post02-yowsean-li

Yowsean Li

11/27/2017

Alternatives to XML for Data Serialization

Introduction

In class, we were introduced to XML (eXtensible Markup Language), a format that makes it possible to store and share data while retaining its structure and key information such as labels for data values. There are many other types of formats that can achieve a similar function to XML, including JSON (JavaScript Object Notation) and YAML (YAML Ain't Markup Language). JSON has become increasingly popular in recent years for storing data objects and is often used instead of XML. YAML is not as commonly used for data storage, but is also capable of replacing the functionality of XML in many cases. In the real world, XML and JSON are the most common formats data is stored for interchanging, or transfering.

In this post, I will go over the differences between the three different formats and the unique properties that make them more preferable to use in certain situations. In addition, I will give a brief overview of how to manipulate and import these formats into usable data in R.

Motivation and Background

The motivation to explore this topic for this post came from the complex form of the XML format. I wanted to find other formats that were simpler, but also could get the job done. This led me to the JSON and YAML formats which both have a more minimal approach and have their advantages to XML, but lack some features.

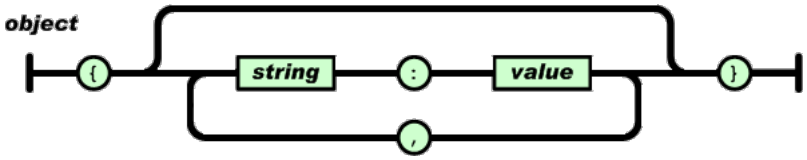
Discussion

In this section, we'll introduce the JSON and YAML formats and discuss the advantages of each.

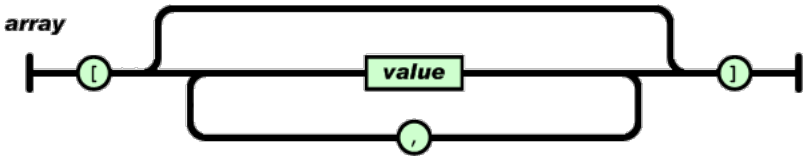
JSON and its advantages

The general format of JSON is with two main structures – key-value pairs and lists. Braces surround key-value pairs to form objects. Square brackets denote lists, where entries are separated by commas. Values can be strings, numbers, boolean values, objects, or arrays. Objects can also be nested in other objects, allowing for a hierarchical structure.

The diagrams below demonstrate the basic idea of JSON syntax.

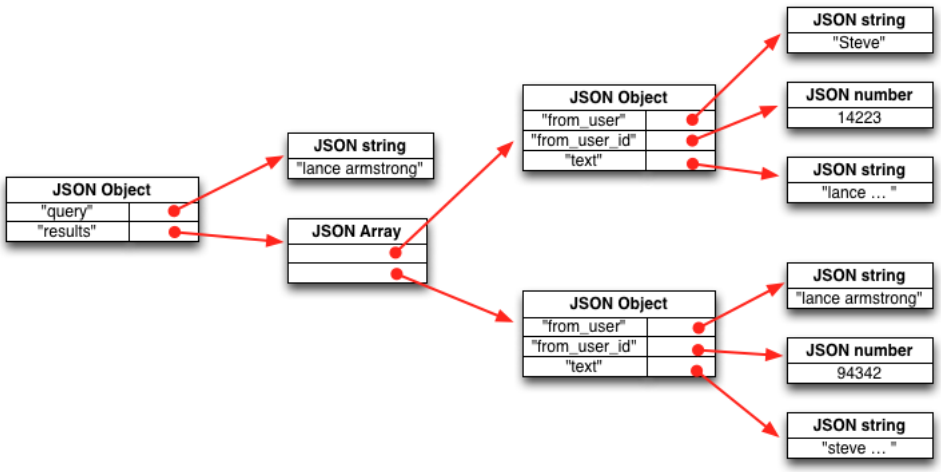


Object



Array

The following image shows an example of a nested JSON structure.



