Integração de Sistemas - 2021/22
MEI

Data Representation and Serialization Formats

Assignment #1

Maria Paula Viegas	2017125592	viegas@student.dei.uc.pt
Rita Fonseca	2018294751	rmfonseca@student.dei.uc.pt

1. Índice

2. Introdução

Nesta experiência, fomos desafiadas a comparar duas tecnologias diferentes de representação de dados, nomeadamente um formato de texto e um formato binário, a fim de chegar a conclusões sobre qual tem uma maior *performance* e qual a diferença entre usar um e não o outro. Assim, optámos por comparar o formato de texto XML (*Extensible Markup Language*) com o formato binário Google Protocol Buffers.

De maneira a chegarmos a algum tipo de conclusão sobre possíveis diferenças entre os dois formatos, tivemos que desenvolver um pequeno programa que, ao ler um ficheiro de *input* com os dados a analisar, fizesse a correspondência entre um dono (*owner*) e cada um dos seus animais de estimação (*pet*). Ambos os programas de geração do formato de texto e do formato binário tiveram como base o mesmo código fonte em *Java* já referido, tendo alterado apenas o essencial, de modo a que a comparação fosse o mais paralela e o mais transparente possível.

Por fim, este programa criaria um ficheiro de output com esta situação aplicada na respetiva tecnologia usada, e o tempo de serialização e de desserialização do programa de modo a poder ser usado como parâmetro de comparação entre as duas tecnologias. Do mesmo modo, foi também usado como elemento de comparação a complexidade do programa, o tamanho do ficheiro gerado como *output*, o tempo de análise do texto de *input* e o tamanho das estruturas usadas na memória.

2.1. XML

Extensible Markup Language, mais conhecido como XML, é um formato de texto flexível que tem um papel importante na troca de grandes quantidades de dados por serviços *web*. É também bastante útil na troca de informação entre diferentes programas, sistemas ou mesmo organizações dada a facilidade em ler o formato em que os dados ficam guardados, podendo ser percebido até mesmo por uma pessoa qualquer.

A ideia base deste formato de texto é uma estrutura em árvore com *tags* associadas a cada nó que guardam as propriedades do mesmo. Caso esta propriedade seja única e tenha um valor único identificador daquele específico dado costumam ser usados atributos.

Para exemplificar, tomando em conta o cenário colocado para o desenrolar desta experiência, podemos ver como é que os dados relativos a um dono Manuel com o identificador único (*id*) 123 e à sua cadela *Betty* com o *id* 321 seriam guardados num ficheiro XML:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<catalog>
        <owners ownerId=”123”>
                <name>Manuel</name>
                <telephone>912912912</telephone>
                <birth>2005-11-05</birth>
                <address>Portugal</address>
                <pets ownerId=”123” petId=”321”>
                        <name>Betty</name>
                        <species>Labrador</species>
                        <gender>female</gender>
                        <birth>2012-08-06</birth>
                        <weight>12</weight>
                        <form_desc>brown</form_desc>
                </pets>
        </owner>
</catalog>
```

Figura 1 - Exemplo de um ficheiro de XML com estrutura em árvore

2.2. Google Protocol Buffers

O Protocol buffers, também conhecido como *Protobuf*, é um mecanismo de extensão desenvolvido pela Google para serializar dados estruturados. Tem como uma das principais vantagens a independência de uma linguagem ou uma plataforma específica. Sendo assim, uma vez definida a estrutura dos dados desejados, é possível usar o código fonte gerado para escrever ou ler os dados para uma variedade de data streams e em diferentes linguagens.

Por ser um protocolo de serialização binário, o seu formato serializado não é legível para humanos. No entanto, a Google afirma que ele pode ser de 3 a 10 vezes mais compacto que o XML correspondente. É flexível e extensível, de forma que a estrutura dos dados pode ser atualizada sem inviabilizar os programas que usam o formato desatualizado. Além disso, o formato binário é gerado e analisado rapidamente.

Para este trabalho, a estruturação pet-owner dos dados acontece num ficheiro *catalog.proto* que, uma vez compilado, gera o código fonte das classes java necessárias para leitura e escrita de dados.

