

# MessagePack

Overview da tecnologia e aplicação utilizando o protocolo  
“TFTP2” criado para a disciplina

Guilherme Medeiros e Rafael Teles



# 1. Overview

# 1.1 Overview - Definição

- Forma de serialização de objetos para binário.
- Troca de dados entre linguagens.
- Alta eficiência.
- “É como JSON, mas mais rápido e menor”
- Baixa utilização de bytes para a representação de cada tipo de dado.

JSON 27 bytes

```
{ "compact": true, "schema": 0 }
```

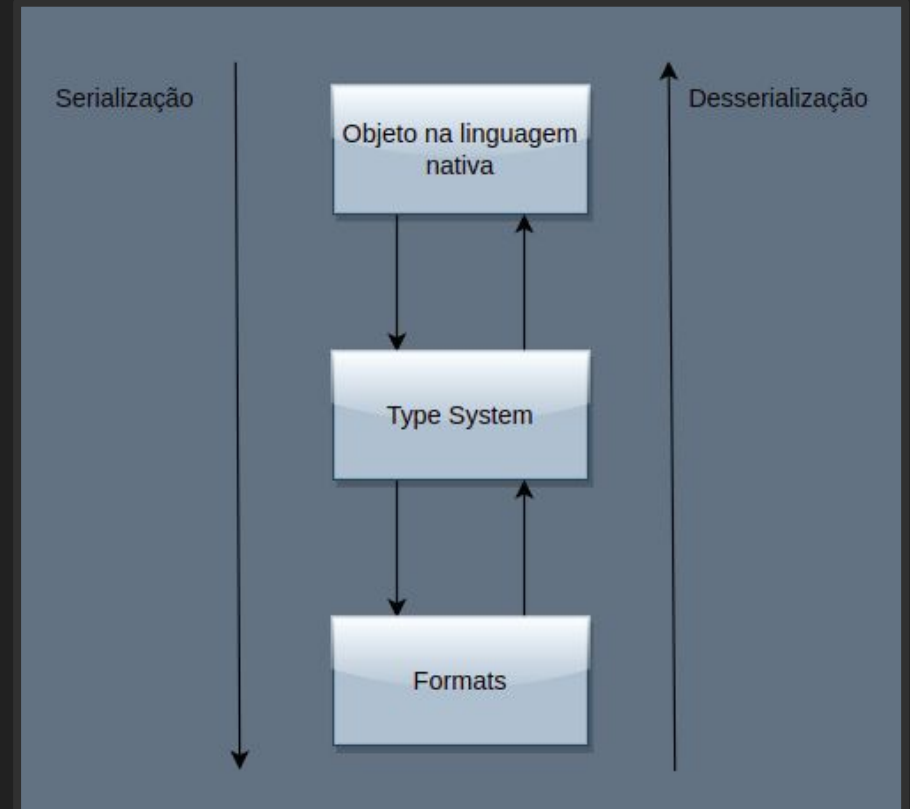
MessagePack 18 bytes



# 1. Overview - Type Systems e Formats

Dois conceitos novos são apresentados nessa tecnologia:

- Type Systems: Tipo de dados intermediários a uma serialização MS, cada Type System assume um Format diferente.
- Formats: Formato em bytes que cada tipo de dado é estruturado.



# 1.1 Overview - Type Systems

- **Integer** - representa um número inteiro
- **Null** - representa o nil
- **Boolean** - representa verdadeiro ou falso
- **Float** - representa números não inteiros, NaN e Infinito
- **Raw:**
  - String - Estende o tipo RAW para strings do tipo UTF-8
  - Binary - Estende o tipo RAW para um byte array
- **Array** - representa uma sequência de objetos
- **Map** - representa um par de objetos do tipo chave-valor

JSON 27 bytes

```
{ "compact": true, "schema": 0 }
```

MessagePack 18 bytes



Limitações não serão mencionadas neste tutorial, mas elas existem quanto ao tamanho máximo que cada type system pode assumir

# 1.1 Overview - Formats

- Sufixo na cadeia de bites em um ou mais byte que representa um Type Form.

