gRPC - Andreas Maurer

What is gRPC and why does it work accross languages and platforms?

gRPC (Google Remote Procedure Call) is an open-source framework for building distributed system independent of programming languages. The use of the Protobuf meta language, makes gRPC language independent. You generate the .proto file into code native to the language you code in, which makes the process convenient. They communicate through HTTP/2 but unlike REST they don't communicate through JSON or XML but rather an internal protocol which is more efficient and smaller in size.

Describe the RPC life cycle starting with the RPC client?

The cycle involves interaction between an RPC client and RPC server. The server waits for an request and answers with an response. And the cycle continues.

> 1. Once the client calls a stub method, the server is notified that the RPC has been invoked with the client's metadata for this call, the method name, and the specified deadline if applicable.
> 2. The server can then either send back its own initial metadata (which must be sent before any response) straight away, or wait for the client's request message. Which happens first, is application-specific.
> 3. Once the server has the client's request message, it does whatever work is necessary to create and populate a response. The response is then returned (if successful) to the client together with status details (status code and optional status message) and optional trailing metadata.
> 4. If the response status is OK, then the client gets the response, which completes the call on the client side.

Describe the workflow of Protocol Buffers?

First you need to specify `syntax = "proto3"`

Then there are services. the

```
syntax = "proto3";

service ServiceName {
  rpc serviceAction(Request) returns (Response);
}

message Request {
    string text = 1;
}

message Response {
    string status = 1;
}

/* the "= 1" specifies the index. For more than 1 attribute you should
increment the index */
```

This will then generate code for your programming language. And when you send and receive messages this will serialize the text in an compact binary format.

What are the benefits of using protocol buffers?

As we said earlier, this makes gRPC language independent. Now every language can generate code native to their own and this makes it really flexible.

When is the use of protocol not recommended?

Some smaller languages might not have proper support for protobuf generation which makes it a hasle to deal with. While JSON on the other hand should be widely supported. Another disadvantage is that the compact binary format is not human readable unlike JSON or XML.

List 3 different data types that can be used with protocol buffers?

1. String
2. float
3. enum

For example:

```
syntax = "proto3";

message Person {
    string name = 1;
    float height = 2;
    enum Status {
        UNKNOWN = 0;
        HAPPY = 1;
        DEPRESSED = 2;
    }
}
```

The implementation

election data protobuf

Here I used the XML as reference to make the protobuf equivalent

```
syntax = "proto3";

service ElectionDataService {
  rpc sendElectionData(ElectionRequest) returns (ElectionResponse);
}

message ElectionRequest {
  Region region = 1;
  repeated Party parties = 2;
}

message ElectionResponse {
  string status = 1;
}

message Region {
  int32 regionID = 1;
  string regionName = 2;
  string regionAddress = 3;
  string regionPostalCode = 4;
  string federalState = 5;
  string timestamp = 6;
}

message Party {
  string partyID = 1;
  int32 amountVotes = 2;
}
```

gRPC java server

Simple server who receives the election data from a client.

```java
import io.grpc.Server;
import io.grpc.ServerBuilder;
import io.grpc.stub.StreamObserver;

import java.io.IOException;

public class ElectionServer
    extends ElectionDataServiceGrpc.ElectionDataServiceImplBase
{
    private final static int PORT = 50051;

    public static void main(String[] args)
        throws InterruptedException, IOException
    {
        Server server = ServerBuilder.forPort(PORT)
            .addService(new ElectionServer())
            .build();
        System.out.println("Server starting on port 50051 ...");
        server.start();
        server.awaitTermination();
    }

    @Override
    public void sendElectionData(
        ElectionData.ElectionRequest request,
        StreamObserver<ElectionData.ElectionResponse> responseObserver)
    {
        // Access the region data from the request
        ElectionData.Region region = request.getRegion();
        System.out.println(
            "Region ID: " + region.getRegionID()
            + "Region Name: " + region.getRegionName()
            + "Region Address: " + region.getRegionAddress()
            + "Region Postal Code: " + region.getRegionPostalCode()
            + "Federal State: " + region.getFederalState()
            + "Timestamp: " + region.getTimestamp());

        // Access the party data from the request
        for (ElectionData.Party party : request.getPartiesList())
            System.out.println(
                "Party: " + party.getPartyID() +
                ", Votes: " + party.getAmountVotes());

        // Respond with a status message
        ElectionData.ElectionResponse response =
ElectionData.ElectionResponse.newBuilder()
            .setStatus("Election data received successfully")
            .build();

        responseObserver.onNext(response);
```

```
            responseObserver.onCompleted();
        }
    }
```

