gRPC и все, все, все

Программная инженерия
16.04.2021

Об чём

- protobuf
- grpc
- coroutines

Protocol Buffers

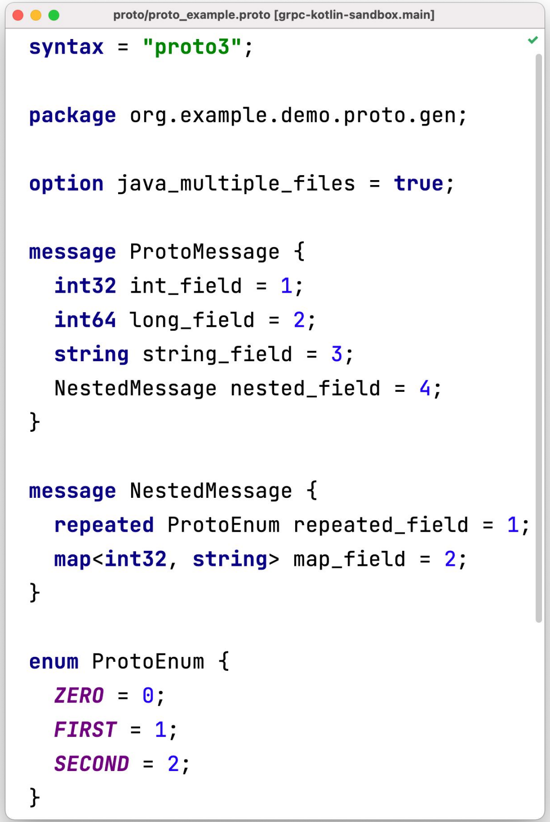
«Protocol buffers are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data – think XML, but smaller, faster, and simpler»

