Universidade Tecnológica Federal do Paraná
Departamento Acadêmico de Computação
Bacharelado em Ciência da Computação

Sistemas Distribuídos

Representação Externa de Dados

Prof. Rodrigo Campiolo

16/08/20

Tópicos

- Introdução
- Marshalling and Unmarshalling
- Representação Externa de Dados
  - Corba CDR e Sun XDR
  - Serialização de Objetos Java
  - XML
  - JSON
  - Protocol Buffers
- Atividades

Introdução

- Na comunicação entre processos as estruturas devem ser representadas por uma **sequência de bytes**.

struct\4\xstring\x05Luffy\x19\x1\xarray\...

```
struct Data {
    name = "Luffy"
    age = 19
    hasAkumaNoMi = true
    nakamas = ["Zoro","Nami","Sanji"]
}
```

- E como tratar das seguintes questões:

Introdução

- Q1: Transmissão de dados de **tipos primitivos** e **tipos compostos**.
  - `int`: 16, 32 ou 64 bits?
  - `arrays`: quantidade e tipo de elementos?
  - `strings`: objeto, array, tipo primitivo?
  - `structs` e `classes`?

Introdução

- Q2: Ordem dos bytes
  - big-endian (bytes **mais** significantes primeiro)
  - little-endian (bytes **menos** significantes primeiro)

```
>>> value = 4
>>> value.to_bytes(4, 'big')
b'\x00\x00\x00\x04'
>>> value.to_bytes(4, 'little')
b'\x04\x00\x00\x00'
```

Introdução

- Q3: Codificação de caracteres (ISO 8859-1, UTF-8, ASCII, Unicode)

```
>>> str = 'a b x y z ç Ç á à ã é ñ ü'
>>> str.encode('utf-8')
b'a b x y z \xc3\xa7 \xc3\x87 \xc3\xa1 \xc3\xa0 \xc3\xa3 \xc3\xa9 \xc3\xb1 \xc3\xbc'
>>> str.encode('iso8859-1')
b'a b x y z \xe7 \xc7 \xe1 \xe0 \xe3 \xe9 \xf1 \xfc'
>>> str.encode('cp850')
b'a b x y z \x87 \x80 \xa0 \x85 \xc6 \x82 \xa4 \x81'
```

* UTF = Unicode Transformation Format

Introdução

- **Representação Externa de Dados**
  - Padrão acordado para a representação de dados entre dois processos.

- Abordagens:
  - Dados são convertidos para um padrão estabelecidos entre o transmissor e receptor.
  - Dados transmitidos no formato do emissor em conjunto com informações que descrevem os dados (metadados).

Introdução

- **Marshalling**: processo de empacotar dados no emissor para um formato adequado para a transmissão.

DADOS → REPRESENTAÇÃO EXTERNA DE DADOS

- **Unmarshalling**: processo de desempacotar dados no receptor e obter os mesmos valores.

REPRESENTAÇÃO EXTERNA DE DADOS → DADOS

Representação Externa Dados

- Dada a estrutura:

```
Person {
    name  = "Smith"      // string
    place = "London"     // string
    year  = 1984         // inteiro
}
```

- Como transmitir entre plataformas heterogêneas ou software escrito em linguagens diferentes?

Corba CDR

- Common Data Representation (CDR)
  - Emissor define a ordenação dos bytes (flag indica a ordenação).
  - Tipos primitivos alinhados aos seus limites naturais (p. ex. char (1 byte), long (4 bytes), double (8 bytes).
  - Tipos compostos (struct, union, array, sequence, strings).
  - IDL – Interface Definition Language.

Corba CDR

- Exemplo

```
struct Person {
    string name;
    string place;
    unsigned long year;
};
```

Compilador de IDL: java, C/C++, python, ...

```
# omniidl -bcxx exemplo_cdr.idl
# idlj exemplo_cdr.idl
```

Sun XDR

- eXternal Data Representation (XDR)
  - Usada no RPC Linux (p. ex. NFS).
  - Representação dos bits em múltiplos de 4 bytes.
  - Ordenação big-endian.
  - Simples para construção de hardware para processar e decodificar.

