Testes de (de)serialização com AVRO

Documentação

Documentação oficial

Avro é um sistema de serialização de dados, provendo estruturas de dados ricas, um formato de dados binário, compacto e rápido.

Schemas

Avro depende de *schemas*. Quando dados Avro são lidos, o *schema* utilizado na escrita está sempre presente. Isso permite que cada dado seja escrito sem *overheads* por valor, fazendo com que a serialização seja rápida e pequena. Isso também facilita o uso de linguagens scripts dinâmicas, já que os dados, com seu *schema*, são completamente auto-descritivos.

Exemplo de schema:

```
{
    "namespace": "exemplo.avro",
    "type": "record",
    "name": "Usuario",
    "fields": [
        {"name": "campo_texto", "type": "string"},
        {"name": "campo_bool", "type": "boolean"},
        {"name": "campo_int", "type": ["int", "null"], "default": "1"},
        {"name": "campo_long", "type": "long"},
        {"name": "campo_float", "type": "float", "default": "0.0"},
        {"name": "campo_double", "type": "double", "default": "0.01"},
        {"name": "campo_bytes", "type": "bytes", "default": "\u00AC\u00FF"},
        {"name": "campo_enum", "type": "enum", "symbols": ["ATIVO", "INATIVO",
"EXCLUIDO"]},
        {"name": "campo_array", "type": "array", "items": "string"},
        {"name": "campo_record", "type": {
            "namespace":"exemplo.avro",
            "type":"record",
            "fields":[
                {"name": "subcampo_texto", "type": "string"},
                {"name": "subcampo_enum", "type": "enum", "symbols": [
                    "op1",
                    "enabled",
                    "disabled"
                ]}
        ]}}

    ]
}
```

namespace

Tipo complexo, uma string JSON que qualifica o nome do schema. É uma sequencia de nomes, separados por pontos.

type

Tipo da informação, podendo ser primitivos ou complexos:

Tipos primitivos:

- *null*: sem valor
- *boolean*: valor binário
- *int*: inteiro de 32 bits (com sinal)
- *long*: inteiro de 64 bits (com sinal)
- *float*: ponto flutuante de 32 bits
- *double*: ponto flutuante de 64 bits
- *bytes*: sequência de bytes
- *string*: sequencia de caracteres unicode

Tipos complexos:

- *record*: Registro para agrupar vários campos (Atributos: **name**, **fields**, namespace, doc, aliases)
- *enum*: Enumerador (Atributos: **name**, namespace, aliases, doc, **symbols**, default)
- *array*: Vetor (Atributo: **items**)
- *map*: Utilizado para dados opcionais e não padronizados
- *union*: Utilizado quando o tipo do dado pode ser dois schemas ou mais
- *fixed*: Informações fixas em relação ao tamanho (Atributos: **size**, **name**)

name

Record, enums e fixed são tipos nomeados. Cada um tem um nome completo composto por duas partes, o nome e o *namespace*. Nomes devem iniciar por letra (maiúscula ou minúscula), número ou *underline* [A-Za-z0-9_]

array

```
/* Schema utilizando array */
{
    "name": "Parent",
    "type": "record",
    "fields": [
        {
            "name": "children",
            "type": {
                "type": "array",
                "items": {
                    "name": "Child",
                    "type": "record",
```

```
                    "fields": [
                        {"name": "name", "type": "string"}
                    ]
                }
            }
        }
    ]
}
/* Objeto */
{
    "children": [
        {"name": "Guionardo"},
        {"name": "Otávio"}
    ]
}
```

map

```
/* Schema utilizando MAP */
{"namespace": "example.avro",
 "type": "record",
 "name": "Log",
 "fields": [
   {"name": "ip", "type": "string"},
   {"name": "timestamp",  "type": "string"},
   {"name": "message", "type": "string"},
   {"name": "additional", "type": {"type": "map", "values": "string"}}
  ]
}
/* Objeto */
{
    "ip": "172.18.80.109",
    "timestamp": "2015-09-17T23:00:18.313Z",
    "message": "blah blash",
    "additional": {
        "microseconds": "123",
        "thread": "http-apr-8080-exec-1147"
    }
}
```

Utilização

Python

Durante a escrita desta documentação, foram utilizados os seguintes pacotes disponíveis pelo PyPI e instaláveis pelo comando abaixo:

```
pip install avro-json-serializer avro-python3
```

Basicamente, a (de)serialização de objetos seguirá com a leitura/escrita de informações em formato binário, pois essa é a forma mais rápida e segura de validação por conter o schema utilizado na serialização.

