

# g + 베운 RPC, gRPC



SpringCamp 2017  
오명운

[homo.efficio@gmail.com](mailto:homo.efficio@gmail.com)

<http://homoefficio.github.io/>



# 여백





swimmers.stream()

## for-loop 를 Stream.forEach() 로 바꾸지 말아야 할 3가지 이유

Jun 26, 2016 in Technique, 벤역

`.filter(...)`

## 대용량 파일을 AsynchronousFileChannel로 다뤄보기

Aug 13, 2016 in Technique

`long endTime = System.nanoTime();``System.err.println("비정상 종료 : " + (endTime -`

## Java8 람다 관련 스펙 정리

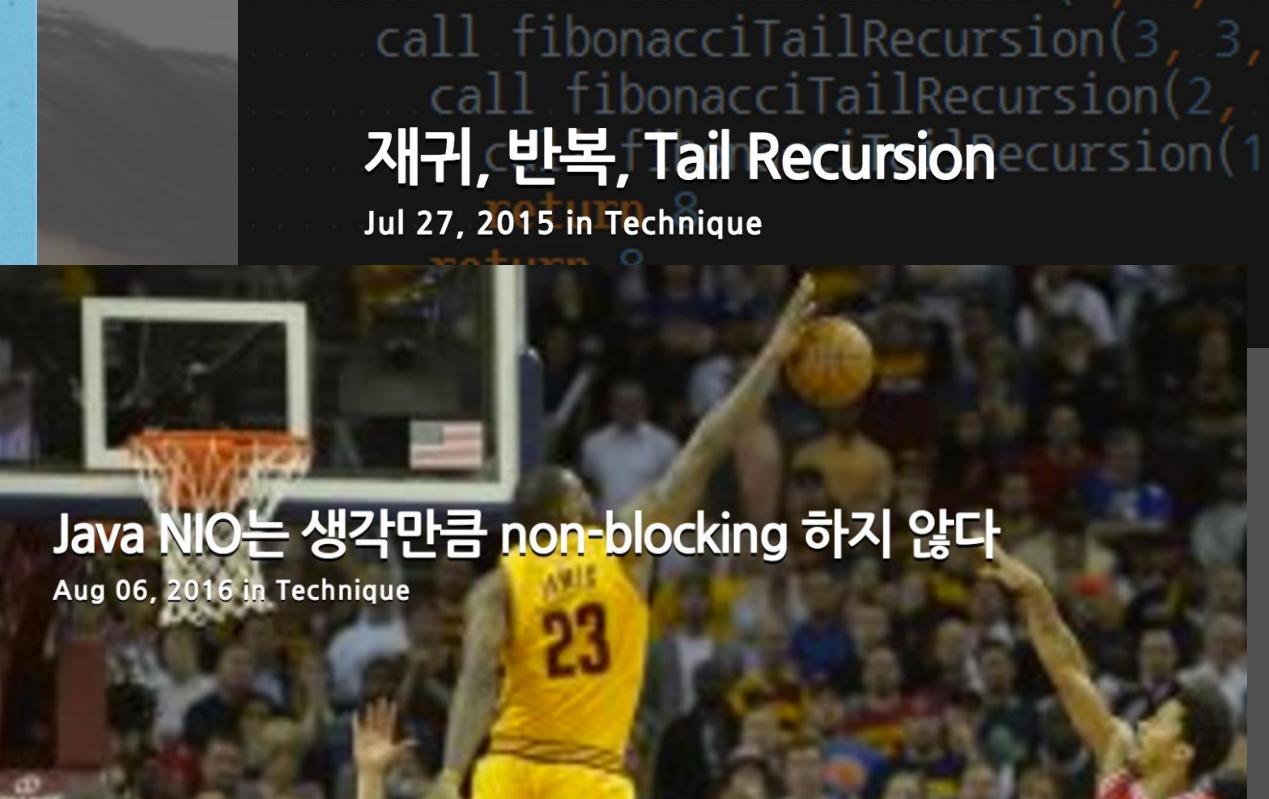
Feb 19, 2017 in Technique

자바는 아직 함수가 독립적으로 어떤 값에 할당되거나, 어떤 함수의 인으로 사용될 수 없다. 대신에 Java8에서부터는 추상 메서드를 한 개만 페이스라는 것을 언어의 기능으로 추가해서 할당, 인자 또는 반환에 사스의 자리에 람다식이나 메서드 레퍼런스를 사용할 수 있게 해서, 간접

[Continue reading](#)

## 재귀, 반복, Tail Recursion

Jul 27, 2015 in Technique



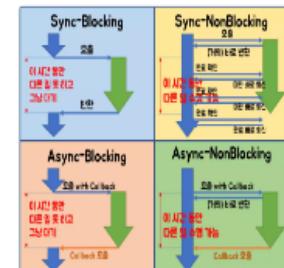
## Java NIO는 생각만큼 non-blocking 하지 않다

Aug 06, 2016 in Technique

## Blocking-NonBlocking-Synchronous-Asynchronous

Feb 19, 2017 in Concepts

꽤 자주 접하는 용어다. 특히나 요즘들어 더 자주 접하게 되는데, 얼추 알고는 있고 알고 있는 게 틀린 것도 아니지만, 막상 명확하게 구분해서 설명하라면 또 만만치가 않은.. 그래서 찾아보면 또 대충 뭔소린지 알아들을 것 같다가도, 구분해서 설명하라면 머뭇거리게 되긴 마찬가지다. 자료마다 미세하나마 조금씩 차이가 있는 것들도 많아서, 정확하고 유일한 구분법은

[Continue reading](#)

for-loop 를  
지 이유

Jun 26, 2016 in Techniques

대용량  
파일

Aug 13, 2016 in Techniques

Java8 람다  
구조

Feb 19, 2017 in Techniques

자바는 아직 함수가

으로 사용될 수 없다.

페이스라는 것을 언어

스의 자리에 람다식이나 메서드 레퍼런스를 사용할 수 있게 해서, 간접

Continue reading

swimmers.stream()



COMMUNITY EXPERIENCE DISTILLED

## Spring Microservices

Build scalable microservices with Spring, Docker, and Mesos

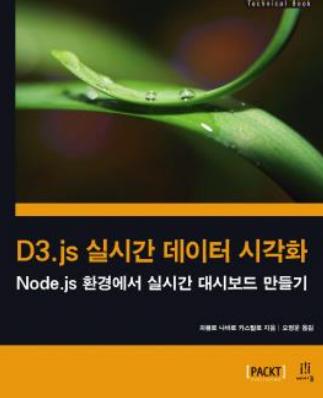
Rajesh RV

[PACKT]  
PUBLISHING

call fibonacciTailRecursion(3, 3,  
call fibonacciTailRecursion(2,  
재귀, 반복, Tail Recursion

Jul 27, 2017

SCORN-PACKT  
TECHNICAL BOOK



D3.js 실시간 데이터 시각화

Node.js 환경에서 실시간 대시보드 만들기

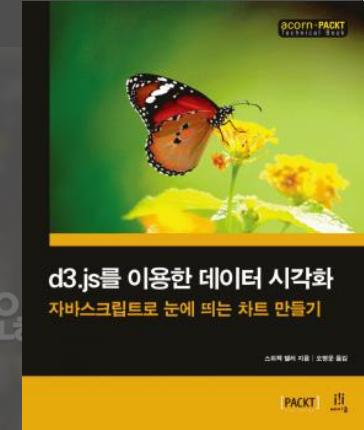
지혜로 나이트 카스텔로 저작 | 오현준 편집



Gerrit 코드 리뷰

웹 기반 협업형 온라인 코드 리뷰 시스템

주저 달라서시오 저작 | 오현준 편집



d3.js를 이용한 데이터 시각화

자바스크립트로 눈에 띠는 차트 만들기

스티브 달리 저작 | 오현준 편집

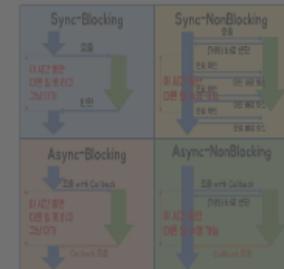
## NonBlocking-Synchronous-

nonblocking

concepts

어다. 특히나 요즘들어 더 자주 접하게 되는데, 얼추 알고는 있고 알고 있는  
게 틀린 것도 아니지만, 막상 명확하게 구분해서 설명하라면 또 만만치가 않은.. 그래서 찾아  
보면 또 대충 뭔소린지 알아들을 것 같다가도, 구분해서 설명하라면 머뭇거리게 되긴 마찬가  
지다. 자료마다 미세하나마 조금씩 차이가 있는 것들도 많아서, 정확하고 유일한 구분법은

Continue reading



2,000,000,000

2,000,000,000

구글이  
1주일 동안 띄우는  
컨테이너의 수



10,000,000,000

# 10,000,000,000



매콤통통 쫀득쫀득  
먹고싶다 밀떡볶이

10,000,000,000



매콤통통 쫀득쫀득  
먹고싶다 밀떡볶이

하루 세번  
3,044년

10,000,000,000

구글이

1초 동안 던지는

원격 호출의 수

원격 호출

Remote Procedure Call

원격 호출

Remote Procedure Call

g | 벼운 RPC, gRPC

# 목 차

- Remote Procedure Call
- gRPC?
- gRPC의 장단점 및 쓸모
- gRPC 속으로

# 오늘의 목표

# 오늘의 목표

뭐야. X라 쉽잖아..

이런 건 나도 하겠다.



# Remote Procedure Call

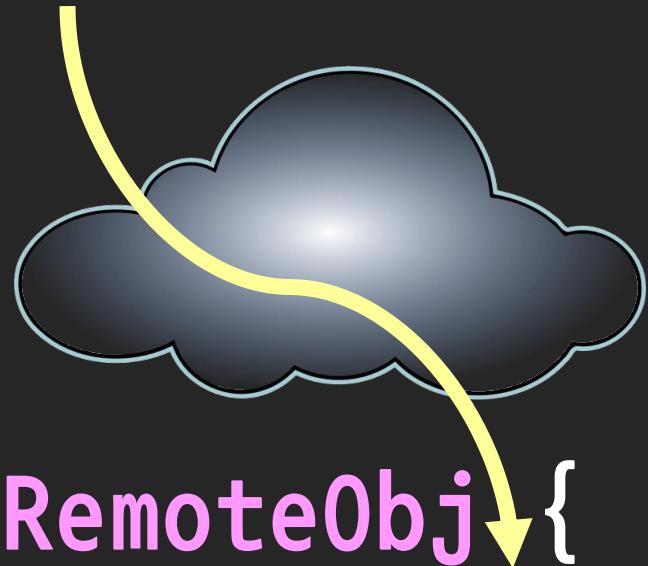
원격에 존재하는  
프로시저(함수)를  
마치 로컬에 존재하는 것처럼  
호출하는 것

# Remote Procedure Call

localObj.drinkBeerWith(chicken)

# Remote Procedure Call

localObj.drinkBeerWith(chicken)



```
public class RemoteObj{  
    public void drinkBeerWith(Anjoo anjoo) {}  
}
```