Sun XDR

índice da sequência de bytes	4 bytes	
0-3	5	tamanho da string
4-7	"Smit"	string
8-11	"h___"	
12-15	6	tamanho da string
16-19	"Lond"	string
20-23	"on__"	
24-27	1984	unsigned int

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
00	00	00	05	S	m	i	t	h	-	-	-	00	00	00	06	L	o	n	d	o	n	-	-	0	0	07	C0

4 bytes

Sun XDR

- Exemplo

```
struct Person {
    string name<>;
    string place<>;
    unsigned int year;
};
```

Compilador XDR:

```
# rpcgen exemploxdr.xdr
```

Para brincar: https://docs.python.org/3.0/library/xdrlib.html

Serialização de Objetos Java

- Encapsula os tipos e dados como uma sequência de bytes.
- Os tipos primitivos são por padrão serializáveis. Classes devem implementar a interface *Serializable.*
- Classes Java:
  - *ObjectInputStream:* leitura de objetos serializados.
  - *ObjectOutputStream:* escrita de objetos serializados.

Serialização de Objetos Java

- Exemplo

```
import java.io.Serializable;


public class Person implements Serializable
{
  private String name;
  private String place;
  private int year;

  /* construtor */
  public Person (String name, String place, int year)
  {
     this.name = name;
     this.place = place;
     this.year = year;
  }

  // métodos set() e get()

} // class Person
```

Serialização de Objetos Java

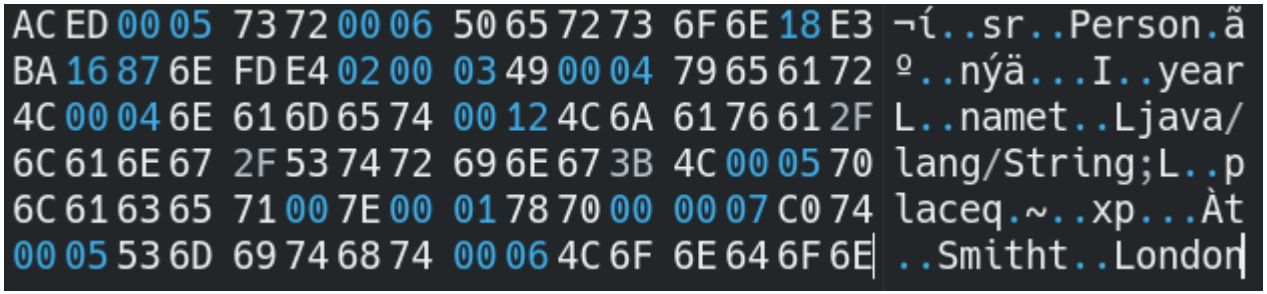
- Exemplo

Serialized values				*Explanation*
Person	8-byte version number		h0	*class name, version number*
3	int year	java.lang.String name	java.lang.String place	*number, type and name of instance variables*
1984	5 Smith	6 London	h1	*values of instance variables*

The true serialized form contains additional type markers; h0 and h1 are handles.

Fonte: Coulouris

Serialização de Objetos Java



AC ED - STREAM_MAGIC
00 05 - STREAM_VERSION
73 – TC_OBJECT (novo objeto)
72 – TC_CLASSDESC (novo descritor de classe)
00 06 – tamanho do nome da classe
50 65 72 73 6F 6E – nome da classe (Person)
18 E3 BA 16 87 6E FD E4 – serial (8 bytes)
02 – objeto serializavel
00 03 – número de atributos
49 – indica tipo primitivo int (I)
00 04 – tamanho nome atributo
79 65 61 72 – nome do atributo (year)
4C – indica tipo Objeto (L)
00 04 – tamanho nome atributo
6E 61 6D 65 – nome do atributo (name)

74 - TC_String
00 12 – tamanho nome tipo (18 bytes)
4C – indica tipo Objeto (L)
6A 61 … 67 3B – java/lang/String;
4C – indica tipo Objeto (L)
00 05 – tamanho nome atributo
70 6C 61 63 65 – nome do atributo (place)
71 00 7E 00 01 78 70 - ?
00 00 07 C0 – valor do atributo (1984)
74 – TC_STRING
00 05 – tamanho do string
53 6D 69 74 68 – valor do atributo (Smith)
74 - TC_STRING
00 06 – tamanho da string
4C 6F 6E 64 6F 6E – valor do atributo (London)

