Desenvolvimento de Software para Persistência

JSON
JavaScript Object Notation

Prof. Regis Pires Magalhães
regismagalhaes@ufc.br

UFC

Serialização

- Processo de capturar uma estrutura de dados de maneira a permitir que ela seja armazenada, transmitida e reconstruída novamente em uma estrutura de dados no futuro.

Tipos Javascript

- Tipos primitivos e não primitivos
  - Primitivos: números, strings, booleanos, undefined, null.
  - Literais
  - typeof
    - tipo do dado.
  - instanceof
    - tipo object de uma instância.
- Notações
  - Ponto
  - Colchetes

instance of

```
function Car(make, model, year) {
  this.make = make;
  this.model = model;
  this.year = year;
}

var mycar = new Car('Honda', 'Accord', 1998);
var a = mycar instanceof Car;    // returns true
var b = mycar instanceof Object; // returns true
```

instance of

```
// defining constructors
function C() {}
function D() {}

var o = new C();

// true, because: Object.getPrototypeOf(o) === C.prototype
o instanceof C;

// false, because D.prototype is nowhere in o's prototype chain
o instanceof D;

o instanceof Object; // true, because:
C.prototype instanceof Object // true
```

Objetos Javascript

- Array - lista ordenada de valores. Entre []
- Objeto - coleção de pares chave/valor.
- Coleções não ordenadas entre {}

JSON

- Formato leve usado para armazenamento e troca de dados.
- Auto-descritivo e de fácil entendimento.
- Independente de linguagem.
  - Usa sintaxe Javascript, mas é um formato texto que pode ser lido e usado por qualquer linguagem.
- Tipos:
  - Array, Boolean, null, Number, Object, String.

Chaves

- Chaves JSON devem ser strings entre aspas duplas.

```
{ "name":"John" }
```

- Em JavaScript chaves podem ser strings, números ou nomes de identificadores.

```
{ name:"John" }
```

stringify

- JSON.stringify(value[, [replacer, space]]);

```
var myObj = { "name":"John", "age":31, "city":"New York"
};
var myJSON = JSON.stringify(myObj);
console.log(myJSON);
```

toJSON

- usado para serialização quando stringify for usado.

```
var obj = {
  foo: 'foo',
  toJSON: function() {
    return 'bar';
  }
};
JSON.stringify(obj);          // '"bar"'
JSON.stringify({ x: obj }); // '{"x":"bar"}'
```

parse

- converte json para objeto JS nativo.

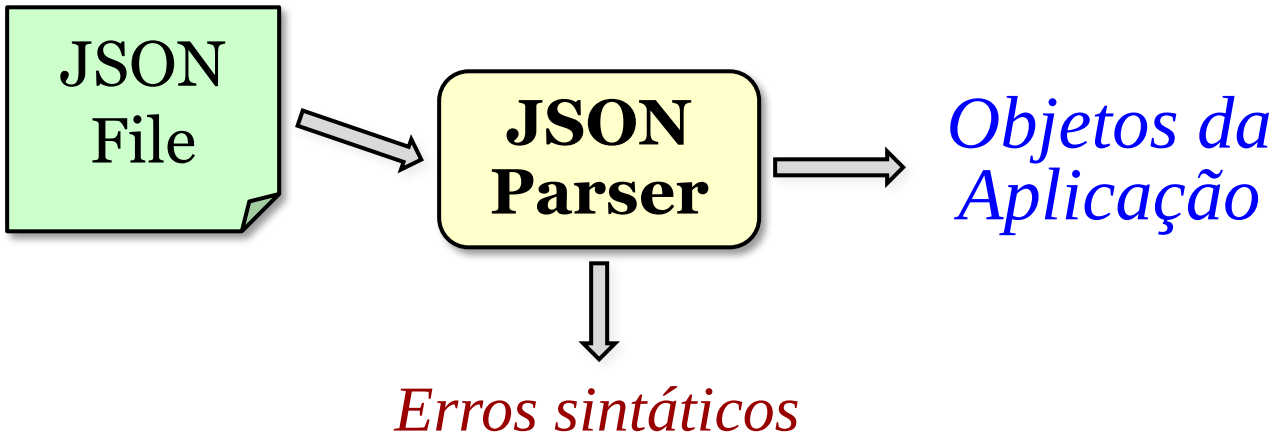
```
var myJSON = '{ "name":"John", "age":31, "city":"New York" }';
var myObj = JSON.parse(myJSON);
document.getElementById("demo").innerHTML = myObj.name;
```