Para isso, foi criado o módulo python *avro_helper* contendo a classe *AvroObject*. E para facilitar ainda mais a utilização, foram criadas duas funções para o processo.

avro_object_to_bin(data, schema) -> tuple

Esta função converte um objeto em sua representação avro em forma binária, utilizando um schema.

Retorno:

```
retorno = (
    sucesso: bool,
    bin_msg: bytes|str,
    schema_info :("namespace":str, "type":str, "name": str, "origin": tuple)
)
```

Seu retorno é uma tupla onde o primeiro elemento é um boolean indicando o sucesso da operação, o segundo é binário da conversão ou um string com a mensagem de erro e o terceiro é a informação do schema.

avro_bin_to_object(data) -> tuple

Esta função converte um binário AVRO para seu objeto de origem. Não é necessário schema, pois ele já está embutido no binário.

```
retorno = (
    sucesso: bool,
    bin_msg: array|str,
    schema_info :("namespace":str, "type":str, "name": str, "origin": tuple)
)
```

Seu retorno é uma tupla onde o primeiro elemento é um boolean indicando o sucesso da operação, o segundo é um array com o(s) objeto(s) original(is) ou um string com a mensagem de erro e o terceiro é a informação do schema.

```
from avro_helper import avro_object_to_bin, avro_bin_to_object

# Definindo um schema
schema = {
    "namespace": "teste",
    "name": "Usuario",
    "type": "record",
    "fields": [
        {"name": "id", "type": "int"},
```

```
        {"name": "nome", "type": "string"},
        {"name": "empresa", "namespace": "teste", "type": {
            "namespace": "teste",
            "name": "empresa",
            "type": "record",
            "fields": [
                {"name": "id", "type": "int"},
                {"name": "nome", "type": "string"}
            ]}}
    ]
}


# Definindo um objeto referente ao schema
objeto = {
    "id": 1,
    "nome": "Guionardo",
    "empresa": {
        "id": 5,
        "nome": "HBSIS"
    }
}
print('Objeto original', objeto, '\n')
# Objeto original {'id': 1, 'nome': 'Guionardo', 'empresa': {'id': 5, 'nome':
'HBSIS'}}

# Convertendo o objeto para um binário avro
# schema pode ser um dict, como no exemplo, mas também pode ser um string JSON, um
nome de arquivo ou uma URL com conteúdo JSON
ret = avro_object_to_bin(objeto, schema)
# O retorno da função é uma tupla, onde o primeiro elemento é boolean
representando o sucesso.
# O segundo elemento é o conteúdo ou uma mensagem de erro, dependendo do sucesso
# O terceiro elemento mostra as informações do schema utilizado

print('Retorno da conversão:', ret)
# Retorno da conversão: (True,
b'Obj\x01\x04\x14avro.codec\x08null\x16avro.schema\x92\x05{"type": "record",
"name": "Usuario", "namespace": "teste", "fields": [{"type": "int", "name": "id"},
{"type": "string", "name": "nome"}, {"namespace": "teste", "type": {"type":
"record", "name": "empresa", "namespace": "teste", "fields": [{"type": "int",
"name": "id"}, {"type": "string", "name": "nome"}]}, "name":
"empresa"}]}\x00\xe4k\xe6O\x8c\x1b\x93\x1b\x1f+\x90UH\xa3\x0b\xb7\x02$\x02\x12Guio
nardo\n\nHBSIS\xe4k\xe6O\x8c\x1b\x93\x1b\x1f+\x90UH\xa3\x0b\xb7')

# Simulando um erro de schema, alterei o type do id de 'int' para 'into'
# Retorno da conversão: (False, "Unknown named schema 'into', known names:
['teste.Usuario', 'teste.empresa'].")


if (ret[0]):
    print('\nConversão com sucesso')
    ret = avro_bin_to_object(ret[1])
    print('Objeto recuperado:', ret[1])
```

```
    # Objeto recuperado: (True, [{'id': 1, 'nome': 'Guionardo', 'empresa': {'id':
5, 'nome': 'HBSIS'}}])
    print('Schema:', ret[2])
    # Schema: {'namespace': 'teste', 'type': 'record', 'name': 'Usuario',
'origin': ('binary', None)}
else:
    print('Conversão com erro:', ret[1])
    # Conversão com erro: Unknown named schema 'into', known names:
['teste.Usuario', 'teste.empresa'].
```