

## **Serviços e Interoperabilidade de Sistemas**

### **JSON (e XML)**

#### **Objetivos:**

- Introdução ao JSON
- Necessidade de representação de dados seguindo um formato standard
- O formato JSON
- Manipulação de JSON em JAVA
- Manipulação de JSON em PHP
- Exercícios
- (Introdução ao XML)

**Duração: 1 aula**

©2020-2021: {bruno.madeira, mario.fernandes, nuno.costa, nuno.simoese}@ipleiria.pt

JSON significa *JavaScript Object Notation* e não é mais do que um formato textual, “leve”, para empacotar e trocar dados entre diferentes aplicações ou sistemas.

Mas, por que razão devemos empacotar os dados num formato bem conhecido antes de os enviar?

Antes de mais devemos considerar que as máquinas não são todas iguais. Por exemplo, as máquinas, por norma, armazenam o tipo inteiro com o mesmo número de bits usado pelo processador (64 bits, 32 bits, etc.). Isto quer dizer que, se uma máquina de 64 bits enviar um número inteiro a uma máquina de 32 bits, provavelmente, o número interpretado na máquina de 32 bits será diferente (“vai cortar o número a meio, em termos de bits”). Para além do número de bits usado para armazenar tipos de dados, existem também diferenças ao nível da ordem pela qual os bytes são armazenados na memória RAM (*little-endian* vs *big-endian*). Estes exemplos mostram-nos que o uso do formato binário nos pode trazer vários problemas em termos de passar informação de uma máquina para outra. Então, o formato textual (apesar de usar mais bits no armazenamento) poderá ter menos problemas em termos de troca de informação já que um byte<sup>1</sup>(letra, número, símbolo) tem o mesmo tamanho em todas as máquinas.

Em resumo, podemos assumir que, quando é usado o formato binário (por exemplo um ficheiro binário) esse formato está associado a determinada aplicação e só a empresa que desenvolveu essa aplicação conhece o formato dos dados (não é nada interoperável!).

Por outro lado, o uso do formato textual, por si só, também não resolve todas as questões da interoperabilidade, pelo menos na semântica, ou seja, se tivermos um ficheiro com:

Labrador;12;33.56

não é fácil inferir o significado do 12 e do 33.56. Mais, se necessitarmos usar o ; no primeiro campo, vai haver problemas na interpretação, já que aparentemente passamos a ter 4 campos. É aqui que entra a meta informação.

A meta informação é informação que é adicionada à informação de modo a caracterizar a mesma.

Por exemplo:

<raça>Labrador</raça><idade>12</idade><peso>33.56</peso>

---

<sup>1</sup> Apesar de determinados sistemas operativos e linguagens poderem usar mais do que um byte para armazenar cada letra, número ou símbolo (utf-8, utf-16, etc.)

No exemplo são visíveis “etiquetas” que caracterizam a informação relacionada. Agora já sabemos o significado de cada um dos campos (meta informação) e, para além disso, já podemos usar qualquer caractere na descrição da raça do cão (;). Contudo, também é visível que agora o ficheiro passou a ter (+/-) o dobro do tamanho.

## O formato JSON

O formato JSON privilegia a economia de memória na representação dos dados, ou seja, as etiquetas (meta informação) usadas foram definidas para usarem o mínimo de caracteres, quando comparado com o XML, por exemplo.

O formato JSON é baseado em pares de **chave:valor**, envolvidos por chavetas (definem um registo), como mostra o exemplo seguinte:

```
{ "raça": "labrador", "idade": 12, "peso": 33.56 }
```

Se for necessário registar vários cães, a criação de múltiplos registos não é a melhor solução, dado que para processar essa informação (no destino) obrigamos à leitura e filtragem de registos.

```
{ "raça": "labrador", "idade": 12, "peso": 33.56 }
```

```
{ "raça": "beagle", "idade": 0, "peso": 23.50 }
```

```
{ "raça": "boxer", "idade": 2, "peso": 6.30 }
```