```
syntax = "proto3";

option java_multiple_files = true;

message Pet {
        optional int32 pet_id = 1;
        optional string name = 2;
        optional string species = 3;
        optional string gender = 4;
        optional int32 weight = 5;
```

```
        optional string birth = 6;
        optional string form_desc = 7;
        optional int32 owner_id = 8;
}

message Owner {
        optional int32 owner_id = 1;
        optional string name = 2;
        optional string birth = 3;
        optional int64 telephone = 4;
        optional string address = 5;
        repeated Pet pets = 6;
 }

message Catalog {
        repeated Owner allOwners = 1;
        repeated Pet allPets = 2;
}
```

Figura 2 - Estrutura dos dados em catalog.proto

3. Condições

Para a realização do trabalho, é necessário estabelecer condições iniciais que se mantêm para todos os testes, independentemente se é formato de texto ou formato binário. Estas condições são descritas nesta parte.

3.1. Estrutura de Dados e Formato do ficheiro de texto

Para a comparação das diferentes tecnologias analisadas, usamos uma estrutura de dados simples com a relação many-to-one pet-owner. Os *Pets* e os *Owners* seguem a estrutura do enunciado, e adicionamos uma estrutura *Catalog* que guarda uma lista de *Pets* e outra de *Owners* para registar os indivíduos e suas respectivas relações. A figura 1 ilustra o diagrama de classes da estrutura definida.

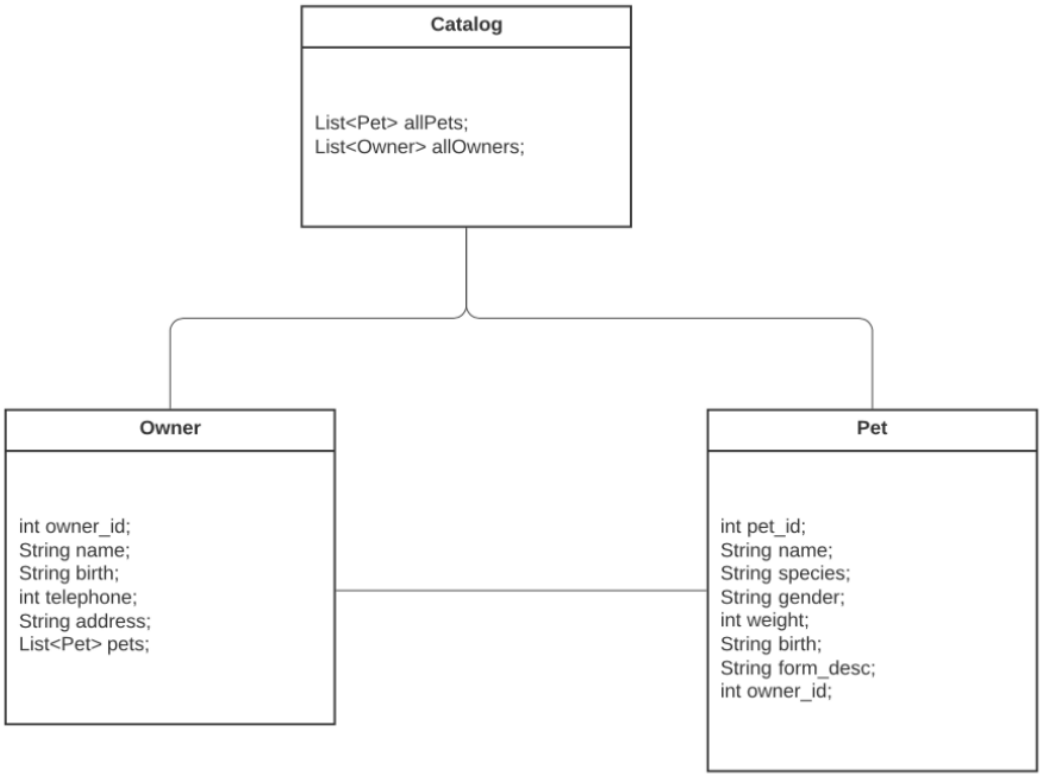

Figura 3 - Diagrama de Classes

Também definimos um ficheiro de texto com nosso set exemplo para realizar os testes. Este ficheiro tem como nome "*data.txt*" e cada linha começa com um sinal de "+" ou um sinal de "-". O sinal determina se as informações da respectiva linha são um *Pet* ("+") ou um *Owner* ("-"). Cada informação da linha é separada pelo sinal de ";". A Tabela 1 ilustra como as linhas do ficheiro devem ser formatadas e a ordem que os parâmetros devem seguir.

Owner	-;owner_id;name;birth;telephone;address
Pet	+;pet_id;name;species;gender;weight;birth;form_desc;owner_id

Figura 4 - Formatação do ficheiro data.txt

Nosso set exemplo consta 34 *pets* e 5 *owners, sendo que cada* owner possui, no máximo, 10 *pets*.

3.2. Características

