Protocol Buffers

mdo@ufc.br

Introdução

- Mecanismo para Serializar dados estruturados.
- Extensível
- Neutro em linguagem e em plataforma

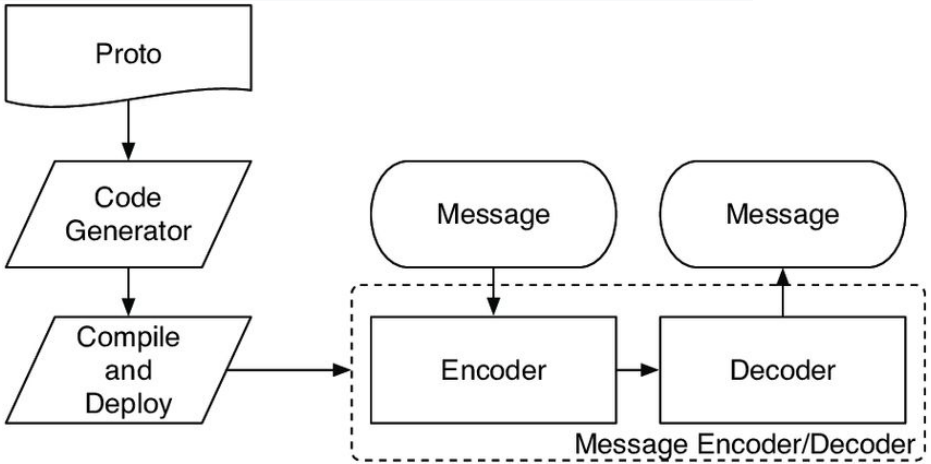
```
message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;
}
```

```
Person john = Person.newBuilder()
    .setId(1234)
    .setName("John Doe")
    .setEmail("jdoe@example.com")
    .build();
output = new FileOutputStream(args[
john.writeTo(output);
```

```
Person john;
fstream input(argv[1],
    ios::in | ios::binary);
john.ParseFromIstream(&input);
id = john.id();
name = john.name();
email = john.email();
```

Por que usar Protocol Buffer?

- Mecanismo flexível, eficiente e automatizado
- Pense em XML, mas menor, mais rápido e mais simples.
- **Defina** seus dados estruturados (.proto)
- **Gere** código para *serializar* e *deserializar* com facilidade seus dados estruturados
- Os dados Estruturados podem ser atualizados
  - Basta recompilar



Como Funciona?

```
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;
}
```

```
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);
```

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

Protocol Buffer vs XML

- São mais simples
- São 3 a 10 vezes menores
- São 20 a 100 vezes mais
- rápidas
- São menos ambíguos
- Código de serialização/deserialização gerado automaticamente e mais fáceis de usar

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

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

Desvantagens em relação ao XML

- O XML é possui leitura e edição amigável para pessoas
  - Mais fácil de “debugar”
- XML é auto descrito
  - Enquanto protocol Buffer devemos conhecer o .proto que descreve o Objeto

Serialização e Deserialização em JAVA

- `byte[] toByteArray();` serializar a mensagem e retornar um array de bytes que contém seus bytes brutos
- `static Person parseFrom(byte[] data);`: deserializar um OBJETO (MESSAGE) a partir de um array de bytes
- `void writeTo(OutputStream output);`: serializar o objeto e o escrever num `OutputStream`.
- `static Person parseFrom(InputStream input);`: ler e deserializar a mensagem a partir de um `InputStream`.

Serialização e Deserialização em JAVA

- `SerializeToString()`: serializar o objeto e o retorna como uma String, mas os dados estão em binário não textual.

- `ParseFromString(data)`: deserializar o objeto a partir de uma String.

Message

- Define um formato/Classe
  - Sintaxe simples
- Campos (Fields) devem ser identificados por índices numéricos
- Campos possuem um nome, um tipo e um descritor (campo obrigatório ou não)
- Messages podem Importar ou Herdar(subclasses) outras Messages

```
message SearchResponse {
  message Result {
    string url = 1;
    string title = 2;
    repeated string snippets = 3;
  }
  repeated Result results = 1;
}
```