2019 IEEE 13th International Conference on Semantic Computing (ICSC)

NodeJS and Angular Tools for JSON-LD

Aaron Sterling
Department of Computer Science
Iowa State University
Ames, Iowa, USA
sterling@iastate.edu

***Abstract*—We report on three open-source, ISC-licensed tools that improve the experience of web programmers who use jsonld.js, the official JSON-LD JavaScript library. Two of these tools are for NodeJS, which is primarily a backend framework, with which one can manage servers using JavaScript. The NodeJS tools are: jldc, a NodeJS command line tool; and node-jsonld, a module that can be plugged into other programs. The two NodeJS tools are written in Typescript 3 and transpiled to JavaScript ES2015. The third tool is written in and for the web framework Angular, which is primarily used in frontend development. The Angular tool is ngx-jsonld-provider. All three tools extend jsonld.js with better error trapping, more verbose help messages, and greater type safety. All three tools are registered with the Node Package Manager (npm). At the time of this writing, web developers have downloaded them from npm more than 500 times combined.**

I. Introduction

This paper reports on three open-source tools that extend the official JSON-LD JavaScript software library, so that more typesafe, better documented JSON-LD operations are available to both frontend and backend web developers.

A significant challenge for web programmers is to ensure data can be correctly understood across many different platforms. For example, are different databases using terms like "name" or "employee" in the same way—and, if not, how best to translate between them? One solution to this problem is Linked Data, which provides context to a term like "name" by attaching to "name" a web address that points to a formal definition of "name," so the term's meaning is unambiguous [1]. JSON-LD is a W3C standard that allows developers to attach linked data to JSON objects [2]. (JSON is a data-interchange format that is often used in web programming [3].) The JSON-LD specification provides algorithms for several operations on JSON data with respect to a linked-data context. There are official software libraries for these algorithms available in several programming languages [4], including JavaScript [5]. However, this JavaScript library is written in "common" JavaScript, and does not take advantage of powerful libraries and techniques that have recently become available.

Over the last five years, the open-source community has put in a tremendous amount of effort to make JavaScript a safer, more mature language. Two open-source projects in that effort are Typescript and Inversify. Typescript, backed by Microsoft, adds static types to JavaScript [6]. Typescript is now extremely popular among web developers, because type checking helps push runtime errors back to compile time, and makes maintaining large projects easier. Inversify is less-known than Typescript, but used by many companies with large-scale Javascript projects [7]. Perhaps Inversify's greatest strength is that it provides Javascript and Typescript programmers with a mechanism for Inversion of Control [8, 9]. Inversion of Control has become a popular design principle in web programming, in part because it focuses on the importance of contracts between modules, without having to know anything about the inner workings of a module.

The two tools written with Typescript and Inversify are interfaces between jsonld.js and the popular server management software NodeJS [10]. NodeJS, written in C++, takes the same Javascript engine that appears in Google's Chrome browser, and runs independent of the ability to render visual content (for example, no HTML). NodeJS is often used as the backend part of a full stack, because developers can use the same language for the backend (Javascript/Typescript) as they do for the frontend, and because queries to servers managed by NodeJS are nonblocking. Therefore, two of the tools we present in this paper are intended to be consumed by a popular backend technology. The third is written for Angular, a frontend Javascript framework.

The web development framework Angular [11], backed by Google, was originally written in an extension of Javascript, AtScript, but Microsoft and Google agreed to collaborate, and extended an earlier version of Typescript so "modern" Typescript subsumes AtScript. Angular makes it easy for programmers to use *dependency injection*, an aspect of Inversion of Control. The third tool we present in this paper is an interface between Angular and jsonld.js.

We present the following tools in the next section: **jldc**[1], a NodeJS command line interface for JSON-LD operations on files; **node-jsonld**[2], a Javascript ES6 module whose code is related to that of jldc, which lets NodeJS programmers perform JSON-LD operations programmatically; and **ngx-jsonld-provider**[3], an Angular provider that functions as a typed interface to the JSON-LD library. All three tools we present in this paper are strongly typed (through Typescript 3, the most recent version), and written consistent with Inversion of Control principles (using either Inversify or Angular's

[1]Source code and documentation for jldc is available at https://github.com/Aaron-Sterling/jldc

[2]Source code and documentation for node-jsonld is available at https://github.com/Aaron-Sterling/node-jsonld

[3]Source code and documentation for ngx-jsonld-provider is available at https://github.com/Aaron-Sterling/ngx-jsonld-provider

978-1-5386-6783-5/19/$31.00 ©2019 IEEE
DOI 10.1109/ICSC.2019.00077

dependency injection). We believe that this adds important scaling and safety features to jsonld.js. All three are open-source, and their source code can be cloned from Github repositories. All are ISC-licensed, so they are free to use for any purpose. Finally, all three tools are packages registered with Node Package Manager (npm—a popular library download service), and, combined, have been downloaded hundreds of times.