# Remote Procedure Call

원격에 존재하는 프로시저(함수)를  
로컬에서처럼 호출하려면

- 원격과의 연결
- 호출/데이터 형식을 어떻게 맞출지

# Remote Procedure Call

- 원격과의 연결
- 호출/데이터 형식을 어떻게 맞출지

S\*

# Remote Procedure Call

- 원격과의 연결
- 호출/데이터 형식을 어떻게 맞출지

Stub

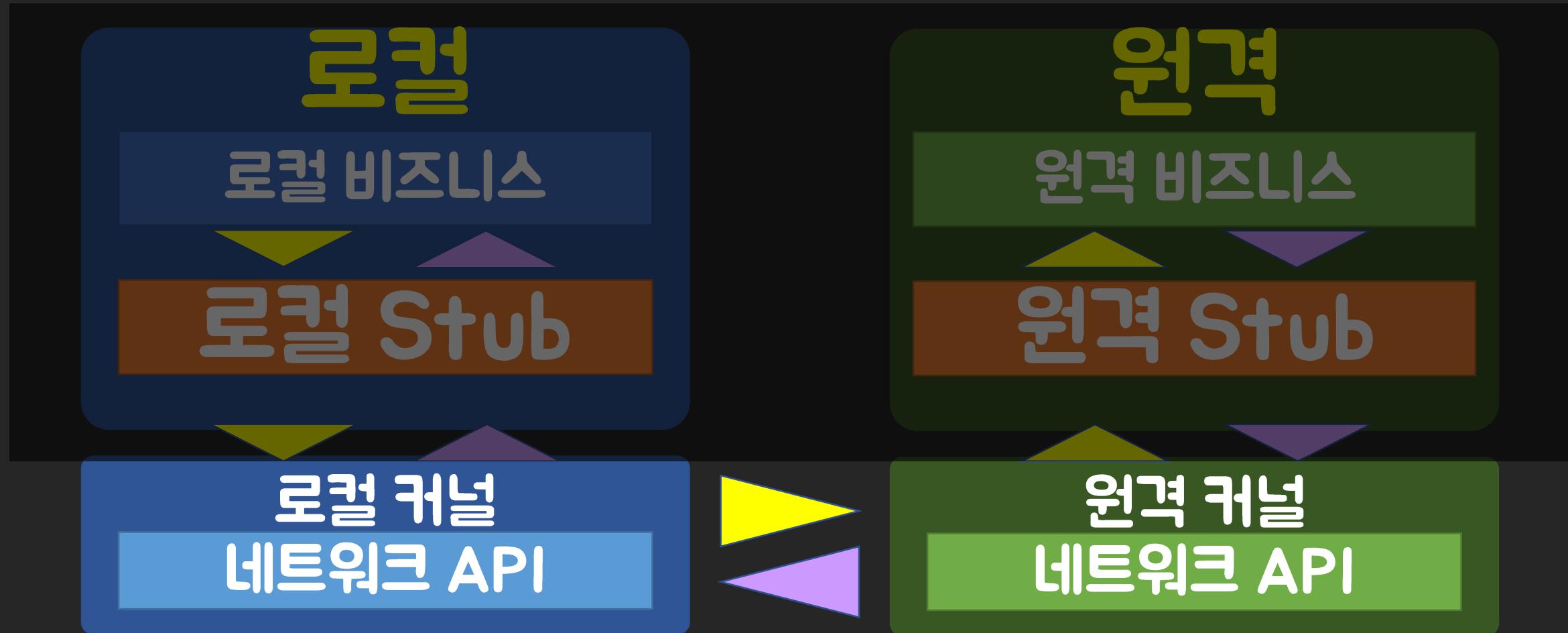
# Remote Procedure Call



# Remote Procedure Call



# Remote Procedure Call



# Remote Procedure Call



# Remote Procedure Call



# Remote Procedure Call



# Remote Procedure Call

CORBA - IIOP

Java RMI, Java RMI-IIOP,

XML-RPC ...

# Remote Procedure Call

CORBA - IIOP

Java RMI, Java RMI-IIOP,

XML-RPC ...

요즘 어떻게 지내세요?

# Remote Procedure Call

높은 복잡도

가파른 학습 곡선

낮은 개발 생산성

# RESTful API Everywhere

낮은 복잡도

완만한 학습 곡선

높은 개발 생산성

편리한 JSON

# RPC는 망했나?



# RPC는 망했나?

아니다.

여러 이유로 구현체가 별로 였을 뿐..

게임, 실시간 금융 거래, 모니터링 등  
(어려움에도 불구하고) 여전히 사용

# RPC는 망했나?

제다가 이젠

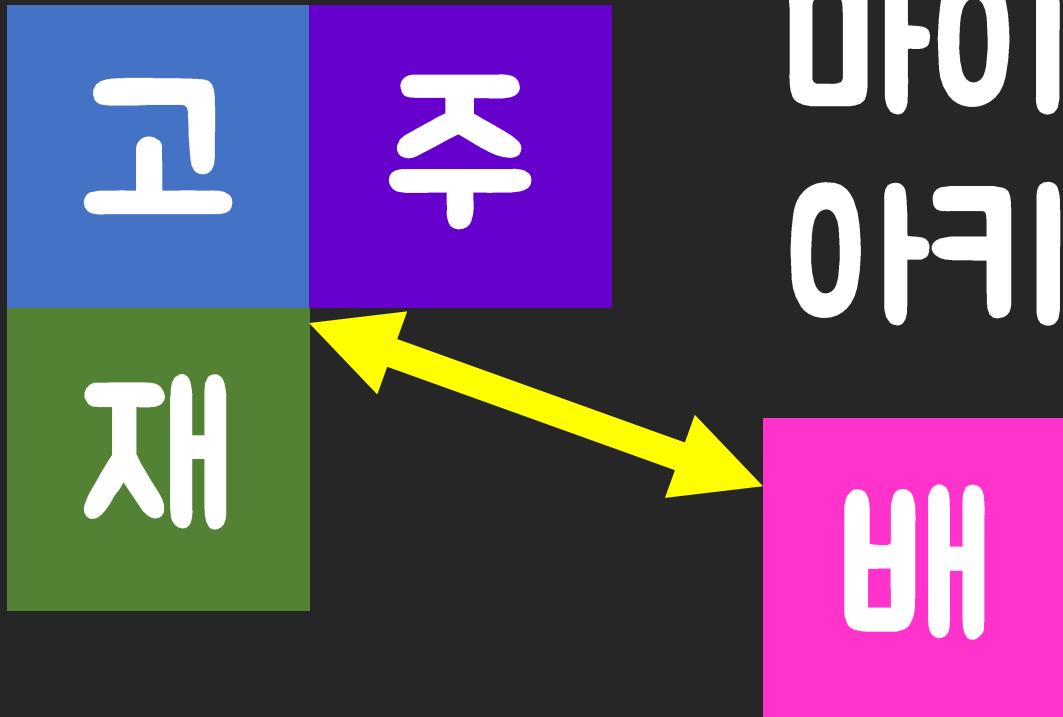
마이크로서비스의 시대 아닌가?

# RPC와 마이크로서비스



Monolithic(일체형)  
아키텍처

# RPC와 마이크로서비스



마이크로서비스  
아키텍처

# RPC와 마이크로서비스



마이크로서비스  
아키텍처

# RPC와 마이크로서비스



# Stubby

우리가 새로 만들었쪄..

RPC 그림에서

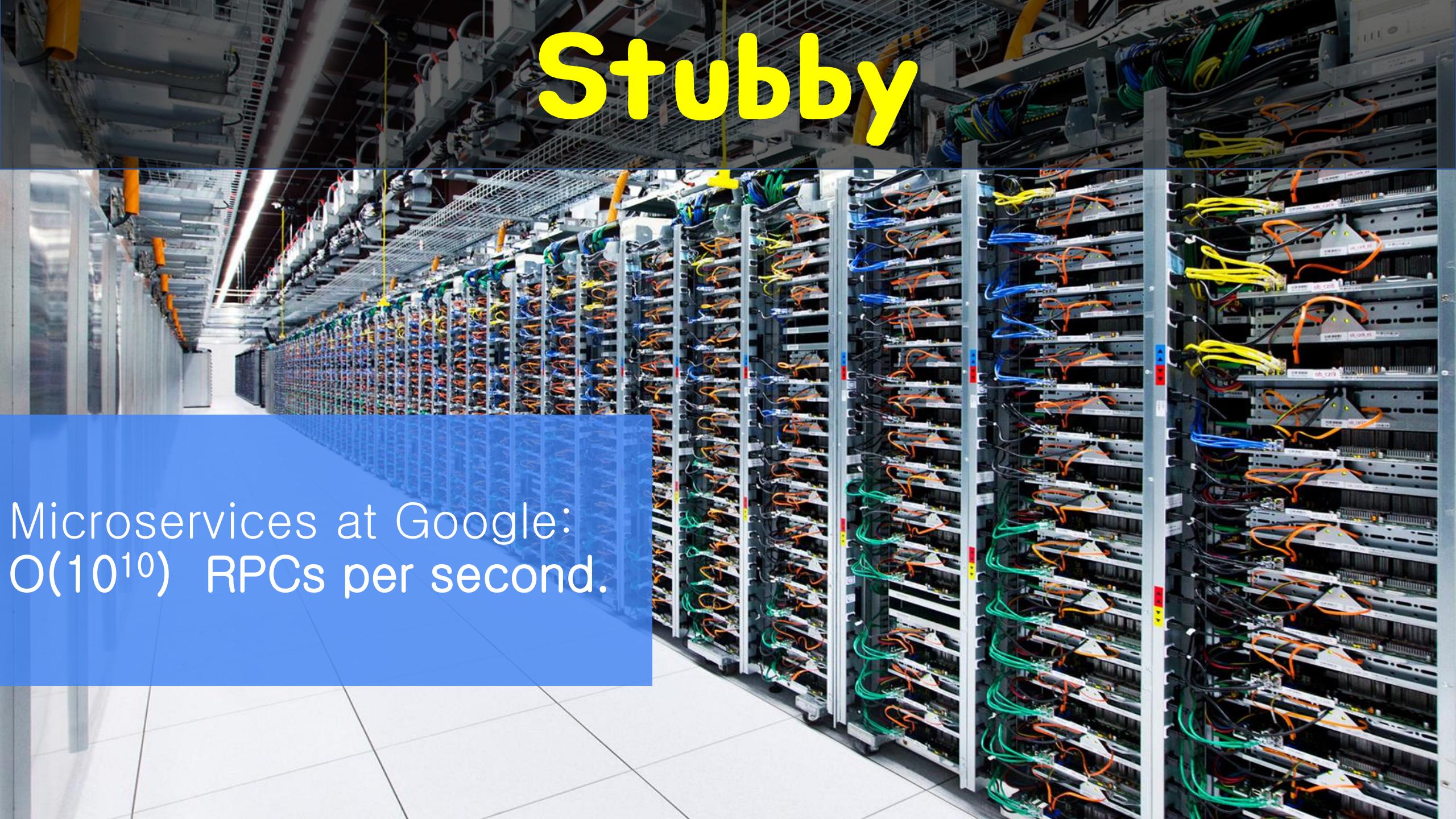
Stub 생각나지?

이름은 Stubby!



# Stubby

Microservices at Google:  
 $O(10^{10})$  RPCs per second.



# Stubby의 오픈 소스 후계자

↑GRPC↓

# gRPC

- ~~자료~~알피씨
- 2016년 8월에 1.0.0 출시(현재 1.2.0)
- BSD(3-clauses) License
- 10개 언어 지원
  - C++, Java, Python, Go, Ruby, PHP
  - C#, Node.js, Android Java, Objective-C

# gRPC가 REST보다 나은 점

gRPC가 REST보다 나은 점

g 툭 뺏다

# gRPC가 REST보다 나은 점

- **Binary Protocol**(Protocol Buffers 3)
  - Text보다 더 적은 데이터 공간으로 처리 가능
    - 네트워크, 메모리 효율성 좋음
  - Text보다 더 적은 Serialize/Deserialize 부하
    - CPU 효율성 좋음

# gRPC가 REST보다 나은 점

- HTTP/2
  - Connection Multiplexing
  - Header Compression 가능
    - 네트워크 효율성 좋음
  - Client/Server 양방향 Streaming 가능
    - 적용 분야 다양성 좋음