gRPC java client

Simple client who sends the Election data to the server.

```java
import io.grpc.ManagedChannel;
import io.grpc.ManagedChannelBuilder;
import java.util.ArrayList;
import java.util.List;

public class ElectionClient {
    private final static int PORT = 50051;

    public static void main(String[] args) {
        ManagedChannel channel =
            ManagedChannelBuilder.forAddress("localhost", PORT)
            .usePlaintext()
            .build();

        ElectionDataServiceGrpc.ElectionDataServiceBlockingStub stub =
            ElectionDataServiceGrpc.newBlockingStub(channel);

        // Create the region
        ElectionData.Region region = ElectionData.Region.newBuilder()
            .setRegionID(33123)
            .setRegionName("Linz Bahnhof")
            .setRegionAddress("Bahnhofsstrasse 27/9")
            .setRegionPostalCode("Linz")
            .setFederalState("Austria")
            .setTimestamp("2024-09-12 11:48:21")
            .build();

        // Create the parties
        List<ElectionData.Party> parties = new ArrayList<>();

parties.add(ElectionData.Party.newBuilder().setPartyID("OEVP").setAmountVotes(322
).build());

parties.add(ElectionData.Party.newBuilder().setPartyID("SPOE").setAmountVotes(301
).build());

parties.add(ElectionData.Party.newBuilder().setPartyID("FPOE").setAmountVotes(231
).build());

parties.add(ElectionData.Party.newBuilder().setPartyID("GRUENE").setAmountVotes(2
11).build());

parties.add(ElectionData.Party.newBuilder().setPartyID("NEOS").setAmountVotes(182
).build());

        // Create the request
        ElectionData.ElectionRequest request =
ElectionData.ElectionRequest.newBuilder()
            .setRegion(region)
            .addAllParties(parties)
```

```
            .build();

        // Send the data and receive the status
        ElectionData.ElectionResponse response = stub.sendElectionData(request);
        System.out.println("Server response: " + response.getStatus());

        channel.shutdown();
    }
}
```

gRPC python client

Simple client doing effectively the same as the Java client.

```python
import grpc
import election_data_pb2
import election_data_pb2_grpc

def run_client():
    server_address = "localhost:50051"
    channel = grpc.insecure_channel(server_address)
    stub = election_data_pb2_grpc.ElectionDataServiceStub(channel)

    # set the region
    region = election_data_pb2.Region(
        regionID=33123,
        regionName="Linz Bahnhof",
        regionAddress="Bahnhofsstrasse 27/9",
        regionPostalCode="Linz",
        federalState="Austria",
        timestamp="2024-09-12 11:48:21"
    )

    # set the parties
    parties = [
        election_data_pb2.Party(partyID="OEVP", amountVotes=322),
        election_data_pb2.Party(partyID="SPOE", amountVotes=301),
        election_data_pb2.Party(partyID="FPOE", amountVotes=231),
        election_data_pb2.Party(partyID="GRUENE", amountVotes=211),
        election_data_pb2.Party(partyID="NEOS", amountVotes=182)
    ]

    request = election_data_pb2.ElectionRequest(region=region, parties=parties)

    # error handeling
    try:
        response = stub.sendElectionData(request)
        print(f"Server response: {response.status}")
    except grpc.RpcError as e:
        print(f"gRPC error: {e.code()} - {e.details()}")

    channel.close()

if __name__ == "__main__":
    run_client()
```