Google Protocol Buffers

by

Dr. Günter Kolousek

Protobuf Buffers

- Mechanismus zum Serialisieren von Daten
- Unabhängig von Plattform
- entwickelt in C++
- aber: unabhängig von Programmiersprache
  - Java, C++, Python, C#, Dart, Go, Objective-C, Ruby, PHP, JavaScript, Go
  - weitere Implementierungen: Julia, Rust, Swift,…
- Basierend auf einem Schema (IDL)
- nicht selbst-beschreibend
  - d.h. ohne IDL…
- Google, aber: open source

Features

- flexibel
- effizient
- einfach
- neue Felder können leicht hinzugefügt werden

Einsatz und Alternativen

- Einsatz
  - Microservices
  - (meist) intern
- Alternativen
  - XML
    - kleiner um Faktor 3 bis 10
    - schneller um Faktor 20 bis 100
  - JSON, YAML,
  - BSON, CBOR, UBJSON, ASN.1,…
  - FlatBuffers, Thrift, MessagePack,…

Datentypen

- variable Längenkodierung
  - `int32`, `in64`
    - wenn negativ, dann besser: `sint32` bzw. `sint64`
    - nur positiv, dann besser: `uint32` bzw. `uint64`
  - `string`
  - `bytes`
- fixe Länge
  - `float`, `double`
  - nur positiv: `fixed32` bzw. `fixed64`
  - auch negativ: `sfixed32` bzw. `sfixed64`
  - `bool`
- nicht vorhanden, dann default-Wert (”Nullwert”)!
- Maps
  - `map<key_type, value_type> map_field = N;`
  - `key_type`: entweder skalar oder `string`!

Beispiel

```
syntax = ”proto3”;
package tutorial;

message Person {
  string name = 1;  // optional per default
  int32 id = 2;  // also:
  string email = 3;

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

  message PhoneNumber {
    string number = 1;
    PhoneType type = 2;
  }

  repeated PhoneNumber phone = 4;
}
```

Workflow

1. Definition der Nachrichten (`.proto`-Datei)
2. aus `.proto`-Datei Zugriffscode Code generieren
3. Implementierung
4. Starten

Definition des Service

messages.proto:

```
syntax = ”proto3”;

message Dummy {
  string text = 1;
}
```

Zugriffscode generieren

```
$ protoc --cpp_out=build --proto_path=.\
  messages.proto
...
```

- erzeugt:
  - `messages.pb.cc`
  - `messages.pb.h`

Implementierung

```
#include <iostream>

#include "messages.pb.h"

using namespace std;

int main() {
    GOOGLE_PROTOBUF_VERIFY_VERSION;
    Dummy *d = new Dummy;
    d->set_text("Hello World");
    cout << d->text() << endl;
    delete d;
    google::protobuf::ShutdownProtobufLibrary();
}
```

Übersetzen & Starten

- Übersetzen: am Besten mittels `meson`
- Starten

```
$ go
Hello World
```

Length-prefix Framing

> *If you want to write multiple messages to a single file or stream, it is up to you to keep track of where one message ends and the next begins. The Protocol Buffer wire format is not self-delimiting, so protocol buffer parsers cannot determine where a message ends on their own. The easiest way to solve this problem is to write the size of each message before you write the message itself. When you read the messages back in, you read the size, then read the bytes into a separate buffer, then parse from that buffer. (If you want to avoid copying bytes to a separate buffer, check out the `CodedInputStream` class (in both C++ and Java) which can be told to limit reads to a certain number of bytes.)*

https://developers.google.com/protocol-buffers/docs/techniques#streaming →
Zählmethode…