gRPC 입문기

최상용 2024.01.27

누구세요 ?

최상용

오늘의집 Backend 개발자

목차

gRPC 입문기

들어가기에 앞서..

들어가기에 앞서

gRPC 에는 unary 방식과 stream 방식이 있다.

gRPC 란 무엇인가 ?

gRPC 란 무엇인가 ?

gRPC는 크로스 플랫폼 고성능 RPC(Remote Procedure Call) 프레임워크

gRPC 란 무엇인가 ?

RPC 는 뭔가요 ?

gRPC 란 무엇인가 ?

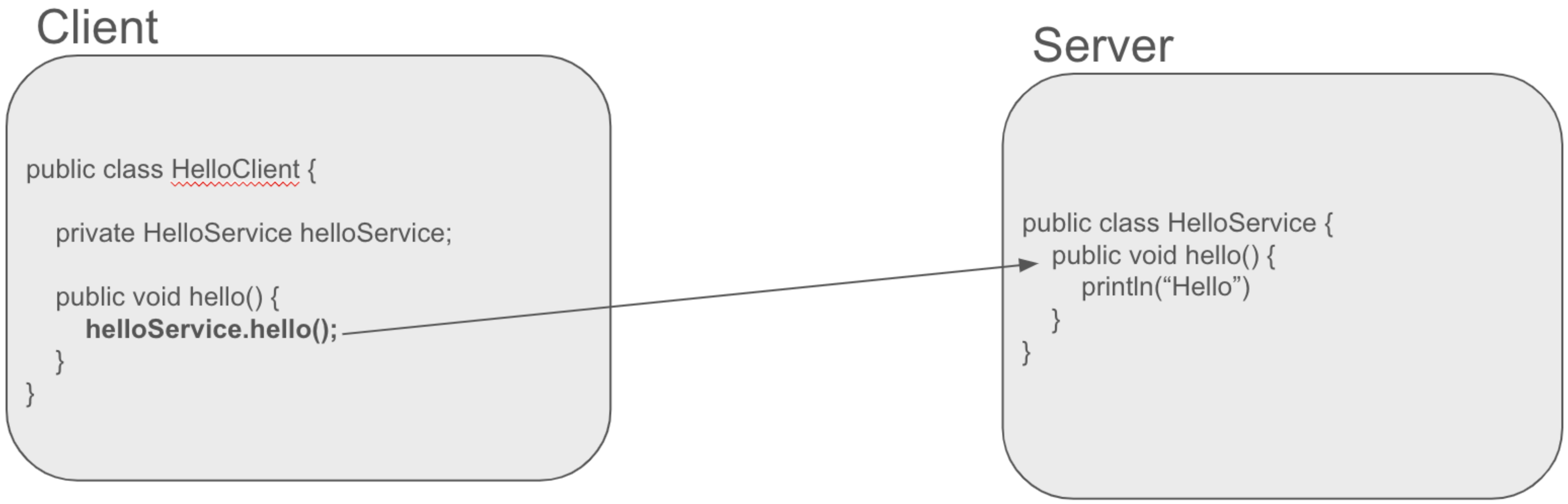
한 프로그램이 네트워크의 다른 컴퓨터에 있는 프로그램에게 서비스를 요청할 수 있는 소프트웨어 통신 프로토콜

원격시스템의 다른 프로세스를 로컬시스템처럼 호출 가능하다는 특징이 있음

gRPC 란 무엇인가 ?

한 프로그램이 네트워크의 다른 컴퓨터에 있는 프로그램에게 서비스를 요청할 수 있는 소프트웨어 통신 프로토콜

원격시스템의 다른 프로세스를 로컬시스템처럼 호출 가능하다는 특징이 있음



```
public class HelloClient {

    private HelloService helloService;

    public void hello() {
        helloService.hello();
    }
}
```

```
public class HelloService {
    public void hello() {
        println("Hello")
    }
}
```

gRPC 란 무엇인가 ?

gRPC는 크로스 플랫폼 고성능 RPC(Remote Procedure Call) 프레임워크

gRPC 란 무엇인가 ?

gRPC 는 Service Interface 와 payload 의 구조를 정의하기 위하여 Protocol Buffers 를 사용

gRPC 란 무엇인가 ?

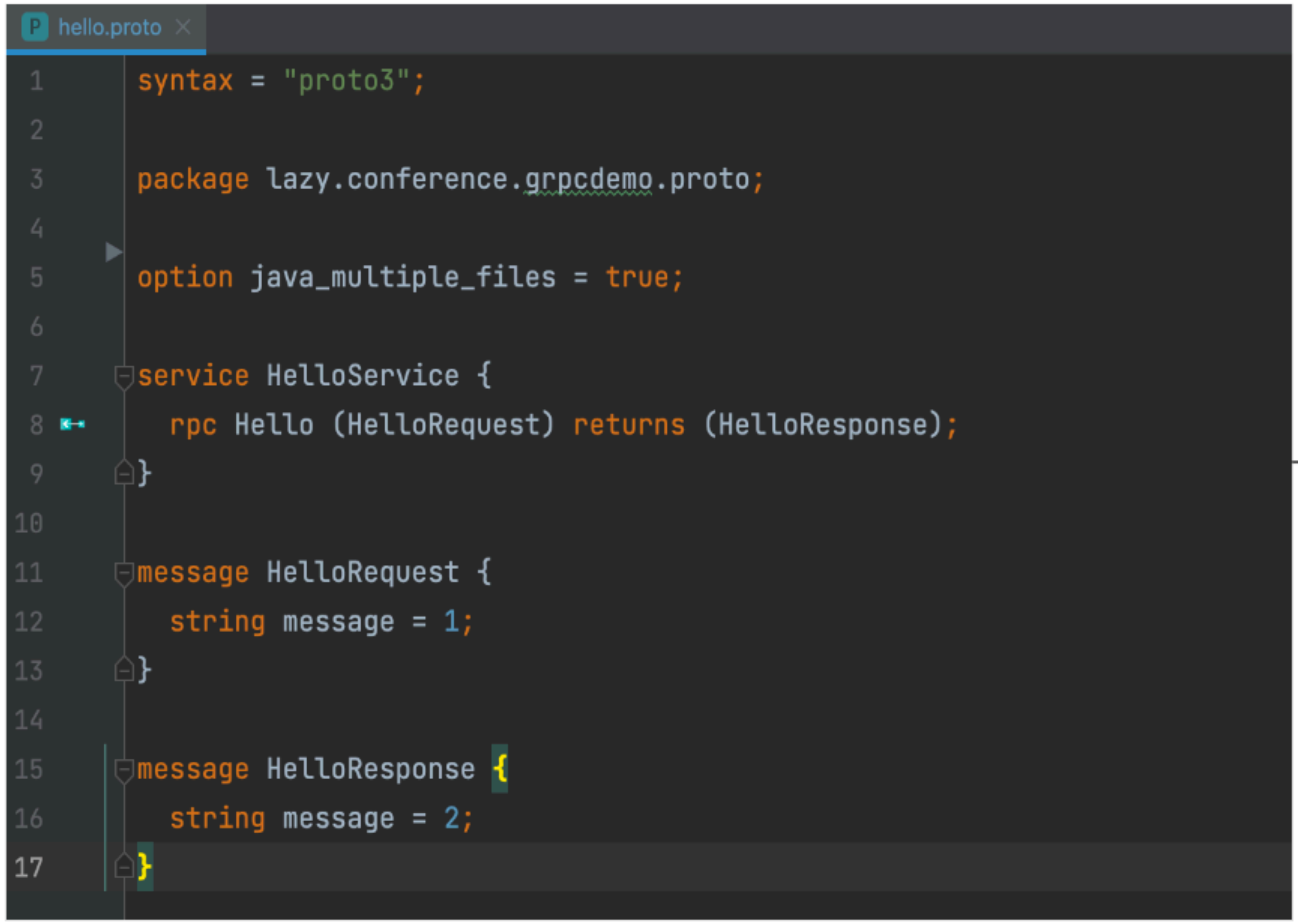
Protocol Buffers ?

gRPC 란 무엇인가 ?

Protocol Buffers are a language-neutral, platform-neutral extensible mechanism for serializing structured data.

gRPC 란 무엇인가 ?

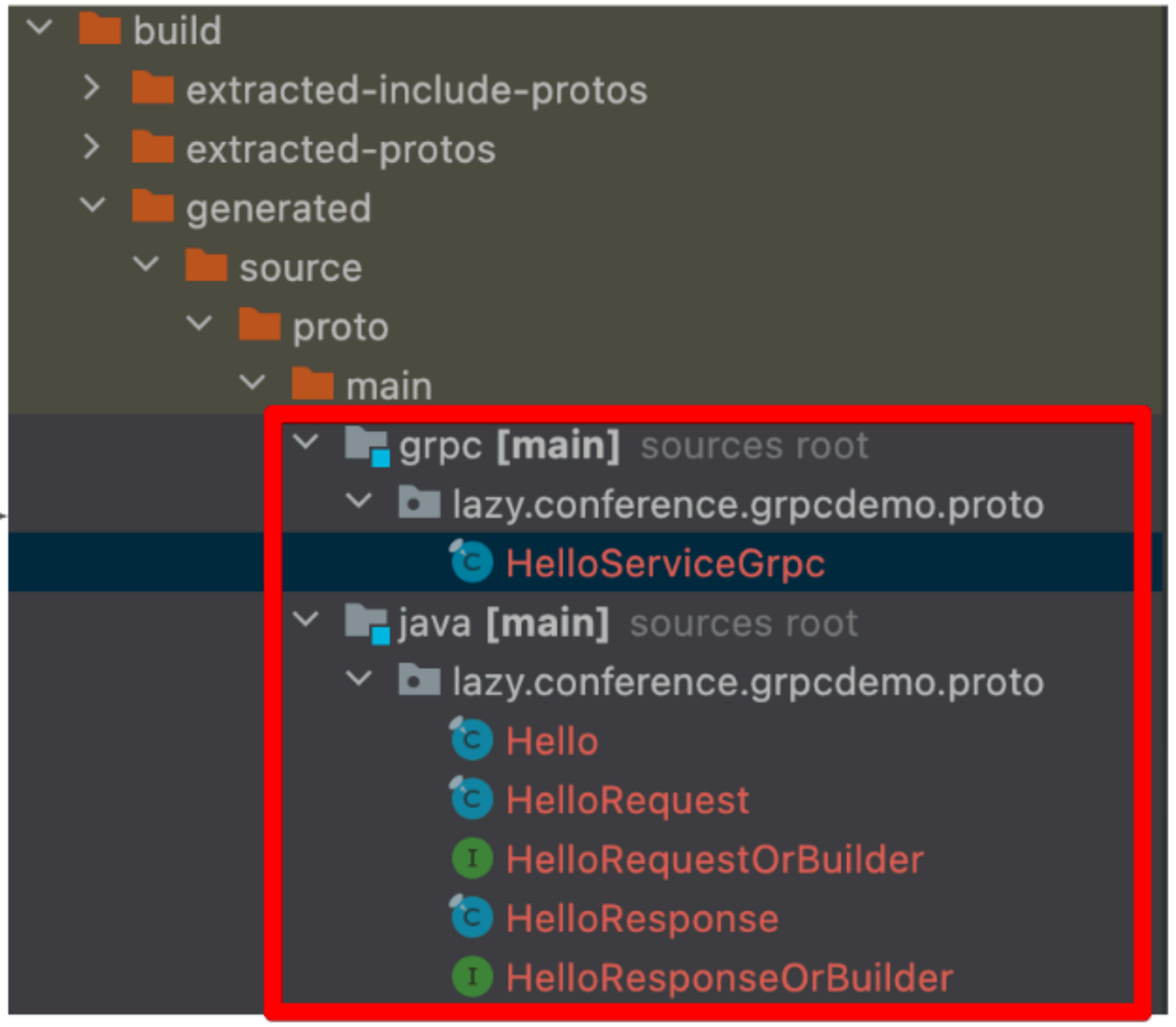
Proto file 을 작성한 후에 generateProto 를 실행하면 파일 생성



hello.proto

```
syntax = "proto3";

package lazy.conference.grpcdemo.proto;

option java_multiple_files = true;

service HelloService {
  rpc Hello (HelloRequest) returns (HelloResponse);
}

message HelloRequest {
  string message = 1;
}

message HelloResponse {
  string message = 2;
}
```