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

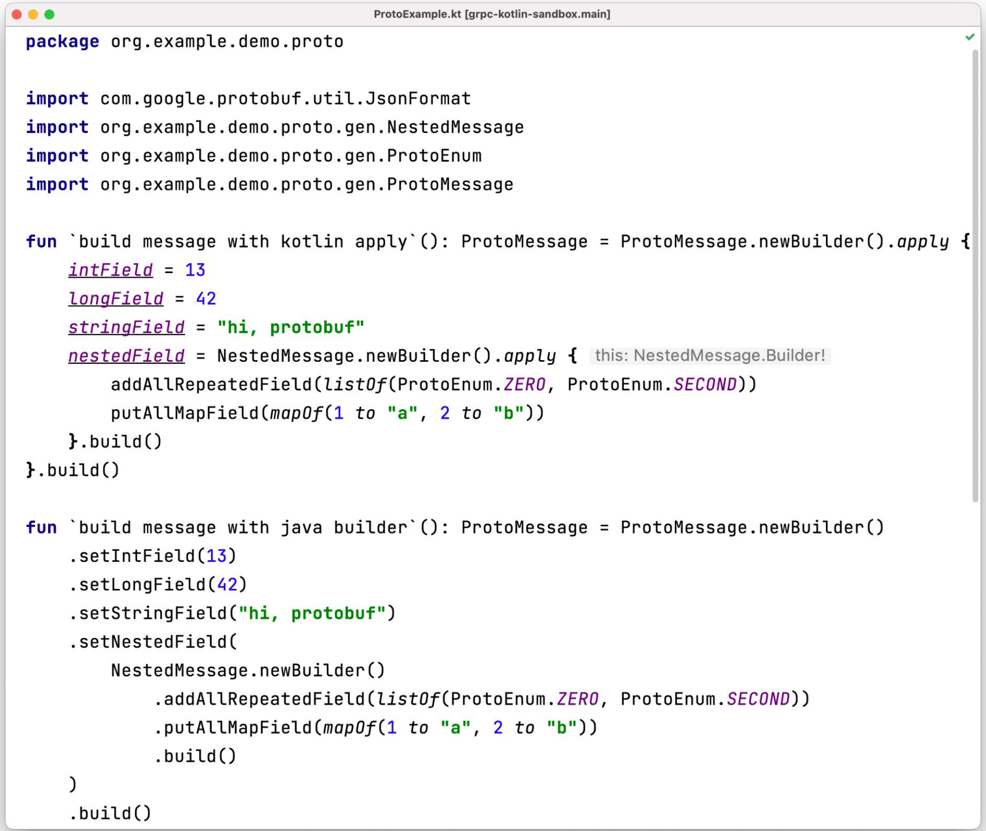
Protocol Buffers

- ✓ fast [de]serialization
- ✓ binary
- ✓ type-safe
- ✓ backward compatibility
- ✓ language-agnostic

Protocol Buffers


proto/proto_example.proto [grpc-kotlin-sandbox.main]

```
syntax = "proto3";

package org.example.demo.proto.gen;

option java_multiple_files = true;

message ProtoMessage {
  int32 int_field = 1;
  int64 long_field = 2;
  string string_field = 3;
  NestedMessage nested_field = 4;
}

message NestedMessage {
  repeated ProtoEnum repeated_field = 1;
  map<int32, string> map_field = 2;
}

enum ProtoEnum {
  ZERO = 0;
  FIRST = 1;
  SECOND = 2;
}
```


ProtoExample.kt [grpc-kotlin-sandbox.main]

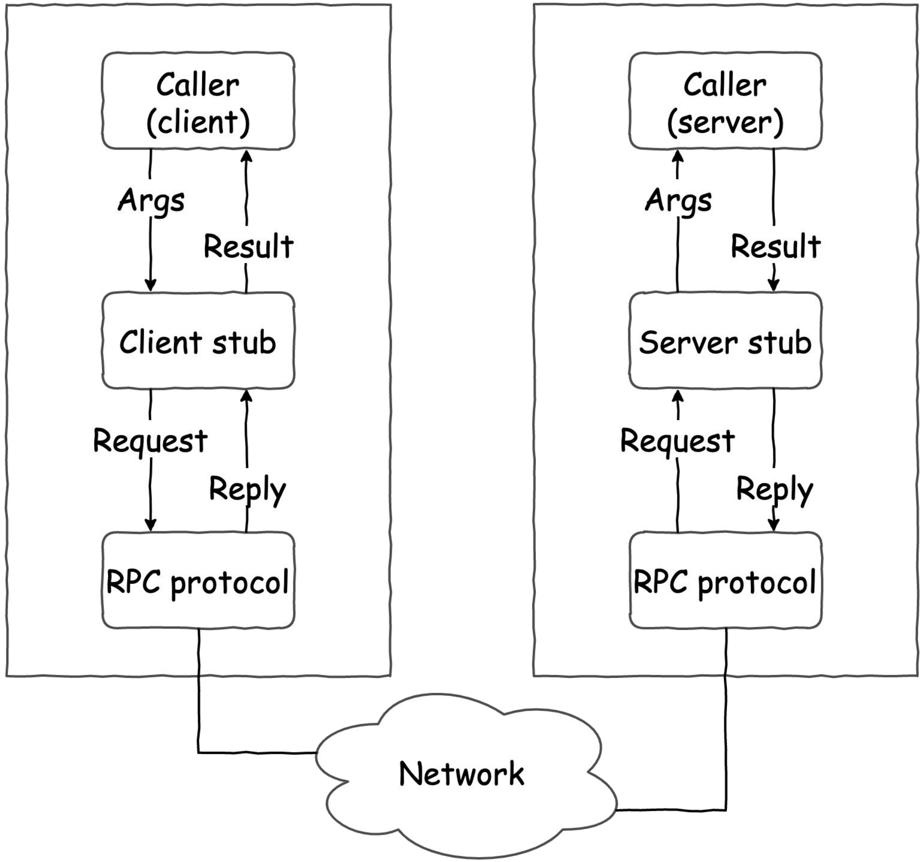
```
package org.example.demo.proto

import com.google.protobuf.util.JsonFormat
import org.example.demo.proto.gen.NestedMessage
import org.example.demo.proto.gen.ProtoEnum
import org.example.demo.proto.gen.ProtoMessage

fun `build message with kotlin apply`(): ProtoMessage = ProtoMessage.newBuilder().apply {
    intField = 13
    longField = 42
    stringField = "hi, protobuf"
    nestedField = NestedMessage.newBuilder().apply {  this: NestedMessage.Builder!
        addAllRepeatedField(listOf(ProtoEnum.ZERO, ProtoEnum.SECOND))
        putAllMapField(mapOf(1 to "a", 2 to "b"))
    }.build()
}.build()

fun `build message with java builder`(): ProtoMessage = ProtoMessage.newBuilder()
    .setIntField(13)
    .setLongField(42)
    .setStringField("hi, protobuf")
    .setNestedField(
        NestedMessage.newBuilder()
            .addAllRepeatedField(listOf(ProtoEnum.ZERO, ProtoEnum.SECOND))
            .putAllMapField(mapOf(1 to "a", 2 to "b"))
            .build()
    )
    .build()
```