JSON structure

The syntax between JSON and XML is vastly different. Compared to JSON, XML is more verbose as it requires tags before and after the data entry which results in a lot of unnecessary text and memory, especially in large datasets. The object and array notation is also unique to JSON, allowing the user to easily see these structures. It is definitely possible to have these structures in XML as well, but it is not as easy to read for the user.

The visual below shows how JSON is more concise than XML. On the left, the JSON format has a label paired with each entry and braces surrounding each object. On the right, we see that the XML format has each label twice, surrounding the entries.

http://localhost:8080/Json/SyncReply/Contacts


```
{
  - Contacts: [
      - {
            FirstName: "Demis",
            LastName: "Bellot",
            Email: "demis.bellot@gmail.com"
        },
      - {
            FirstName: "Steve",
            LastName: "Jobs",
            Email: "steve@apple.com"
        },
      - {
            FirstName: "Steve",
            LastName: "Ballmer",
            Email: "steve@microsoft.com"
        },
      - {
            FirstName: "Eric",
            LastName: "Schmidt",
            Email: "eric@google.com"
        },
      - {
            FirstName: "Larry",
            LastName: "Ellison",
            Email: "larry@oracle.com"
        }
    ]
}
```


http://localhost:8080/Xml/SyncReply/Contacts


```
<ContactsResponse xmlns:i="http://www.w3.org/20
  <Contacts>
    <Contact>
      <Email>demis.bellot@gmail.com</Email>
      <FirstName>Demis</FirstName>
      <LastName>Bellot</LastName>
    </Contact>
    <Contact>
      <Email>steve@apple.com</Email>
      <FirstName>Steve</FirstName>
      <LastName>Jobs</LastName>
    </Contact>
    <Contact>
      <Email>steve@microsoft.com</Email>
      <FirstName>Steve</FirstName>
      <LastName>Ballmer</LastName>
    </Contact>
    <Contact>
      <Email>eric@google.com</Email>
      <FirstName>Eric</FirstName>
      <LastName>Schmidt</LastName>
    </Contact>
    <Contact>
      <Email>larry@oracle.com</Email>
      <FirstName>Larry</FirstName>
      <LastName>Ellison</LastName>
    </Contact>
  </Contacts>
</ContactsResponse>
```


JSON and XML

YAML and its advantages

YAML has a similar format to JSON, but replaces the braces and brackets with indentation. Essentially, the level of indentation indicates the hierarchy of the data. Below is a comparison between JSON and YAML formats. We can see that they are very similar. The main differences lie in the functionality of the two formats.

YAML and JSON

YAML supports many capabilities not permitted in JSON, such as self referencing, complex data types, and comments. While YAML has more features than JSON, it is not commonly used for data storage as its features are not particularly useful in that field. It is a popular format for configuration files, however. Furthermore, YAML is more complicated to use and does not have as much support as JSON or XML.

Advantages of XML

While JSON and YAML have significant advantages over XML in terms of conciseness and readability, XML still has its advantages in certain situations. The use of tags, for instance, allows for metadata to be stored with the data, something that JSON and YAML cannot do efficiently. It is also simpler to query specific data entries from XML without iterating through the file using XPath. All of these data types have slightly different purposes although their abilities may overlap. XML works best for document markup, JSON for structured data interchange, and YAML for specific cases where its features may be needed.

Examples

In this portion of the post, I will go over how to deal with JSON and YAML data in R through some examples. For JSON, we will use the `jsonlite` library and for YAML we will use the `yaml` library.

```r
# Import libraries
library(jsonlite)
library(yaml)
```

