GOOGLE

PROTOBUF

Paulo Henrique, Willian Abdon e Paulo Félix

O QUE É PROTOBUF?

O Protocol Buffer é um método de serialização de dados estruturados criado pela Google que promete ser flexível e eficiente.

Inicialmente utilizado internamente, a Google tinha milhares de formatos de dados diferentes para trocar mensagens pela rede e era necessária uma forma simples de realizar essa tarefa.

O QUE ELA RESOLVE?

O compartilhamento de dados via aplicação,
independente da linguagem utilizada.

Pense em XML, só que menor, mais rápido e mais simples.

MAS POR QUÊ NÃO XML?

NÃO É ESCALÁVEL

Não funcionaria bem quando as máquinas e links de redes estivessem operando em capacidade máxima

COMPLEXIDADE

Se torna pesado escrever código para trabalhar com estruturas mais complicadas

ALÉM DISSO...

- SÃO DE 3 A 10 VEZES MENOR QUE O XML
- DE 20 A 100 VEZES MAIS RÁPIDO
- GERAM CLASSES DE ACESSO DE DADOS MAIS FÁCEIS DE USAR PROGRAMATICAMENTE
- SÃO MENOS AMBÍGUOS

COMO FUNCIONA?

ENTENDENDO O PROTOBUF

Deve ser especificado como a informação que será serializada deve ser estruturada definindo um tipo protobuf message em um arquivo .proto.

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

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    required string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }

  repeated PhoneNumber phone = 4;
}
```

Cada tipo mensagem tem um ou mais campos numerados unicamente e cada campo possui um nome e um tipo de valor. Os tipos de vlaor podem ser *number (integer* ou *floating-point), booleans*, *strings, raw bytes* ou até mesmo outro *protobuf message type*, permitindo que sua estrutura de dados seja hierárquica.

Depois de definir a mensagem, deve-se rodar o compilador do protocol buffer para a linguagem da sua aplicação no arquivo .proto e gerar as classes de acesso de dados. Essas classes geram simples métodos de acesso para cada campo (por exemplo name( ) e set_name( )), como também métodos para serializar/parsear toda a estrutura de/para raw bytes.

EXEMPLO

Se compilarmos nossa mensagem para C++, o exemplo anterior irá gerar uma classe Person e teremos o seguinte:

EXEMPLO

PARA ENVIAR

```
Person person;
person.set_name("John Doe");
person.set_id(1234);
person.set_email("jdoe@example.com");
fstream output("myfile", ios::out | ios::binary);
person.SerializeToOstream(&output);
```

EXEMPLO

AO RECEBER

```
fstream input("myfile", ios::in | ios::binary);
Person person;
person.ParseFromIstream(&input);
cout << "Name: " << person.name() << endl;
cout << "E-mail: " << person.email() << endl;
```

Podemos adicionar novos campos a mensagem sem quebrar compatibilidades anteriores. Os binários antigos simplesmente ignoram o novo campo quando estiverem fazendo o parse. Ou seja, se você utiliza protobuf como seu *data format*, você pode estender seu protocolo sem sem preocupar se o código já existente irá quebrar.

PROTO VS XML

CLASSE PERSON

XML

```xml
<person>
  <name>John Doe</name>
  <email>jdoe@example.com</email>
</person>
```

PROTO

```
# Textual representation of a protocol buffer.
# This is *not* the binary format used on the wire.
person {
  name: "John Doe"
  email: "jdoe@example.com"
}
```

MAS O QUE ESSA DIFERENÇA REPRESENTA?

Quando uma mensagem do protobuf é transformada em binário, provavelmente terá o tamanho de 28 bytes e leva de 100 a 200 nanosegundos para parsear. A versão XML tem pelo menos 69 bytes, se você remover os espaços e provalvemente deve levar de 5000 a 10000 nanosegundos para parsear.

Quando uma mensagem do protobuf é transformada em binário, provavelmente terá o tamanho de 28 bytes e leva de 100 a 200 nanosegundos para parsear. A versão XML tem pelo menos 69 bytes, se você remover os espaços e provalvemente deve levar de 5000 a 10000 nanosegundos para parsear.

PROTO VS XML

CONSUMINDO O ARQUIVO

XML

```
cout << "Name: "
     << person.getElementsByTagName("name")->item(0)->innerText()
     << endl;
cout << "E-mail: "
     << person.getElementsByTagName("email")->item(0)->innerText()
     << endl;
```

PROTO VS XML

CONSUMINDO O ARQUIVO

PROTO

```
cout << "Name: " << person.name() << endl;
cout << "E-mail: " << person.email() << endl;
```

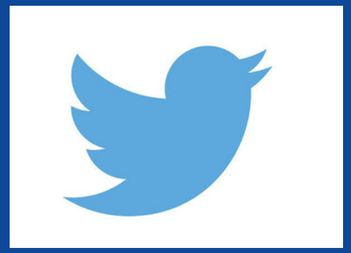
PORÉM...

PROTOBUF NEM SEMPRE É UMA SOLUÇÃO MELHOR QUE XML

Protocol Buffers não é uma boa solução para modelar um documento *text-based* com *markup*, como por exemplo HTML. Além disso, XML é *human-readable* e *human-editable*, enquanto o protobuf, na forma nativa, não é. Um *protocol buffer* só faz sentido se você tiver a definição da mensagem (o arquivo .proto).

ONDE ENCONTRAR?

QUAIS EMPRESAS ESTÃO USANDO PROTOBUF?



ONDE ENCONTRAR?

STACKSHARE.IO/PROTOBUF

Acesse e descubra quais empresas e desenvolvedores
estão usando Protobuf em sua stack.

LEITURA INTERESSANTE

HADOOP AT TWITTER

LEITURA INTERESSANTE

HOW PROTOCOL BUFFERS ARE USED IN MESOS FRAMEWORK DEVELOPMENT

GOOGLE PROTOBUF!

OBRIGADO!

Paulo Henrique, Willian Abdon e Paulo Félix