2025/10/28 17:43 1/4 Protocol Buffers (Protobuf)

## **Protocol Buffers (Protobuf)**

Protocol Buffers (Protobuf) is a method developed by Google for *serializing structured data*, similar to XML or JSON. It is especially beneficial in applications that communicate with servers or store data, where efficiency and the *speed of data transmission* are crucial. Protobuf is designed to be simpler and more efficient than XML and JSON, offering both smaller message sizes and faster processing.

**Protobuf** requires you to define your structured data in a standard format in a **.proto** file, which is then used to generate source code in your *chosen programming language*. This source code is used to write and read your structured data to and from a variety of data streams and using a variety of languages.

## **Key Features of Protobuf**

- **Efficiency**: Protobuf is designed to be more efficient than XML and JSON, both in terms of speed and the size of the serialized data.
- **Cross-language**: Protobuf supports generated code in various programming languages, allowing for easy data exchange between systems written in different languages.
- **Backward compatibility**: Protobuf is designed to maintain compatibility even if the structure of the data changes, allowing old code to read new data formats and vice versa.
- **Less verbose**: Protobuf messages are much less verbose than XML, leading to significant bandwidth savings.

More details can be found here:

https://developers.google.com/protocol-buffers/docs/tutorials

- 1.) Install the translator from the official website. https://github.com/protocolbuffers/protobuf/releases in the case of Windows, unzip the file protoc-XXX.zip.
- 2.) Create a directory called ./proto and the file book.proto with the following content:

```
syntax = "proto3";

message Book {
    int32 id = 1;
    string title = 2;
    string author = 3;
    float price = 4;
}

message Books {
    repeated Book books = 1;
}
```

We have created two messages named Book and Books. Books can contain several Books. = 1, = 2 at

the end of the lines indicates the internal position of the structure field, numbering starts from one.

3.) Run the following command:

```
.\protoc\bin\protoc.exe --python_out=.\ book.proto
```

After running, book\_pb2.py is created, which is generated source code and contains the data interface. This can be used to manage (serialize and de-serialize) the data.

4.) Upgrade protobuf interface

```
pip install —upgrade protobuf
```

5.) Create the server.py file with the following content:

```
import socket
import book pb2
import create books as c
# protoc/bin/protoc --python out=./ book.proto
# pip3 install --upgrade protobuf
books = c.create books()
book store = book pb2.Books()
for book in books:
    book store.books.append(book)
bytes to send = book store.SerializeToString()
#TCP socket server
s = socket.socket(socket.AF INET, socket.SOCK STREAM)
s.bind((socket.gethostname(), 4100))
s.listen(10)
while True:
    client socket, address = s.accept()
    print(f"server> Connection from {address} has been established!\n")
    client socket.send(bytes to send)
    print(f"server> Message sent: {bytes to send}\n")
    msg = client socket.recv(1024)
    print(f"client> {msg}\n")
    client socket.close()
    if msg == b'bye':
        break
```

2025/10/28 17:43 3/4 Protocol Buffers (Protobuf)

```
s.close()
```

6.) Create the create books.py file with the following content:

```
import book_pb2
def create_books():
   books = []
   books.append(book_pb2.Book())
   books[0].id = 1
   books[0].title = "Solaris"
   books[0].author = "Stanislaw Lem"
   books[0].price = 7.54
   books.append(book pb2.Book())
   books[1].id = 2
   books[1].title = "Dune"
   books[1].author = "Frank Herbert"
   books[1].price = 9.87
   books.append(book_pb2.Book())
   books[2].id = 3
   books[2].title = "Foundation"
   books[2].author = "Isaac Asimov"
   books[2].price = 5.07
    return books
```

7.) Create the client.py file with the following content:

```
import socket
import book_pb2
from google.protobuf.json_format import MessageToJson
import json

#TCP socket client
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((socket.gethostname(), 4100))

msg = s.recv(1024)
print(f"server> {msg}\n")
s.sendall(b'bye')
print(f"client> Message sent: {b'bye'}\n")
s.close()
```

```
books = book_pb2.Books()
books.ParseFromString(msg)

json_obj = MessageToJson(books)
print(f"client> The server's message in JSON:\n{json_obj}")

dict_obj = json.loads(json_obj)

with open('data.json', 'w', encoding='utf-8') as f:
    json.dump(dict_obj, f, ensure_ascii=False, indent=4)
    print("client> data.json saved\n")

with open('data.bytes', 'wb') as fb:
    fb.write(msg)
    print("client> data.bytes saved\n")
```

8.) Run the server and client. python server.py then python client.py commands and let's see and analyze what happens?

From

https://edu.iit.uni-miskolc.hu/ - Institute of Information Science - University of Miskolc

Permanent link:

https://edu.iit.uni-miskolc.hu/tanszek:oktatas:iss\_t:modern\_data\_integration\_based\_on\_protocol\_buffer?rev=1710695220

Last update: 2024/03/17 17:07

