promise = new Promise(function(resolve, reject) {
  resolve( // do this on resolution );
  reject( // do this on rejection );
});
```

Our updated code already has a conditional so we just have to call `resolve()` and `reject()` inside of the conditional. You may notice a few additional changes, which we'll discuss further below.

src/index.js

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

    request.addEventListener("loadend", function() {
      const response = JSON.parse(this.responseText);
      if (this.status === 200) {
        resolve([response, city]);
      } else {
        reject([this, response, city]);
      }
    });
  });

  ...
}
```

Both `resolve()` and `reject()` can take only one argument, which means our previous code that uses two and three arguments with the `printElements` and `printError` functions no longer works.

Here's our previous code:

```
if (this.status === 200) {
  printElements(response, city);
} else {
  printError(this, response, city);
}
```

There are a lot of ways that you can resolve this issue. We've opted to add each original argument to an array, and pass the array in as the single argument:

```
if (this.status === 200) {
  resolve([response, city]);
} else {
  reject([this, response, city]);
}
```

Take note that we don't have to pass in an array with 2 – 3 pieces of data. It all depends on what we decide is valuable data to have when we resolve or reject the API call. For example, we could structure our resolve and reject function calls like this:

```
if (this.status === 200) {
  resolve(response);
} else {
  reject(this);
}
```

If we use the above code, when we resolve our API call (it came back successfully), we only pass in the APi response, and not the user-inputted data. Then, when we reject our API call (something went wrong), we only pass in the `XMLHttpRequest` object (represented by `this`), so that we can print the `status` and `statusText`.

For the sake of trying something new, we'll continue with passing in arrays to both `resolve()` and `reject()`, in order to present more data to the user.

This means that we'll also have to update our two UI functions that handle printing the results to the webpage to have just one parameter, and treat that parameter as the array it now is. Here's the updated code:

src/index.js

```
// UI Logic

function printElements(data) {
  document.querySelector('#showResponse').innerText = `The
humidity in ${data[1]} is ${data[0].main.humidity}%.
  The temperature in Kelvins is ${data[0].main.temp} degree
s.`;
}

function printError(error) {
  document.querySelector('#showResponse').innerText = `Ther
e was an error accessing the weather data for ${error[2]}:
${error[0].status} ${error[0].statusText}: ${error[1].messa
ge}`;
}
```

But we haven't even called those functions yet! We need to look to our `promise.then(...)` method call to find those function calls:

src/index.js

```
function getWeather() {
  let promise = new Promise(function(resolve, reject) {
    ...
  });

  promise.then(function(weatherDataArray) {
    printElements(weatherDataArray);
  }, function(errorArray) {
    printError(errorArray);
  });
}
```

In the last lesson, we discussed how `Promise.prototype.then()` takes two arguments. Both are callback functions. The first argument is a function that should run if the promise is resolved. As

we can see, this function calls `printElements`, passing in the `weatherDataArray`.

The second argument is a function that handles a rejected promise. In this case, we call `printError`, passing in the `errorArray` with error information.

As we can see, it didn't take much work at all to incorporate a promise in our code. They're really nice to work with — and they can make working with complex async code much easier.

Example GitHub Repo for API Project with Promises (https://github.com/epicodus-lessons/section-6-js-api-call-with-webpack/tree/2_xhr_api_call_with_promises)

The above link takes you to the branch called `2_xhr_api_call_with_promises`.

Previous (/intermediate-javascript/asynchrony-and-apis/introduction-to-promises)
Next (/intermediate-javascript/asynchrony-and-apis/static-methods-and-properties)

disable dark mode

(http://www.epicodus.com)