423 Week 11 XML JSON
Maja Frydrychowicz
2017-03-13

Working with responses

Our initial examples of Ajax requests focused on plain text that had to be parsed manually. It's most common to receive responses in structures text formats, namely JSON and XML, especially if you're retrieve data from a public API.

JSON

Text written in JSON (JavaScript Object Notation) is structured to look like a JavaScript object literal. The goal of JSON is to make it easy to convert to and from JavaScript objects. (However, it's common to work with JSON strings in many other programming languages as well.)

Here's an example of a JSON string. Note that property names and values are always expressed in **double-quotes**, not single-quotes.

```
{
  "Title": "Arrival",
  "Year": "2016",
  "Rated": "PG-13",
  "Country": "USA",
  "Ratings": [
    {
      "Source": "Internet Movie Database",
      "Value": "8.0/10"
    },
    {
      "Source": "Rotten Tomatoes",
      "Value": "94%"
    },
    {
      "Source": "Metacritic",
      "Value": "81/100"
    }
  ],
  "Metascore": "81",
  "imdbRating": "8.0"
}
```

As you can see above, JSON text is organized into name-value pairs inside braces and separated by a comma. A name is always a string, like "imdbRating" and a value can be a:

- primitive type (number, boolean, string),
- an array of values,
- null,
- another object that respects the above restrictions.

Let's say that the above string was received as a response to a request, so it's stored in `myRequest.responseText`:

To convert between JSON string format and JavaScript object use the built-in `JSON` library:

```
// convert the string to an object
var movie = JSON.parse(myRequest.responseText);
// now you can easily look at the object's properties
console.log(movie.Title);
// as with any object, you can use dot notation or array Notation
console.log(movie["Title"]);
// You can convert any object to a JSON string, too.
var x = {"name": "foo", "items": [1,2,3]};
// produces "{\"name\":\"foo\",\"items\":[1,2,3]}"
var xInJSONFormat = JSON.stringify(x);
// and you can send this string to server in a POST request,
// for example
```

Not all objects can be represented in JSON. For example, you can't include a function in JSON, or a regular expression object. If you try to use `stringify` on such a value, it will just be ignored:

```
var x = {"a": function() { return 1 + 1; }}
JSON.stringify(x)
// empty braces because the value of "a" can't be
// represented in JSON format
"{}"
```

Cross-browser note

Very old browsers don't support `JSON.parse`. Instead, you could use the `eval` built-in function, but that opens your up to security vulnerabilities -- make sure to validate the data before parsing it with `eval`!

XML

XML (Extensible Markup Language) represents data in a hierarchy of nodes, similarly to HTML. It consists of opening and closing tags and it can be parsed into a DOM tree. As in HTML, each node can also have attributes.

Example 1:

```
<LANGUAGES>
    <ENGLISH>
        <FIELD>First Name:</FIELD>
        <FIELD>Last Name:</FIELD>
        <FIELD>Address 1:</FIELD>
        <FIELD>Address 2</FIELD>
        <FIELD>City:</FIELD>
        <FIELD>Province:</FIELD>
        <FIELD>Postal Code:</FIELD>
    </ENGLISH>
</LANGUAGES>
```

Example 2:

```
<root response="True">
  <movie
    title="Arrival"
    year="2016"
    rated="PG-13"
    released="11 Nov 2016"
    runtime="116 min"
    imdbRating="8.0"
    type="movie"/>
</root>
```

The nodes in XML can be anything, like `<movie>`, `<animal>`, `<blob>`. Whoever is writing the XML document designs the *schema*.

XML is very strict: in order to be valid, all nodes must have opening and closing tags, etc. (HTML5 is a loose example of XML, but it is less strict -- e.g. it doesn't always require closing tags, as with `<input>` and `<br>`.)

How do we interact with XML in JavaScript? If you want to get the "year" out of the XML in Example 2, you do that through DOM manipulation methods, just like you've been doing with the HTML DOM so far -- it's the same principle.

If you make a request to a server and the server responds with XML, you can access it via `myRequest.responseXML`, which will be an `XMLDocument` object. The `XMLDocument` object represents the DOM for the XML data received from the server.

Looking at Example 2 again:

```
// getElementsByTagName returns an HTMLCollection
var movie = myRequest.responseXML.getElementsByTagName("movie")[0];
console.log(movie.getAttribute("title"));
// another way: this returns a NodeList
var movie = myRequest.responseXML.querySelector("movie");
```

The above code demonstrates that many of the same DOM manipulation methods are used for XML just as with the HTML DOM: childNodes, parentNode, nodeType, appendChild, removeChild, etc.

You can also parse a string to turn it into an XMLDocument.

```
var text = "<pet><species>dog</species>" +
"<breed>Labrador</breed></pet>";

var parser = new DOMParser();
var xmlDoc = parser.parseFromString(text,"text/xml");
// this will be "dog"
var species = xmlDoc.querySelector("species").childNodes[0].nodeValue
// this will print "#text"
console.log(
  xmlDoc.querySelector("species").childNodes[0].nodeName);
// this will print "species"
console.log(
  xmlDoc.querySelector("species").nodeName);
// documentElement represents the root, in this case <pet>
var root = xmlDoc.documentElement;
```

Node types

It's useful to look at the `nodeType` property of an XML node to distinguish between text nodes, tags and so on. In `<breed>Labrador</breed>`, "Labrador" has `nodeType` 3, and `breed` has `nodeType` 1. (The same node types are used in the HTML DOM as well.)

XML and whitespace

Depending on the browser and the XML data, you may run into inconsistent behaviour wherein whitespace is sometimes treated as a separate child node. Be aware of this while debugging and testing.

```
//note the \n after <pet>
var text = "<pet>\n<species>dog</species>" +
"<breed>Labrador</breed></pet>";

var parser = new DOMParser();
var xmlDoc = parser.parseFromString(text,"text/xml");

// We expect petChildren to be [species, dog]
// but in older browsers it might be [#text, species, dog]
// because of the extra whitespace (\n).
// The \n might be treated as an empty text node
var petChildren = xmlDoc.documentElement.childNodes;
```

XML versus JSON

JSON and XML are different ways of organizing information in a string or plain-text file. They are "human readable", meaning you can just look at the raw text and easily understand its content. They are both used in many kinds of applications, tools and programming languages.

XML is more strictly structured, which can be beneficial, but it's less efficient and less convenient to work with as a result. You will encounter XML a lot more in other settings, like project configuration.

Why JSON is used more often in client-side web development:

- Can express the same idea with fewer characters:
  - `{"movie": {"title": "Snow White"}}`
  - versus `<movie><title>Snow White</title></movie>`
- JSON is better at encoding lists. An example of a list in XML looks like
  - `<address></field>A</field><field>B</field><field>C</field></address>`
  - instead of `{"address": {"fields": [A,B,C]}}`
- JSON is native to JS so it's easy and efficient to parse into JS objects. (In contrast, getting specific data out of XML amounts to manipulating a DOM, which is expensive.)

Last update: 2017-04-06 10:42:29

Source: _javascript_423/lectures/11_xml_json.md