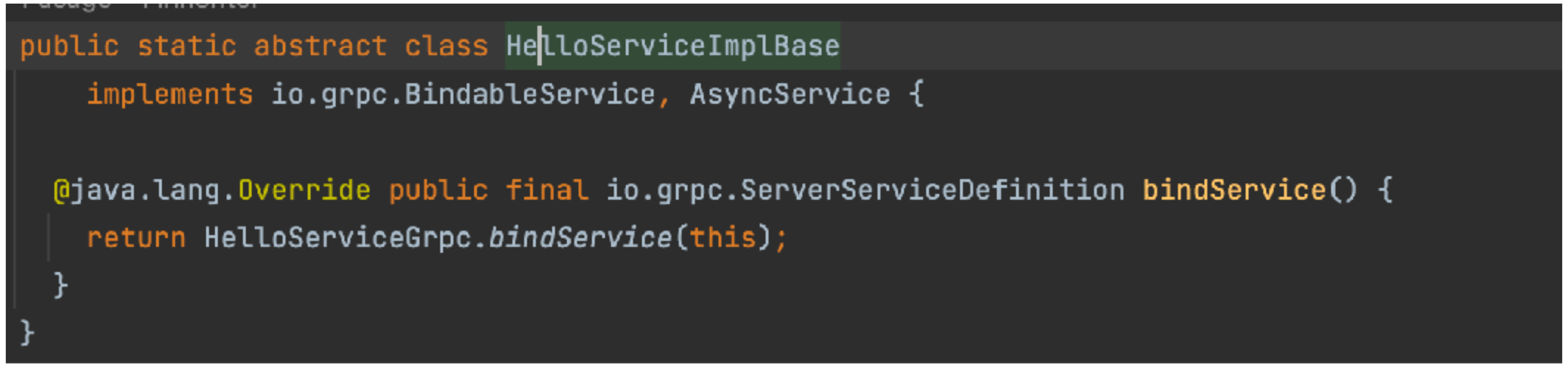
generateProto



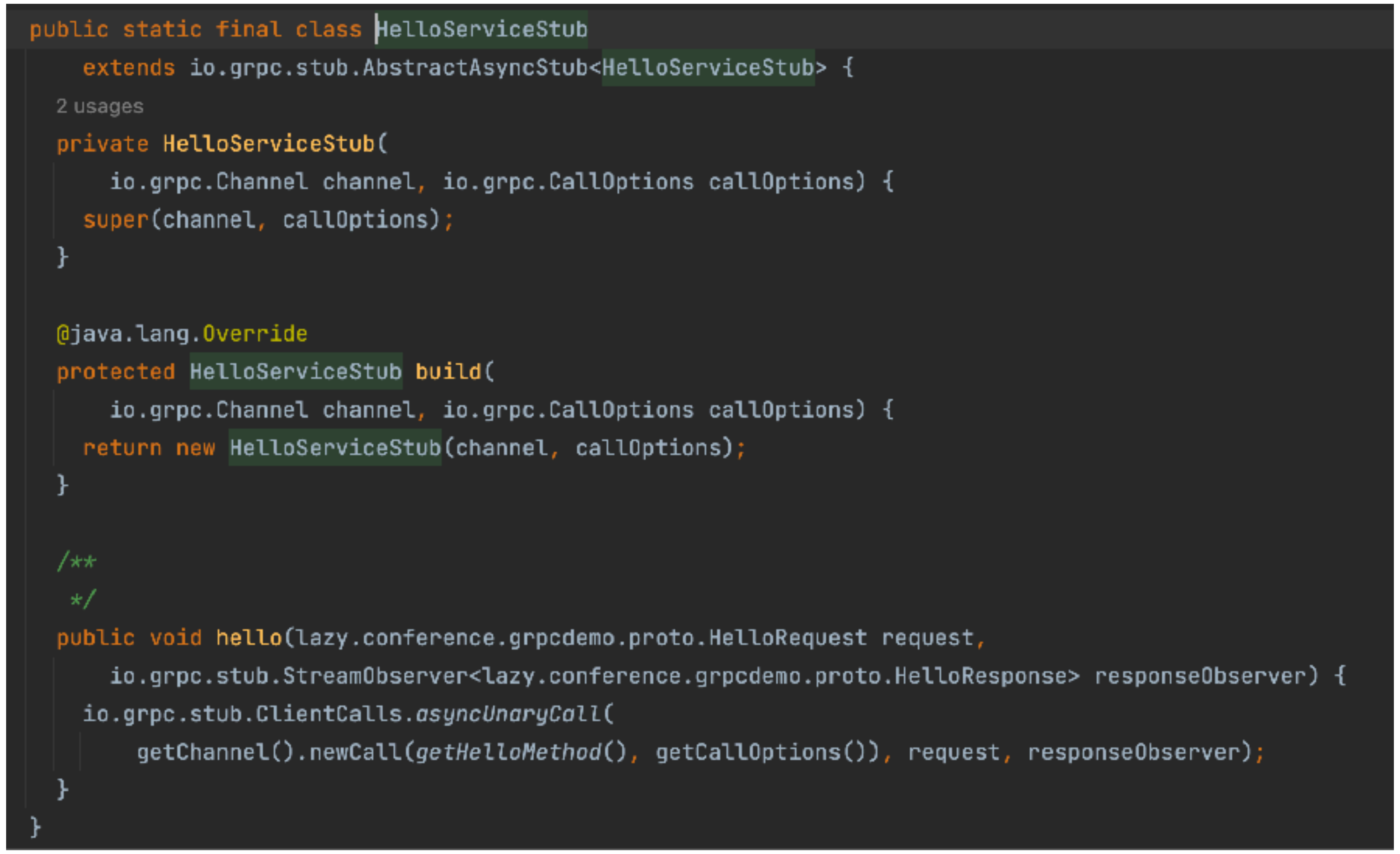
```
build
  extracted-include-protos
  extracted-protos
  generated
    source
      proto
        main
          grpc [main] sources root
            lazy.conference.grpcdemo.proto
              HelloServiceGrpc
          java [main] sources root
            lazy.conference.grpcdemo.proto
              Hello
              HelloRequest
              HelloRequestOrBuilder
              HelloResponse
              HelloResponseOrBuilder
```

gRPC 란 무엇인가 ?

HelloServiceGrpc.java



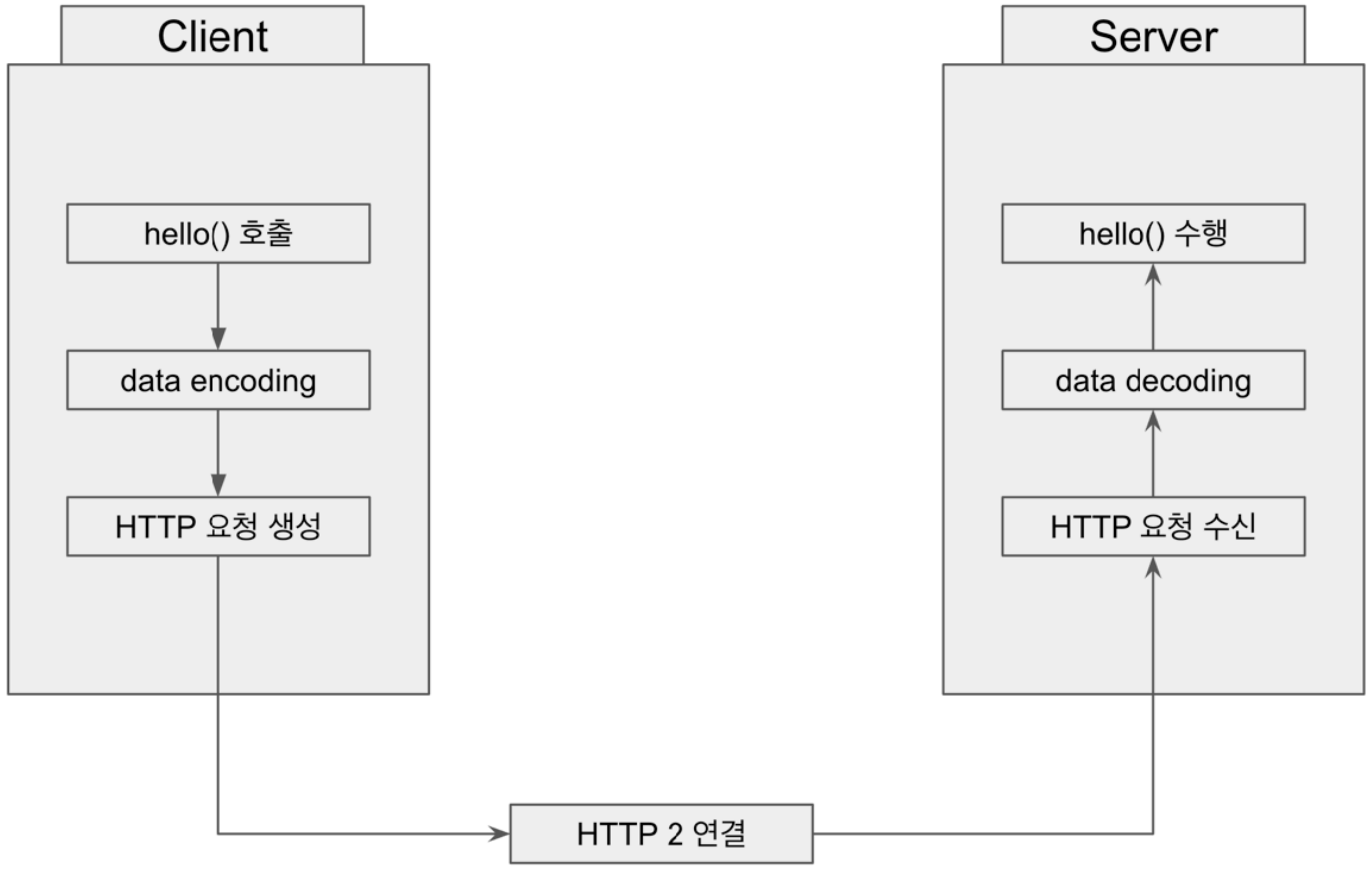
```java
public static abstract class HelloServiceImplBase
    implements io.grpc.BindableService, AsyncService {

  @java.lang.Override public final io.grpc.ServerServiceDefinition bindService() {
    return HelloServiceGrpc.bindService(this);
  }
}
```



```java
public static final class HelloServiceStub
    extends io.grpc.stub.AbstractAsyncStub<HelloServiceStub> {
  2 usages
  private HelloServiceStub(
      io.grpc.Channel channel, io.grpc.CallOptions callOptions) {
    super(channel, callOptions);
  }

  @java.lang.Override
  protected HelloServiceStub build(
      io.grpc.Channel channel, io.grpc.CallOptions callOptions) {
    return new HelloServiceStub(channel, callOptions);
  }

  /**
   */
  public void hello(lazy.conference.grpcdemo.proto.HelloRequest request,
      io.grpc.stub.StreamObserver<lazy.conference.grpcdemo.proto.HelloResponse> responseObserver) {
    io.grpc.stub.ClientCalls.asyncUnaryCall(
        getChannel().newCall(getHelloMethod(), getCallOptions()), request, responseObserver);
  }
}
```