Em vez disso, a solução é usar a notação de *array* (vetor) em JSON, ou seja:

```
{ "cães": [ { "raça": "labrador", "idade": 12, "peso": 33.56 },  
            { "raça": "beagle", "idade": 10, "peso": 23.50 },  
            { "raça": "boxer", "idade": 2, "peso": 6.30 }  
          ]  
}
```

A tabela seguinte [java2s.com] apresenta os tipos suportados pelo JSON:

Tipo	Exemplo
Número	{ "peso" : 12.1 }
String	{ "nome" : "Ola" }
Boolean	{ "valido": true/false }
Vetor	[ "aaa", "bbb", "ccc", ].
Objeto	{ "país" : "PT", "Naturalidade": "Leiria", ..... }
Null	Vazio

### Manipulação de JSON em Java (Netbeans)

O Java não oferece forma nativa para manipulação de JSON. Assim sendo vamos utilizar “Jackson Databind” (<http://www.java2s.com/Code/Jar/j/Downloadjacksonall190jar.htm>). É necessário fazer download do “.JAR” e adicionar a “Libaries” do projecto (carregar com o botão direito e escolher “add JAR”).

A classe que faz a manipulação do JSON deve importar:

```
import org.codehaus.jackson.map.ObjectMapper;
```

Exemplos:

Imagine-se que pretendemos processar a seguinte informação JSON armazenada numa *string*:

```
{"membro" : true, "nome" : "Joao", "idade" : 18}
```

Nota: o nome das propriedades na string JSON deve ser em minúsculas para que o *parser* as interprete corretamente.

Em primeiro lugar teremos que definir uma classe JAVA que se adeque aos dados JSON:

```
public class Utilizador {  
  
    private boolean membro;  
    private String Nome;  
    private int Idade;  
  
    public boolean isMembro() {  
        return membro;  
    }  
  
    public void setMembro(boolean membro) {  
        this.membro = membro;  
    }  
  
    public String getNome() {  
        return Nome;  
    }  
  
    public void setNome(String Nome) {  
        this.Nome = Nome;  
    }  
  
    public int getIdade() {  
        return Idade;  
    }  
  
    public void setIdade(int Idade) {  
        this.Idade = Idade;  
    }  
  
}
```

Nota: deve implementar os getter e setter como indicado para que o *parser* possa funcionar corretamente.

Por último usamos a classe `ObjectMapper` chamando o método `readValue`:

```
try{  
    ObjectMapper mapper = new ObjectMapper();  
  
    Utilizador user = mapper.readValue("{ \"membro\" : true, \"nome\" :  
    \"joao\", \"idade\" : 18}", Utilizador.class);  
  
    System.out.println(user.getNome());  
}  
catch (Exception ex)  
{  
    System.out.println(ex.getMessage());  
}
```

Por outro lado, para serializar um objeto para uma *string* JSON (afim de enviar essa *string* para outra aplicação ou para outro sistema) fazemos:

```
Utilizador user = new Utilizador();

user.setMembro(true);
user.setNome("Joana");
user.setIdade = 20;

String resultadoJSON = mapper.writeValueAsString(user);
System.out.println(resultadoJSON);
```

**De seguida apresenta-se um exemplo para serializar/desserializar vetores:**

```
public class Utilizador {

    private boolean membro;
    private String Nome;
    private int Idade;

    private ArrayList<String> Preferences = new ArrayList<String>();

    ( . . . )

    public void addPreference(String preference)
    {
        getPreferences().add(preference);
    }

    public ArrayList<String> getPreferences() {
        return Preferences;
    }

    public void setPreferences(ArrayList<String> Preferences) {
        this.Preferences = Preferences;
    }
}
```

**E depois:**

```
Utilizador newUser = new Utilizador();

newUser.setNome("Miguel");
newUser.setMembro(true);
newUser.setIdade(19);

newUser.addPreference("Tenis");
newUser.addPreference("Soccer");

ObjectMapper mapper = new ObjectMapper();
String strJson = mapper.writeValueAsString(newUser);
System.out.println(strJson);

Utilizador oldUser = mapper.readValue(strJson, Utilizador.class);
System.out.println(oldUser.getNome());

for(String pref : oldUser.getPreferences())
{
    System.out.println(pref);
}
```

**Para mais informação acerca da biblioteca JACKSON consultar:**  
<https://github.com/FasterXML/jackson-docs>

## Manipulação de JSON em PHP

A manipulação de JSON em PHP é feita recorrendo às funções **json\_encode** e **json\_decode**.

