5954025 - Sistemas Distribuídos

Aula 07 (prática) - Comunicação via HTTP: gRPC

Prof. Dr. Denis M. L. Martins
DCM | FFCLRP | USP

Objetivo da Aula

- Aprender a criar serviços RPC (Remote Procedure Call) utilizando gRPC e Protocol Buffers em Python.
- Entender como definir contratos de serviços, gerar código automático e implementar servidor e cliente.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Instalação

```
pip install --upgrade grpcio grpcio-tools protobuf
```

- **grpcio** é a biblioteca principal de gRPC para Python.
- **grpcio-tools** é necessário para gerar o código a partir dos arquivos **.proto**.

```
python -c "import grpc; print('grpc está instalado')"
```

Se aparecer *grpc está instalado*, está tudo pronto.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exemplo Prático

Contexto: Você está desenvolvendo um sistema de cadastro de livros em uma biblioteca digital.
O sistema precisa permitir:

- Cadastrar um novo livro (título, autor, ISBN, ano, status).
- Listar livros com filtros (autor, título).

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Estrutura do Projeto

```
library_grpc/
├── proto/
│   └── library.proto
├── server.py
├── client.py
```

O servidor implementará os métodos de serviço.

O cliente será responsável por invocar esses métodos.

> Nota: vamos construir passo-a-passo a implementação.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Definindo o Protocol Buffers (protobuf)

Primeiramente define-se a estrutura dos dados em um arquivo **.proto**:

```
syntax = "proto3";

package library;

enum BookStatus {
  UNKNOWN = 0;
  AVAILABLE = 1;
  BORROWED = 2;
}
message Book {
  string title = 1;
  string author = 2;
  string isbn = 3;
  int32 year = 4;
  BookStatus status = 5;
}
```

- **Obrigatório**: A primeira linha define a edição.
- Cada campo tem um número inteiro **único** entre 1 e 536.870.911
- Os números são usados para identificar os campos ao serializar/deserializar. Mais info aqui.
- Use **snake_case** para nome de campo.
- **CamelCase** para mensagem.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Tipos de Campos

- **string** — texto.
- **int32** — número inteiro (32 bits).
- **int64** — número inteiro (64 bits).
- **bool** — verdadeiro/falso.
- **enum** — valor enumerado.
- **oneof** — campo exclusivo.
- **repeated** — vetor de valores.
- **map** — dicionário chave-valor.

```
message CreateBookRequest {
  Book book = 1;
}

message CreateBookResponse {
  bool success = 1;
  string message = 2;
}

message ListBooksRequest {
  string author = 1;
}

message ListBooksResponse {
  repeated Book books = 1;
}
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Definindo Serviços

Um serviço define métodos RPC (Remote Procedure Call) com entrada e saída.

```
service LibraryService {
  rpc CreateBook(CreateBookRequest) returns (CreateBookResponse);
  rpc ListBooks(ListBooksRequest) returns (ListBooksResponse);
}
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Gerando código a partir de **book.proto**

Com **protoc**, você gera código em Python, Java, C++, etc.

Para Python:

```
python -m grpc_tools.protoc \
-I./proto \
--python_out=. \
--grpc_python_out=. \
./proto/library.proto
```

Isso gera:

- **library_pb2.py**: definição de mensagens
- **library_pb2_grpc.py**: stubs de servidor e cliente

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Implementando o Servidor

```
import grpc
from concurrent import futures
import library_pb2
import library_pb2_grpc

class LibraryService(library_pb2_grpc.LibraryServiceServicer):
    def __init__(self):
        self.books = []

    # Continua...
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Implementando o Cliente

```
import grpc
import library_pb2
import library_pb2_grpc

def print_book(book):
    print(book.title, "-", book.author)

def run():
    channel = grpc.insecure_channel('localhost:50051')
    stub = library_pb2_grpc.LibraryServiceStub(channel)
    stub.CreateBook(
        CreateBookRequest(
            # Continua...
        )
    )
    # Continua...
```

Resultado Esperado:

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Execução

Terminal 1:

```
python server.py
```

Terminal 2:

```
python client.py
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Próximos passos

- Ajuste o arquivo **.proto** para definir o serviço **BookService** com dois novos métodos:
  - **GetBookByIsbn**: retorna livro pelo ISBN.
  - **UpdateBookStatus**: atualiza o status do livro.
- Depois gere o código a partir do **.proto** e implemente
- Atualize o código no servidor e no cliente.
- Chame os dois novos métodos com dados de teste e exiba as respostas no console.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Dicas de Debug

- Verifique se o **.proto** está em **syntax = "proto3"**.
- Confira se o servidor está rodando na porta 50051.
- Verifique se os stubs foram importados corretamente.
- Use **logger** para verificar o estado dos dados.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Material Extra

- Tutorial gRPC em Python com implementação de diferentes métodos de chamada (unary, server streaming, etc.): https://www.velotio.com/engineering-blog/grpc-implementation-using-python
- Introdução ao gRPC-Python do CodeLabs da Google: https://codelabs.developers.google.com/grpc/getting-started-grpc-python?hl=pt-br#0
  -Spring Boot + gRPC (YouTube): https://www.youtube.com/watch?v=2Ub-liDVCeI

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br