Em seguida encontra-se uma tabela com as várias características técnicas, relativamente aos computadores da Maria e da Rita, de maneira a enunciar as condições em que a experiência foi realizada, podendo ser um factor influenciador dos resultados finais.

	Maria	**Rita**
Versão do Sistema Operativo	macOS 11.6	macOS 11.6
Processador	2,2 GHz Quad-Core Intel Core i7	2,3 GHz Dual-Core Intel Core i5
Linguagem de Programação	Java	Java
Tecnologias / *Libraries*	java.io protobuf-java 3.18.0	javax.xml java.util java.io
Versão da Linguagem	11.0.10	12.0.2
IDE	IntelliJ IDEA	IntelliJ IDEA

Tabela 1 - Características dos computadores onde foi realizada a experiência

Em cada uma destas máquinas, com as linguagens e tecnologias em cima explicitadas, foram efetuados 30 testes com o mesmo ficheiro de *input* tanto para Google Protocol Buffers (computador da Maria) como para XML (computador da Rita) em cada um dos quais foram medidos os tempos de serialização e desserialização, como já explicado em cima. O ficheiro de *input* não sofreu alterações de modo a que este não fosse um fator influenciador entre as diferenças dos tempos, e de maneira a ter condições de comparação o mais semelhantes possíveis.

4. Parâmetros de Avaliação

Nesta parte, comparamos os formatos XML e Protocol Buffers quanto às respectivas complexidades, tamanhos de serialização, e tempo de serialização e desserialização. Também avaliamos o tamanho das estruturas de dados em memória.

4.1. Complexidade do Programa

Ambos os mecanismos utilizados neste trabalho são conhecidos por serem de simples entendimento e uso.

Quanto ao Protocol Buffers, uma vez que se entende como o mecanismo funciona, é fácil e intuitivo de se aplicar. As classes geradas pelo ficheiro *.proto* já disponibiliza quase tudo que possa ser necessário para a representação e a serialização de dados.

Relativamente ao XML, a única dificuldade de maior cariz foi encontrar as dependências e as bibliotecas corretas de modo a que o programa funcionasse. Os atributos e os elementos (incluindo o elemento de raiz) foram facilmente identificados e a partir daí bastou fazer o *marshall* e o *unmarshall* dos objetos de maneira a gerar o ficheiro de texto.

4.2. Tamanho da serialização

A partir da Tabela 2, podemos ver que o ficheiro gerado com o formato serializado do XML é bem maior que o ficheiro gerado pelo Protobuf. Isso está dentro do esperado porque o ficheiro do Protobuf é codificado em binário.