# gRPC가 REST보다 나은 점

- HTTP 1.1 vs HTTP/2

## HTTP 1.1 vs HTTP/2

링크: <http://http2.golang.org/gophertiles>

# gRPC가 REST보다 나쁜 점

- Browser에서는 아직은 사용 불가

# gRPC가 REST보다 나쁜 점

- Browser에서는 아직은 사용 불가
- Client 쪽 업데이트

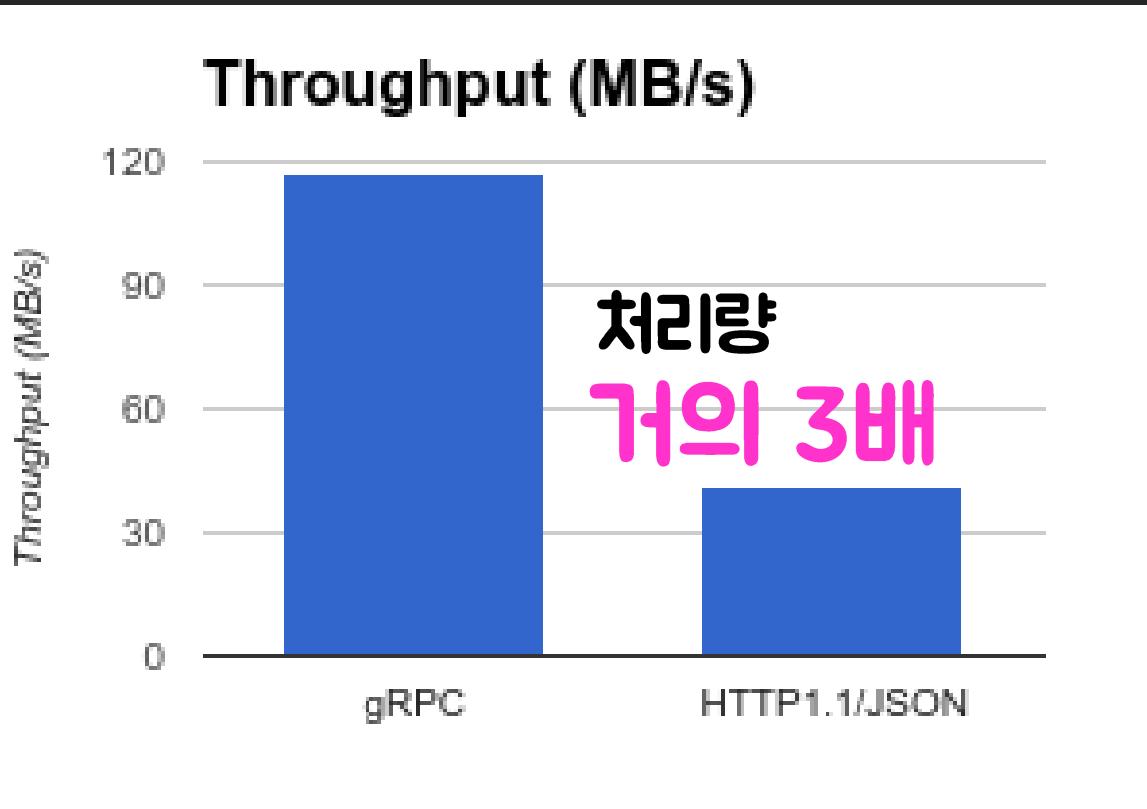
# gRPC가 REST보다 나쁜 점

- Browser에서는 아직은 사용 불가
- Client 쪽 업데이트
- 데이터가 NOT Human Readable

# gRPC가 REST보다 나쁜 점

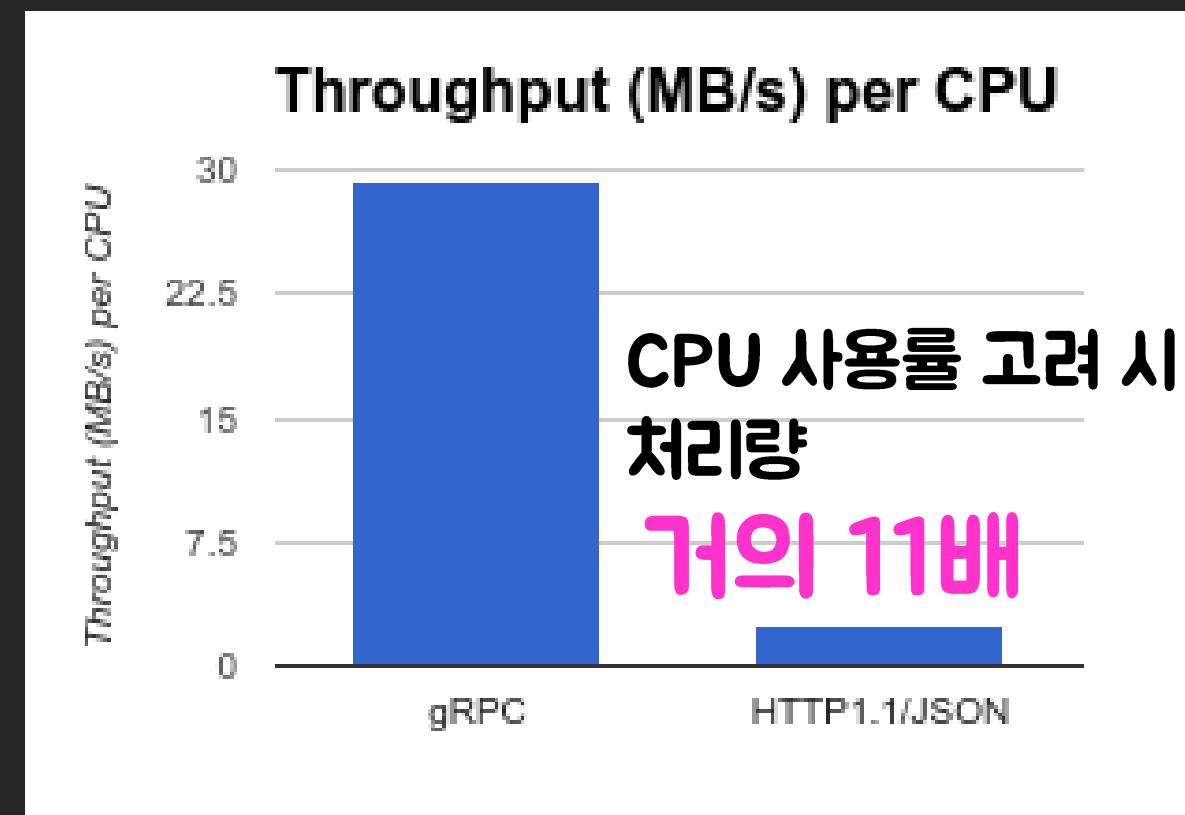
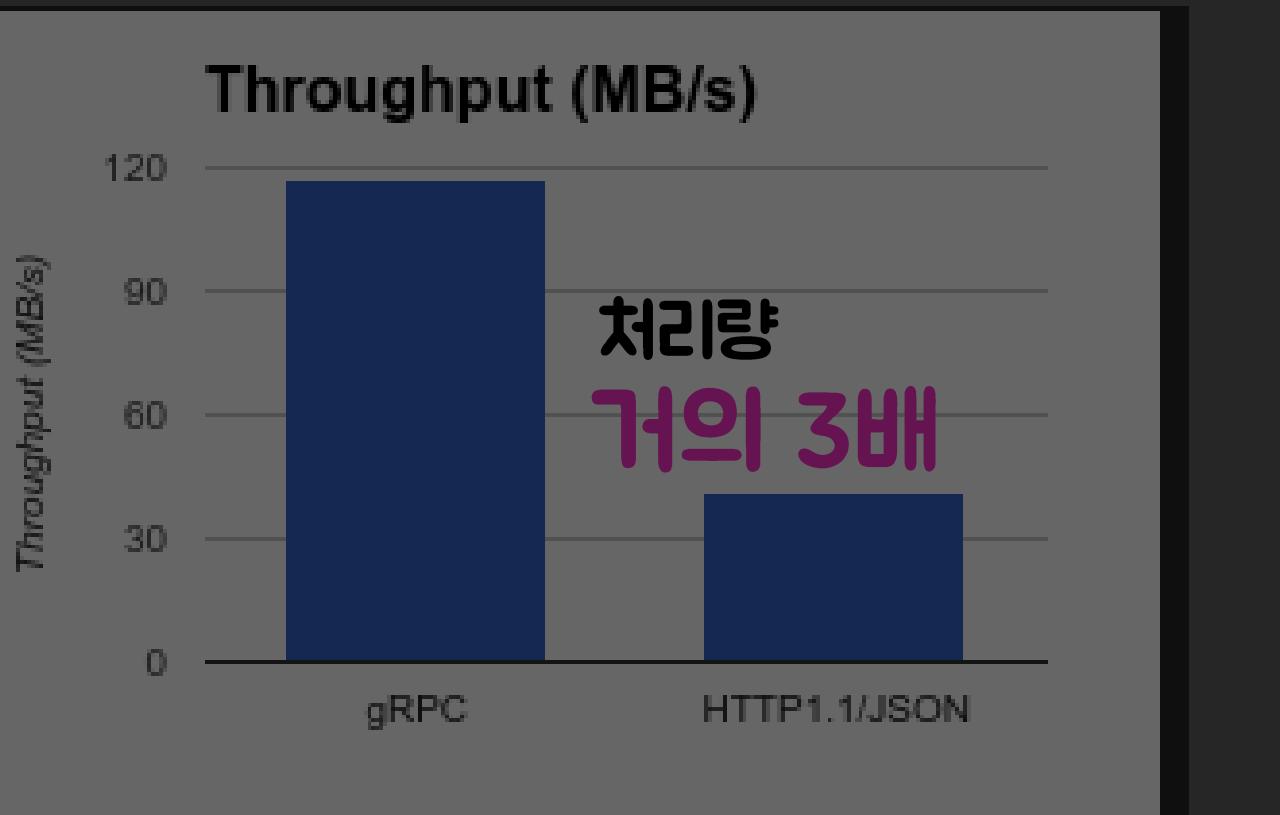
- Browser에서는 아직은 사용 불가
- Client 쪽 업데이트
- 데이터가 NOT Human Readable
- REST보다는 조금 복잡

# gRPC가 REST보다 나은 점



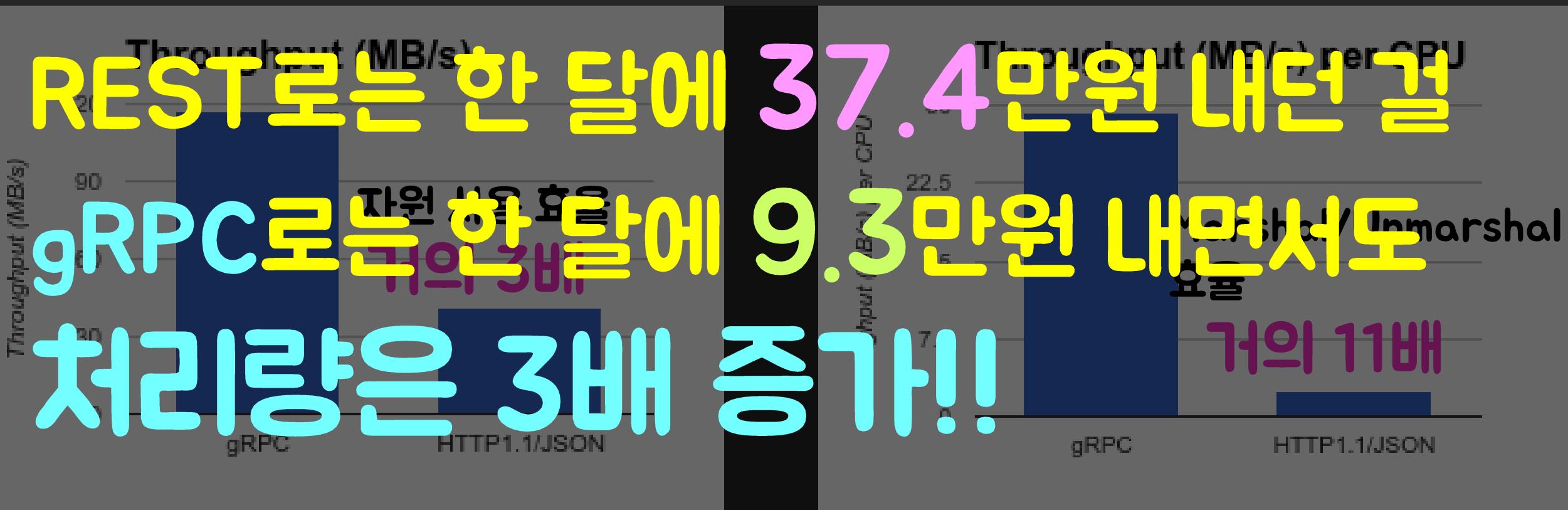
출처: <https://cloud.google.com/blog/big-data/2016/03/announcing-grpc-alpha-for-google-cloud-pubsub>

