Universidade de Coimbra

Faculdade de Ciências e Tecnologia

Mestrado em Engenharia Informática

Integração de Sistemas – 2020/2021

1º Ano – 1º Semestre

Assignment #1

Data Representation and Serialization Formats

Coimbra, 7 de outubro de 2021

2018298731 João Pedro Pacheco Silva joaopedro@student.dei.uc.pt

1. INTRODUÇÃO

Propõe-se neste trabalho realizar um estudo sobre tecnologias de representação e serialização de dados, de forma a obter conhecimentos sobre as características que diferenciam os formatos textuais dos formatos binários. Para isso será feita uma comparação entre o formato XML e o formato MessagePack, tendo em consideração aspetos como a complexidade de programação de cada um, o tamanho do dados serializados e os suas velocidade de serialização e desserialização.

2. DESCRIÇÃO DOS FORMATOS

2.1. XML:

O XML um formato para a criação de documentos com dados organizados de forma hierárquica. Este codifica os dados de forma textoal, sendo legível tanto para pessoas como para máquinas. Ele próprio descreve a sua estrutura e nomes de campos assim como restrições relativamente ao conteúdo do documento, podendo-se assim realizar validações que vão além de uma análise sintática simples. Por ser independente de software e hardware, providencia uma maneira de fazer troca de dados entre sistemas incompatíveis.

2.2. MessagePack:

O MessagePack é um formato de representação de dados que, tal como o XML, é independente de software e de hardware. O seu principal objetivo é ser um formato compacto usado para transferência de dados. Os dados armazenados são escritos no sistema binário, pelo que o seu conteúdo não é elegível para seres humanos.

3. DETALHES DE IMPLEMENTAÇÃO

3.1. Linguagens, bibliotecas e hardware usados:

A implementação dos algoritmos de serialização e desserialização dos dois formatos foi feita inteiramente em java. Relativamente ao MessagePack, a serialização e desserialização foi feita recorrendo à API do MessagePack (versão 0.6.12) presente no Apache Maven. Já no XML a serialização e desserialização foi feita recorrendo à API do JAXB (versão 3.0.0) também presente no Apache Maven.

Em termos de hardware, foi usado um computador com as seguintes especificações:

- Sistema operativo: Windows 11
- Processador: Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz 2.21 GHz
- RAM instalada: 16,0 GB (15,9 GB utilizável)
- Tipo de sistema: Sistema operativo de 64 bits, processador baseado em x64

3.2. Estruturas de dados usadas:

Relativamente às estruturas de dados, foram criadas duas classes chamadas Pet e Owner, responsáveis por guardar os dados dos relacionamentos pet-owner, e uma classe chamada Data, responsável por criar e armazenar os diferentes conjuntos de pets e owners usados em cada experiência.

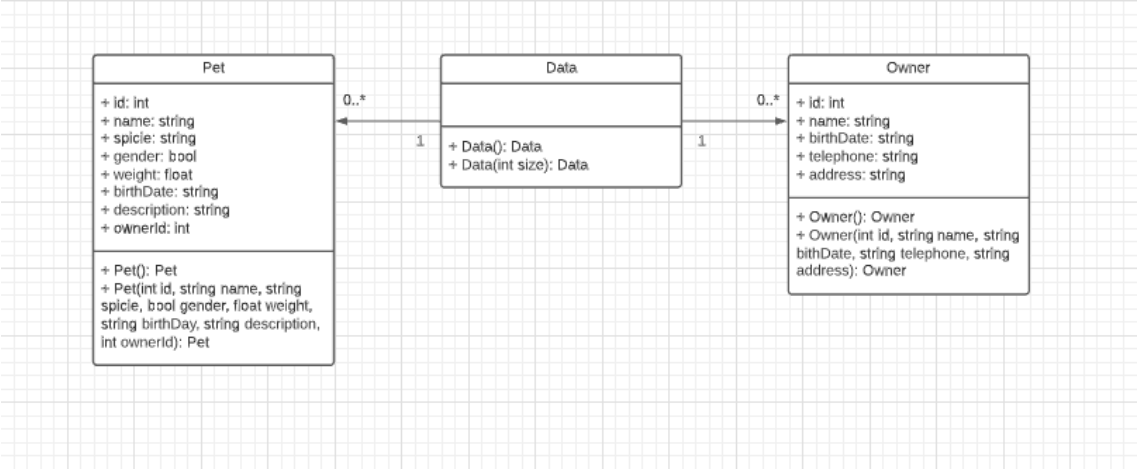

Figura 1- Diagrama UML

3.3. Medições de tempo:

Nas serializações do XML e do MessagePack a medição é iniciada após a criação dos dados e termina assim que o código escrever todos os dados serializados em ficheiro.

```
public static void main(String[] args) throws Exception{

    Data d = new Data( size: 1000000);   ← criação dos dados

    long time = System.currentTimeMillis();   ← inicio da medição

    JAXBContext contextObj = JAXBContext.newInstance(Data.class);

    Marshaller marshallerObj = contextObj.createMarshaller();
    marshallerObj.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
    marshallerObj.marshal(d, new File( pathname: "C:\\Users\\ASUS\\Desktop\\IS\\Projeto_1\\src\\main\\java\\data.xml"));

    time = System.currentTimeMillis() - time;   ← fim da medição

    System.out.println(time);

}
```

