Protocol Buffer
Protobuf - uma alternativa ao JSON e XML

QXD0099 - Desenvolvimento de Software para Persistência

Universidade Federal do Ceará - *Campus* Quixadá

Prof. Francisco Victor da Silva Pinheiro
victorpinheiro@ufc.br

Agenda

- Motivação
- O que é Protobuf?
- Como usar?
  - Instalando o Protobuf
- Definição de schema (arquivo .proto)
- Tipos de dados suportados
- Como Funciona o Protobuf?
  - Serialização e Desserialização
  - Geração de Código
  - Exemplo de Serialização e Desserialização
- Quando utilizar?
- E quando JSON é melhor?
- Protocolo entre dois sistemas

Motivação

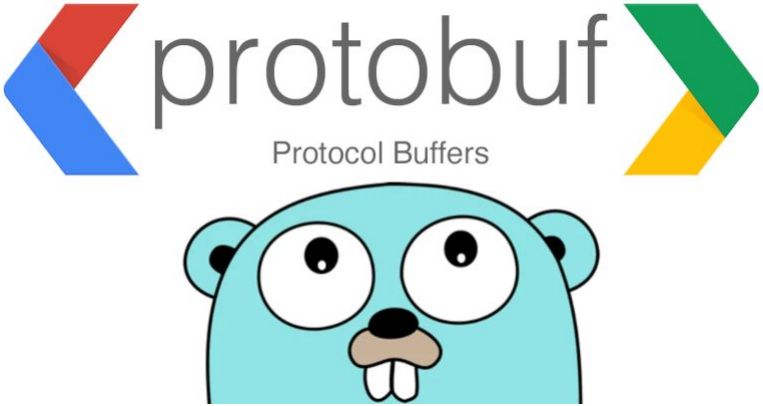
- Normalmente, a comunicação de serviços ocorre através de HTTP usando o formato JSON.
- Embora o JSON possui muitas vantagens óbvias como formato de intercâmbio de dados — ele é legível para humanos, bem compreendido e normalmente funciona bem — ele também tem seus problemas.
- Para os casos no qual os dados não são consumidos diretamente por Javascript em navegadores, como por exemplo em serviços internos, pode ser que formatos estruturados — como o Protobuf — seja uma melhor opção para codificar dados.

O que é Protobuf?

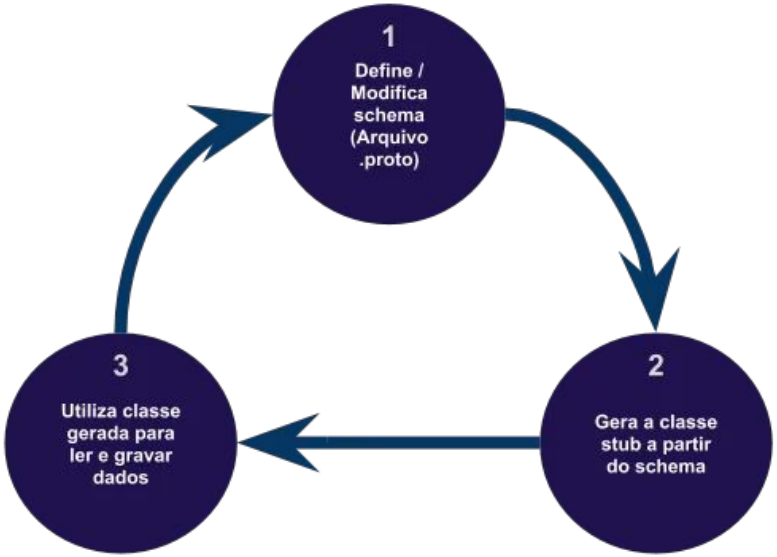
- Protobuf (sigla de Protocol buffers) é um mecanismo criado e usado pelo Google para serializar dados estruturados.

- Compacto e rápido.
- Suporta múltiplas linguagens de programação.
- *Mais eficiente que formatos como JSON e XML em termos de tamanho e velocidade de processamento.*

- **Onde Protobuf é usado?**
  - APIs, microserviços, armazenamento de dados, comunicação entre dispositivos, etc.



O que é Protobuf?