# gRPC가 REST보다 나은 점



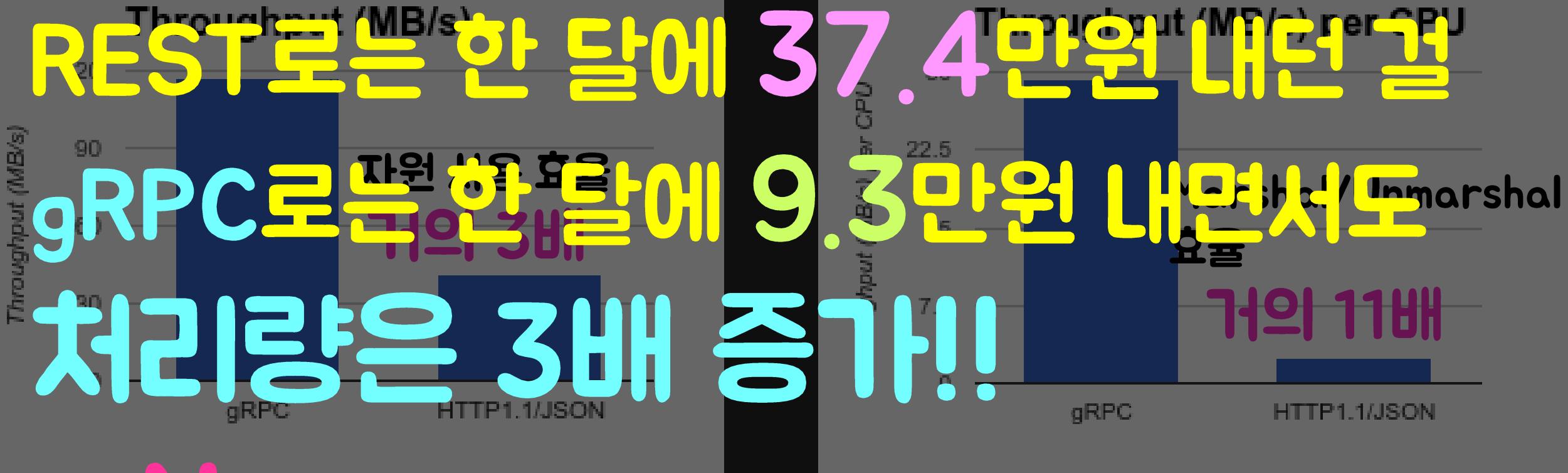
출처: <https://cloud.google.com/blog/big-data/2016/03/announcing-grpc-alpha-for-google-cloud-pubsub>

# gRPC가 REST보다 나은 점



출처: <https://cloud.google.com/blog/big-data/2016/03/announcing-grpc-alpha-for-google-cloud-pubsub>

# gRPC가 REST보다 나은 점



\* N

출처: <https://cloud.google.com/blog/big-data/2016/03/announcing-grpc-alpha-for-google-cloud-pubsub>

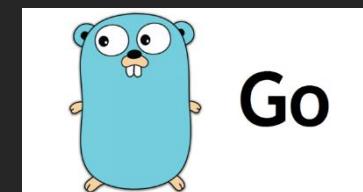
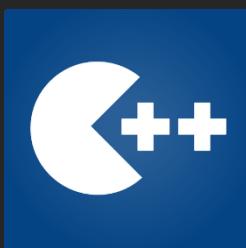
# gRPC가 좋은 점

## proto 파일 하나면



# gRPC가 좋은 점

proto 파일 하나면 10가지 언어로 작성 가능

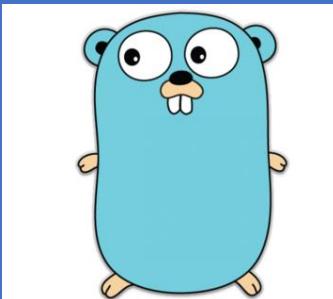


.proto



# gRPC가 좋은 점

Polyglot



Go

gRPC 서버



gRPC 클라이언트



gRPC 클라이언트



python™

gRPC 서버

# gRPC가 좋은 점

JSON

XML

Thrift

# gRPC가 좋은 점

복잡해보이지만  
반은 자동으로 먹고 들어간다.

gRPC는 이런 곳에 쓰면 좋다

# gRPC는 이런 곳에 쓰면 좋다

- 브라우저 필요 없는 백엔드 서버 간 통신

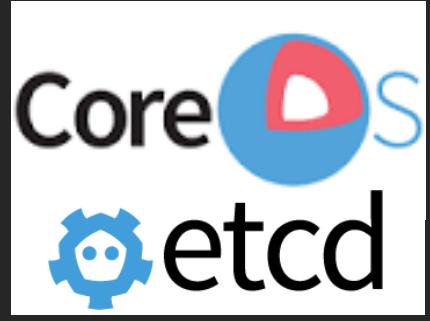
# gRPC는 이런 곳에 쓰면 좋다

- 브라우저 필요 없는 백엔드 서버 간 통신
- 자원이 빈약한 디바이스와 서버 간 통신

# gRPC는 이런 곳에 쓰면 좋다

- 브라우저 필요 없는 백엔드 서버 간 통신
- 자원이 빈약한 디바이스와 서버 간 통신
- 바이트/호출/CPU 수 등으로 과금 되는  
클라우드 환경에서의 비용 절감

# gRPC 사용 사례



클라이언트-서버 통신



클러스터링



Square



マイ크로서비스



네트워크 장비 간 스트리밍

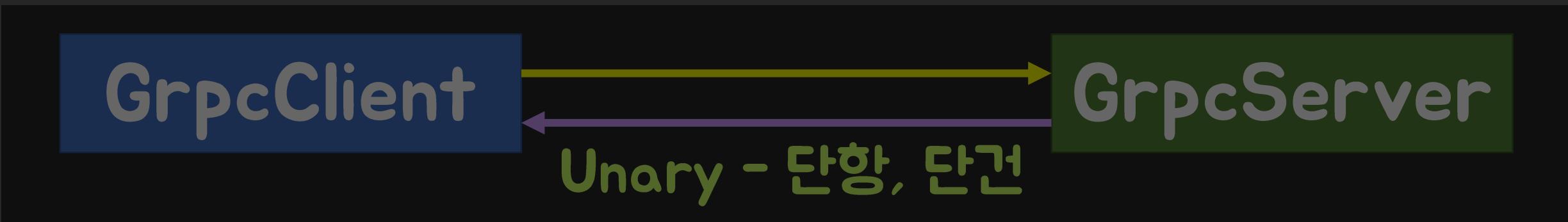
A photograph of a shirtless man with brown hair, wearing red and blue swim trunks, captured mid-air as he jumps off a dark, rocky cliff into a vast, turquoise-blue ocean. His arms are outstretched to his sides, and his legs are bent at the knees. The ocean extends to a distant horizon under a bright blue sky filled with scattered white and grey clouds. In the far distance, a small, dark mountainous island is visible.

*gRPC*

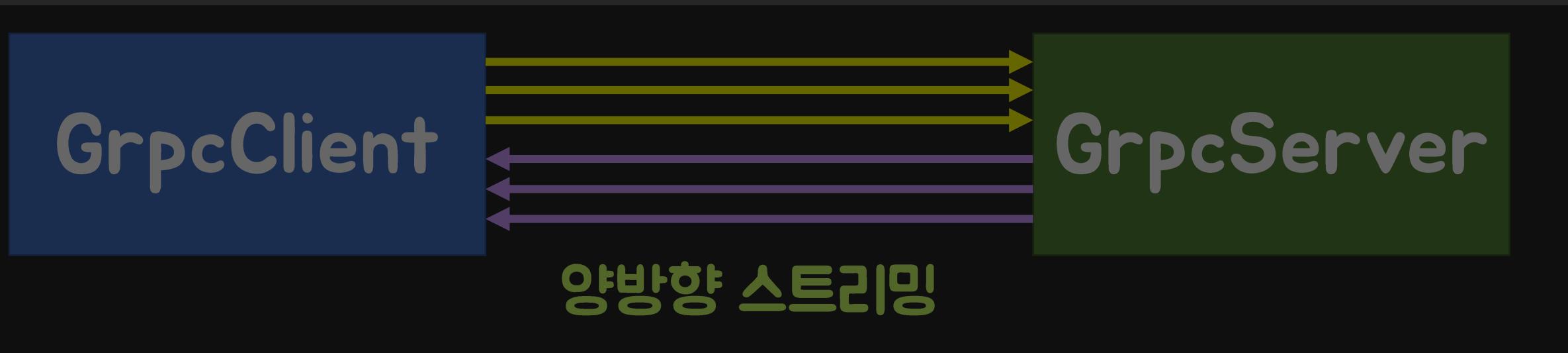
# gRPC 4 가지 스트리밍 방식



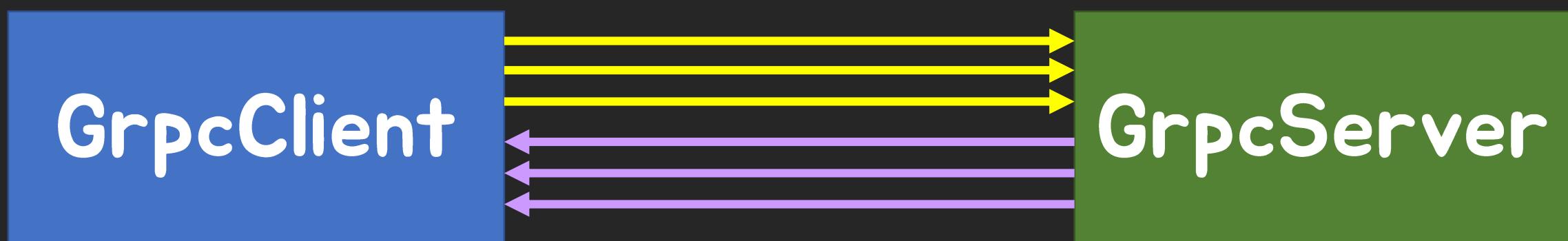
# gRPC 4가지 스트리밍 방식



# gRPC 4 가지 스트리밍 방식



# gRPC 4 가지 스트리밍 방식



양방향 스트리밍

# gRPC 3가지 Stub

BlockingStub

# gRPC 3가지 Stub

BlockingStub

(Async)Stub

# gRPC 3가지 Stub

BlockingStub

(Async)Stub

FutureStub

# gRPC 뭘 해볼까?



# gRPC Hello World



헬로 월드~

# gRPC Hello World



# gRPC Hello World



Async  
헬로 월드~



\$%&@#?

# gRPC Hello World

← 'Hello SpringCamp2017'

GrpcServer

\$%&@#?



# 서버 개발 순서

- 서버 프로젝트 작성
  - build.gradle
  - proto 파일 작성
  - gRPC 기반코드 생성 from proto 파일
  - gRPC Dependency 문제 해결
  - gRPC 서버 작성

# build.gradle

proto 작성

gRPC 기반코드 생성

gRPC Dependency 해결

gRPC 서버 작성

```
1 group 'homo.efficio'
2 version '1.0-SNAPSHOT'
3
4 apply plugin: 'java'
5 apply plugin: 'com.google.protobuf'
6
7 sourceCompatibility = 1.8
8
9 repositories {
10     mavenCentral()
11 }
12
13 buildscript {
14
15     repositories {
16         mavenCentral()
17     }
18
19     dependencies {
20         classpath 'com.google.protobuf:protobuf-gradle-plugin:0.8.1'
21     }
22 }
```

```
24  protobuf {  
25      protoc {  
26          artifact = "com.google.protobuf:protoc:3.2.0"  
27      }  
28      plugins {  
29          grpc {  
30              artifact = 'io.grpc:protoc-gen-grpc-java:1.2.0'  
31          }  
32      }  
33      generateProtoTasks {  
34          all()*.plugins {  
35              grpc {}  
36          }  
37      }  
38  }  
39  
40  dependencies {  
41      compile 'io.grpc:grpc-netty:1.2.0'  
42      compile 'io.grpc:grpc-protobuf:1.2.0'  
43      compile 'io.grpc:grpc-stub:1.2.0'  
44  
45      testCompile group: 'junit', name: 'junit', version: '4.12'  
46  }
```

build.gradle

proto 작성

gRPC 기반코드 생성

gRPC Dependency 해결

gRPC 서버 작성

# Protobuf Support

Sort by: name ▾

Category: All

FORMATTING

Proto Support 67,664 9 months ago

LANGUAGES

ReProtoc 43 4 days ago

TOOLS INTEGRATION

one year ago

LANGUAGES

Protobuf Support

 Install

★★★★★ 67664 downloads

Updated 2017-04-18 v0.8.0

[Google Protobuf](#) support for JetBrains products.

