

Relatório sobre a Implementação de Comunicação Remota via gRPC com Python

Julia da Rosa¹

¹Instituto Federal Catarinense – Campus Rio do Sul – Unidade Urbana (IFC)

julia.rosa.ifc.riodosul@gmail.com

Abstract. TODO

Resumo. TODO

1. Introdução

Neste relatório estaremos explorando o paradigma de comunicação RPC, observando como as entidades se comunicam em um sistema distribuído através de um exemplo didático e funcional em Python utilizando gRPC.

2. Chamada de Procedimento Remoto (RPC)

A Chamada de Procedimento Remoto (RPC, do inglês Remote Procedure Call) constitui um mecanismo essencial na computação distribuída, permitindo que procedimentos localizados em máquinas remotas sejam invocados como se estivessem no mesmo espaço de endereçamento do processo cliente.

Esse modelo abstrai os detalhes da comunicação em rede, ocultando processos complexos como a codificação e decodificação de parâmetros, a troca de mensagens entre os nós da rede e a manutenção da semântica de chamadas locais. Dessa forma, o RPC fornece uma camada de transparência que facilita o desenvolvimento de aplicações distribuídas.

A abordagem RPC é especialmente adequada para arquiteturas cliente-servidor, uma vez que os servidores disponibilizam operações por meio de interfaces bem definidas, e os clientes podem invocá-las diretamente, de maneira transparente, como se fossem funções locais.

Do ponto de vista de Sistemas Distribuídos, o gRPC permite a criação de sistemas robustos, desacoplados e eficientes, fundamentais para arquiteturas modernas como microserviços.

2.1. gRPC

Para a implementação foi escolhido o gRPC, uma estrutura de RPC de código aberto desenvolvida pela Google que utiliza HTTP/2 para transporte e Protocol Buffers (Protobuf) para serialização de mensagens, oferecendo alta performance e suporte a várias linguagens. Sua comunicação é binária, compacta e rápida, com suporte a multiplexação, streaming, compressão e bidirecionalidade.

Além do gRPC ser mais leve e rápido que REST tradicional, (utilizando JSON + HTTP/1.1), a padronização oferecida pela interface que fortemente tipada e automaticamente gerada via .proto facilita a implementação e entendimento do código.

3. Descrição da Implementação

Para tornar a aplicação mais intuitiva, optou-se por adotar um cenário didático inspirado em uma padaria. No lugar da terminologia genérica "produzir" e "consumir", foram definidos dois métodos no serviço gRPC:

`Assar(ProdutoRequest)`: simula a produção de um item de padaria, adicionando-o a um estoque (lista interna).

`Vender(VenderRequest)`: representa a venda de um item ao cliente solicitante, retirando-o do estoque.

Esses métodos fazem parte da interface `ProdutoService`, definida em um arquivo `.proto`, conforme o exemplo a seguir:

```
syntax = "proto3";

package produto;

service ProdutoService {
    rpc Assar(ProdutoRequest) returns (ProdutoResponse);
    rpc Vender(VenderRequest) returns (ProdutoResponse);
}

message ProdutoRequest {
    string nome = 1;
}

message VenderRequest {
    string solicitante = 1;
}

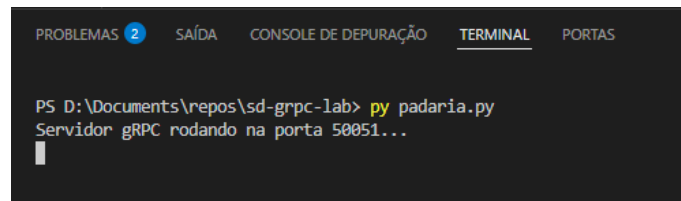
message ProdutoResponse {
    string mensagem = 1;
}
```

O servidor, ou no nosso exemplo *padaria.py*, implementa essa interface utilizando a linguagem Python e a biblioteca gRPC. A classe `ProdutoService` mantém um atributo `estoque`, que é manipulado pelos dois métodos principais. A seguir, um trecho do método `Assar`:

```
def Assar(self, request, context):
    self.estoque.append(request.nome)
    mensagem = f"Produto {request.nome} produzido com sucesso!"
    return produto_pb2.ProdutoResponse(mensagem=mensagem)
```