Figura 2 - Código de serialização do XML

```java
public static void main(String[] args) throws Exception {

    Data d = new Data( size: 10);      ← criação dos dados

    long time = System.currentTimeMillis();      ← inicio da medição

    MessagePack msgpack = new MessagePack();

    byte[] bytes = msgpack.write(d);
    OutputStream outputStream = new FileOutputStream( name: "C:\\Users\\ASUS\\Desktop\\IS\\Projeto_1\\src\\main\\java\\data.bin");
    outputStream.write(bytes);

    time = System.currentTimeMillis() - time;      ← fim da medição

    System.out.println(time);
}
```

Figura 3 - Código de serialização do MessagePack

Nas desserializações do XML e do MessagePack a medição é iniciada antes da leitura dos dados no ficheiro e termina assim que todos os dados forem desserializados.

```java
public static void main(String[] args) throws Exception{

    long time = System.currentTimeMillis();      ← inicio da medição

    File file = new File( pathname: "C:\\Users\\ASUS\\Desktop\\IS\\Projeto_1\\src\\main\\java\\data.xml");
    JAXBContext jaxbContext = JAXBContext.newInstance(Data.class);

    Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller();
    Data d = (Data) jaxbUnmarshaller.unmarshal(file);

    time = System.currentTimeMillis() - time;      ← fim da medição

    System.out.println(time);

}
```

Figura 4 - Código de desserialização do XML

```java
public static void main(String[] args) throws Exception {

    long time = System.currentTimeMillis();      ← inicio da medição

    MessagePack msgpack = new MessagePack();

    InputStream in = new FileInputStream( name: "C:\\Users\\ASUS\\Desktop\\IS\\Projeto_1\\src\\main\\java\\data.bin");
    byte[] bytes = in.readAllBytes();
    Data d = msgpack.read(bytes, Data.class);

    time = System.currentTimeMillis() - time;      ← fim da medição

    System.out.println(time);
}
```

Figura 5 - Código de desserialização do MessagePack

4. ANÁLISE E INTERPRETAÇÃO DOS RESULTADOS

XML:

Número de pets e owners	Tempo médio de serialização (ms)	Tempo médio de desserialização (ms)	Espaço ocupado (Bytes)
10	183.6	212.5	4 496
100	189.0	239.4	45 131
1000	209.9	306.1	459 581
10000	328.9	722.6	4 689 081
100000	843.4	1584.8	47 938 081
1000000	4882.7	8735.9	490 292 081

Tabela 1 – Resultados das experiências de serialização e desserialização do XML

Messagepack:

Número de pets e owners	Tempo médio de serialização (ms)	Tempo médio de desserialização (ms)	Espaço ocupado (Bytes)
10	167.9	183.2	833
100	173.1	195.9	8 757
1000	177.1	198.8	97 305
10000	213.2	246.1	1 037 305
100000	362.2	435.5	11 238 093
1000000	1286.3	1593.4	122 842 093

Tabela 1 – Resultados das experiências de serialização e desserialização do MessagePack

A partir dos dados, observa-se que o XML ocupa mais espaço em memoria que o MessagePack. Esses valores maiores devem-se ao facto do XML se comprometer em ser legível para seres humanos. Por exemplo, na representação do número 100, o XML tem que usar 3 Bytes, um para cada algarismo, de forma obter representação correta em ASCII. Já o MessagePack, por não ter de se preocupar em ser legível consegue representar esse mesmo número com apenas um Byte. Um outro fator que também contribui para o XML ocupar mais espaço é o uso de tags, que é uma forma muito pouco compacta de delimitar os dados.

A partir dos dados, observa-se também que XML apresenta tempos de se serialização e desserialização maiores que o MessagePack. Essa diferença de velocidade deve-se ao facto do XML precisar de fazer parsing dos dados de forma a separar corretamente os dados delimitados pelas tags e fazer tratamento dos mesmo de forma a passar valores numéricos para sequências de caracteres ou sequencias caracteres para valores numéricos. O MessagePack, por outro lado, apenas precisa escrever sequencias de Bytes com os dados e precedê-los com a identificação dos respetivos formatos.

5. REFERÊNCIAS

https://pt.wikipedia.org/wiki/MessagePack

https://pt.wikipedia.org/wiki/XML

https://en.wikipedia.org/wiki/Comparison_of_data-serialization_formats

https://www.w3schools.com/xml/xml_whatis.asp