How-To: Get Started with JSON

What is JSON?
A better format for data exchange

JSON has eclipsed XML as the preferred data interchange format for web applications and web services. Here's why

JSON / Thinkstock

JavaScript Object Notation is a schema-less, text-based representation of structured data that is based on key-value pairs and ordered lists. Although JSON is derived from JavaScript, it is supported either natively or through libraries in most major programming languages. JSON is commonly, but not exclusively, used to exchange information between web clients and web servers.

Over the last 15 years, JSON has become ubiquitous on the web. Today it is the format of choice for almost every publicly available web service, and it is frequently used for private web services as well.

The popularity of JSON has also resulted in native JSON support by many databases. Relational databases like PostgreSQL and MySQL now ship with native support for storing and querying JSON data. NoSQL databases like MongoDB and Neo4j also support JSON, though MongoDB uses a slightly modified, binary version of JSON behind the scenes.

In this article, we'll take a quick look at JSON and discuss where it came from, its advantages over XML, its drawbacks, when you should use it, and when you should consider alternatives. But first, let's dive into the nitty gritty of what JSON looks like in practice.

0 seconds of 30 secondsVolume 0%

JSON example

Here's an example of data encoded in JSON:

```
{
  "firstName": "Jonathan",
  "lastName": "Freeman",
  "loginCount": 4,
  "isWriter": true,
  "worksWith": ["Spantree Technology Group", "InfoWorld"],
  "pets": [
    {
      "name": "Lilly",
      "type": "Raccoon"
    }
  ]
}
```

The structure above clearly defines some attributes of a person. It includes a first and last name, the number of times the person has logged in, whether this person is a writer, a list of companies the person works with, and a list of the person's pets (only one, in this case). A structure like the one above may be passed from a server to a web browser or a mobile application, which will then perform some action such as displaying the data or saving it for later reference.

JSON is a generic data format with a minimal number of value types: strings, numbers, booleans, lists, objects, and null. Although the notation is a subset of JavaScript, these types are represented in all common programming languages, making JSON a good candidate to transmit data across language gaps.

JSON files

JSON data is stored in files that end with the .json extension. In keeping with JSON's human-readable ethos, these are simply plain text files and can be easily opened and examined. As the SQLizer blog explains, this is also a key to JSON's wider interoperability, as just about every language you can name can read and process plain text files, and they're easy to send over the Internet.

Why should I use JSON?

To understand the usefulness and importance of JSON, we'll have to understand a bit about the history of interactivity on the web.

In the early 2000s, interactivity on the web began to transform. At the time, the browser served mainly as a dumb client to display information, and the server did all of the hard work to prepare the content for display. When a user clicked on a link or a button in the browser, a request would be sent to the server, the server would prepare the information needed as HTML, and the browser would render the HTML as a new page. This pattern was sluggish and inefficient, requiring the browser to re-render everything on the page even if only a section of the page had changed.

Because full-page reloads were costly, web developers looked to newer technologies to improve the overall user experience. Meanwhile, the capability of making web requests in the background while a page was being shown, which had recently been introduced in Internet Explorer 5, was proving to be a viable approach to loading data incrementally for display. Instead of reloading the entire contents of the page, clicking the refresh button would trigger a web request that would load in the background. When the contents were loaded, the data could be manipulated, saved, and displayed on the page using JavaScript, the universal programming language in browsers.

REST vs. SOAP: The JSON connection

Originally, this data was transferred in XML format (see below for an example) using a messaging protocol called SOAP (Simple Object Access Protocol). But XML was verbose and difficult to manage in JavaScript. JavaScript already had objects, which are a way of expressing data within the language, so Douglas Crockford took a subset of that expression as a specification for a new data interchange format and dubbed it JSON. JSON was much easier for people to read and for browsers to parse.

Over the course of the '00s, another Web services technology, called Representational State Transfer, or REST, began to overtake SOAP for the purpose of transferring data. One of the big advantages of programming using REST APIs is that you can use multiple data formats — not just XML, but JSON and HTML as well. As web developers came to prefer JSON over

XML, so too did they come to favor REST over SOAP. As Kostyantyn Kharchenko put it on the Svitla blog, "In many ways, the success of REST is due to the JSON format because of its easy use on various platforms."

Today, JSON is the de-facto standard for exchanging data between web and mobile clients and back-end services.