positive fixint stores 7-bit positive integer

```
+-----+
|0XXXXXX|
+-----+
```

negative fixint stores 5-bit negative integer

```
+-----+
|111YYYY|
+-----+
```

float 32 stores a floating point number in IEEE 754 single precision floating point number format:

```
+-----+-----+-----+-----+
| 0xca |XXXXXXXX|XXXXXXXX|XXXXXXXX|XXXXXXXX|
+-----+-----+-----+-----+
```

float 64 stores a floating point number in IEEE 754 double precision floating point number format:

```
+-----+-----+-----+-----+-----+-----+-----+-----+
| 0xcb |YYYYYYYY|YYYYYYYY|YYYYYYYY|YYYYYYYY|YYYYYYYY|YYYYYYYY|YYYYYYYY|
+-----+-----+-----+-----+-----+-----+-----+-----+
```

fixstr stores a byte array whose length is upto 31 bytes:

```
+-----+=====+
|101XXXX| data |
+-----+=====+
```

str 8 stores a byte array whose length is upto  $(2^8)-1$  bytes:

```
+-----+-----+=====+
| 0xd9 |YYYYYYYY| data |
+-----+-----+=====+
```

str 16 stores a byte array whose length is upto  $(2^{16})-1$  bytes:

```
+-----+-----+-----+=====+
| 0xda |ZZZZZZZ|ZZZZZZZ| data |
+-----+-----+-----+=====+
```

str 32 stores a byte array whose length is upto  $(2^{32})-1$  bytes:

```
+-----+-----+-----+-----+=====+
| 0xdb |AAAAAAA|AAAAAAA|AAAAAAA|AAAAAAA| data |
+-----+-----+-----+-----+=====+
```

format name	first byte (in binary)	first byte (in hex)
positive fixint	0xxxxxxx	0x00 - 0x7f
fixmap	1000xxxx	0x80 - 0x8f
fixarray	1001xxxx	0x90 - 0x9f
fixstr	101xxxxx	0xa0 - 0xbf
nil	11000000	0xc0
(never used)	11000001	0xc1
false	11000010	0xc2
true	11000011	0xc3
bin 8	11000100	0xc4
bin 16	11000101	0xc5
bin 32	11000110	0xc6
ext 8	11000111	0xc7
ext 16	11001000	0xc8
ext 32	11001001	0xc9
float 32	11001010	0xca
float 64	11001011	0xcb
uint 8	11001100	0xcc
uint 16	11001101	0xcd

## 2. Tutorial

- Python
- Simplicidade da linguagem



## 2.1 Tutorial - Preparativos

- Pré requisitos:

- Python2
- PIP installer

```
~$ python3 --version  
Python 3.6.9
```

- Instalação do MessagePack

- `pip install msgpack`

```
~$ pip --version  
pip 9.0.1 from /usr/lib/python2.7/dist-packages (python 2.7)
```

```
~$ pip install msgpack  
Collecting msgpack  
Installing collected packages: msgpack  
Successfully installed msgpack-1.0.4
```

## 2.2 Tutorial 1 - Um número inteiro codificado

```
>>> import msgpack
>>> msgpack.packb(1, use_bin_type=True)
'\x01' #Inteiro 1 codificado pelo msgpack

>>> msgpack.unpackb(_, raw=False)
1      #Saída do decodificador 1

>>> msgpack.packb(45, use_bin_type=True)
'-'    #Inteiro 45 codificado pelo msgpack
>>> msgpack.unpackb(_, raw=False)
45     #Saída do decodificador 45

>>> msgpack.packb(23, use_bin_type=True)
'\x17' #Inteiro 23 codificado pelo msgpack
>>> msgpack.unpackb(_, raw=False)
23     #Saída do decodificador 23
>>>
```

## 2.2 Tutorial 2 - Um array de números inteiros

```
>>> import msgpack
>>> msgpack.packb([1, 2, 3], use_bin_type=True)
'\x93\x01\x02\x03'    #Array codificado

>>> msgpack.unpackb(_, raw=False)
[1, 2, 3]              #Array decodificado
>>>
```

## 2.2 Tutorial 3 - Decodificando vários objetos

```
import msgpack
from io import BytesIO

buf = BytesIO()
for i in range(100):
    buf.write(msgpack.packb(i, use_bin_type=True))

buf.seek(0)

unpacker = msgpack.Unpacker(buf, raw=False)
for unpacked in unpacker:
```

## 2.2 Tutorial 4 - Criando um dicionário para ser codificado


```
import datetime
import msgpack

useful_dict = {
    "id": 1,
    "created": datetime.datetime.now(),
}

def decode_datetime(obj):
    if '__datetime__' in obj:
        obj = datetime.datetime.strptime(obj["as_str"], "%Y%m%dT%H:%M:%S.%f")
    return obj

def encode_datetime(obj):
    if isinstance(obj, datetime.datetime):
        return {'__datetime__': True, 'as_str': obj.strftime("%Y%m%dT%H:%M:%S.%f")}
    return obj

packed_dict = msgpack.packb(useful_dict, default=encode_datetime, use_bin_type=True)
print(packed_dict)
this_dict_again = msgpack.unpackb(packed_dict, object_hook=decode_datetime, raw=False)
print(this_dict_again)
```



```
{__id__: 1, __created__: True, __as_str__: '20220727T19:45:42.010458', __datetime__: True}

{'id': 1, 'created': datetime.datetime(2022, 7, 27, 19, 45, 42, 10458)}
```