First, we will take an existing dataframe from R, `mtcars`, and convert it into a JSON format with the `toJSON` function.

```
# Take first 5 entries from mtcars
mtcars <- head(mtcars)
mtcars
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
# Convert dataframe to JSON
mtcars_JSON <- toJSON(mtcars)
mtcars_JSON
```

```
## [{"mpg":21,"cyl":6,"disp":160,"hp":110,"drat":3.9,"wt":2.62,"qsec":16.46,"vs":0,"am":1,"gear":4,"carb":4,"_row"
:"Mazda RX4"},{"mpg":21,"cyl":6,"disp":160,"hp":110,"drat":3.9,"wt":2.875,"qsec":17.02,"vs":0,"am":1,"gear":4,"car
b":4,"_row":"Mazda RX4 Wag"},{"mpg":22.8,"cyl":4,"disp":108,"hp":93,"drat":3.85,"wt":2.32,"qsec":18.61,"vs":1,"am"
:1,"gear":4,"carb":1,"_row":"Datsun 710"},{"mpg":21.4,"cyl":6,"disp":258,"hp":110,"drat":3.08,"wt":3.215,"qsec":19
.44,"vs":1,"am":0,"gear":3,"carb":1,"_row":"Hornet 4 Drive"},{"mpg":18.7,"cyl":8,"disp":360,"hp":175,"drat":3.15,"
wt":3.44,"qsec":17.02,"vs":0,"am":0,"gear":3,"carb":2,"_row":"Hornet Sportabout"},{"mpg":18.1,"cyl":6,"disp":225,"
hp":105,"drat":2.76,"wt":3.46,"qsec":20.22,"vs":1,"am":0,"gear":3,"carb":1,"_row":"Valiant"}]
```

We can make this a more readable format by setting the `pretty` attribute to `TRUE`.

```
# mtcars in readable JSON format
mtcars_JSON <- toJSON(mtcars, pretty=TRUE)
mtcars_JSON
```

```
## [
##   {
##     "mpg": 21,
##     "cyl": 6,
##     "disp": 160,
##     "hp": 110,
##     "drat": 3.9,
##     "wt": 2.62,
##     "qsec": 16.46,
##     "vs": 0,
##     "am": 1,
##     "gear": 4,
##     "carb": 4,
##     "_row": "Mazda RX4"
##   },
##   {
##     "mpg": 21,
##     "cyl": 6,
##     "disp": 160,
##     "hp": 110,
##     "drat": 3.9,
##     "wt": 2.875,
##     "qsec": 17.02,
##     "vs": 0,
##     "am": 1,
##     "gear": 4,
##     "carb": 4,
##     "_row": "Mazda RX4 Wag"
##   },
##   {
##     "mpg": 22.8,
##     "cyl": 4,
##     "disp": 108,
##     "hp": 93,
##     "drat": 3.85,
##     "wt": 2.32,
##     "qsec": 18.61,
##     "vs": 1,
##     "am": 1,
##     "gear": 4,
##     "carb": 1,
##     "_row": "Datsun 710"
##   },
##   {
##     "mpg": 21.4,
##     "cyl": 6,
##     "disp": 258,
##     "hp": 110,
##     "drat": 3.08,
##     "wt": 3.215,
##     "qsec": 19.44,
##     "vs": 1,
```

```
##     "am": 0,
##     "gear": 3,
##     "carb": 1,
##     "_row": "Hornet 4 Drive"
##   },
##   {
##     "mpg": 18.7,
##     "cyl": 8,
##     "disp": 360,
##     "hp": 175,
##     "drat": 3.15,
##     "wt": 3.44,
##     "qsec": 17.02,
##     "vs": 0,
##     "am": 0,
##     "gear": 3,
##     "carb": 2,
##     "_row": "Hornet Sportabout"
##   },
##   {
##     "mpg": 18.1,
##     "cyl": 6,
##     "disp": 225,
##     "hp": 105,
##     "drat": 2.76,
##     "wt": 3.46,
##     "qsec": 20.22,
##     "vs": 1,
##     "am": 0,
##     "gear": 3,
##     "carb": 1,
##     "_row": "Valiant"
##   }
## ]
```

We can write this JSON object into a JSON file with the `write_json` function.

```
write_json(mtcars_JSON, "mtcars.json")
```