gRPC 동작방식

gRPC 동작방식


Client
hello() 호출
data encoding
HTTP 요청 생성
Server
hello() 수행
data decoding
HTTP 요청 수신
HTTP 2 연결

gRPC 동작방식

Protocol Buffers

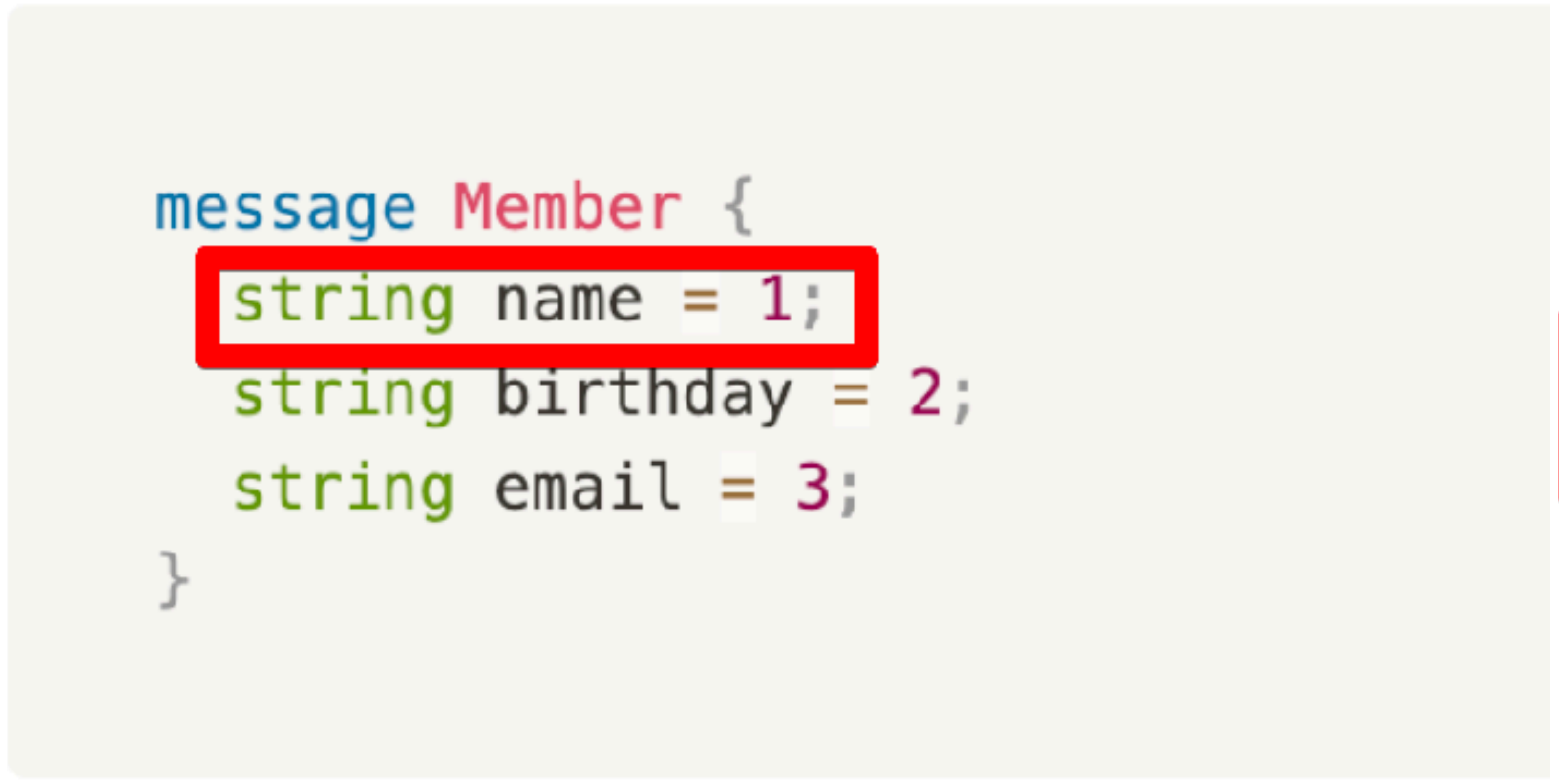
gRPC 동작방식

```
message Member {
  string name = 1;
  string birthday = 2;
  string email = 3;
}
```

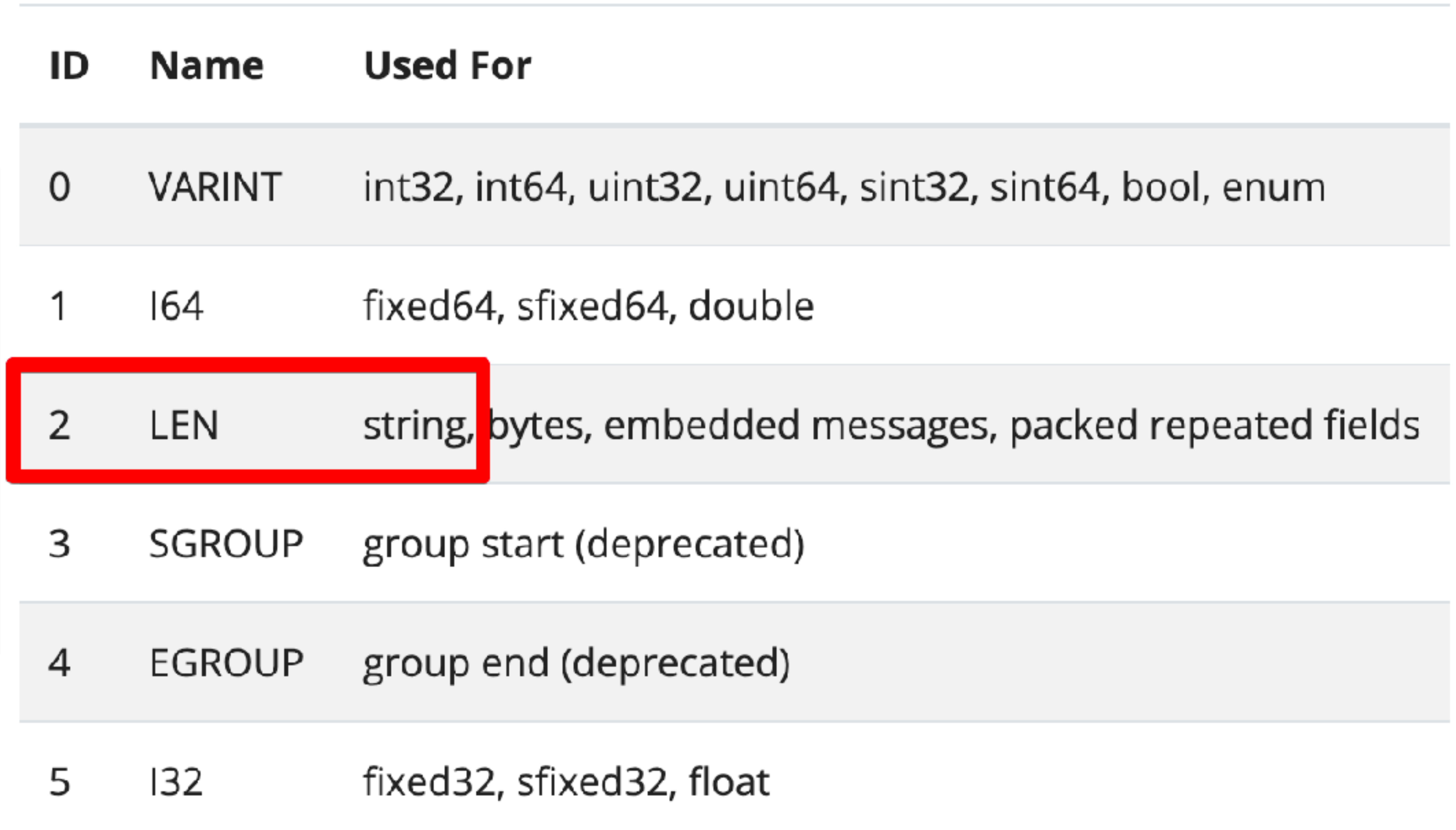
gRPC 동작방식

tag	value	tag	value	0

gRPC 동작방식



```
message Member {
  string name = 1;
  string birthday = 2;
  string email = 3;
}
```



ID	Name	Used For
0	VARINT	int32, int64, uint32, uint64, sint32, sint64, bool, enum
1	I64	fixed64, sfixed64, double
2	LEN	string, bytes, embedded messages, packed repeated fields
3	SGROUP	group start (deprecated)
4	EGROUP	group end (deprecated)
5	I32	fixed32, sfixed32, float

gRPC 동작방식

타입별로 알고리즘 최적화

gRPC 동작방식

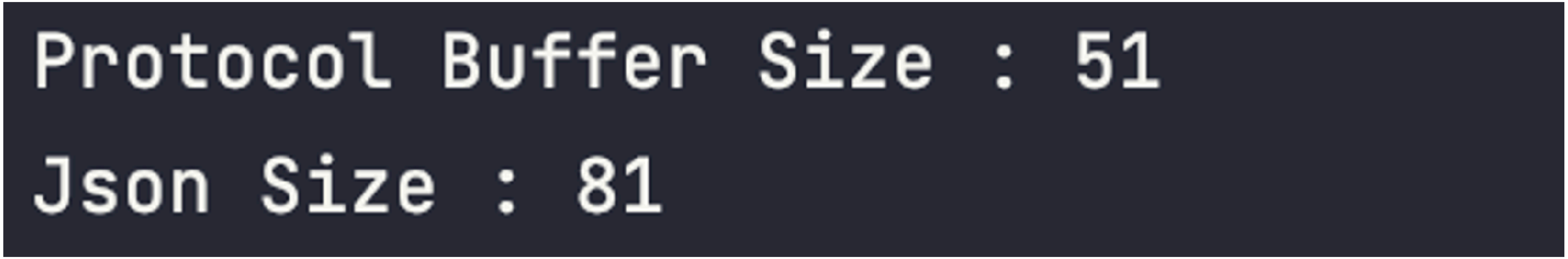
값이 있는 데이터들만 인코딩

gRPC 동작방식

```
message Member {
  string name = 1;
  string birthday = 2;
  string email = 3;
}
```

```
public class Member {
  private final String name;
  private final String birthday;
  private final String email;
}
```