Features:

- Full Proto3 support.
- Custom include path for `proto` files.
- Reference support.
- Usage search for messages/enums.
- Syntax highlighting.
- Syntax validation for `proto2/proto3`.
- Error checks for reserved/duplicated field tags and names.
- Fonts & Colors configuration.
- Structure View.
- Brace matching.
- Line and block commenting.
- Code formatting.
- Navigation to message, enum or service by name (Ctrl+N)
- `Proto` file rename refactoring (imports are updated automatically).

[Roadmap](#) | [Issue tracker](#) | [Donate \(PayPal, BitCoin\)](#)

Change Notes

v0.8.0 (2017-04-14)

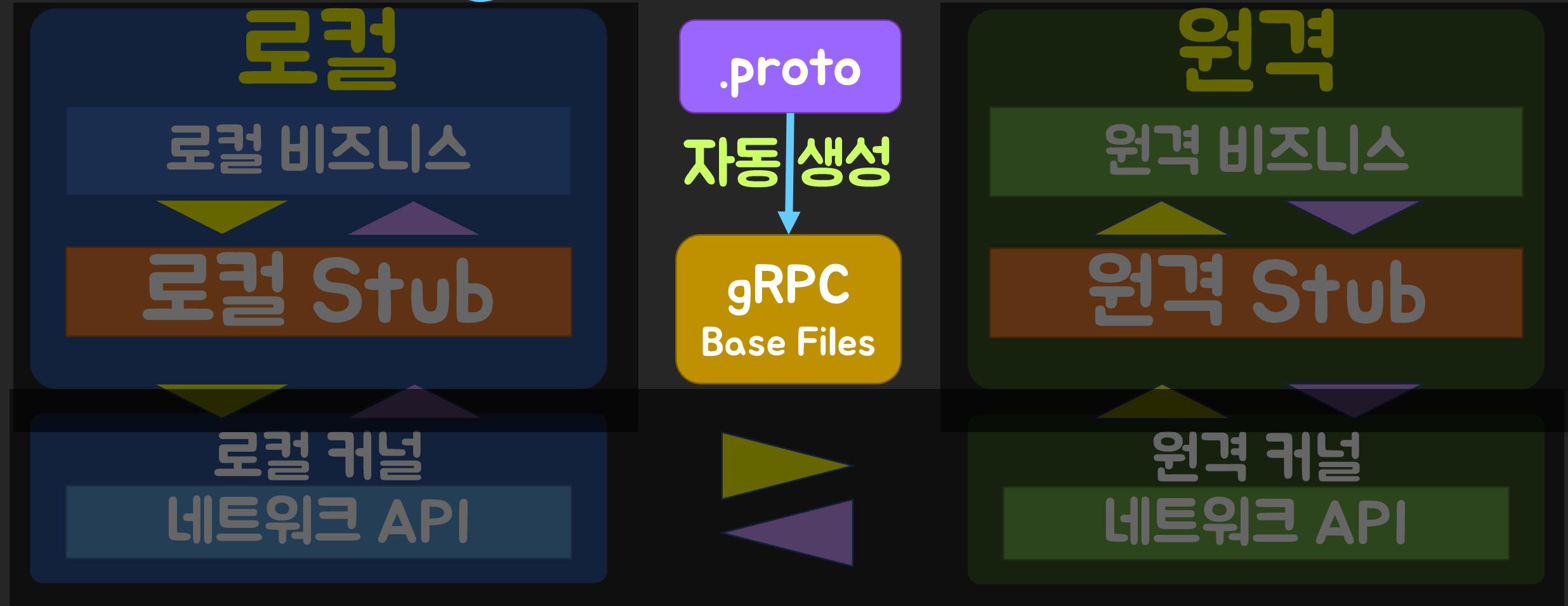
- Support for `proto` file renames - import statements are automatically updated when imported file is renamed.

HTTP Proxy Settings... Manage repositories... Close

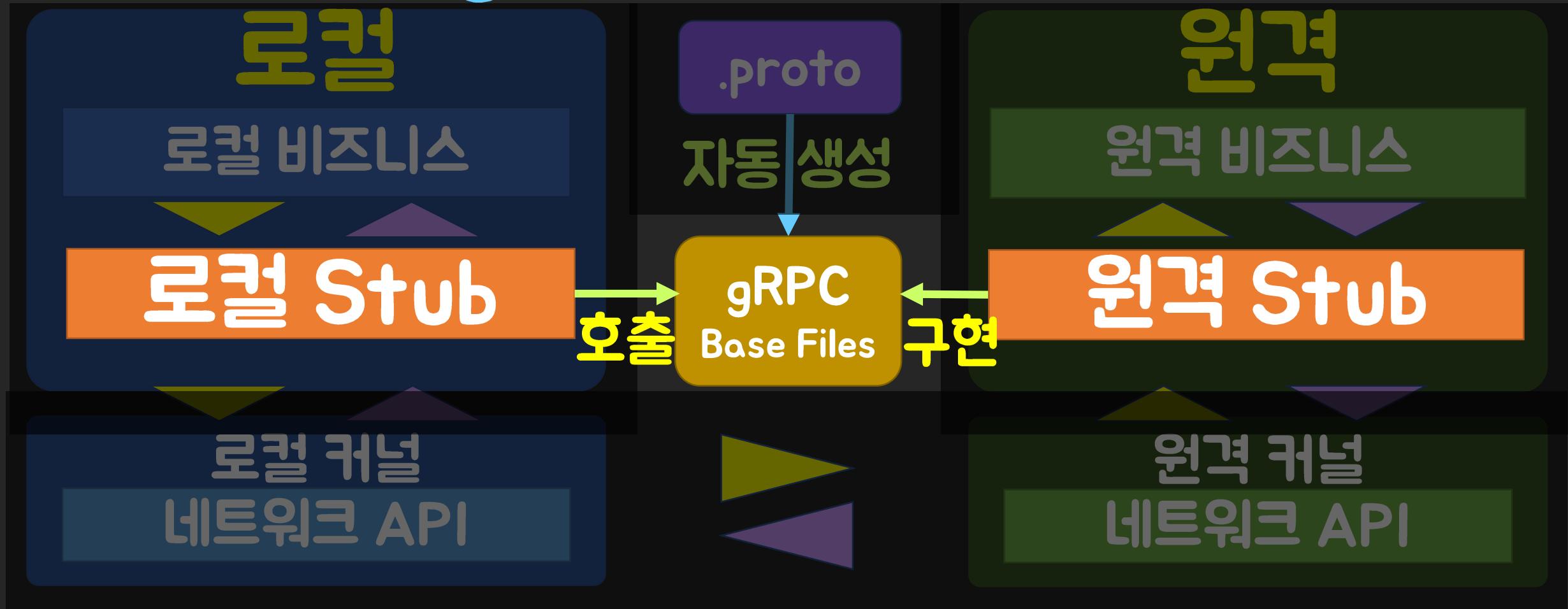
# RPC Model



# gRPC Model



# gRPC Model



# .proto 파일

메타 정보

자료 구조

에서드 선언

## 메타 정보

```
syntax = "proto3";  
  
option java_multiple_files = true;  
option java_outer_classname = "HelloGrpcProto";  
  
package homo.efficio.springcamp2017.grpc.hello;
```

```
message HelloRequest {  
    string request = 1;  
}
```

```
message HelloResponse {  
    string response = 1;  
}
```

```
service HelloSpringCamp {  
  
    rpc unaryHello(HelloRequest) returns (HelloResponse) {}  
}
```

## 메타 정보

```
syntax = "proto3";  
  
option java_multiple_files = true;  
option java_outer_classname = "HelloGrpcProto";  
  
package homo.efficio.springcamp2017.grpc.hello;
```

## 자료 구조

```
message HelloRequest {  
    string request = 1;  
}  
  
message HelloResponse {  
    string response = 1;  
}
```

```
service HelloSpringCamp {  
  
    rpc unaryHello(HelloRequest) returns (HelloResponse) {}  
}
```

## 메타 정보

```
syntax = "proto3";  
  
option java_multiple_files = true;  
option java_outer_classname = "HelloGrpcProto";  
  
package homo.efficio.springcamp2017.grpc.hello;
```

## 자료 구조

```
message HelloRequest {  
    string request = 1;  
}  
string nickname = 2;  
  
message HelloResponse {  
    string response = 1;  
}
```

```
service HelloSpringCamp {  
  
    rpc unaryHello(HelloRequest) returns (HelloResponse) {}  
}
```

## 메타 정보

```
syntax = "proto3";  
  
option java_multiple_files = true;  
option java_outer_classname = "HelloGrpcProto";  
  
package homo.efficio.springcamp2017.grpc.hello;
```

## 자료 구조

```
message HelloRequest {  
    string request = 1;  
}  
string nickname = 2;  
  
message HelloResponse {  
    string response = 1;  
}  
HelloRequest request = 2;
```

```
service HelloSpringCamp {  
  
    rpc unaryHello(HelloRequest) returns (HelloResponse) {}  
}
```

## 메타 정보

```
syntax = "proto3";  
  
option java_multiple_files = true;  
option java_outer_classname = "HelloGrpcProto";  
  
package homo.efficio.springcamp2017.grpc.hello;
```

## 자료 구조

```
message HelloRequest {  
    string request = 1;  
}  
  
message HelloResponse {  
    string response = 1;  
}
```

```
service HelloSpringCamp {  
    rpc unaryHello(HelloRequest) returns (HelloResponse) {}  
}
```