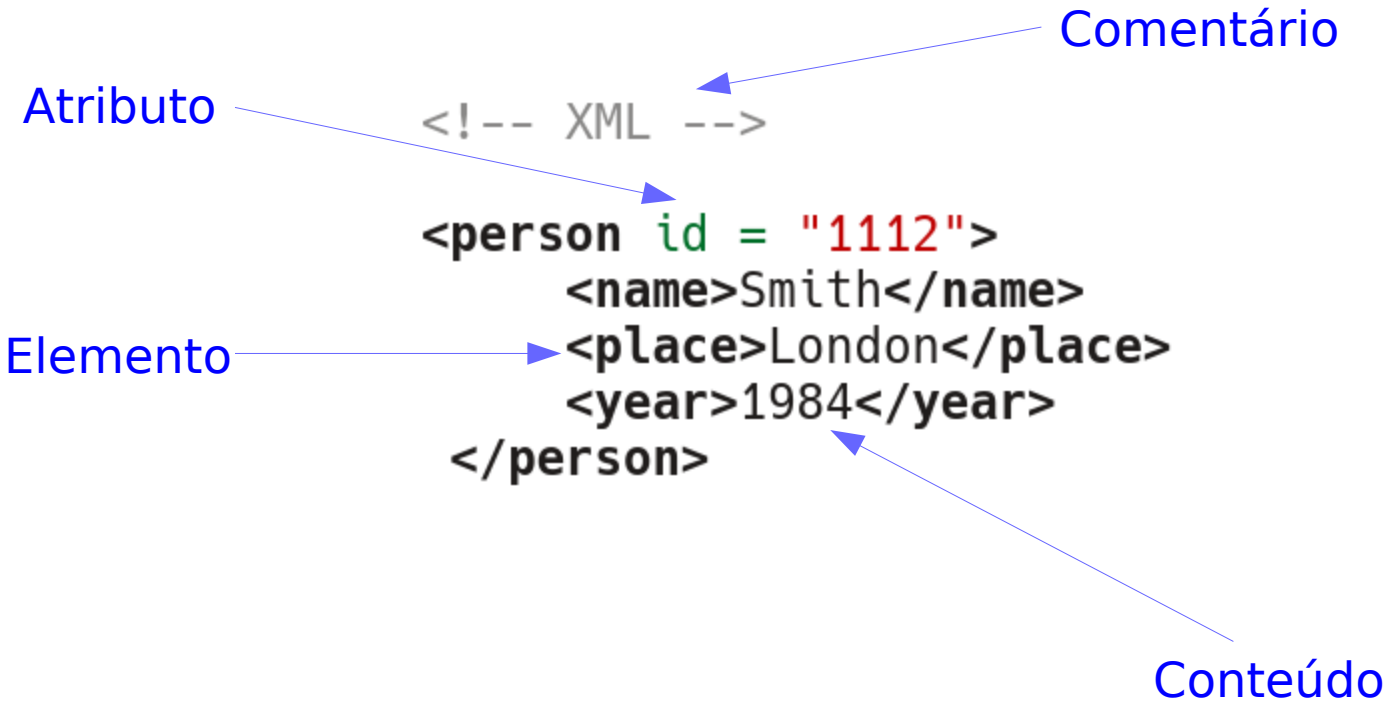
* Protocolo de serialização Java 14: https://docs.oracle.com/en/java/javase/14/docs/specs/serialization/protocol.html

XML

- Extensible Markup Language (XML)
- Linguagem de marcação definida pelo World Wide Web Consortion (W3C)
- Representação textual
- Usuários definem as suas marcações
- Parsers
  - Document Object Model (DOM): leitura/escrita, tree-based
  - Simple API for XML (SAX): leitura, event-based
- Mais info: *https://www.w3schools.com/xml/default.asp*

XML

- Exemplo


Comentário
Atributo
<!-- XML -->
<person id = "1112">
<name>Smith</name>
<place>London</place>
<year>1984</year>
</person>
Elemento
Conteúdo

