JSON: The Basics

Overview

- What is JSON?
- Comparisons with XML
- Syntax
- Data Types
- Usage
- Live Examples

What is JSON?

JSON is…

- A lightweight text based data-interchange format
- Completely language independent
- Based on a subset of the JavaScript Programming Language
- Easy to understand, manipulate and generate

JSON is NOT…

- Overly Complex
- A “document” format
- A markup language
- A programming language

Why use JSON?

- Straightforward syntax
- Easy to create and manipulate
- Can be natively parsed in JavaScript using **eval()**
- Supported by all major JavaScript frameworks
- Supported by most backend technologies

JSON vs. XML

Much Like XML

- Plain text formats
- “Self-describing“ (human readable)
- Hierarchical (Values can contain lists of objects or values)

Not Like XML

- Lighter and faster than XML
- JSON uses typed objects. All XML values are type-less strings and must be parsed at runtime.
- Less syntax, no semantics
- Properties are immediately accessible to JavaScript code

Knocks against JSON

- Lack of namespaces
- No inherit validation (XML has DTD and templates, but there is JSONlint)
- Not extensible
- It's basically just ***not*** XML

JSON Vs. XML

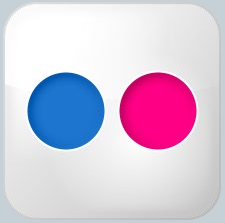
JSON	XML
Data-oriented	Document-oriented
Supports arrays (to define data objects) JSON types: string, number, array, Boolean	Data should be string only
JSON is simple to read and write	XML has a more complex format
JSON is less secure than XML	XML is more secure
For AJAX applications, JSON is faster and easier than XML	XML is comparatively slower owing to it consuming more memory

How & When to use JSON

- Transfer data to and from a server
- Perform asynchronous data calls without requiring a page refresh
- Working with data stores
- Compile and save form or user data for local storage

Where is JSON used today?

- Anywhere and everywhere!



And many, many more!

Syntax

JSON Object Syntax

- Unordered sets of name/value pairs
- Begins with **{** (left brace)
- Ends with **}** (right brace)
- Each name is followed by **:** (colon)
- Name/value pairs are separated by **,** (comma)

JSON Example

```
var employeeData = {
  "employee_id": 1234567,
  "name": “Freyan",
  "hire_date": "1/1/2021",
  "location": "Norwalk, CT",
  "consultant": false
};
```

Arrays in JSON

- An ordered collection of values
- Begins with [(left bracket)
- Ends with] (right bracket)
- Name/value pairs are separated by , (comma)

JSON Array Example

```
var employeeData = {
  "employee_id": 1236937,
  "name": “khyati",
  "hire_date": "1/1/2022",
  "location": "Norwalk, CT",
  "consultant": false,
   "random_nums": [ 24,65,12,94 ]
};
```

Data Types

Data Types: Strings

- Sequence of 0 or more Unicode characters
- Wrapped in "double quotes“
- Backslash escapement

Data Types: Numbers

- Integer
- Real
- Scientific
- No octal or hex
- No NaN or Infinity – Use **null** instead.

Data Types: Booleans & Null

- Booleans: true or false
- Null: A value that specifies nothing or no value.

Data Types: Objects & Arrays

- Objects: Unordered key/value pairs wrapped in { }
- Arrays: Ordered key/value pairs wrapped in [ ]

Simple Example

Simple Demo

- Build a JSON data object in code
- Display raw output
- Display formatted output
- Manipulate via form input

JSON objects

- JSON Array of Numbers
- [12, 34, 56, 43, 95]

- JSON Array of Booleans
- [true, true, false, false, true]

- JSON Array of Objects
- {"employees":[
- {"name":"Ram", "email":"ram@gmail.com", "age":23},
- {"name":"Shyam", "email":"shyam23@gmail.com", "age":28},
- {"name":"John", "email":"john@gmail.com", "age":33},
- {"name":"Bob", "email":"bob32@gmail.com", "age":41}
-]}

JSON Multidimensional Array

- [
- ["a", "b", "c"],
- ["m", "n", "o"],
- ["x", "y", "z"]
-]

Java JSON

- The **json.simple** library allows us to read and write JSON data in Java.
- we can encode and decode JSON object in java using json.simple library.
- The org.json.simple package contains important classes for JSON API.
- JSONValue
- JSONObject
- JSONArray
- JsonString
- JsonNumber

Encoding JSON in Java

- import org.json.simple.JSONObject;
- public class JsonExample1{
- public static void main(String args[]){
- JSONObject obj=new JSONObject();
- obj.put("name","sonoo");
- obj.put("age",new Integer(27));
- obj.put("salary",new Double(600000));
- System.out.print(obj);
- }}

Java JSON Encode using List

```
import java.util.ArrayList;
import java.util.List;
import org.json.simple.JSONValue;
public class JsonExample1{
public static void main(String args[]){
  List arr = new ArrayList();
  arr.add("sonoo");
  arr.add(new Integer(27));
  arr.add(new Double(600000));
  String jsonText = JSONValue.toJSONString(arr);
  System.out.print(jsonText);
}}
```

Java JSON Decode

```
import org.json.simple.JSONObject;
import org.json.simple.JSONValue;
public class JsonDecodeExample1 {
public static void main(String[] args) {
    String s="{\"name\":\"sonoo\",\"salary\":600000.0,\"age\":27}";
    Object obj=JSONValue.parse(s);
    JSONObject jsonObject = (JSONObject) obj;

    String name = (String) jsonObject.get("name");
    double salary = (Double) jsonObject.get("salary");
    long age = (Long) jsonObject.get("age");
    System.out.println(name+" "+salary+" "+age);
}
}
```