build.gradle

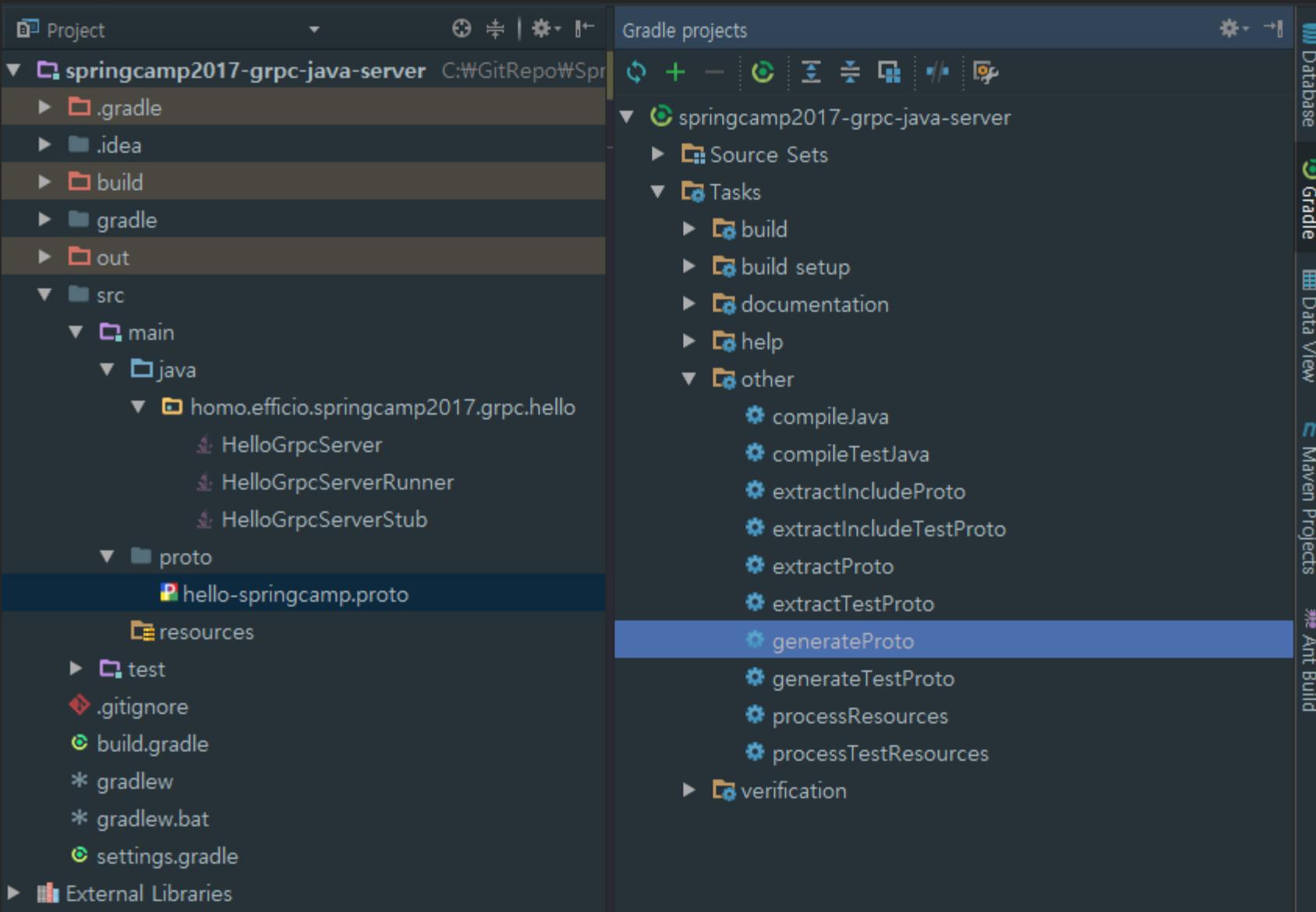
proto 작성

gRPC 기반코드 생성

gRPC Dependency 해결

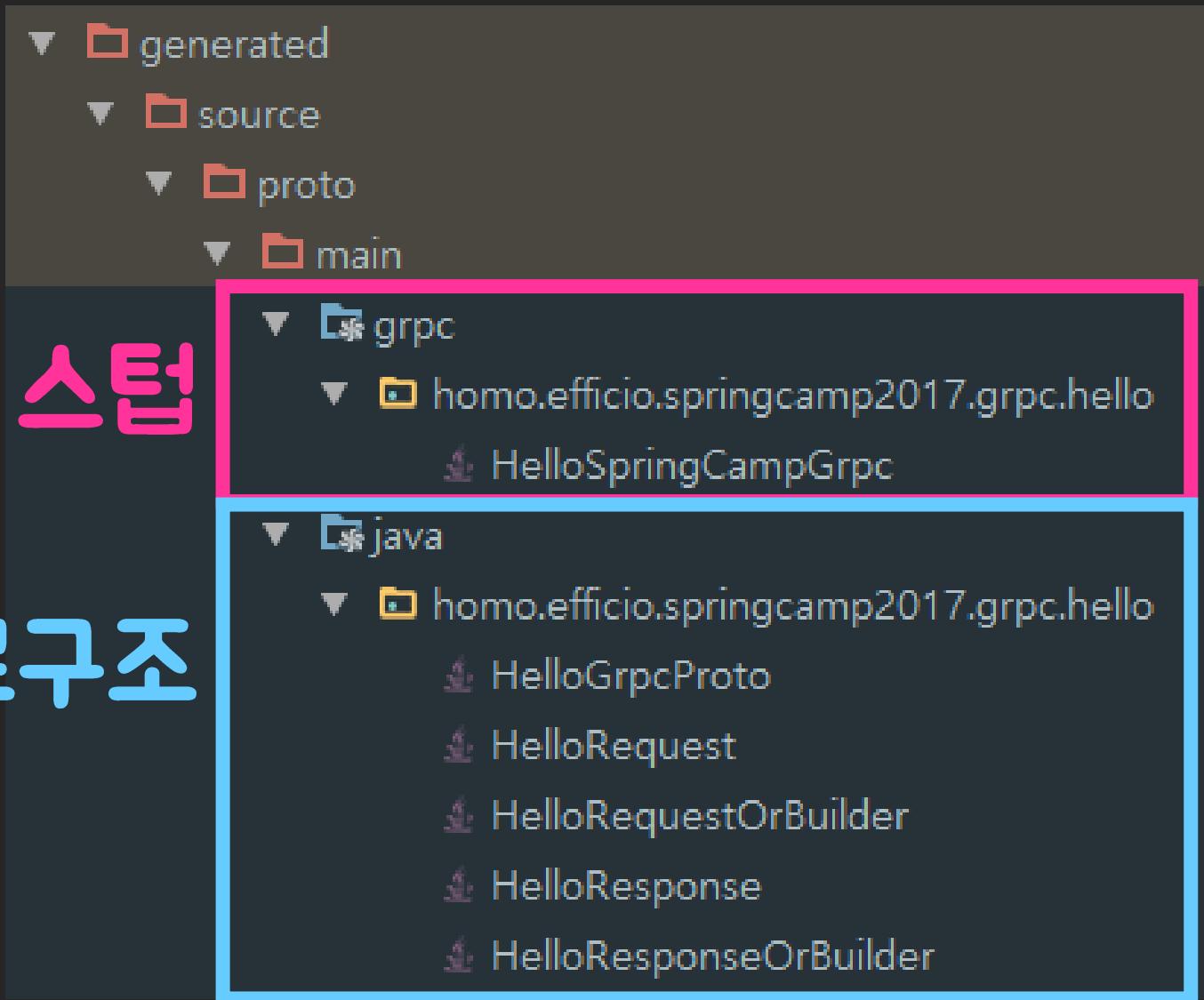
gRPC 서버 작성

# generateProto



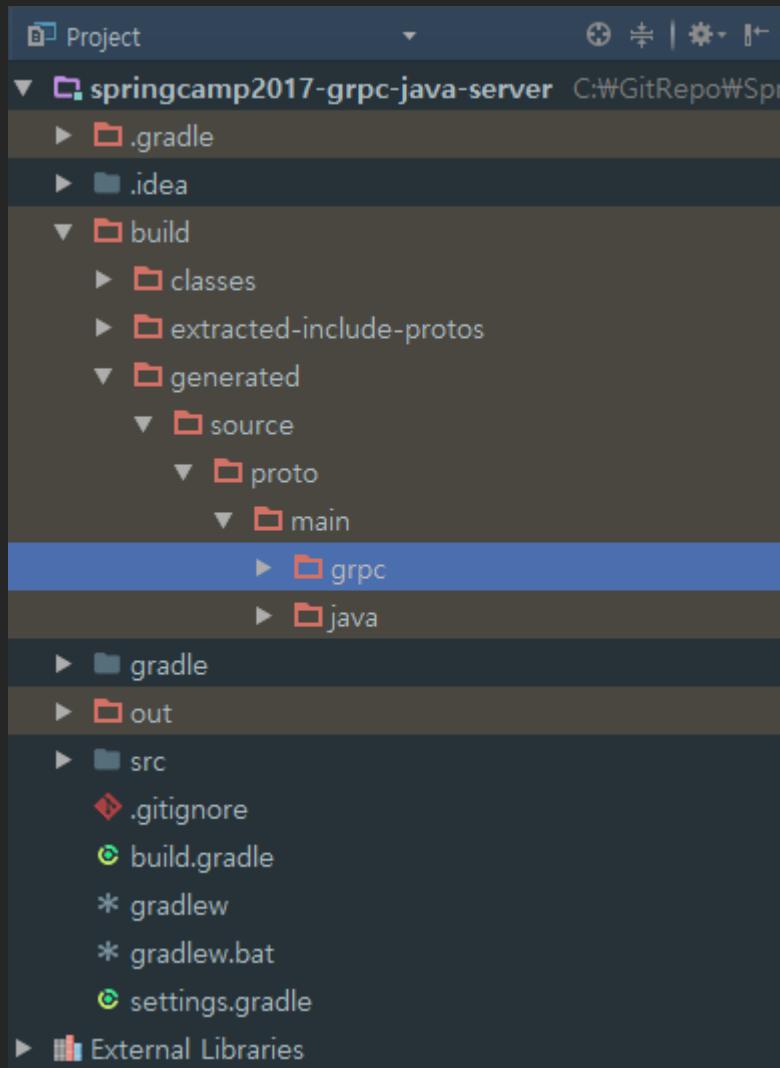
# generateProto

스텝  
자료구조

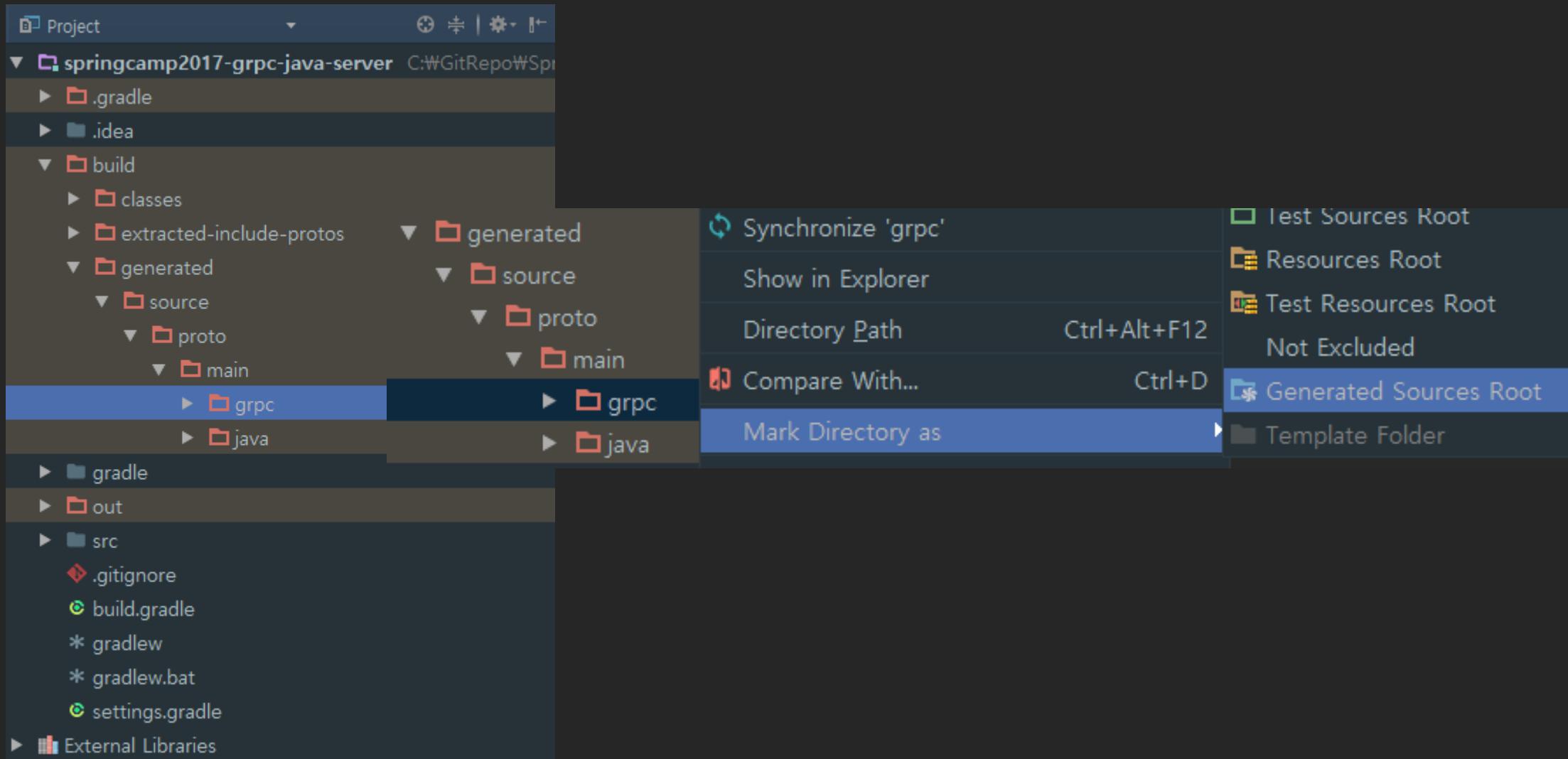


build.gradle  
proto 작성  
gRPC 기반코드 생성  
**gRPC Dependency 해결**  
gRPC 서버 작성

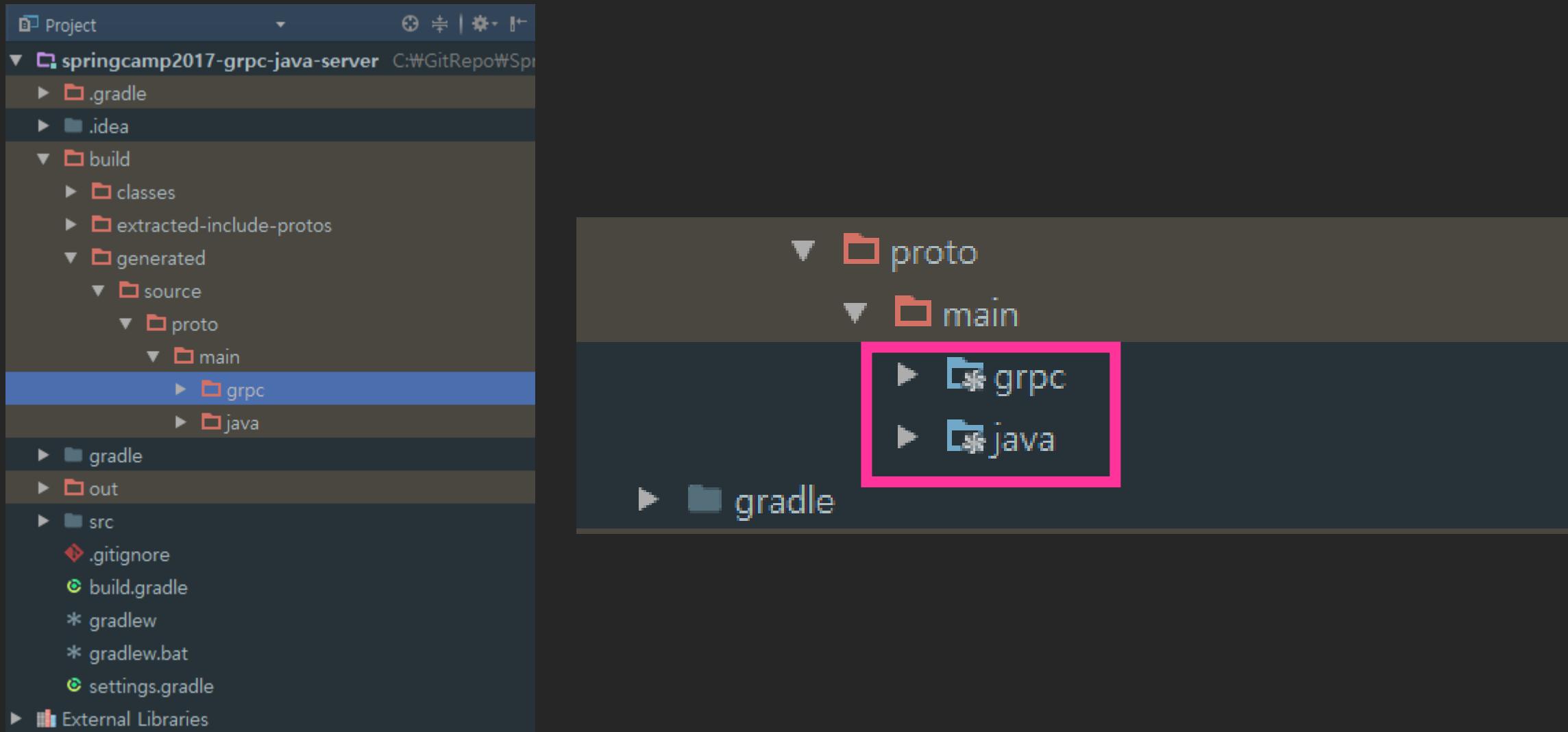
# Generated Sources Root 지정



# Generated Sources Root 지정



# Generated Sources Root 지정



# 컴파일 오류(이름 해석 실패) - 자동 생성된 코드

The screenshot shows an IDE interface with a project structure on the left and a code editor on the right. The project is named 'springcamp2017-grpc-java-server' located at 'C:\GitRepo\SpringCamp2017'. The code editor displays a file named 'HelloSpringCampGrpc.java'.

**Project Tree:**

- Project: springcamp2017-grpc-java-server C:\GitRepo\SpringCamp2017
- .gradle
- .idea
- build
  - classes
  - extracted-include-protos
  - generated
    - source
      - proto
        - main
        - grpc
          - homo.efficio.springcamp2017.grpc.hello
  - java- gradle
- out
- src
  - .gitignore
  - build.gradle
  - \* gradlew
  - \* gradlew.bat
  - settings.gradle
- External Libraries

**Code Editor:**

```
public final class HelloSpringCampGrpc {
    private HelloSpringCampGrpc() {}

    public static final String SERVICE_NAME = "homo.efficio.springcamp2017.g";

    // Static method descriptors that strictly reflect the proto.
    @io.grpc.ExperimentalApi("https://github.com/grpc/grpc-java/issues/1901")
    public static final io.grpc.MethodDescriptor<homo.efficio.springcamp2017.h
        homo.efficio.springcamp2017.grpc.hello.HelloResponse> METHOD_UNARY_H
        io.grpc.MethodDescriptor.create(
            io.grpc.MethodDescriptor.MethodType.UNARY,
            generateFullName(
                "homo.efficio.springcamp2017.grpc.hello.HelloSpringCamp", "un
            io.grpc.protobuf.ProtoUtils.marshaller(homo.efficio.springcamp2017.g
            io.grpc.protobuf.ProtoUtils.marshaller(homo.efficio.springcamp2017.g

    /**
     * Creates a new async stub that supports all call types for the service
     */
}
```

The code editor highlights several lines with red squiggly underlines, indicating syntax errors. The first error is at line 27: `@io.grpc.ExperimentalApi("https://github.com/grpc/grpc-java/issues/1901")`. The second error is at line 31: `public static final io.grpc.MethodDescriptor<homo.efficio.springcamp2017.h`.

# 컴파일 오류(이름 해석 실패) - 자동 생성된 코드

The screenshot shows an IDE interface with a project structure on the left and a code editor on the right. The project structure includes a 'springcamp2017-grpc-java-server' folder containing '.gradle', '.idea', 'build', 'generated', 'src', 'gradlew', 'gradlew.bat', and 'settings.gradle'. The 'generated' folder contains 'source' and 'proto' subfolders, with 'proto' further divided into 'main' and 'grpc' subfolders, which in turn contain 'homo.efficio.springcamp2017.grpc.hello' and 'HelloSpringCampGrpc.java'. The code editor displays the 