Exemplo de conversão de JSON para (objeto) PHP:

```
<?php

$json = '{"titulo": "JSON for Beginners",
"autor": "ESTG",
        "edição": 2
}';

$livro = json_decode($json); //Objeto PHP

echo $livro->titulo;
?>
```

Exemplo de conversão de PHP para JSON:

```
<?php
class pessoa
{
    public $nome = "";
    public $hobbies = "";
    public $nascimento = "";
}

$p = new pessoa();
$p->nome = "joao";
$p->hobbies = "desporto";
$p->birthdate = "8/5/1974 12:20:03";
echo json_encode($p);
?>
```

Para mais informação consulte a ajuda oficial das funções **json\_encode** e **json\_decode**.

## Exercícios

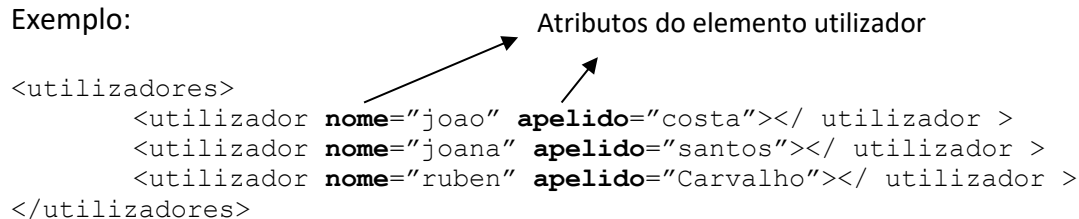
1. Elabore, no papel, o registo de um estudante CTeSP, do segundo ano, relativamente à unidade curricular de Serviços e Interoperabilidade de Sistemas. O registo deve contemplar, pelo menos, o número de estudante, nome, idade e unidades em que está matriculado.
2. De acordo com a alínea anterior, implemente uma aplicação gráfica JAVA, em que pede os dados ao utilizador, preenche a estrutura JSON e guarda num ficheiro, usando a opção de menu “Save”. Implemente também a opção “Open” para ler a partir do ficheiro.
3. Implemente um botão que permita saber apenas os nomes guardados na estrutura JSON no ficheiro.

## Extra - XML

O XML é um formato semelhante ao JSON mas mais “*verbose*” ou seja, o XML consegue ser muito mais completo e claro na caracterização dos dados (meta dados). Contudo ocupa muito mais memória e poderá ser inviável em sistemas embebidos de poucos recursos.

O XML é constituído apenas por elementos e por atributos. Os elementos têm sempre o formato `<elem>xxx</elem>` enquanto que os atributos estão dentro de um elemento.

Exemplo:



The diagram shows an XML snippet with three user elements. Two arrows point from the text 'Atributos do elemento utilizador' to the 'nome' and 'apelido' attributes of the first user element.

```
<utilizadores>
  <utilizador nome="joao" apelido="costa"></ utilizador >
  <utilizador nome="joana" apelido="santos"></ utilizador >
  <utilizador nome="ruben" apelido="Carvalho"></ utilizador >
</utilizadores>
```

Ao contrário de JSON, em XML a noção de vetor não é definida com diferentes símbolos de meta-dados. Continuamos a usar apenas elementos e atributos. No exemplo acima poderemos considerar utilizadores como um vetor de registos de utilizador.

O uso de atributos não é obrigatório. Assim sendo, os dados do exemplo acima também podem ser representados da seguinte forma:

```
<utilizadores>
  <utilizador>
    <nome>joao</nome>
    <apelido>costa</apelido>
  </utilizador>
  <utilizador>
    <nome>joana</nome>
    <apelido>santos</apelido>
  </utilizador>
  <utilizador>
    <nome>ruben</nome>
    <apelido>carvalho</apelido>
  </utilizador>
</utilizadores>
```