II. The Tools

A. *jldc: NodeJS command line interface*

The original JSON-LD operations of jsonld.js accept urls that point to locations on the web for, e.g., a JSON-LD source and a context in order to run the compact algorithm on the source. The first two tools we consider—jldc and node-jsonld—extend the functionality of those operations with the file system library of NodeJS, so JSON-LD operations can be performed on resources read from, and written to, a local disk.

jldc is a NodeJS command line interface for jsonld.js. An official NodeJS command line interface for jsonld.js already exists: jsonld-cli [12]. However, jsonld-cli was written in common Javascript three years ago, before libraries like Inversify (or mature Typescript) were available. Further, the user interface of jsonld-cli is extremely simple; for example, it does not ask for confirmation before overwriting an existing file.

With jldc, we built a command line interface (CLI) that has verbose messages that explain the tasks being performed, any errors encountered, and an interactive confirmation process that requires the user to say "yes" before any existing file is overwritten. (jsonld-cli has none of these features.) The core of jldc is built with jsonld.js and the CLI library *Commander.js* [13]. The jldc interface is built using the libraries *chalk* (which provides colored and formatted text) [14], and *Inquirer.js* (which handles asking the user to confirm overwrite steps) [15].

As one example, the command to run the JSON-LD compact algorithm is

```
jldc compact sourceFile.json
 contextFile.json targetFile.json
```

This command returns distinct errors if the source file or context file do not exist, and will commence a confirmation dialog if the target file *does* exist. If there are no errors, and all user confirmations receive a "yes" response, then jldc will execute the compact algorithm on the contents of sourceFile.json, consistent with the contents of context.json, and write the result to targetFile.json.

As mentioned in the Introduction, jldc is built with Inversify and Typescript 3. Therefore, jldc not only provides a richer user experience than was previously available with jsonld-cli, but also, under the hood, better follows web programming best practices (type safety and Inversion of Control).

One disadvantage of jldc is that is can only be used at the command line, not as a module within a larger program. We have repurposed the core code of jldc to create just such a module, which we consider next.

B. *node-jsonld: NodeJS plugin module*

node-jsonld can be thought of as the code for jldc, with the command line interface replaced by a Javascript ES6 module with methods both to perform JSON-LD operations, and to get in-depth help messages. The consuming code imports the module `nodeJsonLd`, and the command to perform the compact operation looks like:

```
nodeJsonLd.compact(source, context,
target);
```

Here `source`, `context` and `target` are names of files. All JSON-LD operations, including `nodeJsonLd.compact`, are asynchronous calls, of type `Promise<string>`. This type is a Promise from the Javascript ES2015 spec [16]: the call eventually returns a string, or throws an error if one occurred. The node-jsonld API guarantees that the string will be `'successful'` if the JSON-LD operation was successful, or an error code if the operation was unsuccessful. (A technical note: this aspect of the API is an improvement over both the type of Promise available from jsonld.js alone, which relies on a browser polyfill and does not play well with Typescript 3; and over the Promise API of the NodeJS file system module, which is still experimental.)

Like jldc, the code of node-jsonld is separated into injectable modules, and then injected as needed, using Inversify. There are two main pieces of code: the JSON-LD Operations Service, which handles the interface to jsonld.js; and Help Message Service, which provides extended help messages for each JSON-LD operation that the consuming code can use as desired (for example in tooltips). There is a getter for each JSON-LD operation, such as `getCompactHelpMessage()` and `getFlattenHelpMessage()`. These all return strings.

We now turn our attention from the backend (NodeJS) to a JSON-LD tool for a frontend framework (Angular).

C. *ngx-jsonld-provider: Angular provider*

Roughly speaking, the structure of Angular permits two kinds of code objects: *components* and *providers*. A component contains code that renders something on the screen. A web page is a component, for example, as are much smaller things that might be embedded into a page, such as a button. By contrast, a provider is a piece of code that does not render data for the user. Providers may read data from a disk or a database, or process information, and then convey data to components, which are responsible for deciding what parts of that data to render.

The previous work that is closest to ngx-jsonld-provider is ngx-json-ld by Rylan [17]. ngx-json-ld is an Angular component that displays results of JSON-LD operations in HTML. While we find the project interesting, we also believe its strategy is limited, because JSON-LD operations are inherently data transformations, and so their natural location in Angular is within a provider, not a component. Therefore, we have built a JSON-LD provider, and left all rendering questions to the consuming Angular code.

To use ngx-jsonld-provider, one registers the provider with Angular in the core app module (a standard step with any

globally-scoped provider), and then calls it within a particular component as shown in the following toy example.

```
import {NgxJsonLdProvider}
       from 'ngx-jsonld-provider';

const name={'http://schema.org/name':
            'John Doe'};
const context={'name':
               'http://schema.org/name'};
class foo {
  constructor(jsonld: NgxJsonLdProvider) {
    jsonld.compact(name, context)
          .then(res => console.log(res));
  }
}
```

In an instance of Angular's dependency injection, `NgxJsonLdProvider` is injected into the constructor of the class `foo`. The methods of the provider are then available to `foo`. `jsonld.compact` calls the compact operation of jsonld.js. Unlike the NodeJS tools, `jsonld.compact` returns type `JsonLd`, so the variable res is a `JsonLd` object.