	XML	***Protobuf***
Size	*10 814 bytes*	*2 473 bytes*

Tabela 2 - Tamanho da serialização

4.3. Tempo de Serialização

Analisando o gráfico seguinte (Figura 2) consegue-se chegar à conclusão que, não só o XML tem tempos de serialização mais instáveis e variáveis, como o Protobuf é bem mais rápido e eficaz ao gerar o *output* a partir dos objetos que contêm os dados guardados, sendo que o primeiro tem uma média de 492.6 ms e o segundo de ≈120.03 ms .

Para medir o tempo de serialização, marcamos como tempo de início do processo o momento em que o programa termina de ler o dataset exemplo e,

como tempo final, o momento que o programa termina de escrever o ficheiro de output com os dados serializados.

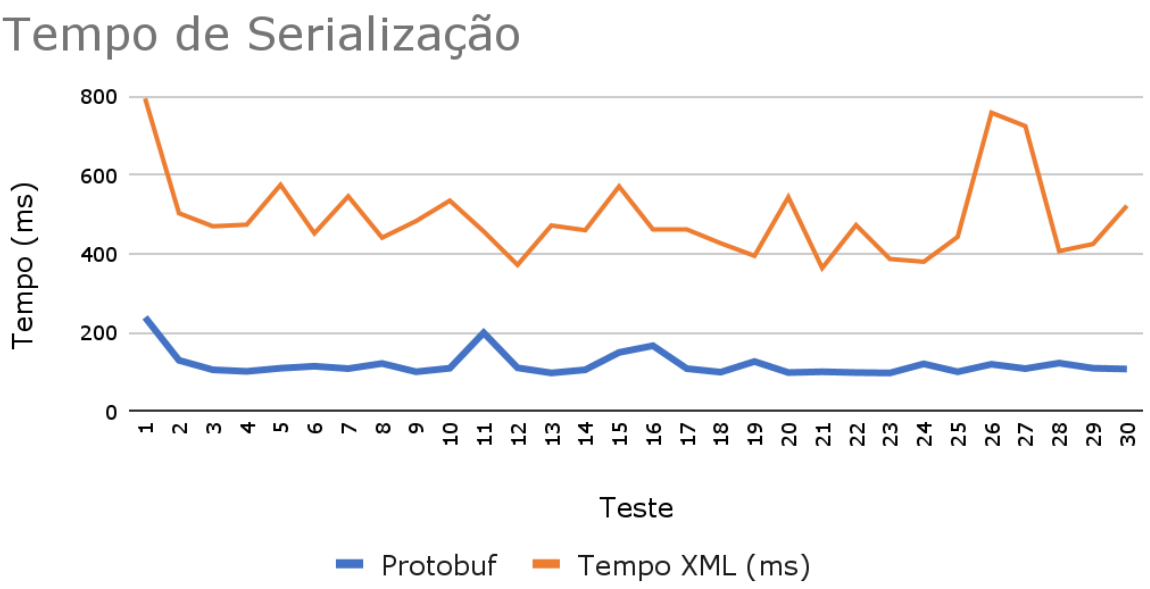

Gráfico 1 - Tempos de serialização do XML e do Protobuf

4.4. Tempo de Desserialização

Observando o gráfico que se segue conseguimos concluir que, mais uma vez, o XML ultrapassou por bastante as velocidades do protobuf, tendo o primeiro uma média de ≈158.96 ms e o segundo uma média de ≈8.43 ms.

De facto, sendo o Protobuf um formato binário comprimido, seria expectável que este fosse igualmente a desserializar como a serializar relativamente ao XML, que por sua vez, se trata de um formato de texto mais demorado a interpretar por uma máquina.

Consideramos como tempo inicial de desserialização o momento antes de ler o ficheiro de output - sendo, para o XML, o comando de *unmarshal()* da classe *Unmarshaller* e, para o Protobuf, o comando de *parseFrom()* da classe *Catalog* gerada pelo protobuf. O tempo final é marcado depois desses comandos.

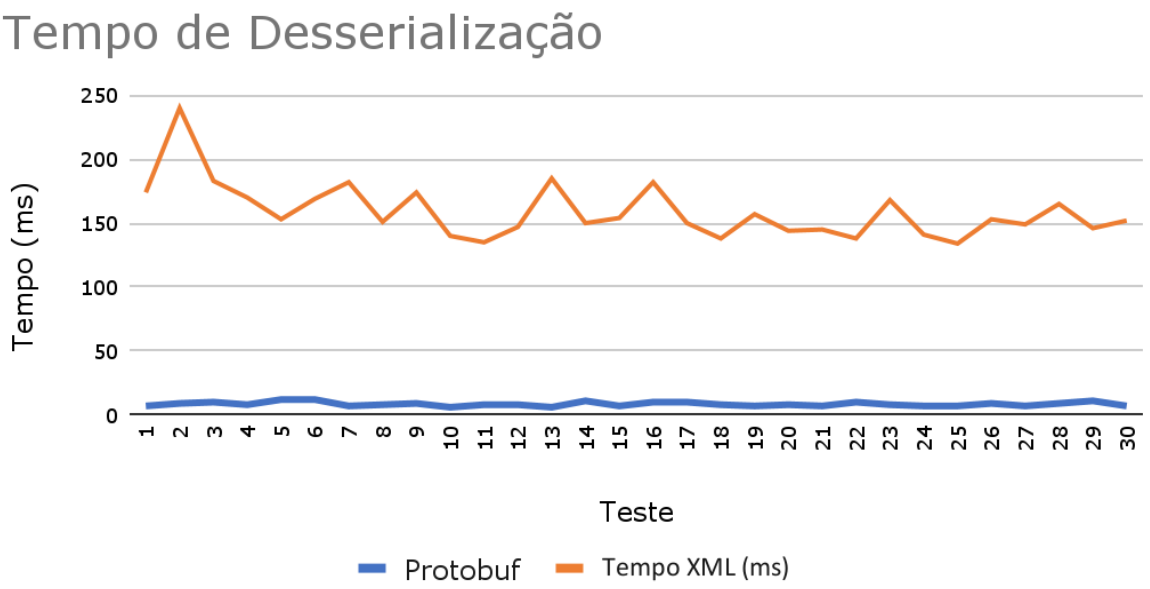

Gráfico 2 - Tempos de desserialização do XML e do Protobuf

4.5. Tamanho das estruturas de dados em memória

No desenvolvimento desta experiência, foram criadas 3 classes de objetos: *Pet* - que armazena informação relativa ao *pet* em questão, *Owner* - que armazena informação relativa ao *owner* em questão e uma lista com os seus *pets*, e *Catalog* - que armazena duas listas: uma com os *pets* já registados no sistema e outra com os *owners* já registados no sistema. Fora estes objetos, ainda existem 4 variáveis relevantes que servem para contar o tempo que passou durante a serialização e a desserialização.