Armazenando e recuperando dados

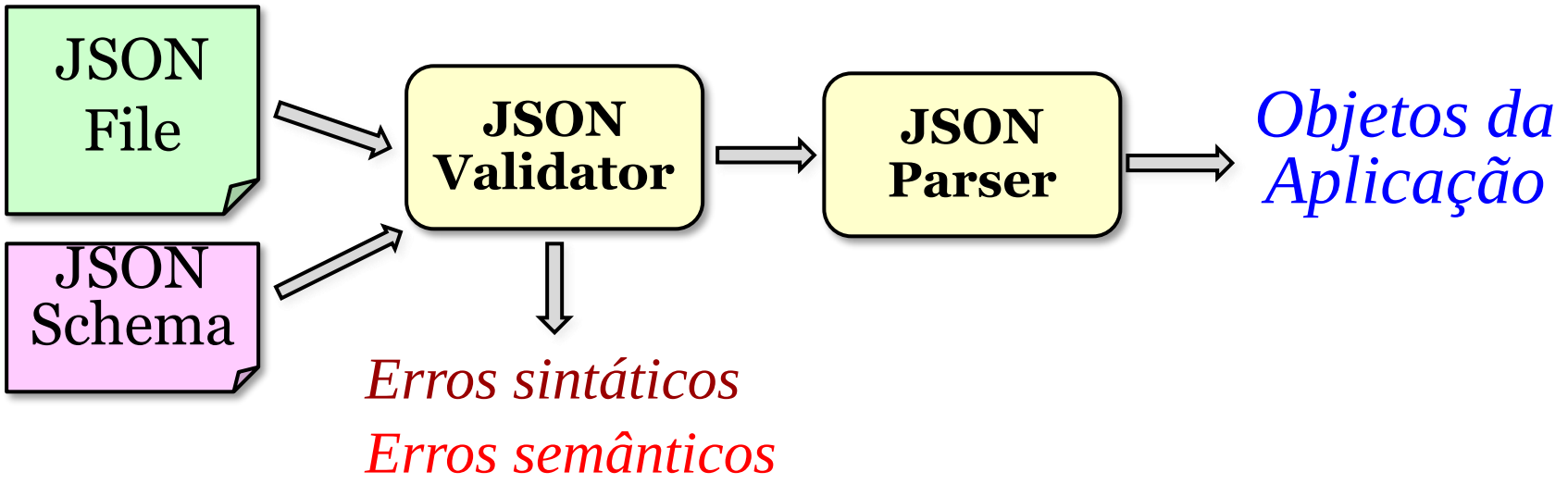
```
myObj = { "name":"John", "age":31, "city":"New York" };
myJSON = JSON.stringify(myObj);
localStorage.setItem("testJSON", myJSON);
```

```
text = localStorage.getItem("testJSON");
obj = JSON.parse(text);
document.getElementById("demo").innerHTML = obj.name;
```

Validação sintática



Validação sintática + semântica



JSON Schema

http://json-schema.org/

```
{
  "type": "object",
  "properties": {
    "first_name": { "type": "string" },
    "last_name": { "type": "string" },
    "birthday": { "type": "string", "format": "date-time" },
    "address": {
      "type": "object",
      "properties": {
        "street_address": { "type": "string" },
        "city": { "type": "string" },
        "state": { "type": "string" },
        "country": { "type" : "string" }
      }
    }
  }
}
```

JSON Schema

http://json-schema.org/

```
{
  "name": "George Washington",
  "birthday": "February 22, 1732",
  "address": "Mount Vernon, Virginia, United States"
}
```

```
{
  "first_name": "George",
  "last_name": "Washington",
  "birthday": "22-02-1732",
  "address": {
    "street_address": "3200 Mount Vernon Memorial Highway",
    "city": "Mount Vernon",
    "state": "Virginia",
    "country": "United States"
  }
}
```

JSON Schema

Java

- json-schema-validator *supports draft 4* (LGPLv3)
- json-schema (implementation based on the org.json API) *supports draft 4, draft 6* (Apache License 2.0)
- json-schema-validator *supports draft 4* (Apache License 2.0)

draft-06 meta-schemas - published on 2017-04-15.