gRPC 동작방식



```
Protocol Buffer Size : 51
Json Size : 81
```

gRPC 동작방식

- 5G 를 능가하는 Serialization , Deserialization 속도
- 먼지보다 작은 data size

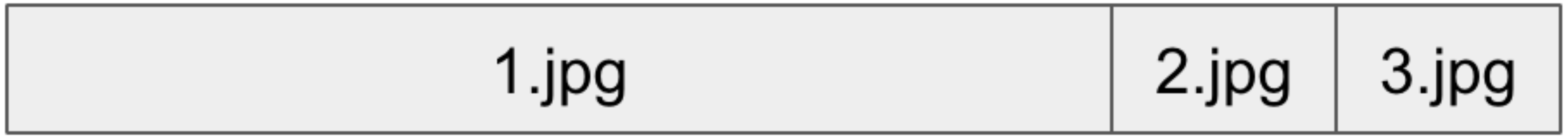
gRPC 동작방식

HTTP 2

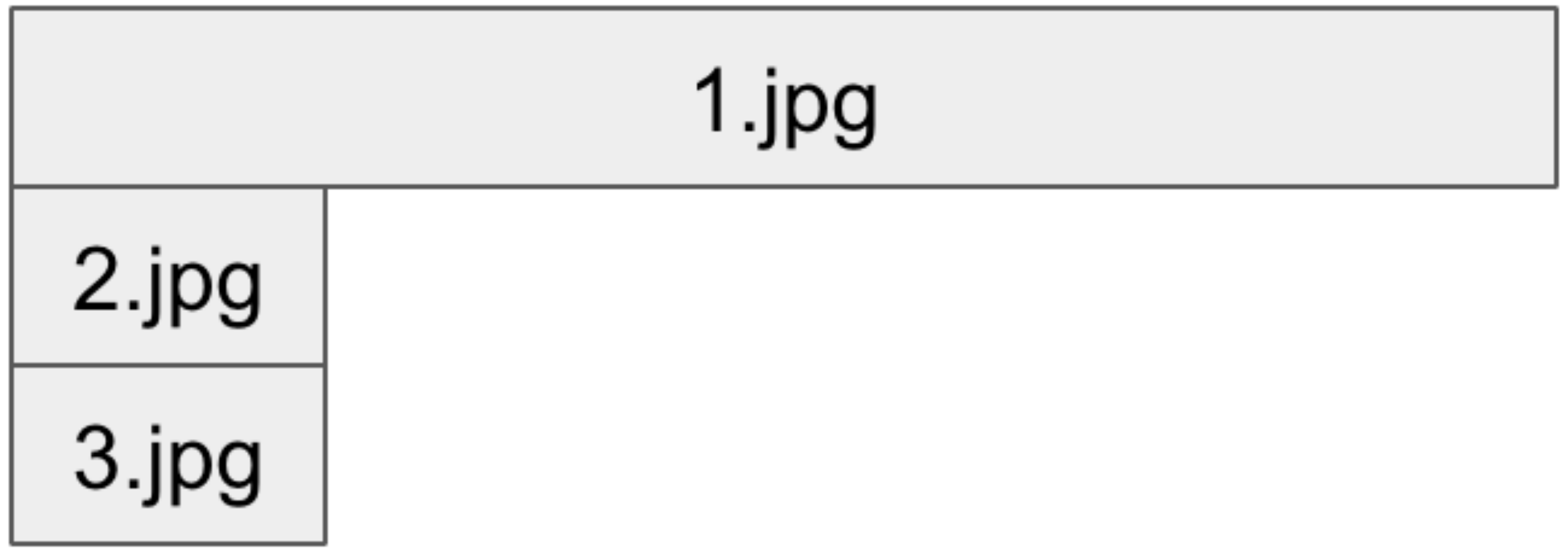
gRPC 동작방식

HOL 블로킹 문제 해결

gRPC 동작방식



gRPC 동작방식



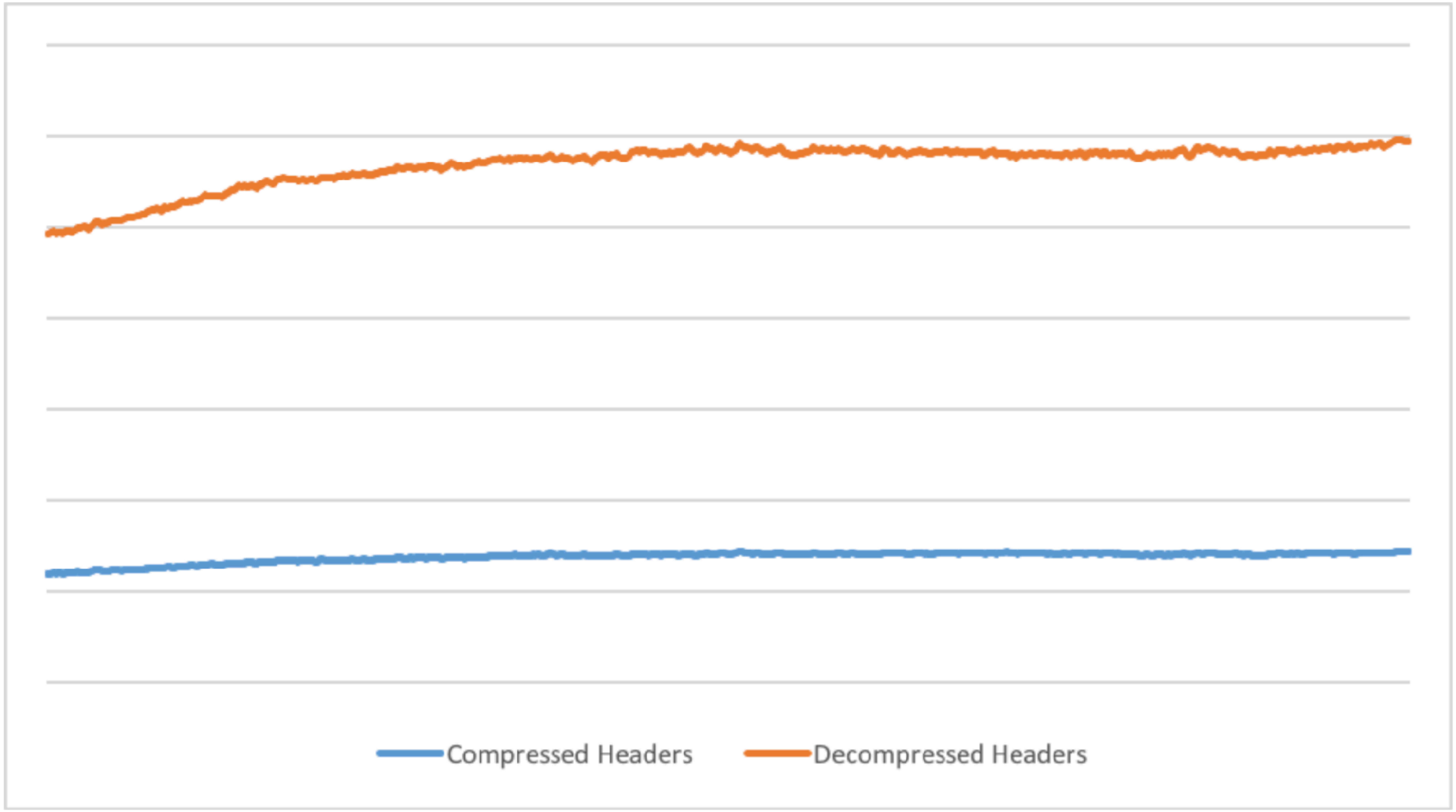
gRPC 동작방식

Multiplexing

gRPC 동작방식

헤더압축

gRPC 동작방식

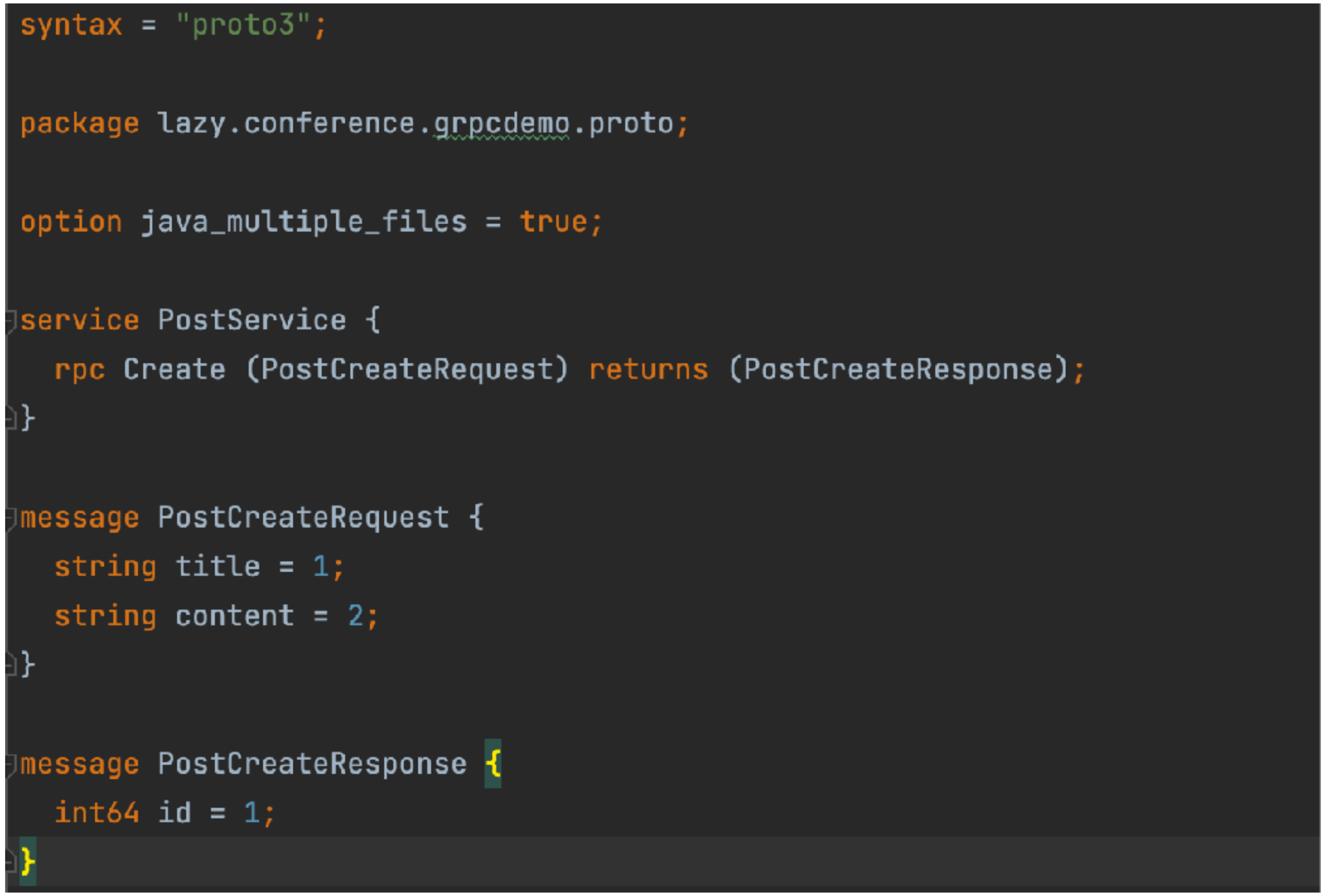


gRPC 동작방식

Protocol Buffers 와 HTTP 2

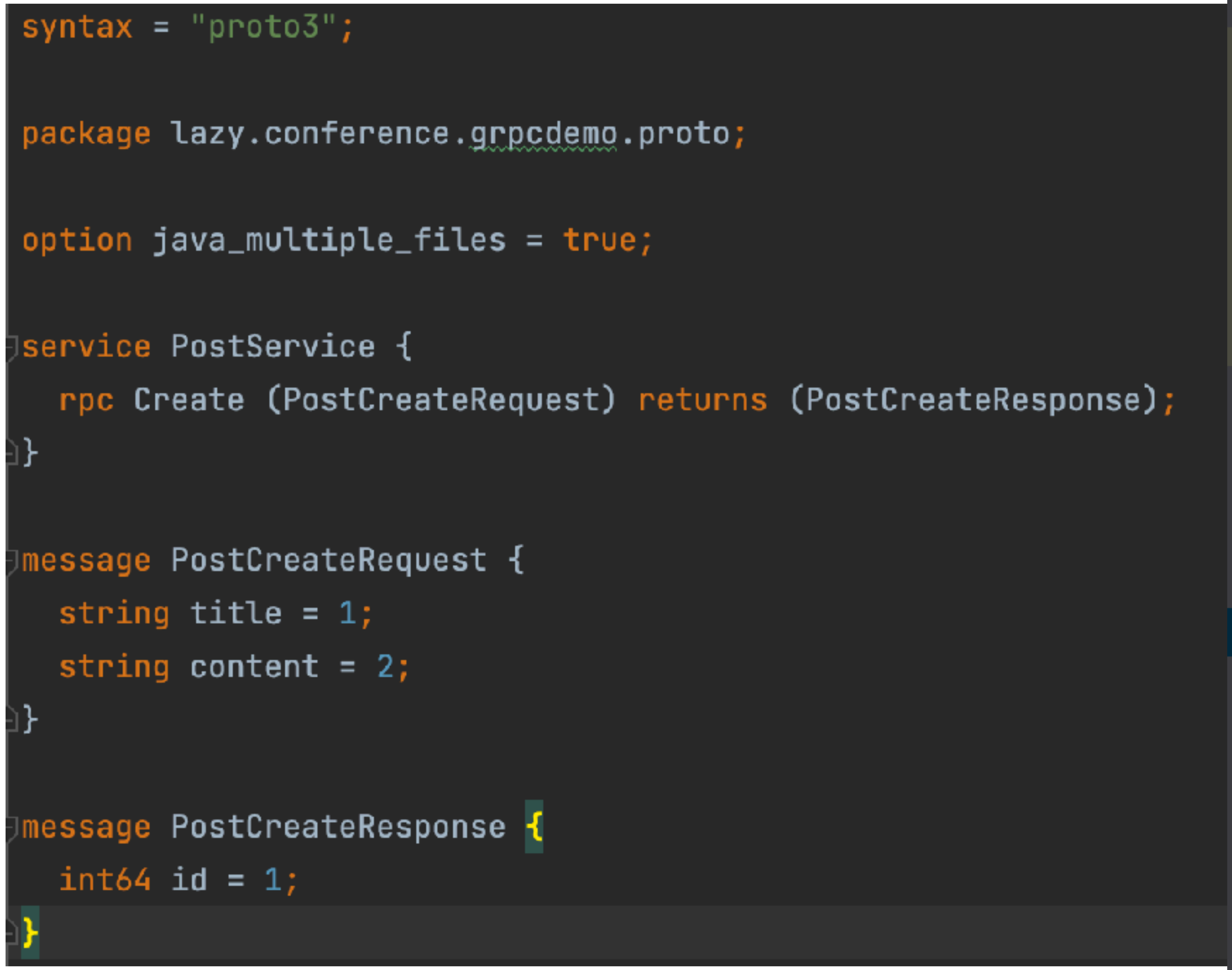
gRPC 로 구현하기

gRPC 로 구현하기

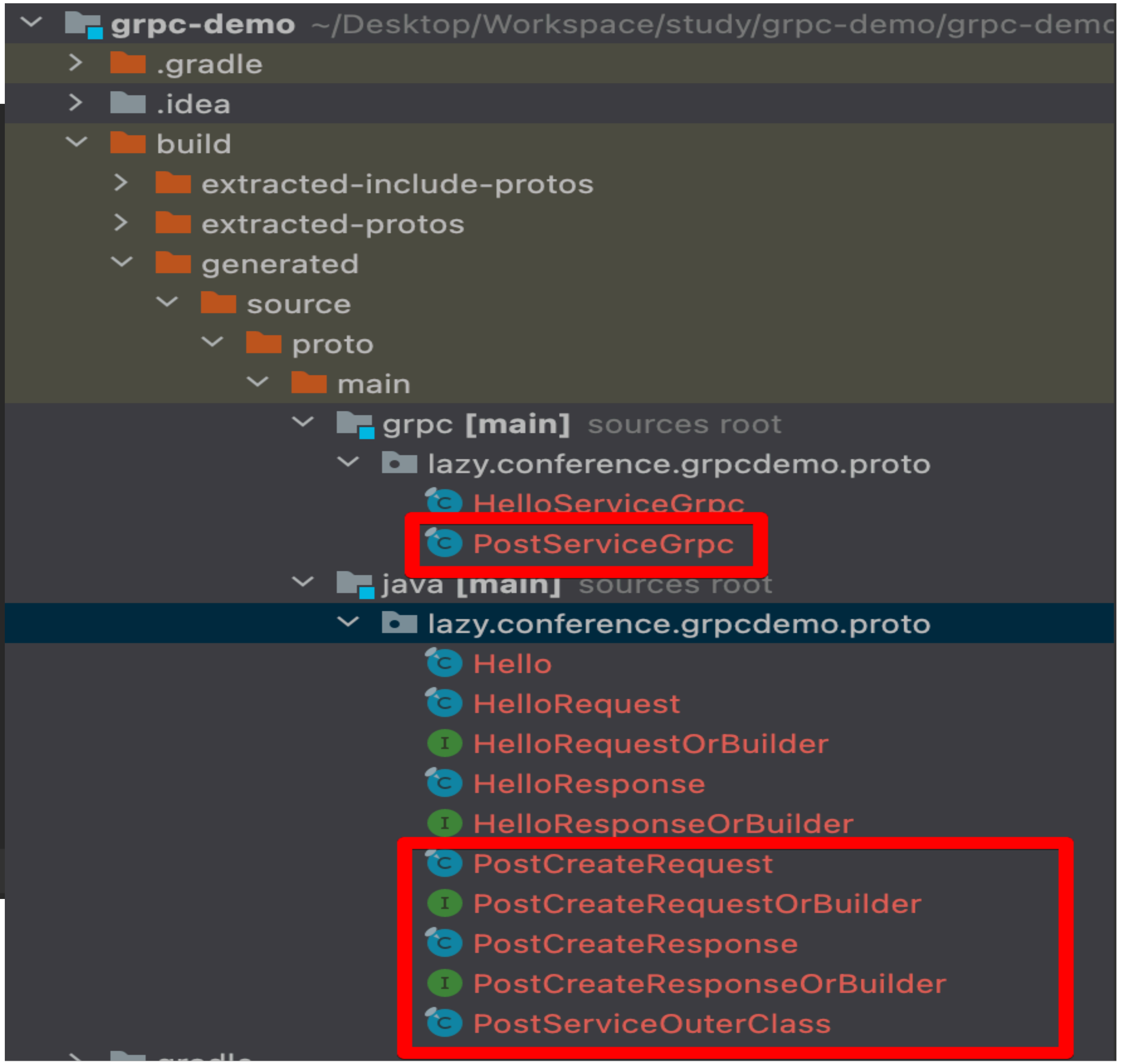


```
syntax = "proto3";

package lazy.conference.grpcdemo.proto;

option java_multiple_files = true;

service PostService {
  rpc Create (PostCreateRequest) returns (PostCreateResponse);
}

message PostCreateRequest {
  string title = 1;
  string content = 2;
}

message PostCreateResponse {
  int64 id = 1;
}
```

gRPC 로 구현하기

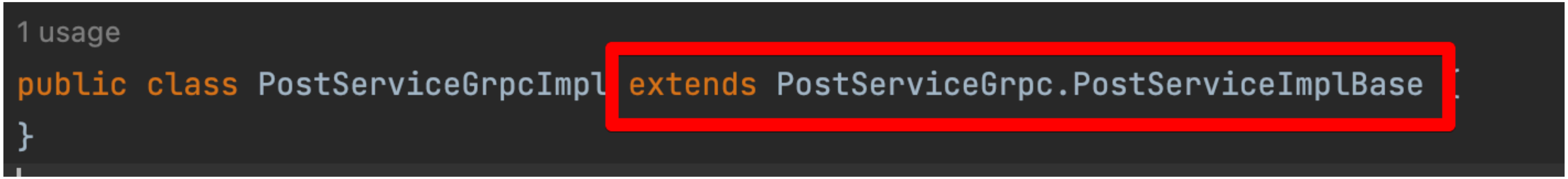


```
syntax = "proto3";

package lazy.conference.grpcdemo.proto;

option java_multiple_files = true;

