gRPC

DNP lab 4
Innopolis University
Fall 2022

grpc

Remote Procedure Calls by Google

Based on **protobuf** aka **Protocol Buffers** (also developed by Google)

Workflow

1. Install gRPC
2. Describe your protocol in **.proto** file
3. Compile your .proto file to generate **stubs**
4. Create your **request handler** using generated stubs
5. Create and start your server using **gRPC** and **request handler**
6. Create your client and connect it to server using **gRPC**

1. Install gRPC

Optionally: create venv (virtual python environment)

Command: **pip3 install grpcio grpcio-tools**

grpcio - main library for working with grpc

grpcio-tools - auxiliary tools, including the **Protobuf compiler**

https://grpc.io/docs/languages/python/quickstart/

2. Describe your protocol in .proto file

```
1   syntax = "proto3";
2
3   service SimpleService {
4     rpc GetServerResponse(Message) returns (MessageResponse);
5   }
6
7   message Message {
8     string message = 1;
9   }
10
11  message MessageResponse {
12    string message = 1;
13    bool received = 2;
14  }
15
```

2. Describe your protocol in .proto file

```
1   syntax = "proto3";

2

3   service SimpleService {
4     rpc GetServerResponse(Message) returns (MessageResponse);
5   }

6

7   message Message {
8     string message = 1;
9   }

10

11  message MessageResponse {
12    string message = 1;
13    bool received = 2;
14  }

15
```

Specify the syntax version to protobuf3

2. Describe your protocol in .proto file

```
1   syntax = "proto3";
2
3   service SimpleService {
4     rpc GetServerResponse(Message) returns (MessageResponse);
5   }
6
7   message Message {
8     string message = 1;
9   }
10
11  message MessageResponse {
12    string message = 1;
13    bool received = 2;
14  }
15
```

Specify the syntax version to protobuf3

Declare your service

2. Describe your protocol in .proto file

```
1   syntax = "proto3";

2

3   service SimpleService {
4     rpc GetServerResponse(Message) returns (MessageResponse);
5   }

6

7   message Message {
8     string message = 1;
9   }

10

11  message MessageResponse {
12    string message = 1;
13    bool received = 2;
14  }

15
```

Specify the syntax version to protobuf3

Declare your service

Declare methods in your service

2. Describe your protocol in .proto file

```
syntax = "proto3";

service SimpleService {
  rpc GetServerResponse(Message) returns (MessageResponse);
}

message Message {
  string message = 1;
}

message MessageResponse {
  string message = 1;
  bool received = 2;
}
```

Specify the syntax version to protobuf3

Declare your service

Declare methods in your service

Describe messages of your service

3. Compile your .proto file to generate stubs

- python3 -m grpc_tools.protoc **SimpleService.proto** --proto_path=. --python_out=. --grpc_python_out=.

SimpleService.proto

```
syntax = "proto3";

service SimpleService {
  rpc GetServerResponse(Message) returns (MessageResponse);
}

message Message {
  string message = 1;
}

message MessageResponse {
  string message = 1;
  bool received = 2;
}

```

SimpleService_pb2.py

SimpleService_pb2_grpc.py

3. Compile your .proto file to generate stubs

SimpleService_pb2.py

- Serialization and message parsing logic

SimpleService_pb2_grpc.py

- Stub for request handler
- Stub for client

4. Create your request handler using generated stubs

```python
import SimpleService_pb2_grpc as pb2_grpc
import SimpleService_pb2 as pb2

class SimpleHandler(pb2_grpc.SimpleServiceServicer):
  def GetServerResponse(self, request, context):
    msg = request.message
    reply = {"message": msg, "received": True}
    return pb2.MessageResponse(**reply)
```

```proto
syntax = "proto3";

service SimpleService {
  rpc GetServerResponse(Message) returns (MessageResponse);
}

message Message {
  string message = 1;
}

message MessageResponse {
  string message = 1;
  bool received = 2;
}
```

4. Create your request handler using generated stubs

```python
import SimpleService_pb2_grpc as pb2_grpc
import SimpleService_pb2 as pb2

class SimpleHandler(pb2_grpc.SimpleServiceServicer):
    def GetServerResponse(self, request, context):
        msg = request.message
        reply = {"message": msg, "received": True}
        return pb2.MessageResponse(**reply)
```

```proto
syntax = "proto3";

service SimpleService {
  rpc GetServerResponse(Message) returns (MessageResponse);
}

message Message {
  string message = 1;
}

message MessageResponse {
  string message = 1;
  bool received = 2;
}
```

5. Create and start your server using gRPC

```
1   import SimpleService_pb2_grpc as pb2_grpc
2   import SimpleService_pb2 as pb2
3   import grpc
4   from concurrent import futures

13  if __name__ == "__main__":
14      server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
15      pb2_grpc.add_SimpleServiceServicer_to_server(SimpleHandler(), server)
16      server.add_insecure_port("127.0.0.1:5555")
17      server.start()
18      try:
19          server.wait_for_termination()
20      except KeyboardInterrupt:
21          print("Shutting down")
```

6. Create your client and connect it to server using gRPC

```python
import grpc
import SimpleService_pb2 as pb2
import SimpleService_pb2_grpc as pb2_grpc


if __name__ == "__main__":
    channel = grpc.insecure_channel("127.0.0.1:5555")
    stub = pb2_grpc.SimpleServiceStub(channel)

    msg = pb2.Message(message="Hello there!")
    response = stub.GetServerResponse(msg)

    print(response)
    print(response.received)
    print(response.message)
```

```proto
syntax = "proto3";

service SimpleService {
  rpc GetServerResponse(Message) returns (MessageResponse);
}

message Message {
  string message = 1;
}

message MessageResponse {
  string message = 1;
  bool received = 2;
}
```

6. Create your client and connect it to server using gRPC

```python
import grpc
import SimpleService_pb2 as pb2
import SimpleService_pb2_grpc as pb2_grpc


if __name__ == "__main__":
    channel = grpc.insecure_channel("127.0.0.1:5555")
    stub = pb2_grpc.SimpleServiceStub(channel)

    msg = pb2.Message(message="Hello there!")
    response = stub.GetServerResponse(msg)

    print(response)
    print(response.received)
    print(response.message)
```

```proto
syntax = "proto3";

service SimpleService {
  rpc GetServerResponse(Message) returns (MessageResponse);
}

message Message {
  string message = 1;
}

message MessageResponse {
  string message = 1;
  bool received = 2;
}
```

Your task

Develop a server and a client using gRPC

Server

> python3 server.py 5555

Has following functions:

- **reverse(text: str) -> str** – returns reversed string
- **split(text: str, delim:str) -> (int, [str])** – splits the text by delimiter. Returns number of parts and parts themself
- **isprime(num: int) -> str** – checks if number is prime or not. This is a **stream** function, which means it accepts a stream of numbers and returns a stream of answers

Server must support multiple clients

Client

> python3 client.py 127.0.0.1:5555

Continuously listens to client input and executes commands:

- **reverse <some text>** – rpc call
- **split <some text>** – rpc call – split text by whitespaces
- **isprime <list of numbers>** – rpc call
  Example: isprime 2 10 5555 14567
- **exit** – stops loop and exits program

Submission

Submit your:

- server.py
- client.py
- service.proto

as a single **.zip** archive

Example

python3 client.py

```
> reverse Hello there!
message: "!ereht olleH"

> split Hello there!
number: 2
parts: "Hello"
parts: "there!"

> isprime 2 5 7 9 10
2 is prime
5 is prime
7 is prime
9 is not prime
10 is not prime
> exit
Shutting down
```