- Primeiro é definido como deseja que os dados sejam estruturados — em um arquivo de extensão .proto.
- Em seguida, esta definição é compilada e o resultado é um código-fonte automaticamente gerado na linguagem desejada — no momento que escrevo este post, as linguagens compatíveis são C++, C#, Go, Java e Python.
- Finalmente, código-fonte gerado é utilizado para gravar e ler os dados estruturados.
- Sempre que houver mudança na estrutura dos dados, o ciclo se repetirá.

Como usar?

- Primeiramente será necessário configurar o seu ambiente, para isto será necessário instalar o Protobuf, para então ter acesso ao protoc , que é compilador utilizado para gerar as classes stub.

- **Instalando o Protobuf**
  - Acesse este link https://github.com/protocolbuffers/protobuf/releases e baixe o pacote com o nome protoc.
  - Extraia o conteúdo do pacote em qualquer local e siga as instruções do README do pacote.

- **Dica para usuários linux:** Rode os comandos abaixo, que servem para mover os arquivos para as pastas corretas e configurar permissão.
  - *sudo mv protoc3/bin/* /usr/local/bin/*
  - *sudo mv protoc3/include/* /usr/local/include/*
  - *sudo chown [user] /usr/local/bin/protoc*
  - *sudo chown -R [user] /usr/local/include/google*

- Com tudo movido e configurado, rode o comando **protoc --version** e verifique se está tudo funcionando.
- Após o ambiente preparado, vamos começar pelo início, como descrito acima, criando um arquivo **.proto**

Definição de schema (arquivo .proto)

- O arquivo .proto define como os dados serão estruturados

- syntax = "proto3"; - Define a versão do Protobuf.
- message - Define um tipo de dado (como uma classe ou estrutura).
- Campos numéricos após o tipo de dado (name = 1, id = 2, etc.) indicam o identificador único do campo.

```
syntax = "proto3";

message Person {
  string name = 1;
  int32 id = 2;
  string email = 3;
}
```

(arquivo .proto)

```
syntax = "proto3";

package movie;

message Movie {
  enum Role {
    CHARACTER = 0;
    SCREENPLAY = 1;
    DIRECTOR = 2;
  }
  message Person {
    string name = 1;
    Role role = 2;
  }
  string name = 1;
  double releaseDate = 2;
  string overview = 3;
  repeated Person featuredCrew = 4;
}

message MovieList {
  repeated Movie movies = 1;
}
```

- **Proto3 Syntax:**
  - Declaração inicial syntax = "proto3"; define que o código usa a versão Proto3, onde todos os campos são opcionais por padrão.
- **Definição de Pacote:**
  - package movie; organiza as mensagens em um namespace para evitar conflitos.
- **Mensagens e Campos:**
  - Movie e MovieList são os tipos principais.
  - Campos são identificados por nomes e números únicos para facilitar a serialização/deserialização.
- **Tipos de Campos:**
  - Tipos simples como string, double.
  - repeated permite listas de valores (ex.: featuredCrew e movies).
- **Enum e Submensagem:**
  - Role: Enumeração de funções relacionadas a filmes.
  - Person: Submensagem dentro de Movie, representando pessoas e seus papéis.

Tipos de dados suportados

- Primitivos (string, int32, bool, etc.).
- Tipos compostos (message, enum).
- Repetições de campos (listas).
- **Versão 2 vs Versão 3 do Protobuf:**
  - Proto3 é mais simples e moderno, removendo elementos como required e valores padrão explícitos, enquanto Proto2 oferece maior flexibilidade, mas é mais complexo de usar e validar.

Como Funciona o Protobuf?

- **Serialização**
  - Os dados definidos no esquema são convertidos em um formato binário compacto.
  - Vantagem: O binário é mais eficiente que JSON ou XML em termos de tamanho e velocidade.

- **Deserialização:**
  - O receptor usa o mesmo esquema para interpretar o binário e reconstruir os dados originais.

```
import person_pb2

# Criar uma instância de Person
person = person_pb2.Person()
person.name = "Alice"
person.id = 1234
person.email = "alice@example.com"
```

```
# Serializar
data = person.SerializeToString()

# Desserializar
new_person = person_pb2.Person()
new_person.ParseFromString(data)
print(new_person)
```

Como Funciona o Protobuf?