JSON objects can also be converted to R dataframes which is more convenient to use in this environment. We can take the JSON file we just saved and convert it to a dataframe with the `fromJSON` function.

```
# Read JSON file
mtcars2_JSON <- read_json("mtcars.json", simplifyVector = TRUE)
mtcars2_JSON
```

```
## [1] "[\n  {\n    \"mpg\": 21,\n    \"cyl\": 6,\n    \"disp\": 160,\n    \"hp\": 110,\n    \"drat\": 3.9,\n    \
"wt\": 2.62,\n    \"qsec\": 16.46,\n    \"vs\": 0,\n    \"am\": 1,\n    \"gear\": 4,\n    \"carb\": 4,\n    \"_row
\": \"Mazda RX4\"\n  },\n  {\n    \"mpg\": 21,\n    \"cyl\": 6,\n    \"disp\": 160,\n    \"hp\": 110,\n    \"drat\
": 3.9,\n    \"wt\": 2.875,\n    \"qsec\": 17.02,\n    \"vs\": 0,\n    \"am\": 1,\n    \"gear\": 4,\n    \"carb\":
4,\n    \"_row\": \"Mazda RX4 Wag\"\n  },\n  {\n    \"mpg\": 22.8,\n    \"cyl\": 4,\n    \"disp\": 108,\n    \"hp\
": 93,\n    \"drat\": 3.85,\n    \"wt\": 2.32,\n    \"qsec\": 18.61,\n    \"vs\": 1,\n    \"am\": 1,\n    \"gear\"
: 4,\n    \"carb\": 1,\n    \"_row\": \"Datsun 710\"\n  },\n  {\n    \"mpg\": 21.4,\n    \"cyl\": 6,\n    \"disp\"
: 258,\n    \"hp\": 110,\n    \"drat\": 3.08,\n    \"wt\": 3.215,\n    \"qsec\": 19.44,\n    \"vs\": 1,\n    \"am\
": 0,\n    \"gear\": 3,\n    \"carb\": 1,\n    \"_row\": \"Hornet 4 Drive\"\n  },\n  {\n    \"mpg\": 18.7,\n    \"
cyl\": 8,\n    \"disp\": 360,\n    \"hp\": 175,\n    \"drat\": 3.15,\n    \"wt\": 3.44,\n    \"qsec\": 17.02,\n
\"vs\": 0,\n    \"am\": 0,\n    \"gear\": 3,\n    \"carb\": 2,\n    \"_row\": \"Hornet Sportabout\"\n  },\n  {\n
\"mpg\": 18.1,\n    \"cyl\": 6,\n    \"disp\": 225,\n    \"hp\": 105,\n    \"drat\": 2.76,\n    \"wt\": 3.46,\n
\"qsec\": 20.22,\n    \"vs\": 1,\n    \"am\": 0,\n    \"gear\": 3,\n    \"carb\": 1,\n    \"_row\": \"Valiant\"\n
}\n]"
```

```
# Convert to dataframe
mtcars2 <- fromJSON(mtcars2_JSON)
mtcars2
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Using the `yaml` library, we can execute the same methods for YAML files.

First, convert the existing R dataframe `iris` to a YAML format using the `as.yaml` function.

```
# Subset first 5 entries
iris <- head(iris)
iris
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
# Convert to yaml
iris_yml <- as.yaml(iris, column.major = FALSE)
iris_yml
```

```
## [1] "- Sepal.Length: 5.1\n  Sepal.Width: 3.5\n  Petal.Length: 1.4\n  Petal.Width: 0.2\n  Species: setosa\n- Sepal.Length: 4.9\n  Sepal.Width: 3.0\n  Petal.Length: 1.4\n  Petal.Width: 0.2\n  Species: setosa\n- Sepal.Length: 4.7\n  Sepal.Width: 3.2\n  Petal.Length: 1.3\n  Petal.Width: 0.2\n  Species: setosa\n- Sepal.Length: 4.6\n  Sepal.Width: 3.1\n  Petal.Length: 1.5\n  Petal.Width: 0.2\n  Species: setosa\n- Sepal.Length: 5.0\n  Sepal.Width: 3.6\n  Petal.Length: 1.4\n  Petal.Width: 0.2\n  Species: setosa\n- Sepal.Length: 5.4\n  Sepal.Width: 3.9\n  Petal.Length: 1.7\n  Petal.Width: 0.4\n  Species: setosa\n"
```

We can write the new YAML object to a file with the following code.