### 3. Especificando o TFTP

1. Analisar formato de cada tipo de mensagem
2. Criar dicionário para cada dado
3. Escrever programas em python que codifiquem uma mensagem, imprimam em um arquivo, leiam o arquivo, decodifique e nos mostre o que foi “transmitido”.
4. Todos as mensagens estão escrevendo no arquivo `data.msgpack`

## 3.1 RRQ:

```
import msgpack

# Define data
rrq = {
    "opcode": 1,
    "filename": "File/to/path",
    "mode": "netascii",
}

# Write msgpack file
with open("data.msgpack", "wb") as outfile:
    packed = msgpack.packb(rrq)
    outfile.write(packed)

# Read msgpack file
with open("data.msgpack", "rb") as data_file:
    byte_data = data_file.read()

data_loaded = msgpack.unpackb(byte_data)
print(data_loaded)
print(rrq == data_loaded)
```

=> data.msgpack



filename

File/to/path opcode modenetascii

=> Saída: data.msgpack decodificado

python2 rrq.py

```
{'filename': 'File/to/path', 'opcode': 1, 'mode': 'netascii'}
True
```

## 3.1 WRQ:

```
import msgpack

# Define data
wrq = {
    "opcode": 2,
    "filename": "File/to/path",
    "mode": "netascii",
}

# Write msgpack file
with open("data.msgpack", "wb") as outfile:
    packed = msgpack.packb(wrq)
    outfile.write(packed)

# Read msgpack file
with open("data.msgpack", "rb") as data_file:
    byte_data = data_file.read()

data_loaded = msgpack.unpackb(byte_data)
print(data_loaded)
print(wrq == data_loaded)
```

=> data.msgpack

❖❖ filename❖  
File/to/path❖opcode❖modenetascii

=> Saída: data.msgpack decodificado

python2 wrq.py

{'filename': 'File/to/path', 'opcode': 2, 'mode': 'netascii'}  
True



## 3.1 DATA:

```
import msgpack

# Define data
data = {
    "opcode": 3,
    "block": 255,
    "mode": b'coisaQualquer',
}

# Write msgpack file
with open("data.msgpack", "wb") as outfile:
    packed = msgpack.packb(data)
    outfile.write(packed)

# Read msgpack file
with open("data.msgpack", "rb") as data_file:
    byte_data = data_file.read()

data_loaded = msgpack.unpackb(byte_data)
print(data_loaded)
print(data == data_loaded)
```

=> data.msgpack

coisaQualquer block

=> Saída: data.msgpack decodificado

python2 data.py

```
{'opcode': 3, 'mode': 'coisaQualquer', 'block': 255}
True
```

## 3.1 ACK:

```
import msgpack

# Define data
ack = {
    "opcode": 4,
    "block": 200,
}

# Write msgpack file
with open("data.msgpack", "wb") as outfile:
    packed = msgpack.packb(ack)
    outfile.write(packed)

# Read msgpack file
with open("data.msgpack", "rb") as data_file:
    byte_data = data_file.read()

data_loaded = msgpack.unpackb(byte_data)
print(data_loaded)
print(ack == data_loaded)
```

=> data.msgpack

❖❖opcode❖❖block❖❖

=> Saída: data.msgpack decodificado

python2 ack.py

```
{'opcode': 4, 'block': 200}
True
```

## 3.1 ERROR:

```
import msgpack

# Define data
error = {
    "opcode": 5,
    "errorcode": 7,
    "errmsg": "No such user",
}

# Write msgpack file
with open("data.msgpack", "wb") as outfile:
    packed = msgpack.packb(error)
    outfile.write(packed)

# Read msgpack file
with open("data.msgpack", "rb") as data_file:
    byte_data = data_file.read()

data_loaded = msgpack.unpackb(byte_data)
print(data_loaded)
print(error == data_loaded)
```

=> data.msgpack

❖❖ errorcodeerrmsg❖  
No such user❖opcode

=> Saída: data.msgpack decodificado

python2 error.py

{'errorcode': 7, 'errmsg': 'No such user', 'opcode': 5}  
True

## 3.1 LIST:

```
import msgpack

# Define data
list = {
    "opcode": 10,
    "caminho": "File/to/path",
}

# Write msgpack file
with open("data.msgpack", "wb") as outfile:
    packed = msgpack.packb(list)
    outfile.write(packed)

# Read msgpack file
with open("data.msgpack", "rb") as data_file:
    byte_data = data_file.read()

data_loaded = msgpack.unpackb(byte_data)
print(data_loaded)
print(list == data_loaded)
```