RPC

«In distributed computing, a remote procedure call (RPC) is when a computer program causes a procedure (subroutine) to execute in a different address space (commonly on another computer on a shared network), which is coded as if it were a normal (local) procedure call»

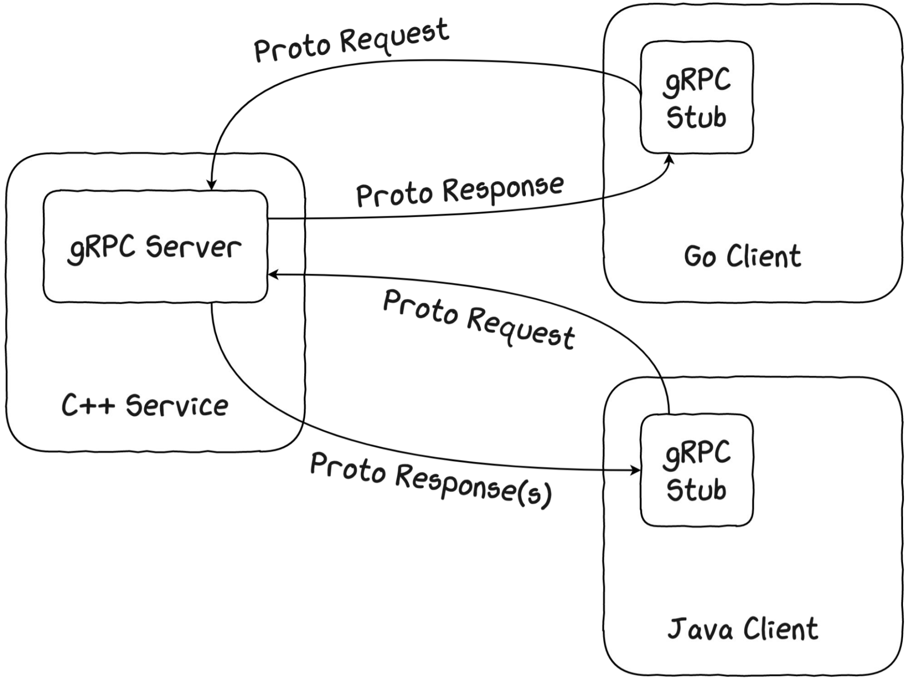
https://en.wikipedia.org/wiki/Remote_procedure_call

RPC


Caller (client)
Args
Result
Client stub
Request
Reply
RPC protocol
Caller (server)
Args
Result
Server stub
Request
Reply
RPC protocol
Network