service PostService {
  rpc Create (PostCreateRequest) returns (PostCreateResponse);
}

message PostCreateRequest {
  string title = 1;
  string content = 2;
}

message PostCreateResponse {
  int64 id = 1;
}
```



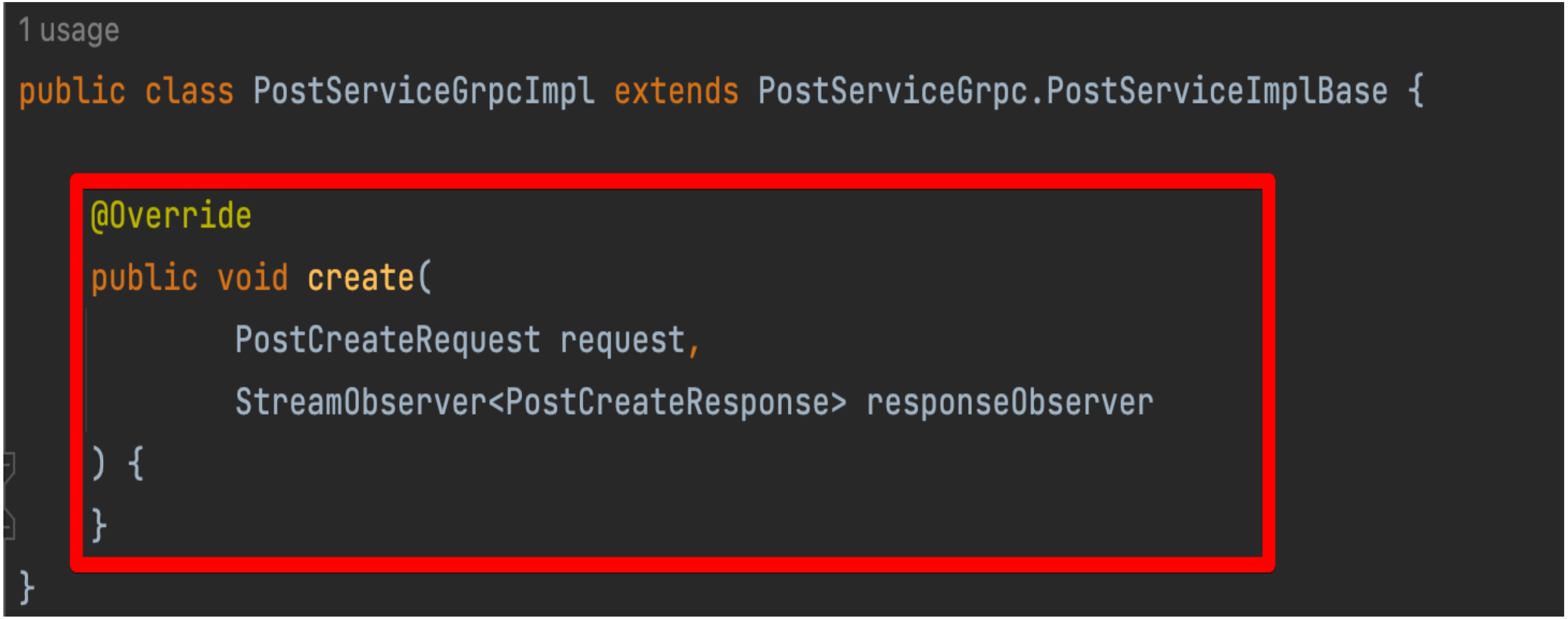
```
grpc-demo ~/Desktop/Workspace/study/grpc-demo/grpc-demo
  .gradle
  .idea
  build
    extracted-include-protos
    extracted-protos
    generated
      source
        proto
          main
            grpc [main] sources root
              lazy.conference.grpcdemo.proto
                HelloServiceGrpc
                PostServiceGrpc
            java [main] sources root
              lazy.conference.grpcdemo.proto
                Hello
                HelloRequest
                HelloRequestOrBuilder
                HelloResponse
                HelloResponseOrBuilder
                PostCreateRequest
                PostCreateRequestOrBuilder
                PostCreateResponse
                PostCreateResponseOrBuilder
                PostServiceOuterClass
```