JSON

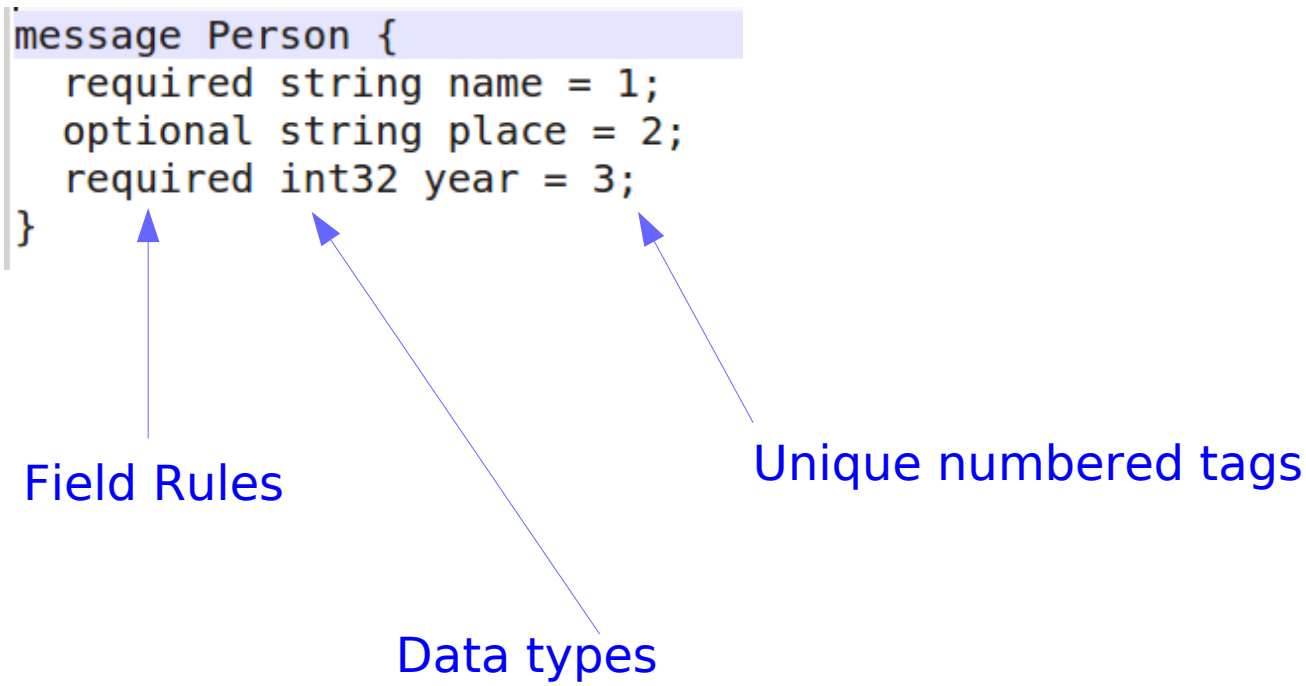
- JavaScript Object Notation (JSON)
- Formato leve para troca de dados
- Mais info:
  - https://www.json.org/

```
{
  "person": {
    "name": "Smith",
    "place": "London",
    "year": 1984
  }
}
```

Protocol Buffers

- Formato leve para serializar estruturas de dados.
- Desenvolvido pela Google.

```
message Person {
  required string name = 1;
  optional string place = 2;
  required int32 year = 3;
}
```



Outros formatos

- pickle: serialização binária em Python.

```
>>> data = {'name':'Smith','place':'London','year':1984}
>>> pickle.dumps(data)
b'\x80\x03}q\x00(X\x04\x00\x00\x00nameq\x01X\x05\x00\x00\x00Smithq\x02X\x0
0\x00placeq\x03X\x06\x00\x00\x00Londonq\x04X\x04\x00\x00\x00yearq\x05M\xc0
```

- YAML: *YAML Ain’nt Markup Language* é uma linguagem simples para serialização baseada em texto puro *(human-readable)*.

```
Person:
    name: Smith
    place: London
    year: 1984
```

- Comparação informal entre diferentes formatos de serialização neste link.

Atividades

- Implementar a comunicação entre dois processos remotos usando Serialização Java, XML, JSON e Protocol Buffers.

Referências

COULOURIS, George F; DOLLIMORE, Jean; KINDBERG, Tim; BLAIR, Gordon. **Sistemas distribuídos: conceitos e projeto**. 5. ed. Porto Alegre: Bookman, 2013.