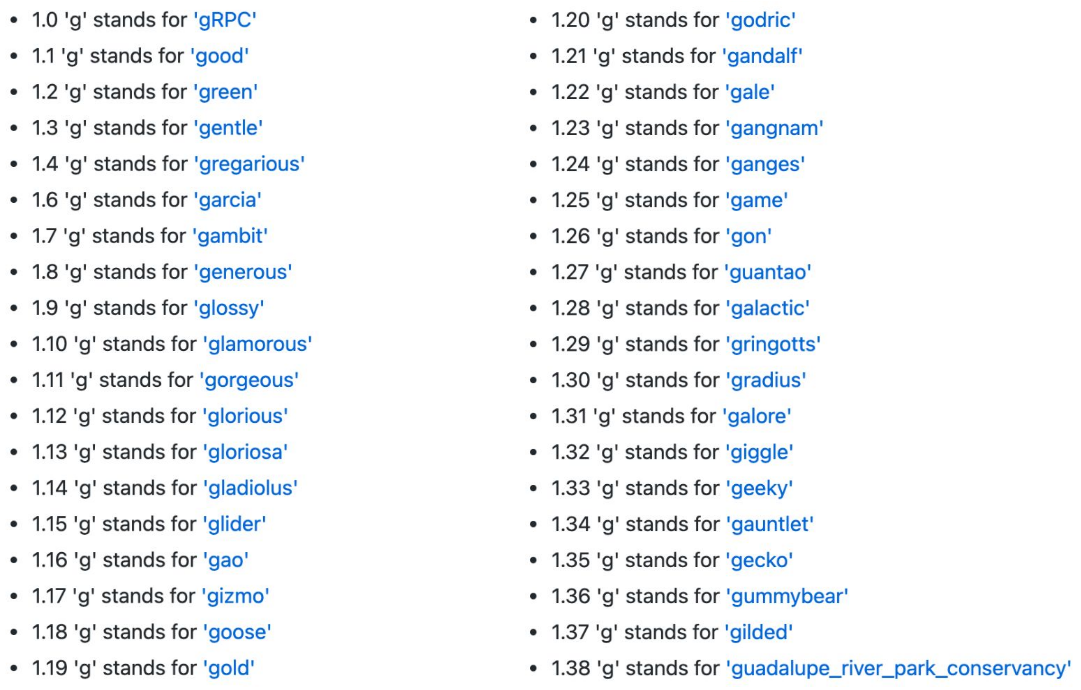
gRPC

- RPC on Protocol Buffers
- HTTP/2
- streaming
    - client-side
    - server-side
    - bidirectional
- language-agnostic



gRPC – *means gRPC Remote Procedure Call*

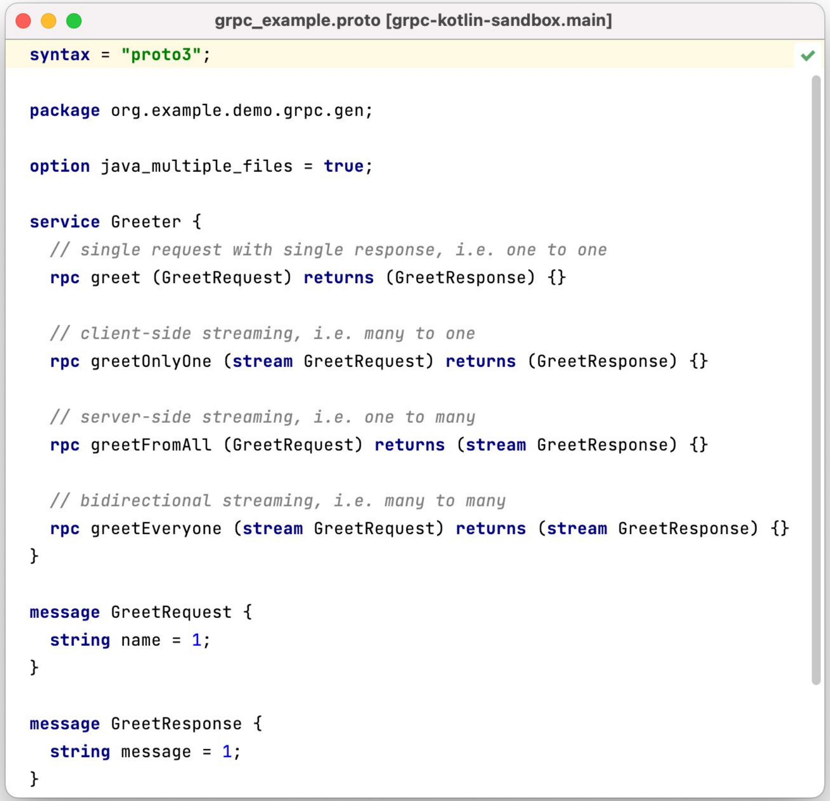
'g' stands for something different every gRPC release:



- 1.0 'g' stands for 'gRPC'
- 1.1 'g' stands for 'good'
- 1.2 'g' stands for 'green'
- 1.3 'g' stands for 'gentle'
- 1.4 'g' stands for 'gregarious'
- 1.6 'g' stands for 'garcia'
- 1.7 'g' stands for 'gambit'
- 1.8 'g' stands for 'generous'
- 1.9 'g' stands for 'glossy'
- 1.10 'g' stands for 'glamorous'
- 1.11 'g' stands for 'gorgeous'
- 1.12 'g' stands for 'glorious'
- 1.13 'g' stands for 'gloriosa'
- 1.14 'g' stands for 'gladiolus'
- 1.15 'g' stands for 'glider'
- 1.16 'g' stands for 'gao'
- 1.17 'g' stands for 'gizmo'
- 1.18 'g' stands for 'goose'
- 1.19 'g' stands for 'gold'
- 1.20 'g' stands for 'godric'
- 1.21 'g' stands for 'gandalf'
- 1.22 'g' stands for 'gale'
- 1.23 'g' stands for 'gangnam'
- 1.24 'g' stands for 'ganges'
- 1.25 'g' stands for 'game'
- 1.26 'g' stands for 'gon'
- 1.27 'g' stands for 'guantao'
- 1.28 'g' stands for 'galactic'
- 1.29 'g' stands for 'gringotts'
- 1.30 'g' stands for 'gradius'
- 1.31 'g' stands for 'galore'
- 1.32 'g' stands for 'giggle'
- 1.33 'g' stands for 'geeky'
- 1.34 'g' stands for 'gauntlet'
- 1.35 'g' stands for 'gecko'
- 1.36 'g' stands for 'gummybear'
- 1.37 'g' stands for 'gilded'
- 1.38 'g' stands for 'guadalupe_river_park_conservancy'