gRPC 로 구현하기



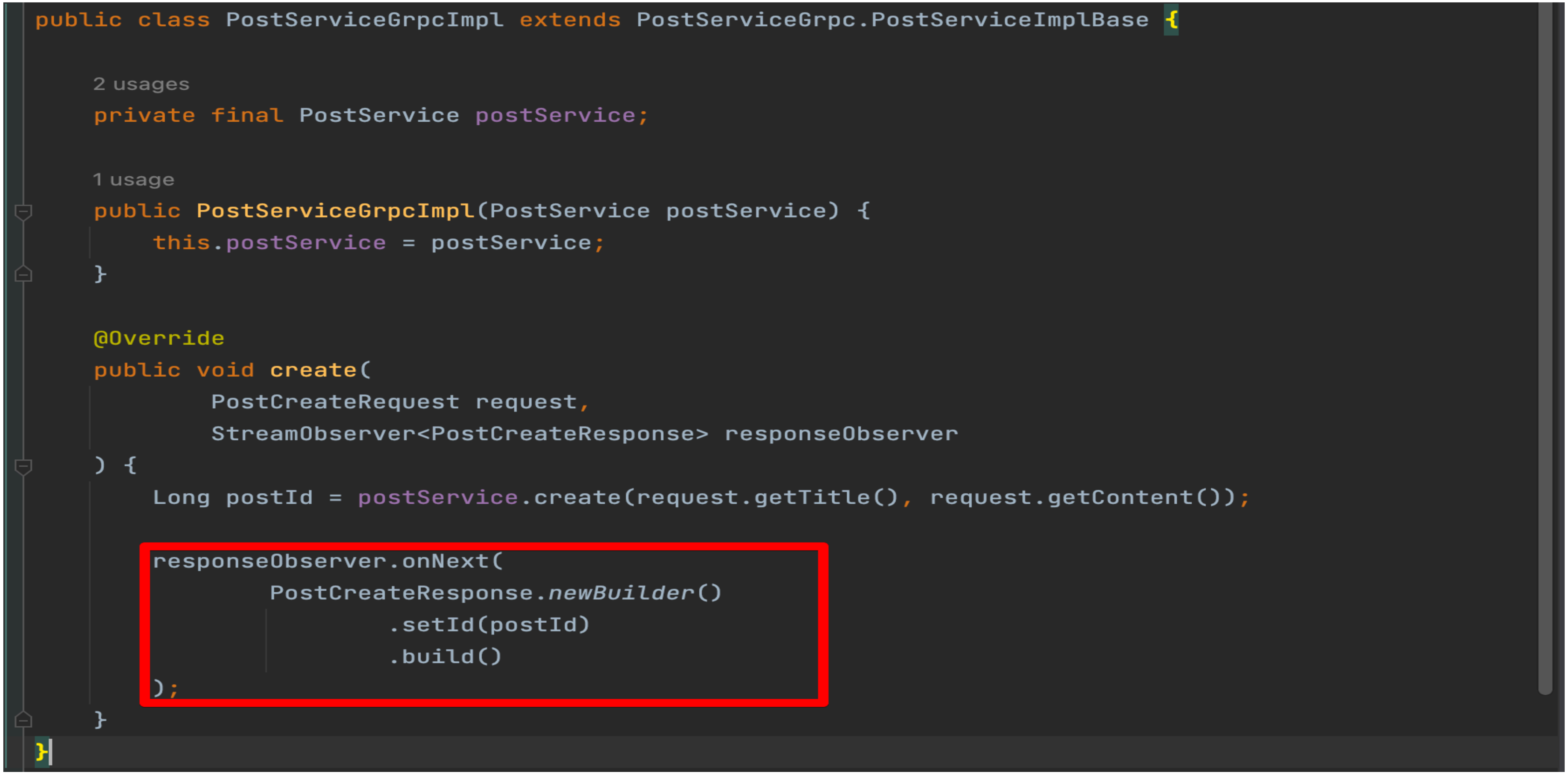
```
1 usage
public class PostServiceGrpcImpl extends PostServiceGrpc.PostServiceImplBase
}
```

gRPC 로 구현하기



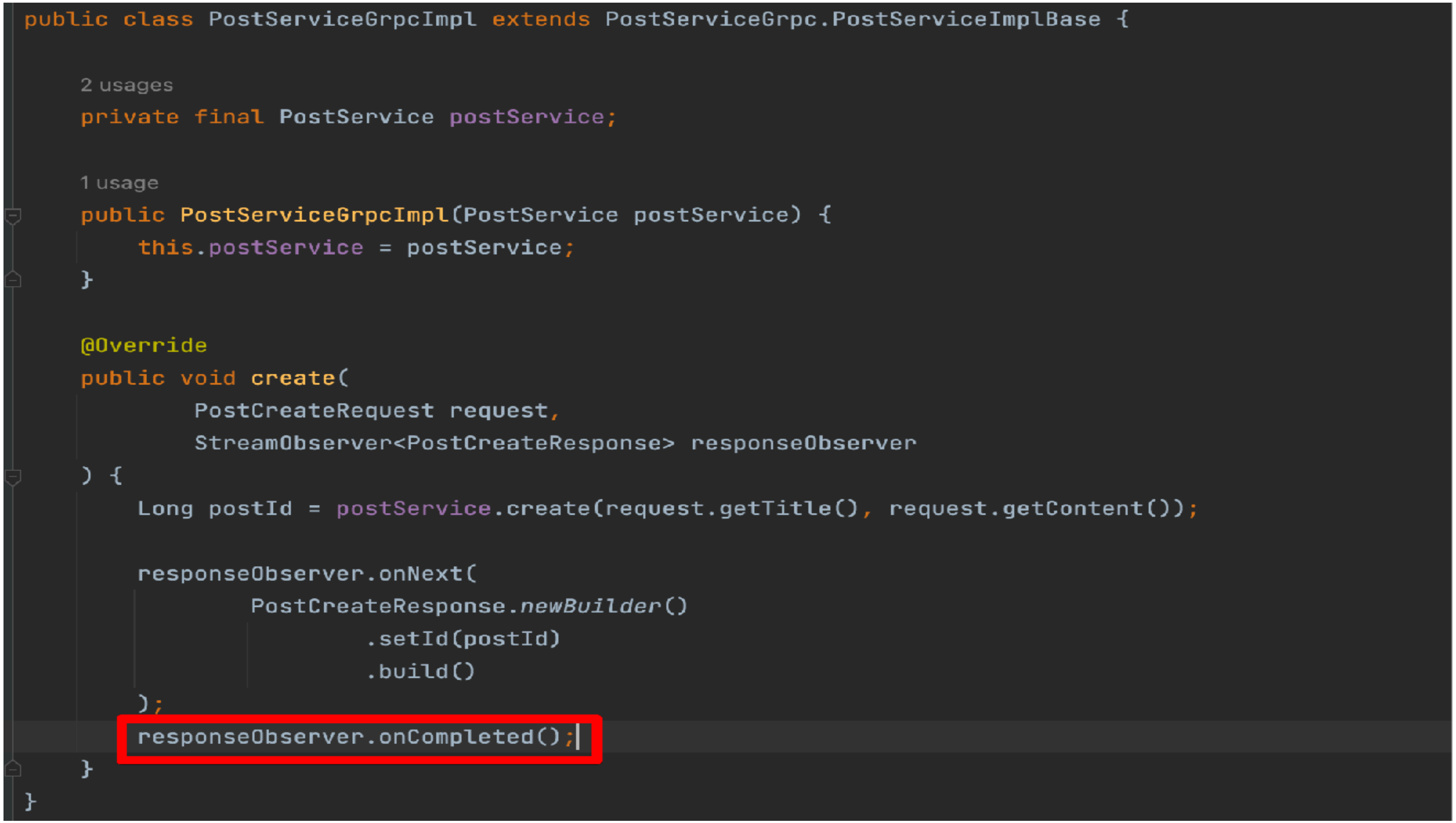
```
1 usage
public class PostServiceGrpcImpl extends PostServiceGrpc.PostServiceImplBase {

    @Override
    public void create(
            PostCreateRequest request,
            StreamObserver<PostCreateResponse> responseObserver
    ) {
    }
}
```

gRPC 로 구현하기



```java
public class PostServiceGrpcImpl extends PostServiceGrpc.PostServiceImplBase {

    2 usages
    private final PostService postService;

    1 usage
    public PostServiceGrpcImpl(PostService postService) {
        this.postService = postService;
    }

    @Override
    public void create(
            PostCreateRequest request,
            StreamObserver<PostCreateResponse> responseObserver
    ) {
        Long postId = postService.create(request.getTitle(), request.getContent());

        responseObserver.onNext(
                PostCreateResponse.newBuilder()
                        .setId(postId)
                        .build()
        );
    }
}
```

