Protocol Buffers: Marshalling
Introduction to Middleware

Vojtěch Horký    Petr Tůma

Department of Distributed and Dependable Systems
Faculty of Mathematics and Physics
Charles University

2010 – 2021
Outline

1. Technology Overview
2. Assignment Part I
3. Message Encoding
4. Message Specification
5. Message Manipulation
6. Assignment Part II
Technology Overview

Goals
Provide platform independent structured data serialization framework.

Features
- Platform independent data description language.
- Serialization code generation for multiple languages (C++, Java, Python, Go, Ruby, JavaScript, Objective C, C# ...).
- Binary transport format with compact data representation.
- Textual transport using JSON.

... http://developers.google.com/protocol-buffers
Examples To Play With ...

> git clone http://github.com/d-iii-s/teaching-introduction-middleware.git

**C**

> cd teaching-introduction-middleware/src/protocol-buffers-basic-usage/c
> cat README.md

**Java**

> cd teaching-introduction-middleware/src/protocol-buffers-basic-usage/java
> cat README.md

**Python**

> cd teaching-introduction-middleware/src/protocol-buffers-basic-usage/python
> cat README.md
syntax = "proto3";

package example;

message AnExampleMessage {
  uint32 some_integer = 1;
  sint32 another_integer = 2;
  string some_string = 8;
  repeated string some_more_strings = 11;
}

message MoreExampleMessages {
  repeated AnExampleMessage messages = 1;
}
Outline

1. Technology Overview
2. Assignment Part I
3. Message Encoding
4. Message Specification
5. Message Manipulation
6. Assignment Part II
Assignment

Server
Implement a server that will provide information on current time.
- The server should accept a spec of what fields to return.
- Fields should be standard YYYY-MM-DD HH:MM:SS.

Client
Implement a client that will query server time:
- Pick a random combination of fields.
- Query information on current time.
- Print the time.

Interoperability
Implement compatible clients and servers in two languages.
Outline

1. Technology Overview
2. Assignment Part I
3. Message Encoding
4. Message Specification
5. Message Manipulation
6. Assignment Part II
Message Encoding

Goals
Compact structure with support for field removal and addition.

Features
- Sequence of field key value pairs.
- Key is field index and type indication.
  - One of variable integer, explicit length, fixed length.
  - Not enough to tell the exact field type!
- Primitive repeated fields packed.
- Total length not included!
**Variable Length Encoding**

### Goals
Support integers clustered around zero more efficiently.

### Features
- Integer stored as variable number of 7 bit values.
- High bit set to zero for last byte.
- Little endian byte order.
- Signed variant.
Outline

1. Technology Overview
2. Assignment Part I
3. Message Encoding
4. Message Specification
5. Message Manipulation
6. Assignment Part II
# Primitive Field Types

## Integer Types

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s)fixed(32</td>
<td>64)</td>
</tr>
<tr>
<td>(u)int(32</td>
<td>64)</td>
</tr>
<tr>
<td>sint(32</td>
<td>64)</td>
</tr>
</tbody>
</table>

## Floating Point Types

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>IEEE 754 32 bit float.</td>
</tr>
<tr>
<td>double</td>
<td>IEEE 754 64 bit float.</td>
</tr>
</tbody>
</table>

## Additional Primitive Types

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>Boolean.</td>
</tr>
<tr>
<td>bytes</td>
<td>Arbitrary sequence of bytes.</td>
</tr>
<tr>
<td>string</td>
<td>Arbitrary sequence of UTF-8 characters.</td>
</tr>
</tbody>
</table>
More Field Types

Oneof Type

```protobuf
message AnExampleMessage {
  oneof some_oneof_field {
    int32 some_integer = 1;
    string some_string = 2;
  }
}
```

Enum Type

```protobuf
define AnEnum {
  INITIAL = 0;
  RED = 1;
  BLUE = 2;
  GREEN = 3;
  WHATEVER = 8;
}
```
More Field Types

