JSON and Graph model

Lecturer: Jiaheng Lu

Autumn 2016
Review and outline

• History of databases (Why we use databases)

• Data models
  • Three features of data models
  • Relational model
  • Semi-structure model
    – XML model
    – JSON model
  • Graph model
Trends in XML and JSON usage

Based on directory of 11,000 web APIs listed at Programmable Web, December 2013

Wow! I better have expertise in both XML and JSON
Example of XML-formatted data

The below XML document contains data about a book: its title, authors, date of publication, and publisher.

```xml
<Book>
  <Title>Parsing Techniques</Title>
  <Authors>
    <Author>Dick Grune</Author>
    <Author>Ceriel J.H. Jacobs</Author>
  </Authors>
  <Date>2007</Date>
  <Publisher>Springer</Publisher>
</Book>
```
Same data, JSON-formatted

```json
{
    "Book": {
        "Title": "Parsing Techniques",
        "Authors": [ "Dick Grune", "Ceriel J.H. Jacobs" ],
        "Date": "2007",
        "Publisher": "Springer"
    }
}
```
XML and JSON, side-by-side

<Book>
  <Title>Parsing Techniques</Title>
  <Authors>
    <Author>Dick Grune</Author>
    <Author>Ceriel J.H. Jacobs</Author>
  </Authors>
  <Date>2007</Date>
  <Publisher>Springer</Publisher>
</Book>

{ "Book": { "Title": "Parsing Techniques", "Authors": [ "Dick Grune", "Ceriel J.H. Jacobs" ], "Date": "2007", "Publisher": "Springer" } }
An XML document is a tree

- Book
  - Title: Parsing Techniques
  - Authors: Dick Grune, Ceriel J.H. Jacobs
  - Date: 2007
  - Publisher: Springer
A JSON Object is a tree

Book

Title

Parsing Techniques

Authors

[“Dick Grune”, “Ceriel J.H. Jacobs”]

Date

2007

Publisher

Springer
JSON

- JavaScript Object Notation
- Textual
- Light-weight.
- Easy to parse.
Values

- Strings
- Numbers
- Objects
- Arrays
- true
- false
- null
Value

string
number
object
array
true
false
null
This is a legal JSON instance

42
so is this

"Hello World"
and so is this

true
and this

[ true, null, 12, "ABC" ]
Whitespace is irrelevant

```json
{
    "name": "John Doe",
    "age": 30,
    "married": true
}
```

equivalent

```json
{"name":"John Doe","age":30,"married":true}
```
No multiline strings

JSON does not allow multiline strings.

Legal:
```
{
    "comment": "This is a very, very long comment"
}
```

Not legal:
```
{
    "comment": "This is a very, very long comment"
}
```
No multiline strings (cont.)

Legal:

```json
{
    "comment": "This is a very, \n very long comment"
}
```

Just 2 symbols
Objects in values

• Objects are unordered containers of key/value pairs
• Objects are wrapped in {   } (curly brackets)
• , (comma) separates key/value pairs
• : (colon) separates keys and values
• Keys are strings
• Values are JSON values
Object

{ 
  string : value 
}

/
Object

```json
{
    "format": {
        "type": "rect",
        "width": 1920,
        "height": 1080,
        "interlace": false,
        "framerate": 24
    }
}
```
No duplicate keys

- You should consider JSON objects as containing key/value pairs.
- Just as in a database the primary keys must be unique, so too in a JSON object the keys must be unique.
- This JSON object has duplicate keys:

  ```json
  { "Title": "A story by Mark Twain",
    "Title": "The Adventures of Huckleberry Finn"
  }
  ```
Array in values

- Arrays are ordered sequences of values
- Arrays are wrapped in `[ ]` (square bracket)
- `,` (comma) separates values
Array

["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"]

[
  [0, -1, 0],
  [1, 0, 0],
  [0, 0, 1]
]
Using JSON you can define arbitrarily complex structures

```
{
  "Book": {
    "Title": "Parsing Techniques",
    "Authors": [ "Dick Grune", "Ceriel J.H. Jacobs" ]
  }
}
```

```
{
  "Book": {
    "Title": "Parsing Techniques",
    "Authors": [
      {"name": "Dick Grune", "university": "Vrije Universiteit"},
      {"name": "Ceriel J.H. Jacobs", "university": "Vrije Universiteit"}
    ]
  }
}
```
7 simple JSON components, assemble to generate unlimited complexity

null

object

array

string

number

true

false
JSON Schema is used to define a valid JSON document

```json
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object",
    "properties": {
        "Book": {
            "type": "object",
            "properties": {
                "Title": {"type": "string"},
                "Authors": {"type": "array", "minItems": 1, "maxItems": 5, "items": { "type": "string" }},
                "Date": {"type": "string"},
                "Publisher": {"type": "string", "enum": ["Springer", "MIT Press", "Harvard Press"]}
            },
            "required": ["Title", "Authors", "Date"],
        },
        "required": ["Book"],
        "additionalProperties": false
    }
}
```
"$schema" keyword

{
    "$schema": "http://json-schema.org/draft-04/schema#",
    ...
}

"$schema" keyword

{  
  "$schema": "http://json-schema.org/draft-04/schema#",
  ...
} 

The $schema keyword says: This object is a JSON schema, conforming to the schema at http://json-schema.org/draft-04/schema#. 
Order of keywords is irrelevant

equivalent!