gRPC 로 구현하기

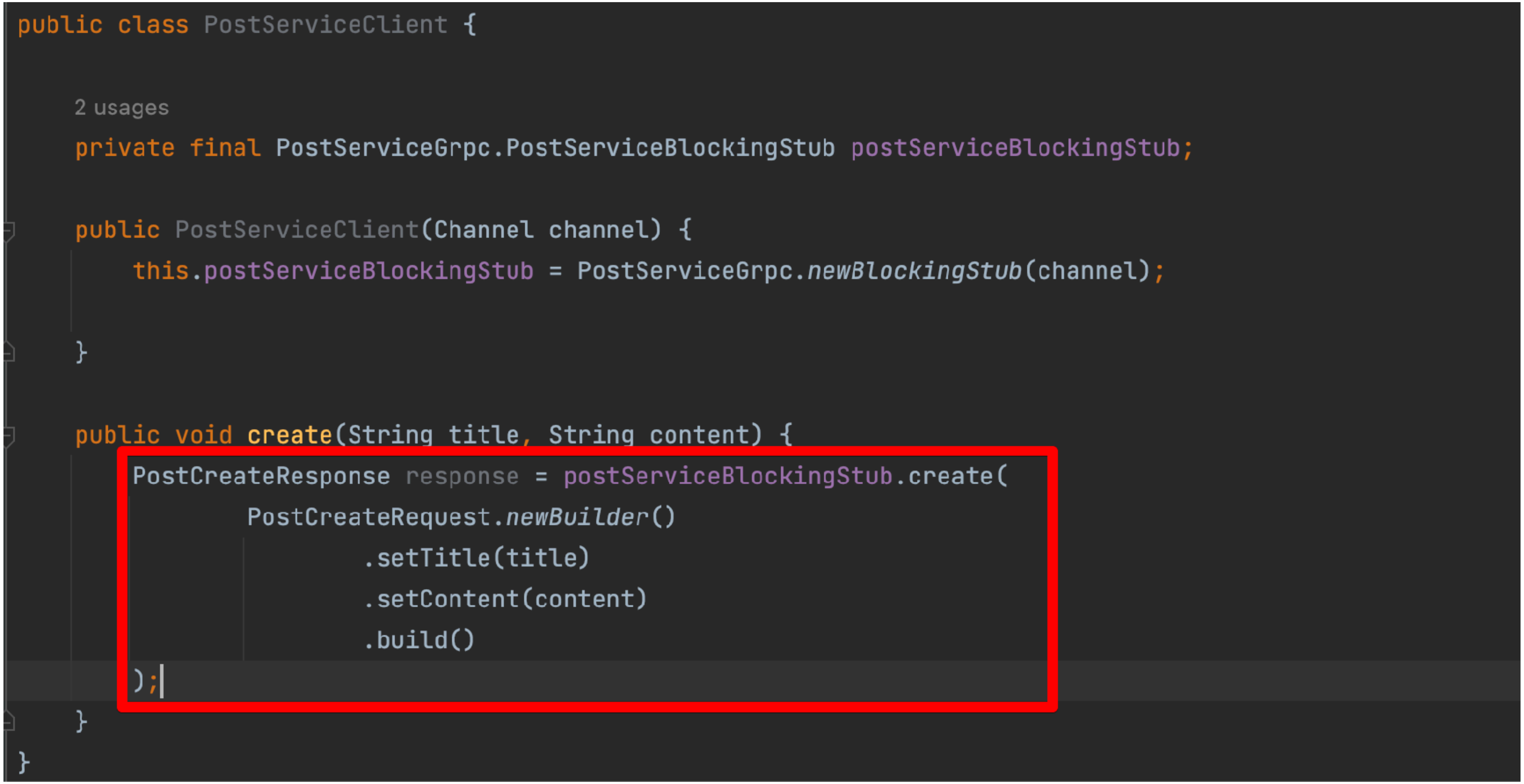


```java
public class PostServiceGrpcImpl extends PostServiceGrpc.PostServiceImplBase {

    2 usages
    private final PostService postService;

    1 usage
    public PostServiceGrpcImpl(PostService postService) {
        this.postService = postService;
    }

    @Override
    public void create(
            PostCreateRequest request,
            StreamObserver<PostCreateResponse> responseObserver
    ) {
        Long postId = postService.create(request.getTitle(), request.getContent());

        responseObserver.onNext(
                PostCreateResponse.newBuilder()
                        .setId(postId)
                        .build()
        );
        responseObserver.onCompleted();
    }
}
```

gRPC 로 구현하기

```
public class PostServiceClient {

    1 usage
    private final PostServiceGrpc.PostServiceBlockingStub postServiceBlockingStub;

    public PostServiceClient(Channel channel) {
        this.postServiceBlockingStub = PostServiceGrpc.newBlockingStub(channel);

    }
}
```

gRPC 로 구현하기



```
public class PostServiceClient {

    2 usages
    private final PostServiceGrpc.PostServiceBlockingStub postServiceBlockingStub;

    public PostServiceClient(Channel channel) {
        this.postServiceBlockingStub = PostServiceGrpc.newBlockingStub(channel);

    }

    public void create(String title, String content) {
        PostCreateResponse response = postServiceBlockingStub.create(
                PostCreateRequest.newBuilder()
                        .setTitle(title)
                        .setContent(content)
                        .build()
        );
    }
}
```

주요 기능

주요 기능

Synchronous and Asynchronous

주요 기능

Metadata

주요 기능

Channel

Best practice

Best Practice

Reusing Channel

Best Practice

Use keep-alive pings

Best Practice

Use non-blocking Stub

REST vs gRPC

REST vs gRPC - HTTP Version

HTTP vs HTTP 2.0

REST vs gRPC - HTTP METHOD

[GET, POST, PATCH, DELETE] vs POST

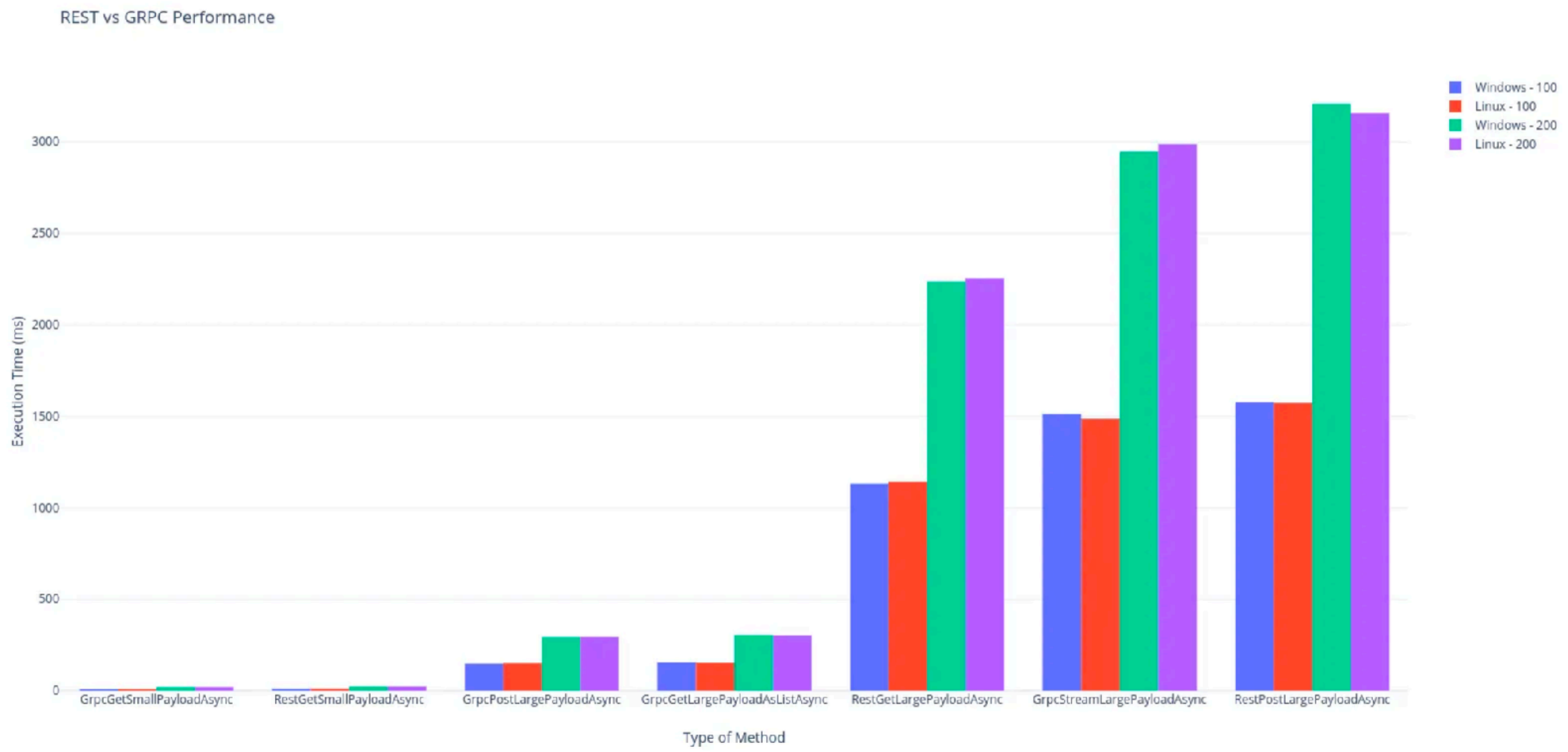
REST vs gRPC - 데이터 형식

JSON vs Binary

REST vs gRPC - 구현속도

빠름 vs 느림

REST vs gRPC - 성능비교



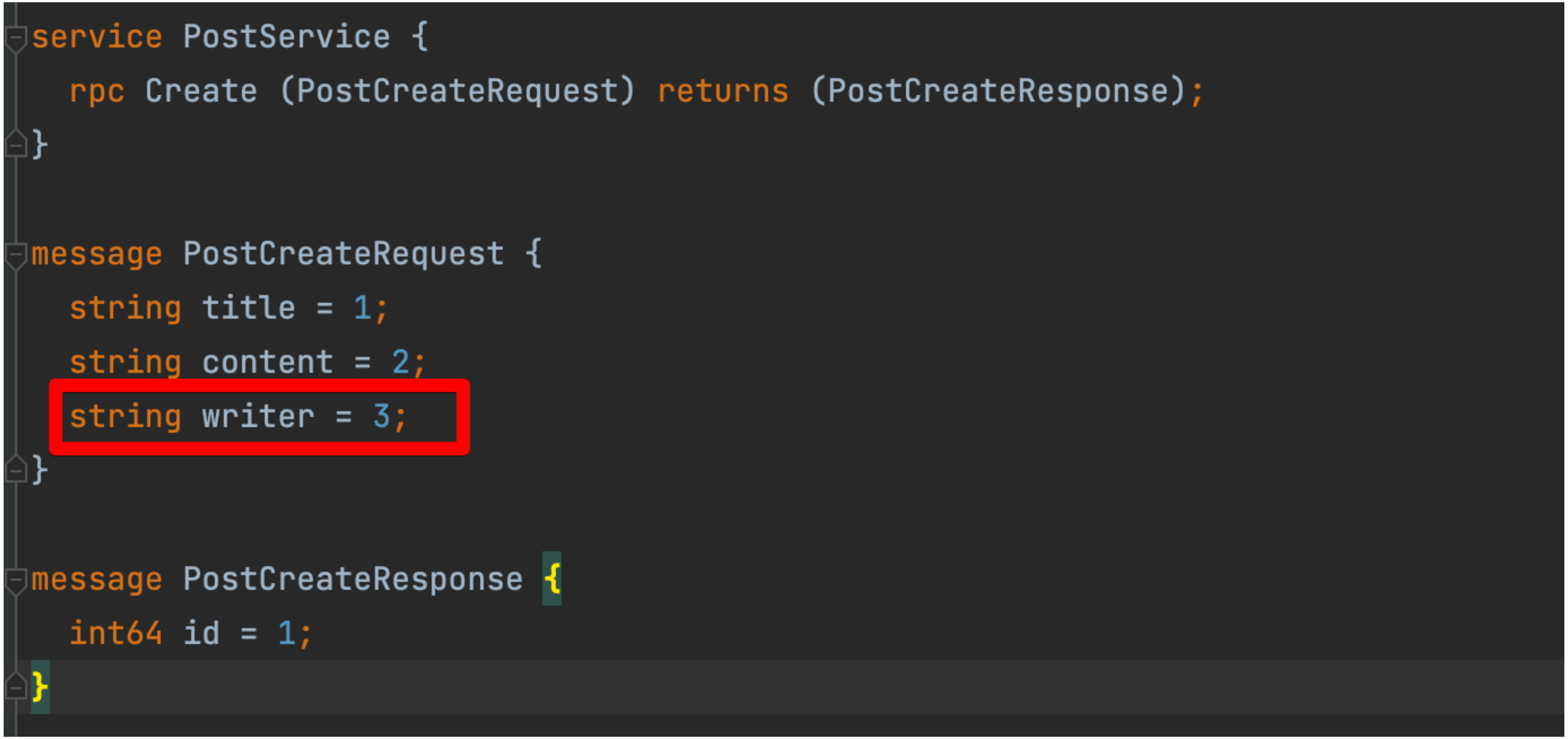
출처 : https://medium.com/@EmperorRXF/evaluating-performance-of-rest-vs-grpc-1b8bdf0b22da