- **Geração de Código**
  - O Protobuf gera classes em linguagens como Python, Java, C++, etc.
    - Essas classes incluem métodos para serializar e deserializar os dados.
  - Como usar o compilador protoc para gerar código em várias linguagens (Python, Java, C++, Go, etc.).
  - Exemplo de comando para gerar código Python
    - protoc --python_out=. person.proto

Exemplo de Serialização e Desserialização

- **Arquivo .proto**:
  - Defina a estrutura dos dados em um arquivo chamado **person.proto**

```
syntax = "proto3";

message Person {
  string name = 1;
  int32 age = 2;
  string email = 3;
}
```

- **Gerar o Código Python**
- Use o compilador Protobuf (protoc) para gerar o arquivo Python com as classes necessárias:
  - protoc --python_out=. person.proto
  - C:\protoc\bin\protoc --python_out=. person.proto

Exemplo de Serialização e Desserialização

```python
import person_pb2  # Importa o arquivo gerado pelo Protobuf

# Serialização
person = person_pb2.Person()
person.name = "Alice"
person.age = 30
person.email = "alice@example.com"

# Converte para binário (serialização)
serialized_data = person.SerializeToString()
print(f"Serialized data: {serialized_data}")

# Desserialização
new_person = person_pb2.Person()
new_person.ParseFromString(serialized_data)

print("\nDeserialized data:")
print(f"Name: {new_person.name}")
print(f"Age: {new_person.age}")
print(f"Email: {new_person.email}")
```

- **Serialização:**
  - A estrutura de dados Person é convertida para um formato binário compacto.
  - Usamos SerializeToString() para gerar os dados serializados.
- **Desserialização:**
  - O binário é interpretado de volta na estrutura de dados original.
  - Usamos ParseFromString() para reconstruir o objeto.

- **Pacote Protobuf**: Certifique-se de instalar o Protobuf em Python:
  - pip install protobuf

Exemplo de Serialização e Desserialização

- Saída Esperada
  - Após executar o código, você verá algo como:
  - **Dados Serializados:**

```
Serialized data: b'\n\x05Alice\x10\x1e\x1a\x11alice@example.com'
```

  - **Dados Desserializados:**

```
Deserialized data:
Name: Alice
Age: 30
Email: alice@example.com
```

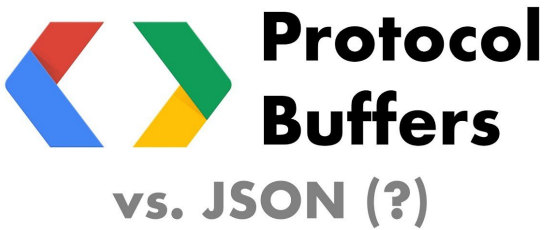
Quando utilizar?

- **Objetivo principal:**
  - JSON é simples e amplamente suportado, mas é focado apenas na troca de dados em texto.
  - Protobuf é mais completo, com suporte a esquemas, validação e compactação binária.

- **Razões para usar Protobuf:**
  - **Schemas definidos:** Garante integridade dos dados durante o transporte.
  - **Compatibilidade de versões:** Facilita a evolução do esquema sem quebras.
  - **Menos duplicação de código:** Classes geradas automaticamente a partir do esquema eliminam a necessidade de escrever parsing manual.
  - **Validação e flexibilidade:** Suporte a palavras-chave como required e optional para controle rigoroso de dados.
  - **Interoperabilidade:** Suporte nativo para várias linguagens, simplificando integração entre serviços.

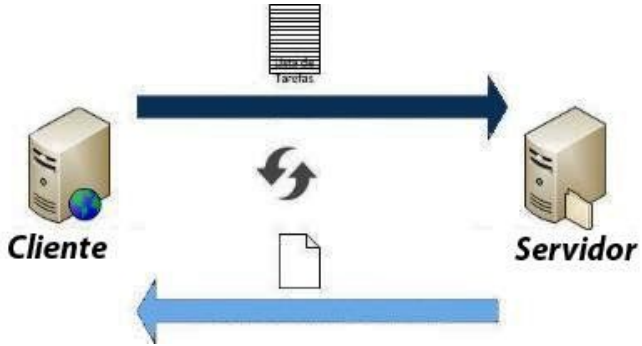
E quando JSON é melhor?