Assim, verificamos que as estruturas de dados na memória são 2 listas de *pets* e de *owners*, uma lista de *pets* por cada *owner,* e finalmente 4 constantes com os valores do tempo armazenados, entre outras quaisquer auxiliares que possam existir.

5. Conclusão

Tal como nos foi sugerido, após o desenvolvimento de um código de base que iria ser aproveitado tanto para XML como para Protobuf (com as devidas alterações em cada um dos casos), foram efetuados 30 testes com XML e com Google Protocol Buffers, tentando minimizar diferentes fatores que fossem prejudicar os resultados finais. De facto, podemos observar que, como esperado, havia uma grande diferença entre os dois formatos.

Apesar das vantagens já indicadas do XML de poder ser facilmente interpretado por diferentes máquinas e/ou até mesmo seres humanos, o Google Protocol Buffers conseguiu, como esperado, ter uma melhor performance de serialização e de espaço de armazenamento (como observado, o ficheiro resultante ocupa menos bytes que o ficheiro de resultante de XML), sendo este bem mais compactado que o XML. No entanto, embora seja um formato analisado e gerado rapidamente, sendo este um formato binário, não é legível para humanos, o que em determinadas situações pode vir a mostrar-se um problema.

Assim, concluímos que para comunicações entre máquinas de empresas diferentes, o Google Protocol Buffers seria o formato indicado a recorrer devido ao pouco espaço ocupado, proteção contra terceiros, e aos baixos tempos de serialização e desserialização. No entanto, caso houvesse um intermediário humano, a escolha mais legível seria utilizar um formato de texto XML para que este conseguisse interpretar facilmente os dados lá representados e, com isso, evitar perder tempo em tentar descodificar o que estava escrito em binário.

6. Referências Bibliográficas

- Paulo Marques e Filipe Araújo, Slides da cadeira de Integração de Sistemas 2021/2022
- Extensible Markup Language (XML), acedido em 4 de Outubro de 2021, <https://www.w3.org/XML/>
- Protocol Buffers, acedido em 4 de Outubro de 2021, <https://developers.google.com/protocol-buffers>
- Binary Serialization with Google Protocol Buffers, acedido em 4 de Outubro de 2021, <https://blog.grijjy.com/2017/04/25/binary-serialization-with-google-protocol-buffers/>