gRPC

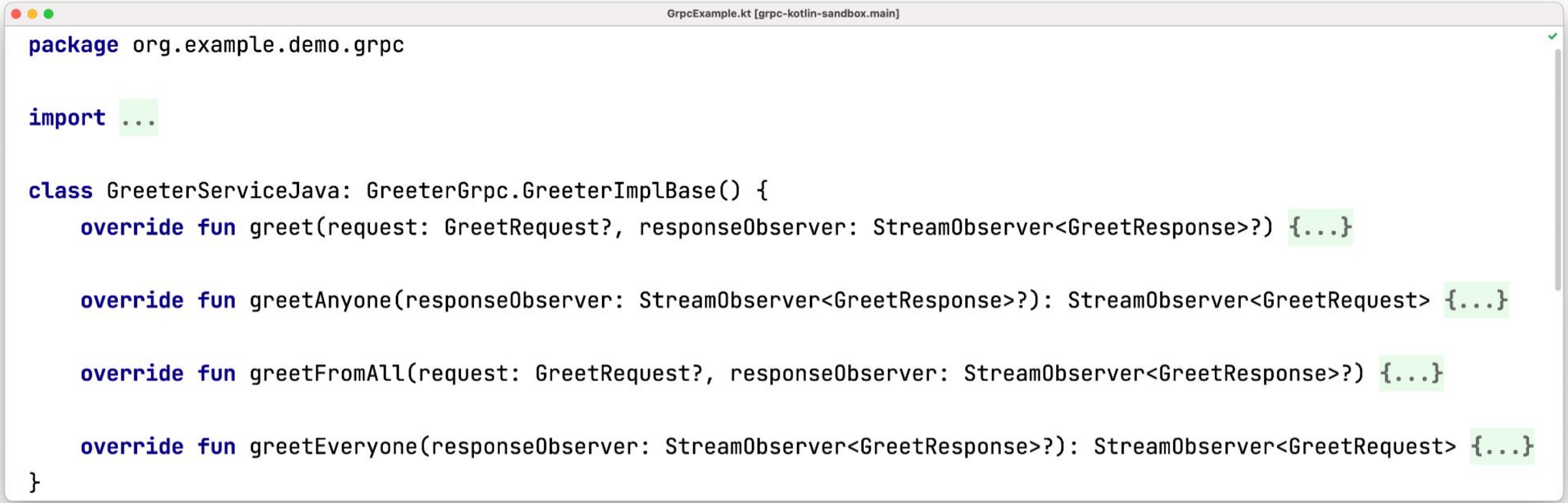
1. define service in *.proto*
2. run *protoc* with grpc plugin
   a. or delegate it to gradle/maven
3. implement server logic
4. create and use client



grpc_example.proto [grpc-kotlin-sandbox.main]

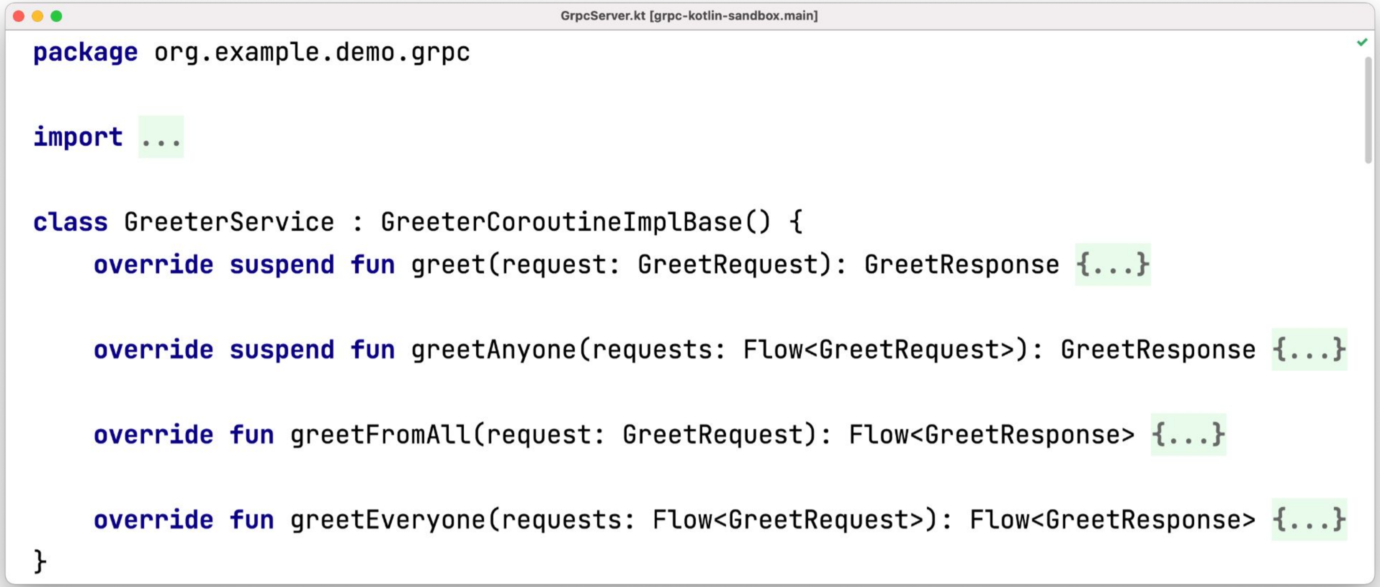
```
syntax = "proto3";

package org.example.demo.grpc.gen;

option java_multiple_files = true;

service Greeter {
  // single request with single response, i.e. one to one
  rpc greet (GreetRequest) returns (GreetResponse) {}

  // client-side streaming, i.e. many to one
  rpc greetOnlyOne (stream GreetRequest) returns (GreetResponse) {}

  // server-side streaming, i.e. one to many
  rpc greetFromAll (GreetRequest) returns (stream GreetResponse) {}

  // bidirectional streaming, i.e. many to many
  rpc greetEveryone (stream GreetRequest) returns (stream GreetResponse) {}
}

message GreetRequest {
  string name = 1;
}

message GreetResponse {
  string message = 1;
}
```