{  
"$schema": "http://json-schema.org/draft-04/schema#",  
"type": "boolean" 
}

{  
  "type": "boolean",  
  "$schema": "http://json-schema.org/draft-04/schema#" 
}
Number type

{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "number"
}

This schema constrains instances to contain only a number (integer, decimal, number with exponent).
Only numeric instances are valid

```json
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "number"
}
```

12.3

JSON Schema Validator

Valid!
Examples of numeric values

12  99.1  12.123e3  12.123E3

12.123 × 10³
"enum" keyword

• The value of enum is an array.
• The items in the array is a list of values that instances may have.
• The following schema says that the only allowable values in instances are the numbers: 2, 4, 6, 8, 10.

```json
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "number",
    "enum": [2, 4, 6, 8, 10]
}
```
"minimum" and "maximum" keywords

- A range of values can be specified using the "minimum" and "maximum" keywords.
- The following schema constrains instances to numbers in the range 0-100, inclusive.

```json
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "number",
    "minimum": 0,
    "maximum": 100
}
```
"multipleOf" keyword

- Instances can be constrained to a number that is a multiple of a number.
- The following schema says that a JSON instance must be a number 0-100, inclusive, and must be a multiple of 2.

```json
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "number",
    "minimum": 0,
    "maximum": 100,
    "multipleOf": 2
}
```

Here are four schema-valid values: 0  2  4  100
Integer type

```json
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "integer"
}
```

This schema constrains instances to contain only an integer.

Here are two schema-valid values:   -900   129
String type

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "string"
}
```

This schema constrains instances to contain only a string.
"maxLength" keyword

The maximum length of a string is constrained using the "maxLength" keyword.

```json
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "string",
  "maxLength": 20
}

Here is a schema-valid value: "Hello World"
```
Value of "maxLength"

```json
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "string",
    "maxLength": 5
}
```

Must be an integer, $\geq 0$
"minLength" keyword

- The "minLength" keyword is used to specify the shortest string length allowed.
- The value of "minLength" must be an integer, greater than or equal to 0.
- The default value is 0.

```json
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "string",
    "minLength": 5,
    "maxLength": 20
}
```
The set of characters that can be used in a string can be constrained using the "pattern" keyword, whose value is a regular expression.

The following schema constrains the set of characters to the lower- and upper-case letters of the English alphabet, plus the space character.

```json
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "string",
    "maxLength": 20,
    "pattern": "^[a-zA-Z ]*$"
}
```
"pattern" value is a regular expression

- The value of "pattern" is a regular expression.
- The regular expressions are not implicitly anchored so you must use the start and end anchors (^…$).

```json
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "string",
    "maxLength": 20,
    "pattern": "^[a-zA-Z ]*$"
}
```

This symbol indicates that a string must **end** with the letters of the alphabet plus space.

This symbol indicates that a string must **start** with the letters of the alphabet plus space.
The string in a JSON instance cannot have a length greater than 20 characters and the characters must be a-z, A-Z, or space.
Components of regular expressions

<table>
<thead>
<tr>
<th>Basic pattern</th>
<th>Matching string</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>The character x</td>
</tr>
<tr>
<td>.</td>
<td>Any character, except newline</td>
</tr>
<tr>
<td>[xyz ...]</td>
<td>Any of the characters, x, y, z, ...</td>
</tr>
</tbody>
</table>

Repetition operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R?</td>
<td>R is optional (zero or one occurrence)</td>
</tr>
<tr>
<td>R*</td>
<td>Zero or more occurrences of R</td>
</tr>
<tr>
<td>R+</td>
<td>One or more occurrences of R</td>
</tr>
</tbody>
</table>

Compositional operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R₁R₂</td>
<td>An R₁ followed by an R₂</td>
</tr>
<tr>
<td>R₁</td>
<td>R₂</td>
</tr>
</tbody>
</table>

Acknowledgement: This table comes from *Modern Compiler Design* by Grune et al, p62.
Use a JSON Schema validator from a Java program

Want to validate a JSON instance against a JSON Schema from a Java program? Download:

   json-schema-validator-2.2.5-lib.jar

from:

   https://bintray.com/fge/maven/json-schema-validator/view

and then read the documentation on the Java API.
• Watch a video on JSON tutorial

• https://www.youtube.com/watch?v=_NFkzw6oFtQ
Outline

• History of databases

• Data models
  • Relational model
  • Semi-structure model
    – XML model
    – JSON model
  • Graph model
Graph data model

Nodes table:
1. John
2. Mary
3. Anna

Edge table:
John, Mary, Friend
Mary, Anna, Sister
Graphs from the Real World

- Biological Network
- Ecological Network
- Social Network
- Chemical Network
- Program Flow
- Web Graph
Some operations on graphs

- Graph union and graph intersection

![Graph union and graph intersection](image)
Some operations on graphs

- Graph union and graph intersection
Some operations on graphs

- Graph union and graph intersection
- GetNeighbour

Neighbour of Node 3 is Node 1
Some operations on graphs

- Graph union and graph intersection
- GetNeighbour
- Community detection

FIG. 1  A simple graph with three communities, enclosed by the dashed circles. Reprinted figure with permission from Ref. (Fortunato and Castellano, 2009). ©2009 by Springer.
Some operations on graphs

- Graph union and graph intersection
- GetNeighbour
- Community detection
Defining Communities

• Intuition: There are more edges inside a community than edges connected with the rest of the graph.

FIG. 1 A simple graph with three communities, enclosed by the dashed circles. Reprinted figure with permission from Ref. (Fortunato and Castellano, 2009). ©2009 by Springer.
General Challenges for community detection

- Many clustering problems are NP-hard. Even polynomial time approaches may be too expensive
  - Call for scalable solutions
- Concepts of “cluster”, “community” are not quantitatively well defined
Graph data model

First watch a video on graph modelling

https://www.youtube.com/watch?v=NH6WoJHN4UA
Summary

- JSON is a widely used semi-structure model like XML
- Graph model is an important data model in NoSQL databases