JSON vs. XML

As noted above, the main alternative to JSON is XML. However, XML is becoming less and less common in new systems, and it's easy to see why. Below is a version of the data you saw above, this time in XML:

```
<?xml version="1.0"?>
<person>
  <first_name>Jonathan</first_name>
  <last_name>Freeman</last_name>
  <login_count>4</login_count>
  <is_writer>true</is_writer>
  <works_with_entities>
    <works_with>Spantree Technology Group</works_with>
    <works_with>InfoWorld</works_with>
  </works_with_entities>
  <pets>
    <pet>
      <name>Lilly</name>
      <type>Raccoon</type>
    </pet>
  </pets>
</person>
```

In addition to being more verbose (exactly twice as verbose in this case), XML also introduces some ambiguity when parsing into a JavaScript-friendly data structure. Converting XML to a JavaScript object can take from tens to hundreds of lines of code and ultimately requires customization based on the specific object being parsed. Converting JSON to a JavaScript object takes one line of code and doesn't require any prior knowledge about the object being parsed.

Limitations of JSON

Although JSON is a relatively concise, flexible data format that is easy to work with in many programming languages, there are some drawbacks to the format. Here are the five main limitations:

1. No schema. On the one hand, that means you have total flexibility to represent the data in any way you want. On the other, it means you could accidentally create misshapen data very easily.
2. Only one number type: the IEEE-754 double-precision floating-point format. That's quite a mouthful, but it simply means that you cannot take advantage of the diverse and nuanced number types available in many programming languages.
3. No date type. This omission means developers must resort to using string representations of dates, leading to formatting discrepancies, or must represent dates in the form of milliseconds since the epoch (January 1, 1970).
4. No comments. This makes it impossible to annotate fields inline, requiring additional documentation and increasing the likelihood of misunderstanding.
5. Verbosity. While JSON is less verbose than XML, it isn't the most concise data interchange format. For high-volume or special-purpose services, you'll want to use more efficient data formats.

When should I use JSON?

If you're writing software that communicates with a browser or native mobile application, you should use JSON as the data format. Using a format like XML is an out-of-date choice and a red flag to front-end and mobile talent you'd otherwise like to attract.

In the case of server-to-server communication, you might be better off using a serialization framework like Apache Avro or Apache Thrift. JSON isn't a bad choice here, and still might be exactly what you need, but the answer isn't as clear as for web and mobile communication.

If you're using NoSQL databases, you're pretty much stuck with whatever the database gives you. In relational databases that support JSON as a type, a good rule of thumb is to use it as little as possible. Relational databases have been tuned for structured data that fits a particular schema. While most now support more flexible data in the form of JSON, you can expect a performance hit when querying for properties within those JSON objects.

JSON is the ubiquitous, de facto format for sending data between web servers and browsers and mobile applications. Its simple design and flexibility make it easy to read and understand, and in most cases, easy to manipulate in the programming language of your choice. The lack of a strict schema enables flexibility of the format, but that flexibility sometimes makes it difficult to ensure that you're reading and writing JSON properly.

JSON parser

The part of an application's code that transforms data stored as JSON into a format the application can use is called a *parser.* JavaScript, as you'd expect, includes a native parser, the JSON.parse() method.

You may have to do a little more work to work with JSON in strongly typed languages like Scala or Elm, but the widespread adoption of JSON means there are libraries and utilities to help you through all of the hardest parts.

The json.org website includes a comprehensive list of code libraries you can use to parse, generate, and manipulate JSON, in languages as diverse as Python, C#, and COBOL.

JSON utilities

If you're looking to manipulate or examine JSON-encoded data directly, without writing code yourself, there are a number of online utilities that can help you. All of programmatic equivalents in the code libraries linked to above, but you can cut and paste JSON code into these browser-based tools to help you understand JSON better or perform quick-and-dirty analysis:

- **JSON Formatter:** JSONLint will format and validate arbitrary JSON code.
- **JSON Viewer:** Stack.hu has a site that will create an interactive tree to help you understand the structure of your JSON code.
- **JSON Beautifier:** If you want to "pretty print" your JSON code, with syntax coloring and the like, Prettydiff can help you out.
- **JSON Converter:** Need to quickly move data from a JSON format into something else? Convertcsv.com has tools that can convert JSON to CSV (which can then be opened in Excel) or XML.

JSON tutorial

Ready to dive in and learn more about how work with JSON in your interactive applications? The Mozilla Developer Network has a great tutorial that will get you started with JSON and JavaScript. If you're ready to move on to other languages, check out tutorial on using JSON with Java (from Baeldung), with Python (from DataCamp), or with C# (from Software Testing Help). Good luck!

Courtesy: https://www.infoworld.com/article/3222851/what-is-json-a-better-format-for-data-exchange.html

Modified: 2021.10.04.1.50.PM

Dököll Solutions, Inc.