gRPC | java server


GrpcExample.kt [grpc-kotlin-sandbox.main]

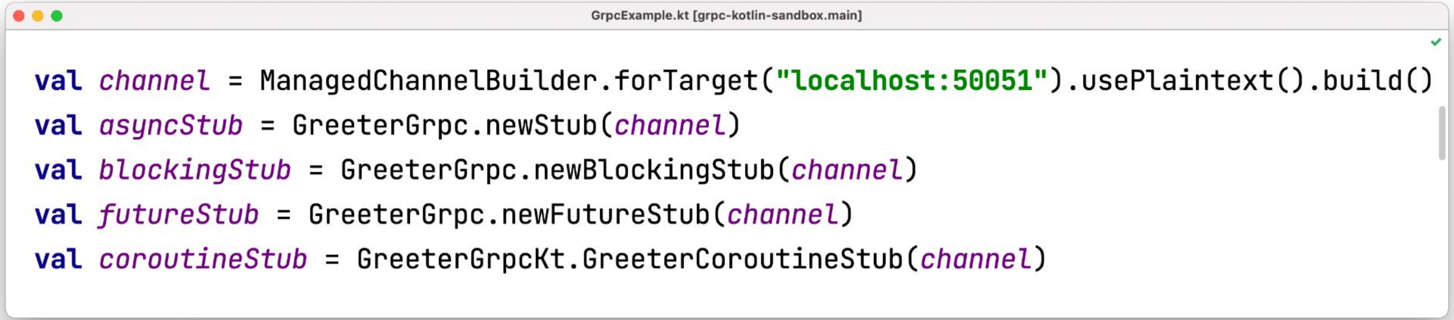
```
package org.example.demo.grpc

import ...

class GreeterServiceJava: GreeterGrpc.GreeterImplBase() {
    override fun greet(request: GreetRequest?, responseObserver: StreamObserver<GreetResponse>?) {...}

    override fun greetAnyone(responseObserver: StreamObserver<GreetResponse>?): StreamObserver<GreetRequest> {...}

    override fun greetFromAll(request: GreetRequest?, responseObserver: StreamObserver<GreetResponse>?) {...}

    override fun greetEveryone(responseObserver: StreamObserver<GreetResponse>?): StreamObserver<GreetRequest> {...}
}
```