JSON Schema Core	defines the basic foundation of JSON Schema
JSON Schema Validation	defines the validation keywords of JSON Schema
JSON Hyper-Schema	defines the hyper-media keywords of JSON Schema

JSON Schema

http://json-schema.org/

```
{
    "$schema": "http://json-schema.org/draft-06/schema#",
    "title": "Product set",
    "type": "array",
    "items": {
        "title": "Product",
        "type": "object",
        "properties": {
            "id": {
                "description": "The unique identifier for a product",
                "type": "number"
            },
            "name": {
                "type": "string"
            },
            "price": {
                "type": "number",
                "exclusiveMinimum": 0
            },
            "tags": {
                "type": "array",
                "items": {
                    "type": "string"
                },
                "minItems": 1,
                "uniqueItems": true
            },
            ...
```

```
...
"dimensions": {
                "type": "object",
                "properties": {
                    "length": {"type": "number"},
                    "width": {"type": "number"},
                    "height": {"type": "number"}
                },
                "required": ["length", "width", "height"]
            },
            "warehouseLocation": {
                "description": "Coordinates of the warehouse with the product",
                "$ref": "http://json-schema.org/geo"
            }
        },
        "required": ["id", "name", "price"]
    }
}
```

JSON Schema

http://json-schema.org/


```
[
    {
        "id": 2,
        "name": "An ice sculpture",
        "price": 12.50,
        "tags": ["cold", "ice"],
        "dimensions": {
            "length": 7.0,
            "width": 12.0,
            "height": 9.5
        },
        "warehouseLocation": {
            "latitude": -78.75,
            "longitude": 20.4
        }
    },
    {
        "id": 3,
        "name": "A blue mouse",
        "price": 25.50,
        "dimensions": {
            "length": 3.1,
            "width": 1.0,
            "height": 1.0
        },
        "warehouseLocation": {
            "latitude": 54.4,
            "longitude": -32.7
        }
    }
]
```

http://json-schema.org/

JSON Schema

- JSON Schema é um vocabulário que permite anotar e validar documentos JSON.

```
{ "type": "string" }
```

```
"I'm a string"
```

```
42
```

Serialização de JSON em Java

- Principais APIs para manipulação de JSON em Java:
  - Jackson
    - https://github.com/FasterXML/jackson-databind
    - Usado por padrão no SpringFramework.
    - Usado por padrão em implementações da especificação JAX-RS (Jersey, Apache CXF, RESTEasy, Restlet).
  - gson (by Google)
- Jackson vs Gson
  - http://www.baeldung.com/jackson-vs-gson

Serialização de JSON em Java

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>br.ufc</groupId>
  <artifactId>json-java</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <dependencies>
      <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.9.1</version>
      </dependency>
  </dependencies>
</project>
```

Serialização de JSON em Java

```
public class Contato {
    private int id;
    private String nome;
    private String email;

    public Contato() {}

    public Contato(int id, String nome, String email) {
        super();
        this.id = id;
        this.nome = nome;
        this.email = email;
    }

     // Getters e Setters

    @Override
    public String toString() {
        return this.id + "," + this.nome + "," + this.email;
    }
}
```

Serialização de JSON em Java

```
import java.io.File;
import java.util.ArrayList;
import java.util.List;

import com.fasterxml.jackson.databind.ObjectMapper;

public class Serializa {

	public static void main(String[] args) throws Exception {
		List<Contato> l = new ArrayList<Contato>();
		l.add(new Contato(1, "João", "joao@gmail.com"));
		l.add(new Contato(1, "Maria", "maria@gmail.com"));
		l.add(new Contato(1, "José", "jose@gmail.com"));

		ObjectMapper mapper = new ObjectMapper();
		String homeDir = System.getProperty("user.home");
		mapper.writeValue(new File(homeDir + "/result.json"), l);

	}

}
```

Deserialização de JSON em Java

```
import java.io.File;
import java.util.List;

import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.ObjectMapper;

public class Deserializa {

    public static void main(String[] args) throws Exception {
        ObjectMapper mapper = new ObjectMapper();

        String homeDir = System.getProperty("user.home");
        TypeReference<List<Contato>> mapType = new TypeReference<List<Contato>>() {};
        List<Contato> l = mapper.readValue(
                                    new File(homeDir + "/result.json"), mapType);
        for (Contato c : l) {
            System.out.println(c);
        }

    }

}
```

Referências

- SMITH, Ben. JSON básico: conheça o formato de dados preferido da web. São Paulo: Novatec, 2015.
- https://en.wikipedia.org/wiki/JSON
- https://www.w3schools.com/js/js_json_intro.asp
- https://spacetelescope.github.io/understanding-json-schema/
- http://json-schema.org/
- https://github.com/FasterXML/jackson-databind
- http://www.mkyong.com/java/jackson-2-convert-java-object-to-from-json/
- Jackson vs Gson (com exemplos)
  - http://www.baeldung.com/jackson-vs-gson

Obrigado!

Dúvidas, comentários, sugestões?

Regis Pires Magalhães
regispires@lia.ufc.br
@regispires

UFC