```
writeLines(iris_yml, "iris.yaml")
```

Now we can read the saved YAML file back into R and convert it back to a dataframe with the `yaml.load` function.

```
# Import yaml as list
iris2_yml <- yaml.load_file("iris.yaml")
iris2_yml
```

```
## [[1]]
## [[1]]$Sepal.Length
## [1] 5.1
##
## [[1]]$Sepal.Width
## [1] 3.5
##
## [[1]]$Petal.Length
## [1] 1.4
##
## [[1]]$Petal.Width
## [1] 0.2
##
## [[1]]$Species
## [1] "setosa"
##
##
## [[2]]
## [[2]]$Sepal.Length
## [1] 4.9
##
## [[2]]$Sepal.Width
## [1] 3
##
## [[2]]$Petal.Length
## [1] 1.4
##
## [[2]]$Petal.Width
## [1] 0.2
##
## [[2]]$Species
## [1] "setosa"
##
##
## [[3]]
## [[3]]$Sepal.Length
## [1] 4.7
##
## [[3]]$Sepal.Width
## [1] 3.2
##
## [[3]]$Petal.Length
## [1] 1.3
##
## [[3]]$Petal.Width
## [1] 0.2
##
## [[3]]$Species
## [1] "setosa"
##
##
## [[4]]
## [[4]]$Sepal.Length
```

```
## [1] 4.6
##
## [[4]]$Sepal.Width
## [1] 3.1
##
## [[4]]$Petal.Length
## [1] 1.5
##
## [[4]]$Petal.Width
## [1] 0.2
##
## [[4]]$Species
## [1] "setosa"
##
##
## [[5]]
## [[5]]$Sepal.Length
## [1] 5
##
## [[5]]$Sepal.Width
## [1] 3.6
##
## [[5]]$Petal.Length
## [1] 1.4
##
## [[5]]$Petal.Width
## [1] 0.2
##
## [[5]]$Species
## [1] "setosa"
##
##
## [[6]]
## [[6]]$Sepal.Length
## [1] 5.4
##
## [[6]]$Sepal.Width
## [1] 3.9
##
## [[6]]$Petal.Length
## [1] 1.7
##
## [[6]]$Petal.Width
## [1] 0.4
##
## [[6]]$Species
## [1] "setosa"
```

The `yaml` library loads the file as a named list so we have to use `ldply` from the `plyr` package to convert that to a dataframe.

```
library(plyr)
# Convert to dataframe
iris2 <- ldply(iris2_yml, data.frame, stringsAsFactors=FALSE)
iris2
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

By importing these files into R as dataframes, you can manipulate the data the same way you would with any other dataframe in R.

Conclusion

XML, JSON, and YAML files are all formats that statisticians may encounter when retrieving or scraping data from an external source. Each type has different benefits over the others. When choosing which type of format to use, it is important to keep in mind what features you need and what your priorities are. Generally speaking, XML and JSON are the two primary options to choose from. If you prioritize simplicity, JSON will probably be the better option. However, if you need additional features, such as storing metadata or data querying, XML may be more effective.

References

- https://www.w3schools.com/js/js_json_intro.asp
- http://www.json.org/xml.html
- https://cran.r-project.org/web/packages/jsonlite/jsonlite.pdf
- http://www.simonstl.com/articles/whyxml.htm
- http://zevross.com/blog/2015/02/12/using-r-to-download-and-parse-json-an-example-using-data-from-an-open-data-portal/
- https://www.r-bloggers.com/better-handling-of-json-data-in-r/

- https://cran.r-project.org/web/packages/yaml/yaml.pdf