Lesson Weekend

Intermediate JavaScript (/intermediate-javascript) / Asynchrony and APIs (/intermediate-javascript/asynchrony-and-apis) / Parsing JSON

Text

Before we continue on to making API calls with JavaScript, let's take a look at an example of a potential response from the OpenWeather API. Often an API call will return complex, deeply-nested JSON. Because JSON is just a set of key-value pairs, we can navigate through it just as we would JavaScript properties of an object.

While we've already worked with JavaScript objects, actually navigating JSON can still be challenging for students at first. A big part of this is the fact that the JSON we are working with is usually more complex than anything we've worked with before. But don't worry — it's not that complex once we understand how to navigate it.

Parsing JSON

We'll often need to get very specific properties from an API response and at the very least we'll probably want to take this JSON data and convert it to a format that looks pleasing on a website. To do that, we need to **parse** the data. That just means breaking it down further to extract the data we need.

We'll use a potential response from the weather API as an example:

```
{
    "coord": {
        "lon": -70.25,
        "lat": 43.66
    },
    "weather": [
        {
            "id": 801,
            "main": "Clouds",
            "description": "few clouds",
            "icon": "02d"
        }
    ],
    "base": "stations",
    "main": {
        "temp": 293.48,
        "pressure": 1019,
        "humidity": 45,
        "temp_min": 291.15,
        "temp_max": 295.93
    },
    "visibility": 16093,
    "wind": {
        "speed": 2.1
    },
    "clouds": {
        "all": 20
    },
    "dt": 1567702485,
    "sys": {
        "type": 1,
        "id": 5454,
        "message": 0.008,
        "country": "US",
        "sunrise": 1567678140,
        "sunset": 1567725058
    },
    "timezone": -14400,
    "id": 4975802,
    "name": "Portland",
    "cod": 200
}
```

Note that this JSON object is mostly a collection of nested key-value pairs. Sometimes the value is a set of yet more key-value pairs. For instance, `"lon"` is a key in an object. This object is itself a value of `"coord"`.

If we were to save this JSON object in a `const` variable called `jsonWeather`, we can access `"coord"` using either bracket notation or dot notation — just as we would access properties of other JavaScript objects.

Go ahead and open the DevTools console and then store the JSON data above in a variable `const jsonWeather` so you can follow along with these examples.

Here's an example using dot notation:

```
> jsonWeather.coord;
{lon: -70.25, lat: 43.66}
```

Here's an example using bracket notation:

```
> jsonWeather["coord"];
{lon: -70.25, lat: 43.66}
```

In general, dot notation is cleaner, however if a property name uses a special character, we must use bracket notation. For example, if there were a key called `"current-weather"` we would have to access it with bracket notation: `jsonWeather["current-weather"]`. Otherwise, we'll get a syntax error.

To access the `"lon"` property that's nested inside of `"coord"`, we'll need to do the following, assuming we are using dot notation:

```
> jsonWeather.coord.lon;
-70.25
```

With a very deeply nested object, we might have to continue specifying properties. In pseudocode, this would look something like this:

```
object.property.nested_property.deeply_nested_property
```

Chaining dot notation in this way will allow us to access most of the properties in a JSON object.

Accessing Array Keys

However, if we look more closely at the JSON object above, we'll see that there's an array inside of the `"weather"` property:

```
"weather": [
        {
            "id": 801,
            "main": "Clouds",
            "description": "few clouds",
            "icon": "02d"
        }
    ],
...
```

Let's see what happens if we try to access it in the same way we'd access other key-value pairs:

```
> jsonWeather.weather.id;
undefined
```

As we can see, `object.weather.id` is `undefined`. This is because `"id"` is not a property of `"weather"`. Instead, it's an element in an array. We can access that element just as we'd access an element in any other JavaScript array:

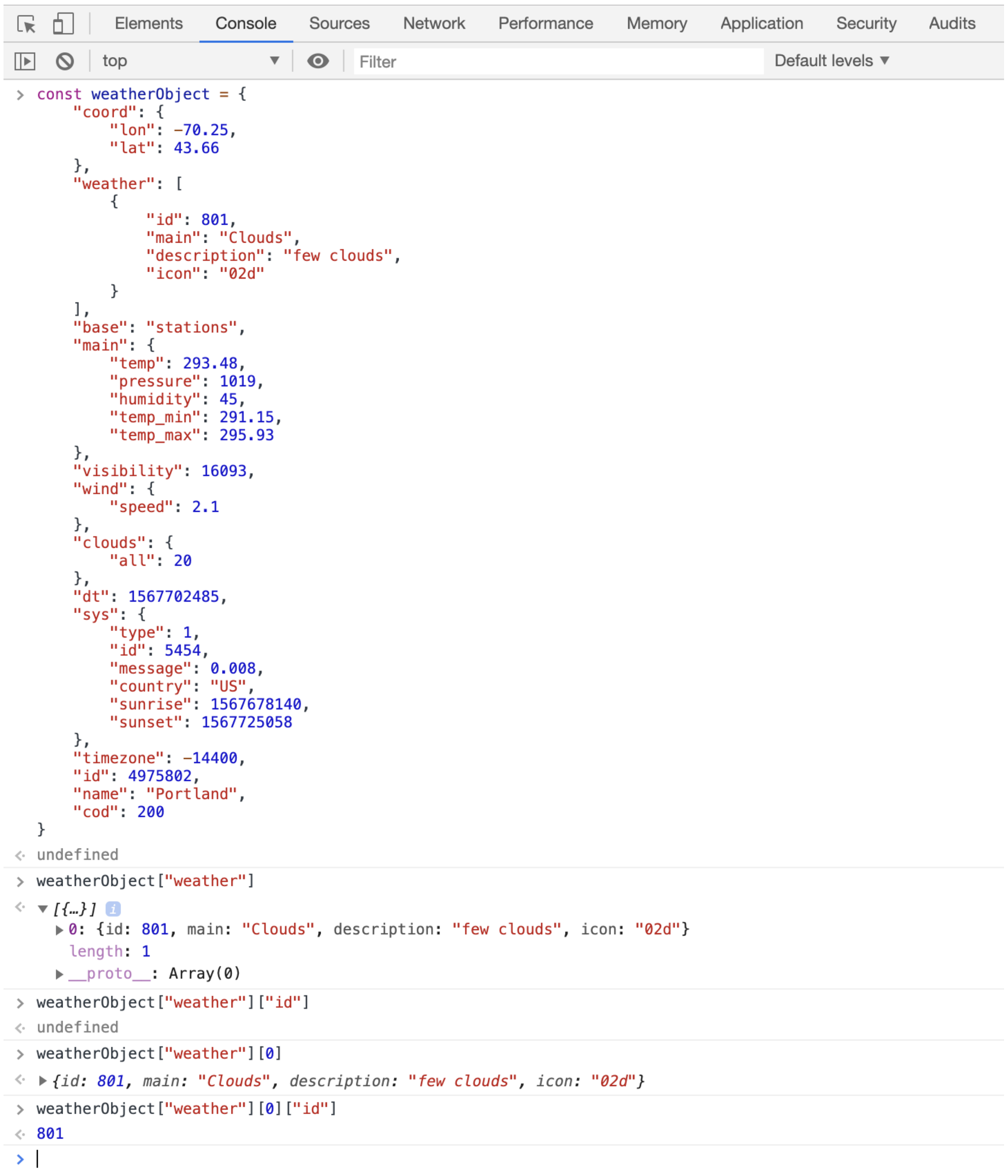
```
> jsonWeather.weather[0];
{id: 801, main: "Clouds", description: "few clouds", icon:
"02d"}
```

We specify that we want the first (and only) element of the array by adding `[0]`. Now we can access more deeply nested key-value pairs within that array element, just as we did before:

```
> jsonWeather.weather[0].main;
"Clouds"
```

Pay close attention to how key-value pairs are nested inside of JSON objects. If you are having issues getting a deeply nested value, make the API call in Postman and then copy the response's JSON body and paste it into the DevTools console, saving it in a variable.

For instance, here is the process we might use to find the `["id"]` property in the OpenWeather API's response. Remember that we can use either dot notation or bracket notation. The example below uses bracket notation — just to ensure you are familiar with both forms of notation:



```
> const weatherObject = {
      "coord": {
          "lon": -70.25,
          "lat": 43.66
      },
      "weather": [
          {
              "id": 801,
              "main": "Clouds",
              "description": "few clouds",
              "icon": "02d"
          }
      ],
      "base": "stations",
      "main": {
          "temp": 293.48,
          "pressure": 1019,
          "humidity": 45,
          "temp_min": 291.15,
          "temp_max": 295.93
      },
      "visibility": 16093,
      "wind": {
          "speed": 2.1
      },
      "clouds": {
          "all": 20
      },
      "dt": 1567702485,
      "sys": {
          "type": 1,
          "id": 5454,
          "message": 0.008,
          "country": "US",
          "sunrise": 1567678140,
          "sunset": 1567725058
      },
      "timezone": -14400,
      "id": 4975802,
      "name": "Portland",
      "cod": 200
  }
< undefined
> weatherObject["weather"]
< ▼[{…}]
    ▶0: {id: 801, main: "Clouds", description: "few clouds", icon: "02d"}
     length: 1
    ▶__proto__: Array(0)
> weatherObject["weather"]["id"]
< undefined
> weatherObject["weather"][0]
< ▶{id: 801, main: "Clouds", description: "few clouds", icon: "02d"}
> weatherObject["weather"][0]["id"]
< 801
> |
```

Looping Through Arrays

Finally, don't forget that once we reach the level of an array in JSON data, we can loop through it just like we would any other array. For instance, let's say we have some JSON that looks like this:

```
{
    "data": [
        {
            "id": 1,
            "name": "Jayne"
        },
        {
            "id": 2,
            "name": "Jasmine"
        },
        {
            "id": 3,
            "name": "Ada"
        },
    ]
}
```

If this were saved inside `const json`, we could do the following:

```
json.data.map(function(person) {
  return person.name;
});
```

This would return the array `["Jayne", "Jasmine", "Ada"]`. We could use any other type of loop as well, such as `Array.prototype.forEach()`.

If you're ever unsure of how to handle data inside JSON, just remember that the usual JavaScript rules about objects and arrays apply. And don't forget to use the console and other tools to parse JSON. That way, it will be much easier to parse JSON in your code as well.

Previous (/intermediate-javascript/asynchrony-and-apis/testing-api-calls-with-postman)

Next (/intermediate-javascript/asynchrony-and-apis/making-api-calls-with-javascript)

Lesson 6 of 33
Last updated more than 3 months ago.

disable dark mode