The NodeJS compact operation wrote the result of the operation to a file and returned a string that reported on the entire process, including the file reads and writes. Since Angular is a frontend framework, the expectation is that this code will be running in a browser, and one security feature of modern browsers is that they do not have access to the local file system. So the inputs to `jsonld.compact` are strings or JSON, the output is of type `JsonLd`, and it is the responsibility of the consuming code to determine what to do with the operation's output.

One advantage an Angular provider has over either a command line interface or an ES6 module that could be consumed by many kinds of code, is the guarantee of stable storage in between operation calls. ngx-jsonld-provider has a functionality the other tools do not: the ability to reset the default options for each JSON-LD operation. There is a getter for each operation, for example `getCompactOptions()`, which return an object of type specific to the options of that operation, such as `CompactOptions`, `FlattenOptions`, etc. The getters report on the current value of the options object, and, using setters like `setCompactOptions(newOptions: CompactOptions)`, a developer can change the default options programmatically.

Like the other two tools, ngx-jsonld-provider gives access to verbose help messages. These are available through the same getters that appear in node-jsonld (`getCompactHelpMessage()` etc.).

III. Conclusion

We created these tools because we were trying to build mobile apps that used computational ontologies, and we found that the most robust tools for ontology manipulation, like the OWL API [18], were very desktop-centric—written in Java without web-friendly interfaces. (Even WebProtege [19], a web portal to a well known ontology editor, is dedicated to one user using one online tool, instead of distributed programming or other characteristics of web computing.) JSON-LD is a file format that is sometimes used for ontologies, and we believe it is a good fit for web manipulation of ontologies, because a principal motivation for the design of JSON-LD was to improve web APIs. Therefore, we designed backend and frontend JSON-LD tools for our own use, and we believe we stumbled onto a general community interest, because jldc alone was downloaded over 300 times in the week after it was published to npm. We hope our work inspires the open source community to build more web tools for JSON-LD, and for computational ontologies.

References

[1] T. Berners-Lee, "Linked data," https://www.w3.org/DesignIssues/LinkedData.html, July 2006, accessed 27 September 2018.

[2] M. Sporny, G. Kellogg, M. Lanthaler *et al.*, "JSON-LD 1.0: A JSON-based serialization for linked data," https://www.w3.org/TR/json-ld/, January 2014, accessed 27 September 2018.

[3] "Introducing JSON," https://www.json.org/, December 1999, accessed 27 September 2018.

[4] "JSON for linking data," https://json-ld.org/, accessed 27 September 2018.

[5] "jsonld.js," https://github.com/digitalbazaar/jsonld.js/, accessed 27 Septenber 2018.

[6] "Typescript: Javascript that scales," https://www.typescriptlang.org/, October 2012, accessed 28 September 2018.

[7] "InversifyJS," http://inversify.io/, accessed 28 September 2018.

[8] M. Mattsson, "Object oriented frameworks, a survey of methodoligical issues," Master's thesis, Blekinge Institute of Technology, February 1996.

[9] S. Mazzocchi, "On Inversion of Control," https://web.archive.org/web/20040202120126/http://www.betaversion.org/~stefano/linotype/news/38/, January 2004, accessed 28 September 2018.

[10] "NodeJS," https://nodejs.org, accessed 30 September 2018.

[11] "Angular," https://angular.io/, September 2016, accessed 28 September 2018.

[12] D. Lehn, "jsonld-cli," https://github.com/digitalbazaar/jsonld-cli, September 2015, accessed 30 September 2018.

[13] "Commander.js," https://github.com/tj/commander.js, accessed 30 September 2018.

[14] "chalk," https://github.com/chalk/chalk, accessed 30 September 2018.

[15] "Inquirer.js," https://github.com/SBoudrias/Inquirer.js/, accessed 30 September 2018.

[16] "ECMAScript 2015: Promise objects," https://www.ecma-international.org/ecma-262/6.0/#sec-promise-objects, accessed 30 September 2018.

[17] C. Rylan, "ngx-json-ld," https://github.com/coryrylan/ngx-json-ld, March 2018, accessed 30 September 2018.

[18] M. Horridge and S. Bechhofer, "The OWL API: A Java API for OWL ontologies," *Semantic Web Journal: Special Issue on Semantic Web Tools and Systems*, vol. 2, no. 1, pp. 11–21, 2011.

[19] T. Tudorache, C. Nyulas, N. F. Noy, and M. A. Musen, "WebProtégé: A collaborative ontology editor and knowledge acquisition tool for the web," *Semantic Web*, vol. 4, no. 1, pp. 88–99, 2013.