'HelloSpringCampGrpc.java' file, specifically the 'HelloSpringCampGrpc' class definition. A tooltip box is overlaid on the screen, centered over the line of code that starts with '@io.grpc.ExperimentalApi("https://github.com/grpc/grpc-java/issues/1901")'. The tooltip text reads 'Add library ... to classpath' and lists three options: 'Make 'private'', 'Make 'protected'', and 'Make package-private'. The background code in the editor shows static method descriptors and marshaller implementations.

```
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
```

23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41

```
HelloSpringCampGrpc METHOD_UNARY_HELLO
public final class HelloSpringCampGrpc {
    private HelloSpringCampGrpc() {}

    public static final String SERVICE_NAME = "homo.efficio.springcamp2017.g

    // Static method descriptors that strictly reflect the proto.
    @io.grpc.ExperimentalApi("https://github.com/grpc/grpc-java/issues/1901")
    public static final io.grpc.MethodDescriptor<homo.efficio.springcamp2017.grpc.hello.HelloSpringCampGrpc, io.grpc.protobuf.ProtoUtils.marshaller> METHOD_UNARY_H

    public static final io.grpc.MethodDescriptor<homo.efficio.springcamp2017.grpc.hello.HelloSpringCampGrpc, io.grpc.protobuf.ProtoUtils.marshaller> METHOD_UNARY_H

    /**
     * Creates a new async stub that supports all call types for the service
     */
}
```

Add library ... to classpath

- Make 'private'
- Make 'protected'
- Make package-private

# 컴파일 오류(이름 해석 실패) - 작성 코드

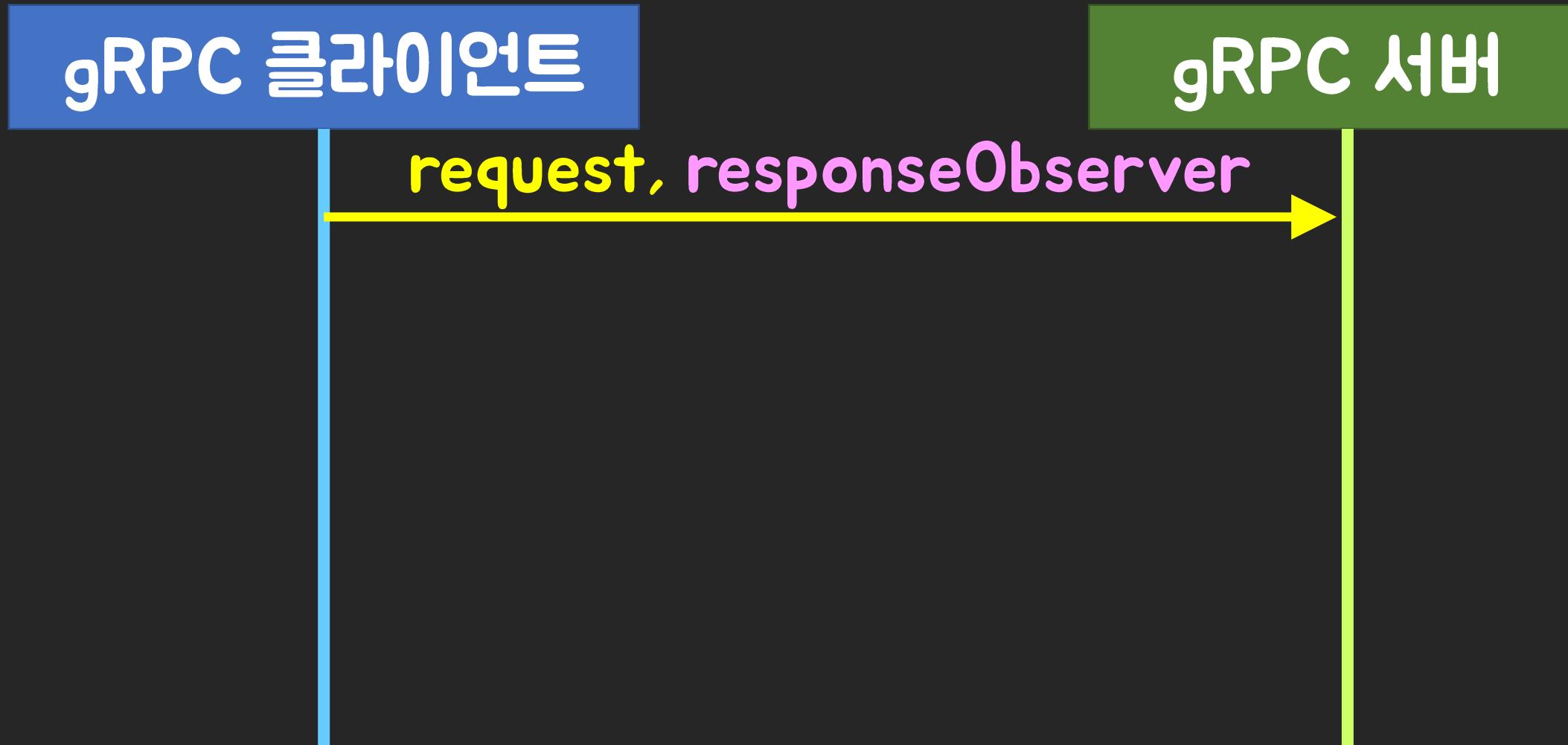
The screenshot shows an IDE interface with a project structure on the left and a code editor on the right. The code editor displays a Java file named `HelloGrpcServerStub.java`. The cursor is positioned at line 16, where the method `unaryHello` is being defined. A tooltip box is open over the word `Unary`, which is underlined with a red squiggle. The tooltip contains the message "Add dependency on module 'springcamp2017-grpc-java-server'" and a title "Add dependency on module ...". Below the title, there are several suggestions:

- Create inner class 'HelloRequest'
- Create interface 'HelloRequest'
- Move 'homo.efficio.springcamp2017.grpc.hello.HelloRequest' from ...
- Bind method parameters to fields
- Create field for parameter 'request'
- Generate overloaded method with default parameter values

```
public class HelloGrpcServerStub extends HelloSpringCampGrpc.HelloSpr
    private final Logger logger = Logger.getLogger(HelloGrpcServerStub.c
    @Override
    public void unaryHello(HelloRequest request, StreamObserver<HelloR
        logger.info("Unary 메시지 수신됨");
        HelloResponse helloResponse = HelloRe
    responseObserver.onNext(helloResponse);
    responseObserver.onC
}
```

build.gradle  
proto 작성  
gRPC 기반코드 생성  
gRPC Dependency 해결  
**gRPC 서버 작성**

# Async Unary Preview



# Async Unary Preview



gRPC 서버

입금해 형ㅋ, 입금 완료되면 카톡 보내



# Async Unary Preview



입금해 형ㅋ, 입금 완료되면 카톡 보내



# Async Unary Preview



입금해 형ㅋ, 입금 완료되면 카톡 보내

아놔 미친..  
1초

# Async Unary Preview



입금해 형ㅋ, 입금 완료되면 카톡 보내

아놔 미친..