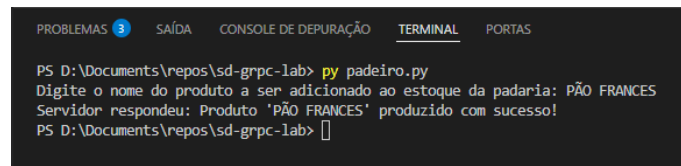
E, para o método `Vender`:

```
def Vender(self, request, context):
    if self.estoque:
        produto = self.estoque.pop(0)
```

A terminal window with tabs for PROBLEMAS (2), SAÍDA, CONSOLE DE DEPURACÃO, TERMINAL, and PORTAS. The terminal shows the command 'py padaria.py' being executed in the directory 'D:\Documents\repos\sd-grpc-lab'. The output is 'Servidor gRPC rodando na porta 50051...' followed by a cursor.

```
PS D:\Documents\repos\sd-grpc-lab> py padaria.py
Servidor gRPC rodando na porta 50051...
```

Figura 1. Executando *pypadaria.py* no terminal.

A terminal window with tabs for PROBLEMAS (3), SAÍDA, CONSOLE DE DEPURACÃO, TERMINAL, and PORTAS. The terminal shows the command 'py padeiro.py' being executed. The output prompts for a product name, receives 'PÃO FRANCES', and returns 'Produto 'PÃO FRANCES' produzido com sucesso!' followed by a cursor.

```
PS D:\Documents\repos\sd-grpc-lab> py padeiro.py
Digite o nome do produto a ser adicionado ao estoque da padaria: PÃO FRANCES
Servidor respondeu: Produto 'PÃO FRANCES' produzido com sucesso!
PS D:\Documents\repos\sd-grpc-lab> 
```

Figura 2. Executando *pypadeiro.py* no terminal.

```
mensagem = f"{{request.solicitante}}■comprou■o■produto■"{{produto}}
else:
mensagem = "Estoque■vazio.■Nenhum■produto■para■vender."
return produto_pb2.ProdutoResponse(mensagem=mensagem)
```

Dois clientes foram desenvolvidos:

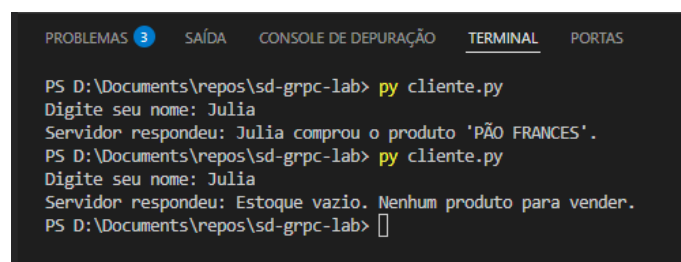
O cliente *"padeiro"* realiza chamadas ao método *Assar* para adicionar produtos ao estoque.

O cliente *"consumidor"* realiza chamadas ao método *Vender*, consumindo os produtos disponíveis.

Para executar esse cenário precisamos também preparar o ambiente com a execução do comando `"python -mgrpc_tools.protoc -I = protos --python_out = . --grpc_python_out = . produto.proto"` no projeto de tal forma que *grpc_python_out* seja o caminho relativo para o arquivo *produto.proto* a partir do diretório onde o comando foi executado para que os arquivos *produto_pb2.py* e *produto_pb2_grpc.py* sejam criados automaticamente onde o comando foi executado. Esses arquivos devem estar no mesmo diretório onde o servidor, *padaria.py*, está ou em um local visível via import. Para que então o servidor possa ser executado.

4. Conclusão

TODO

A terminal window with tabs for PROBLEMAS (3), SAÍDA, CONSOLE DE DEPURACÃO, TERMINAL, and PORTAS. The terminal shows the command 'py cliente.py' being executed. The output prompts for a name, receives 'Julia', and returns 'Julia comprou o produto 'PÃO FRANCES''. Then it prompts for a name again, receives 'Julia', and returns 'Estoque vazio. Nenhum produto para vender.' followed by a cursor.

```
PS D:\Documents\repos\sd-grpc-lab> py cliente.py
Digite seu nome: Julia
Servidor respondeu: Julia comprou o produto 'PÃO FRANCES'.
PS D:\Documents\repos\sd-grpc-lab> py cliente.py
Digite seu nome: Julia
Servidor respondeu: Estoque vazio. Nenhum produto para vender.
PS D:\Documents\repos\sd-grpc-lab> 
```

Figura 3. Executando *pycliente.py* – *semestoque.py* no terminal.

Referências

COULOURIS, George et al. Sistemas distribuídos: conceitos e projeto. 5. ed. Porto Alegre: Bookman, 2013. ISBN 978-85-8260-054-2.