- Algumas vezes que o JSON é melhor que o Protobuf, como por exemplo nas situações em que:
  - É necessário que os dados sejam legíveis para humanos.
  - Os dados do serviço são consumidos diretamente por um web browser.
  - Sua aplicação server side é escrita em JavaScript.
  - Você não está preparado para vincular o modelo de dados a um esquema, por exemplo, talvez seus dados são dinâmicos.
  - A sua aplicação não consome tanta banda assim.
  - E provavelmente muitas outras situações.



Protocolo entre dois sistemas

- Simulação de um serviço que envia dados com Protobuf e outro que os recebe:
  - Usar o Python para criar um simples servidor e cliente usando sockets para enviar/receber dados serializados em Protobuf.



Servidor

```python
import socket
import person_pb2

def create_person():
    person = person_pb2.Person()
    person.name = "Bob"
    person.id = 5678
    person.email = "bob@example.com"
    return person.SerializeToString()

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(('localhost', 5000))
server.listen(1)
print("Servidor aguardando conexão...")

conn, addr = server.accept()
print(f"Conexão de {addr}")

data = create_person()
conn.sendall(data)
conn.close()
```

- **Importações**:
  - socket: Para comunicação de rede.
  - person_pb2: Contém a definição da classe Person gerada pelo Protobuf.
- **Função create_person()**:
  - Cria um objeto Person, preenche seus campos (name, id, email) e o serializa para uma string de bytes.
- **Configuração do servidor**:
  - Cria um socket TCP, associa-o ao endereço local (localhost) e à porta 5000, e coloca o servidor em modo de escuta.
- **Aceitação da conexão**:
  - Aguarda um cliente se conectar, aceita a conexão e exibe o endereço do cliente.
- **Envio de dados**:
  - Serializa os dados da pessoa e envia-os para o cliente. Após o envio, a conexão é fechada.

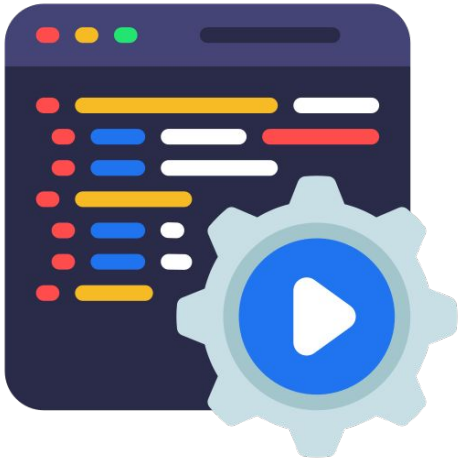
Cliente

```
import person_pb2
import socket

# Conectar ao servidor
client = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
client.connect(('localhost', 5000))

# Receber os dados
data = client.recv(1024)

# Desserializar os dados
person = person_pb2.Person()
person.ParseFromString(data)

# Verificar os campos
print(f"Recebido: {person.name}, {person.id},
{person.email}")  # Isso deve funcionar
client.close()
```

- **Protocol Buffers**, desserializa esses dados e exibe os campos.
  - **Conectar ao servidor**: O cliente cria um socket TCP/IP e se conecta ao servidor na porta 5000.
  - **Receber dados**: O cliente recebe até 1024 bytes de dados enviados pelo servidor.
  - **Desserializar**: Os dados recebidos são convertidos de volta para um objeto Person usando Protobuf.
  - **Exibir os dados**: O cliente exibe os campos name, id e email do objeto Person.
  - **Fechar a conexão**: A conexão com o servidor é fechada.

Referências

- GOOGLE. Protocol Buffers Documentation. Disponível em: https://protobuf.dev/. Acesso em: 21 nov. 2024.
- TEMPORIN, Tiago. O que é e como utilizar protocol buffers. Aprenda Golang, 22 jun. 2023. Disponível em: https://aprendagolang.com.br/o-que-e-e-como-utilizar-protocol-buffers/. Acesso em: 21 nov. 2024.
- JEAN, Clément. Protocol Buffers Handbook: Getting deeper into Protobuf internals and its usage. Birmingham: Packt Publishing, 2024.



Obrigado!

Dúvidas?

Universidade Federal do Ceará - *Campus* Quixadá

Prof. Francisco Victor da Silva Pinheiro
victorpinheiro@ufc.br