사용해본 경험

사용해본 경험

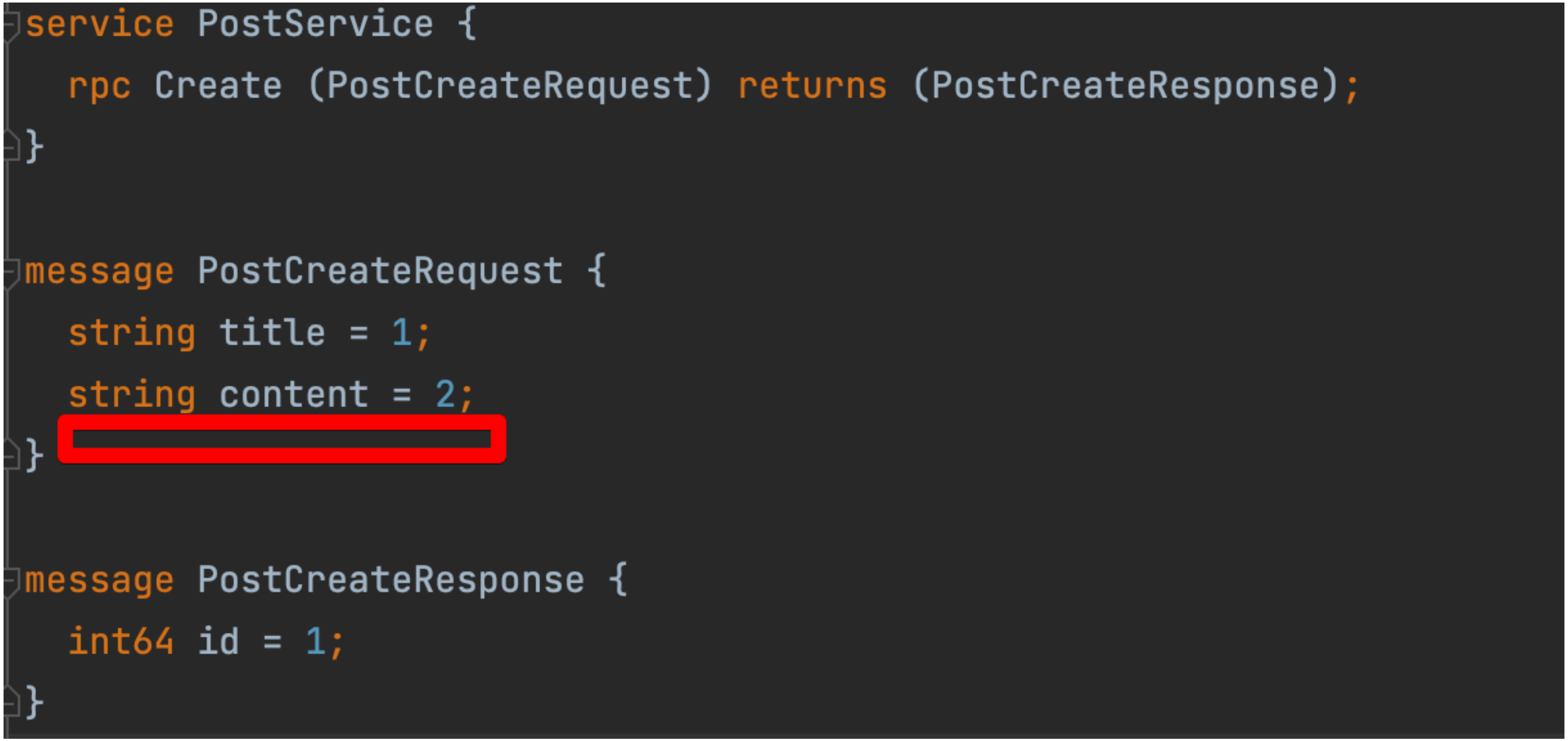
Proto 관리를 잘 해야한다.

사용해본 경험



```
service PostService {
  rpc Create (PostCreateRequest) returns (PostCreateResponse);
}

message PostCreateRequest {
  string title = 1;
  string content = 2;
  string writer = 3;
}

message PostCreateResponse {
  int64 id = 1;
}
```

사용해본 경험

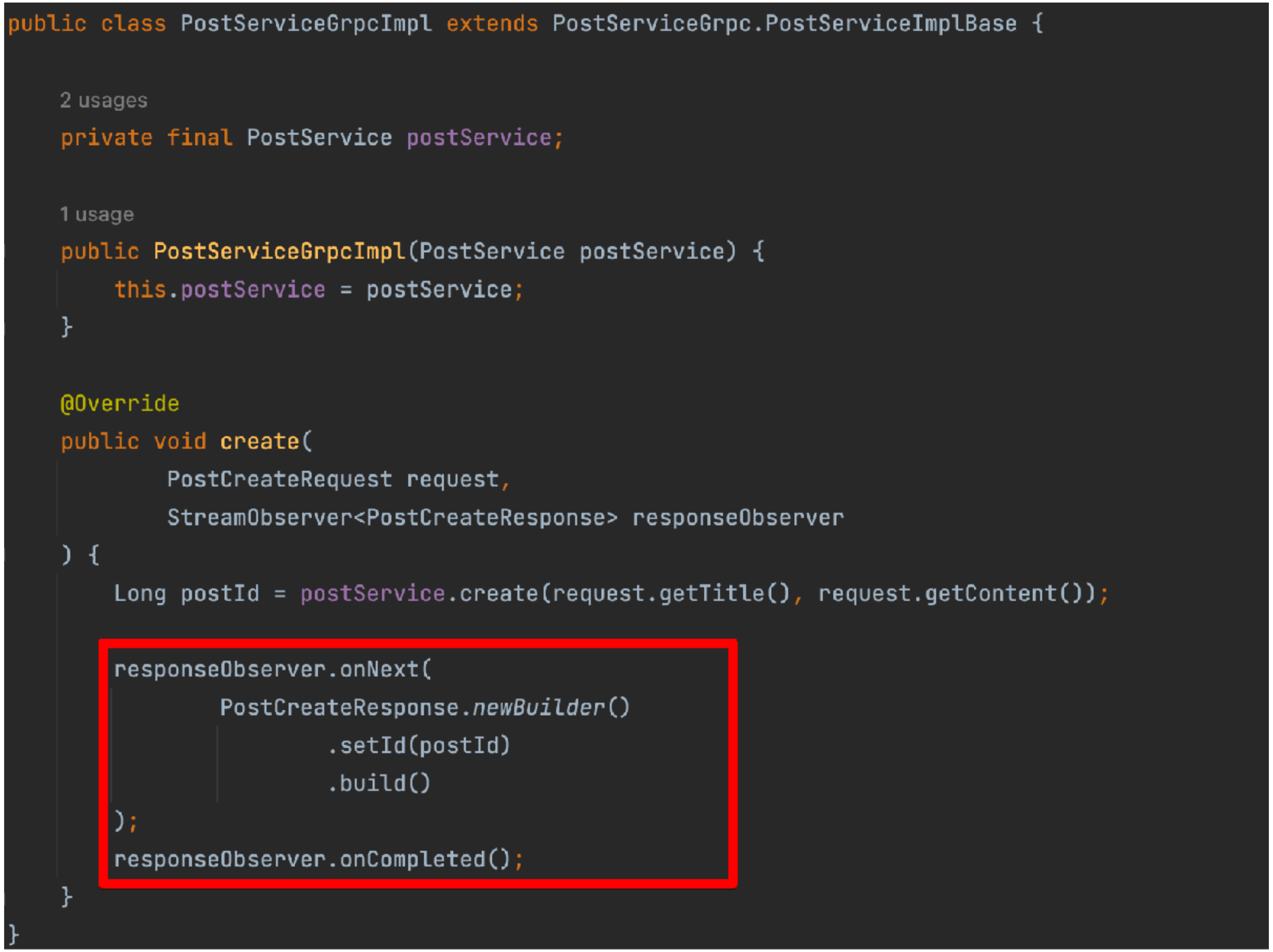


```
service PostService {
  rpc Create (PostCreateRequest) returns (PostCreateResponse);
}

message PostCreateRequest {
  string title = 1;
  string content = 2;
}

message PostCreateResponse {
  int64 id = 1;
}
```

사용해본 경험

Kotlin 과 함께라면 REST 를 개발하는 것과 비슷하게 개발할 수 있다.

사용해본 경험



```java
public class PostServiceGrpcImpl extends PostServiceGrpc.PostServiceImplBase {

    2 usages
    private final PostService postService;

    1 usage
    public PostServiceGrpcImpl(PostService postService) {
        this.postService = postService;
    }

    @Override
    public void create(
            PostCreateRequest request,
            StreamObserver<PostCreateResponse> responseObserver
    ) {
        Long postId = postService.create(request.getTitle(), request.getContent());

        responseObserver.onNext(
                PostCreateResponse.newBuilder()
                        .setId(postId)
                        .build()
        );
        responseObserver.onCompleted();
    }
}
```

사용해본 경험

```
class HelloServer : HelloServiceGrpcKt.HelloServiceCoroutineImplBase() {

    override suspend fun hello(request: HelloRequest): HelloResponse {
        return HelloResponse.newBuilder()
            .setMessage("success!!")
            .build()
    }
}
```

사용해본 경험

Armeria 와 같은 Framework 를 사용한다면 편리한 기능들을 사용할 수 있다.

사용해본 경험

REST 로 서비스가 쾌적하게 운영되며 gRPC 를 운영해본 구성원이 존재하지 않는다.

사용해본 경험

고성능 애플리케이션이 필요해졌다.

사용해본 경험

전송해야 하는 데이터가 너무 크다.

참고자료

- https://www.techtarget.com/searchapparchitecture/definition/Remote-Procedure-Call-RPC
- https://protobuf.dev/
- https://grpc.io/docs/
- https://www.yes24.com/Product/Goods/94489227?pid=123487&cosemkid=go16049119980413337&gad_source=1&gclid=CjwKCAiAyp-sBhBSEiwAWWzTnkDDrVxiZdZsD8Re2lOMw-Z13qBONFWj04UBrAgBH0iAUec6zwx5ehoCTeIQAvD_BwE
- https://medium.com/@EmperorRXF/evaluating-performance-of-rest-vs-grpc-1b8bdf0b22da
- https://blog.cloudflare.com/hpack-the-silent-killer-feature-of-http-2

마무리

Q&A

Q&A

E.O.D