=> data.msgpack

❖❖caminho❖  
File/to/path❖opcode

=> Saída: data.msgpack decodificado

python2 list.py

{'opcode': 10, 'caminho': 'File/to/path'}  
True

## 3.1 LISTANS (resposta do LIST):

```
import msgpack

# Define data
listAns = {
    "opcode": 11,
    "elementos": ["filename", ("filename",
10000)],
}

# Write msgpack file
with open("data.msgpack", "wb") as outfile:
    packed = msgpack.packb(listAns)
    outfile.write(packed)

# Read msgpack file
with open("data.msgpack", "rb") as data_file:
    byte_data = data_file.read()

data_loaded = msgpack.unpackb(byte_data)
print(data_loaded)
print(listAns == data_loaded)
```

=> data.msgpack

❖❖opcode  
❖ elementos❖filename❖filename❖

=> Saída: data.msgpack decodificado

python2 listaAns.py

{'opcode': 11, 'elementos': ['filename', ['filename', 10000]]}  
False

## 3.1 MKDIR:

```
import msgpack

# Define data
mkdir = {
    "opcode": 12,
    "caminho": "path/to/dir",
}

# Write msgpack file
with open("data.msgpack", "wb") as outfile:
    packed = msgpack.packb(mkdir)
    outfile.write(packed)

# Read msgpack file
with open("data.msgpack", "rb") as data_file:
    byte_data = data_file.read()

data_loaded = msgpack.unpackb(byte_data)
print(data_loaded)
print(mkdir == data_loaded)
```

=> data.msgpack

◆◆caminho◆  
path/to/dir◆opcode

=> Saída: data.msgpack decodificado

python2 mkdir.py

{'opcode': 12, 'caminho': 'path/to/dir'}  
True

## 3.1 MKDIRANS (Resposta ao MKDIR):

```
import msgpack

# Define data
mkdirAns = {
    "opcode": 5,
    "errorcode": 8,
    "errmsg": "directory done",
}

# Write msgpack file
with open("data.msgpack", "wb") as outfile:
    packed = msgpack.packb(mkdirAns)
    outfile.write(packed)

# Read msgpack file
with open("data.msgpack", "rb") as data_file:
    byte_data = data_file.read()

data_loaded = msgpack.unpackb(byte_data)
print(data_loaded)
print(mkdirAns == data_loaded)
```

=> data.msgpack

❖❖ errorcoderrormsg❖directory done❖opcode

=> Saída: data.msgpack decodificado

python2 mkdirAns.py

{'errorcode': 8, 'errmsg': 'directory done', 'opcode': 5}  
True

## 3.1 MOVE:

```
import msgpack

# Define data
move = {
    "opcode": 13,
    "nome_original": "nomeLegal",
    "novo_nome": "nomeMaisLegal",
}

# Write msgpack file
with open("data.msgpack", "wb") as outfile:
    packed = msgpack.packb(move)
    outfile.write(packed)

# Read msgpack file
with open("data.msgpack", "rb") as data_file:
    byte_data = data_file.read()

data_loaded = msgpack.unpackb(byte_data)
print(data_loaded)
print(move == data_loaded)
```

=> data.msgpack

nome\_original♦omnomeLegal

=> Saída: data.msgpack decodificado

python2 move.py

```
{'opcode': 13, 'novo_nome': 'nomeMaisLegal', 'nome_original':  
'nomeLegal'}
```

True



## 3.1 MOVEANS (Resposta ao MOVE):

```
import msgpack

# Define data
moveAns = {
    "opcode": 5,
    "errorcode": 9,
    "errmsg": "sucesso",
}

# Write msgpack file
with open("data.msgpack", "wb") as outfile:
    packed = msgpack.packb(moveAns)
    outfile.write(packed)

# Read msgpack file
with open("data.msgpack", "rb") as data_file:
    byte_data = data_file.read()

data_loaded = msgpack.unpackb(byte_data)
print(data_loaded)
print(moveAns == data_loaded)
```

=> data.msgpack



errorcode

errmsg♦sucesso♦opcode

=> Saída: data.msgpack decodificado

python2 moveAns.py

```
{'errorcode': 9, 'errmsg': 'sucesso', 'opcode': 5}
True
```

## 4. Referências

Especificação do msgpack: <https://msgpack.org/#languages>

Tutorial em python: <https://github.com/msgpack/msgpack-python>