gRPC | kotlin server



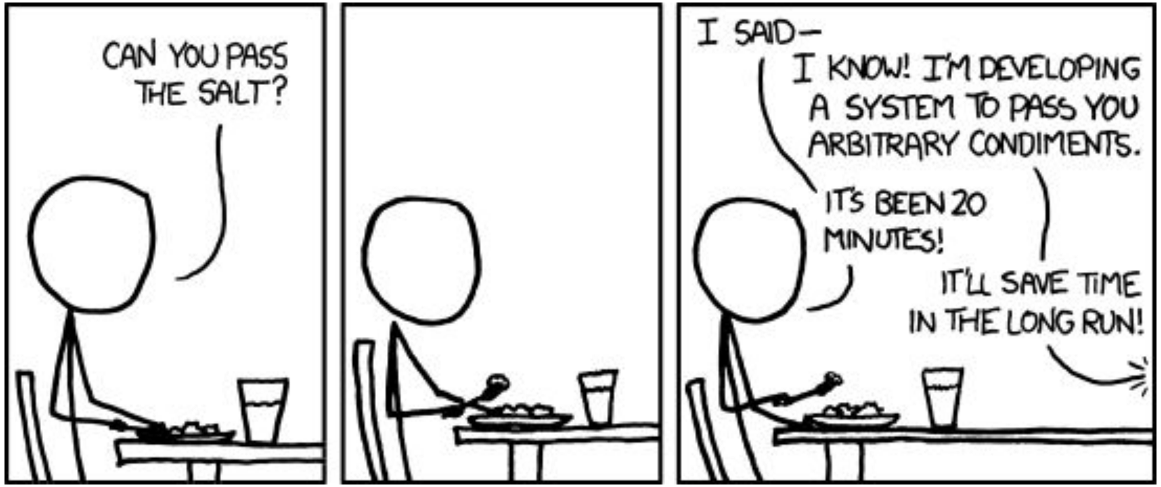
```kotlin
package org.example.demo.grpc

import ...

class GreeterService : GreeterCoroutineImplBase() {
    override suspend fun greet(request: GreetRequest): GreetResponse {...}

    override suspend fun greetAnyone(requests: Flow<GreetRequest>): GreetResponse {...}

    override fun greetFromAll(request: GreetRequest): Flow<GreetResponse> {...}

    override fun greetEveryone(requests: Flow<GreetRequest>): Flow<GreetResponse> {...}
}
```

gRPC | clients


GrpcExample.kt [grpc-kotlin-sandbox.main]

```kotlin
val channel = ManagedChannelBuilder.forTarget("localhost:50051").usePlaintext().build()
val asyncStub = GreeterGrpc.newStub(channel)
val blockingStub = GreeterGrpc.newBlockingStub(channel)
val futureStub = GreeterGrpc.newFutureStub(channel)
val coroutineStub = GreeterGrpcKt.GreeterCoroutineStub(channel)
```

Demo


CAN YOU PASS THE SALT?
I SAID—
I KNOW! I'M DEVELOPING A SYSTEM TO PASS YOU ARBITRARY CONDIMENTS.
IT'S BEEN 20 MINUTES!
IT'LL SAVE TIME IN THE LONG RUN!

github.com/skoret/grpc-kotlin-sandbox

Почитать-посмотреть

- Protocol Buffers | Google Developers
  - github.com/protocolbuffers/protobuf
  - kotlin support request #3742
- Kotlin | gRPC
  - github.com/grpc/grpc-kotlin
  - Coroutines
- Announcing Open Source gRPC Kotlin
- Building Microservices with Kotlin and gRPC
- github.com/marcoferrer/kroto-plus
- github.com/streem/pbandk

НЕСМОТРЯ НА ДЕТАЛЬНЫЙ АНАЛИЗ ТЕКУЩЕЙ СИТУАЦИИ, Я ТАК И НЕ СМОГ СОСТАВИТЬ ЧЁТКОЕ ПРЕДСТАВЛЕНИЕ ОБ ОБСУЖДАЕМОЙ ПРОБЛЕМЕ В СИЛУ ВОЗНИКШЕГО КОНГИТИВНОГО ДИССОНАНСА.