**Any Type**

```protobuf
import "google/protobuf/any.proto";
message AnExampleMessage {
  repeated google.protobuf.Any whatever = 8;
}
```

**Map Type**

```protobuf
message AnExampleMessage {
  map<int32, string> keywords = 8;
}
```
Outline

1. Technology Overview
2. Assignment Part I
3. Message Encoding
4. Message Specification
5. Message Manipulation
6. Assignment Part II
C++ Message Basics

Construction

```cpp
AnExampleMessage message;
AnExampleMessage message (another_message);
message.CopyFrom (another_message);
```

Singular Fields

```cpp
cout << message.some_integer ();;
message.set_some_integer (1234);
```

Repeated Fields

```cpp
int size = messages.messages_size ();
const AnExampleMessage &message = messages.messages (1234);
AnExampleMessage *message = messages.mutable_messages (1234);
AnExampleMessage *message = messages.add_messages ();
```
C++ Message Serialization

**Byte Array**

```cpp
char buffer [BUFFER_SIZE];
message.SerializeToArray (buffer, sizeof (buffer));
message.ParseFromArray (buffer, sizeof (buffer));
```

**Standard Stream**

```cpp
message.SerializeToOstream (&stream);
message.ParseFromIstream (&stream);
```
Java Message Basics

Construction

AnExampleMessage.Builder messageBuilder;
messageBuilder = AnExampleMessage.newBuilder ();
messageBuilder = AnExampleMessage.newBuilder (another_message);
AnExampleMessage message = messageBuilder.build ();

Singular Fields

System.out.println (message.getSomeInteger ());
messageBuilder.setSomeInteger (1234);

Repeated Fields

int size = messages.getMessagesCount ();
AnExampleMessage message = messages.getMessages (1234);
List<AnExampleMessage> messageList = messages.getMessagesList ();
messagesBuilder.addMessages (messageBuilder);
messagesBuilder.addMessages (message);
Java Message Serialization

**Byte Array**

```java
byte [] buffer = message.toByteArray ();
try {
    AnExampleMessage message = AnExampleMessage.parseFrom (buffer);
} catch (InvalidProtocolBufferException e) {
    System.out.println (e);
}
```

**Standard Stream**

```java
message.writeTo (stream);
AnExampleMessage message = AnExampleMessage.parseFrom (stream);
```
Python Message Basics

Construction

```python
message = AnExampleMessage()
message.CopyFrom(another_message)
```

Singular Fields

```python
print(message.some_integer)
message.some_integer = 1234
```

Repeated Fields

```python
size = len(messages.messages)
message = messages.messages[1234]
message = messages.messages.add()
```
Python Message Serialization

**Byte Array**

```python
buffer = message.SerializeToString()
message.ParseFromString(buffer)
message = AnExampleMessage.FromString(buffer)
```

**Standard Stream**

```python
file.write(message.SerializeToString())
message.ParseFromString(file.read())
AnExampleMessage.FromString(file.read())
```
I've decided to let the coders do their own thing on this project.

They've been shut away for days, and when they finally come out, we should have a Release Candidate ready!

 Apparently, they'll be more efficient and they'll gain a greater feeling of ownership of the project.

Ah, here they are now!

So? Are you done?

After a lot of work, a lot of discussion, and a few all-nighters, we finally managed it!

AMAZING! Well done guys!

Yeah! It wasn't easy, but we finally agree on the framework we'll use for the project!

Of course!

We start the real development tomorrow!

Outline

1 Technology Overview
2 Assignment Part I
3 Message Encoding
4 Message Specification
5 Message Manipulation
6 Assignment Part II
Assignment

Performance
Measure the performance of your implementation.

Experiment Design
Stick to the following, or provide arguments for why not:
- Random field mix, each field with probability 1/2.
- Measure at least two minutes long traffic.
- Report average invocation throughput.
- No printing during measurement.
- Compare with past assignments.
Submission

**GitLab**


**Requirements**

- Use the assignment subdirectory.
- Write brief report in SOLUTION.md.
- Include build scripts with instructions.
- Do not commit binaries or temporary build artifacts.