1초

입금  
1초

# Async Unary Preview



입금해 형ㅋ, 입금 완료되면 카톡 보내

블로킹이 아니니까  
그동안 핵이나 쏘자 ㅋㅋ



아놔 미친..

1초

입금  
1초

# Async Unary Preview



입금해 형ㅋ, 입금 완료되면 카톡 보내



블로킹이 아니니까  
그동안 핵이나 쏘자 ㅋㅋ

-----  
입금 했다 미친XX야  
이번이 마지막이야

아놔 미친..

1초

입금  
1초

# Async Unary Preview



입금해 형ㅋ, 입금 완료되면 카톡 보내



블로킹이 아닙니까  
그동안 핵이나 쏘자 ㅋㅋ

-----  
responseObserver.onNext()  
responseObserver.onCompleted()

아놔 미친..

1초

입금  
1초

# gRPC 서버 구성

ServerService

# gRPC 서버 구성

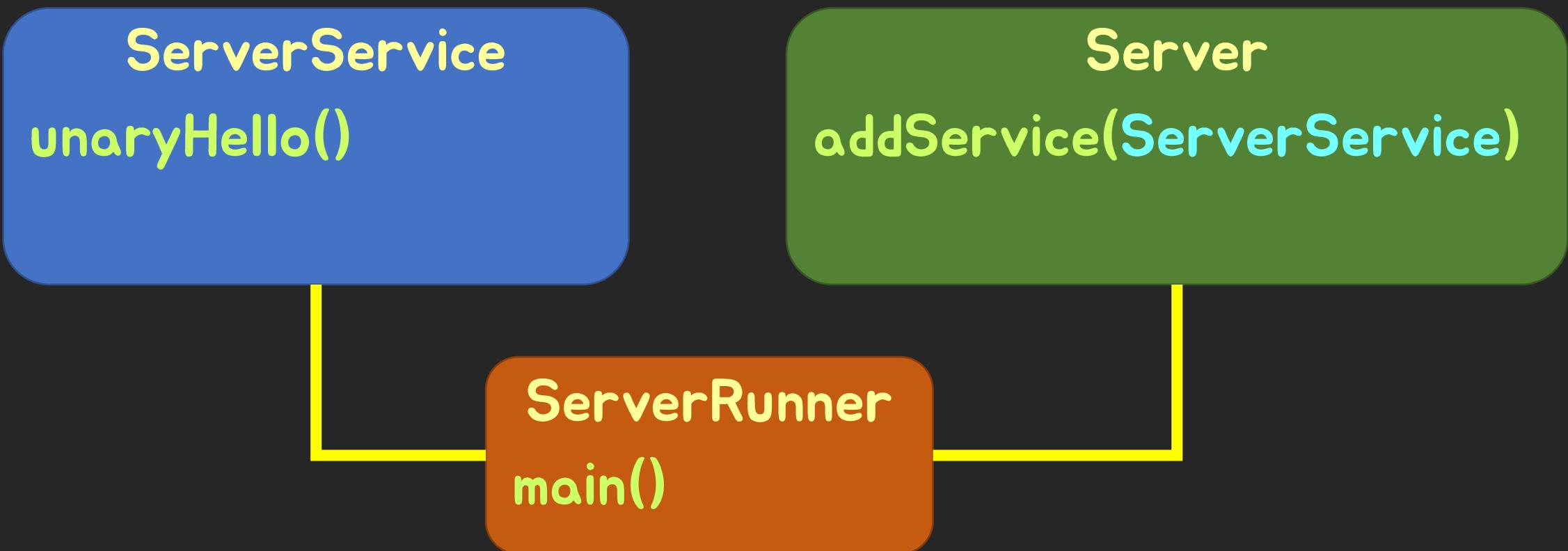
ServerService  
     unaryHello()

# gRPC 서버 구성

ServerService  
UnaryHello()

Server  
addService(ServerService)

# gRPC 서버 구성



Talk is cheap.  
Show me the code.

Talk is cheap.  
Show me the code.

- Linus **도발**s



# gRPC 서버 구성

ServerService  
unaryHello()

Server  
addService(ServerService)

ServerRunner  
main()

# HelloGrpcServerService.java

```
public class HelloGrpcServerService extends HelloSpringCampGrpc.HelloSpringCampImplBase {  
    private final Logger logger = Logger.getLogger(HelloGrpcServerService.class.getName());  
  
    @Override  
    public void unaryHello(HelloRequest request, StreamObserver<HelloResponse> responseObserver) {  
        logger.info(msg: "[트럼프] 요청 받음 : " + request.getRequest());  
  
        // 1초 동안 비즈니스 로직 처리 후에 응답한다고 가정  
        try { Thread.sleep(millis: 1000); } catch (InterruptedException e) { e.printStackTrace(); }  
  
        HelloResponse helloResponse =  
            HelloResponse.newBuilder()  
                .setResponse("입금했다 미친XX야!!")  
                .build();  
  
        // 응답 시작  
        responseObserver.onNext(helloResponse);  
        // 응답 시작 후 1초 후에 응답 완료된다고 가정  
        try { Thread.sleep(millis: 1000); } catch (InterruptedException e) { e.printStackTrace(); }  
        // 응답 완료  
        responseObserver.onCompleted();  
    }  
}
```

# HelloGrpcServerService.java

```
public class HelloGrpcServerService extends HelloSpringCampGrpc.HelloSpringCampImplBase {  
  
    private final Logger logger = Logger.getLogger(HelloGrpcServerService.class.getName());  
  
    @Override  
    public void unaryHello(HelloRequest request, StreamObserver<HelloResponse> responseObserver) {  
        logger.info(msg: "[트럼프] 요청 받음 : " + request.getRequest());  
  
        // 1초 동안 비즈니스 로직 처리 후에 응답한다고 가정  
        try { Thread.sleep(millis: 1000); } catch (InterruptedException e) { e.printStackTrace(); }  
  
        HelloResponse helloResponse =  
            HelloResponse.newBuilder()  
                .setResponse("입금했다 미친XX야!!")  
                .build();  
  
        // 응답 시작  
        responseObserver.onNext(helloResponse);  
        // 응답 시작 후 1초 후에 응답 완료된다고 가정  
        try { Thread.sleep(millis: 1000); } catch (InterruptedException e) { e.printStackTrace(); }  
        // 응답 완료  
        responseObserver.onCompleted();  
    }  
}
```

# HelloGrpcServerService.java

```
public class HelloGrpcServerService extends HelloSpringCampGrpc.HelloSpringCampImplBase {  
    private final Logger logger = Logger.getLogger(HelloGrpcServerService.class.getName());  
  
    @Override  
    public void unaryHello(HelloRequest request, StreamObserver<HelloResponse> responseObserver) {  
        logger.info(msg: "[트럼프] 요청 받음 : " + request.getRequest());  
  
        // 1초 동안 비즈니스 로직 처리 후에 응답한다고 가정  
        try { Thread.sleep( millis: 1000); } catch (InterruptedException e) { e.printStackTrace(); }  
  
        HelloResponse helloResponse =  
            HelloResponse.newBuilder()  
                .setResponse("입금했다 미친XX야!!")  
                .build();  
  
        // 응답 시작  
        responseObserver.onNext(helloResponse);  
        // 응답 시작 후 1초 후에 응답 완료된다고 가정  
        try { Thread.sleep( millis: 1000); } catch (InterruptedException e) { e.printStackTrace(); }  
        // 응답 완료  
        responseObserver.onCompleted();  
    }  
}
```

# HelloGrpcServerService.java

```
public class HelloGrpcServerService extends HelloSpringCampGrpc.HelloSpringCampImplBase {  
    private final Logger logger = Logger.getLogger(HelloGrpcServerService.class.getName());  
  
    @Override  
    public void unaryHello(HelloRequest request, StreamObserver<HelloResponse> responseObserver) {  
        logger.info(msg: "[트럼프] 요청 받음 : " + request.getRequest());  
  
        // 1초 동안 비즈니스 로직 처리 후에 응답한다고 가정  
        try { Thread.sleep( millis: 1000); } catch (InterruptedException e) { e.printStackTrace(); }  
  
        HelloResponse helloResponse =  
            HelloResponse.newBuilder()  
                .setResponse("입금했다 미친XX야!!")  
                .build();  
  
        // 응답 시작  
        responseObserver.onNext(helloResponse);  
        // 응답 시작 후 1초 후에 응답 완료된다고 가정  
        try { Thread.sleep( millis: 1000); } catch (InterruptedException e) { e.printStackTrace(); }  
        // 응답 완료  
        responseObserver.onCompleted();  
    }  
}
```

# HelloGrpcServerService.java

```
public class HelloGrpcServerService extends HelloSpringCampGrpc.HelloSpringCampImplBase {  
    private final Logger logger = Logger.getLogger(HelloGrpcServerService.class.getName());  
  
    @Override  
    public void unaryHello(HelloRequest request, StreamObserver<HelloResponse> responseObserver) {  
        logger.info(msg: "[트럼프] 요청 받음 : " + request.getRequest());  
  
        // 1초 동안 비즈니스 로직 처리 후에 응답한다고 가정  
        try { Thread.sleep(millis: 1000); } catch (InterruptedException e) { e.printStackTrace(); }  
  
        HelloResponse helloResponse =  
            HelloResponse.newBuilder()  
                .setResponse("입금했다 미친XX야!!")  
                .build();  
  
        // 응답 시작  
        responseObserver.onNext(helloResponse);  
        // 응답 시작 후 1초 후에 응답 완료된다고 가정  
        try { Thread.sleep(millis: 1000); } catch (InterruptedException e) { e.printStackTrace(); }  
        // 응답 완료  
        responseObserver.onCompleted();  
    }  
}
```

# HelloGrpcServerService.java

```
public class HelloGrpcServerService extends HelloSpringCampGrpc.HelloSpringCampImplBase {  
    private final Logger logger = Logger.getLogger(HelloGrpcServerService.class.getName());  
  
    @Override  
    public void unaryHello(HelloRequest request, StreamObserver<HelloResponse> responseObserver) {  
        logger.info(msg: "[트럼프] 요청 받음 : " + request.getRequest());  
  
        // 1초 동안 비즈니스 로직 처리 후에 응답한다고 가정  
        try { Thread.sleep(millis: 1000); } catch (InterruptedException e) { e.printStackTrace(); }  
  
        HelloResponse helloResponse =  
            HelloResponse.newBuilder()  
                .setResponse("입금했다 미친XX야!!")  
                .build();  
  
        // 응답 시작  
        responseObserver.onNext(helloResponse);  
        // 응답 시작 후 1초 후에 응답 완료된다고 가정  
        try { Thread.sleep(millis: 1000); } catch (InterruptedException e) { e.printStackTrace(); }  
        // 응답 완료  
        responseObserver.onCompleted();  
    }  
}
```

# HelloGrpcServerService.java

```
public class HelloGrpcServerService extends HelloSpringCampGrpc.HelloSpringCampImplBase {  
    private final Logger logger = Logger.getLogger(HelloGrpcServerService.class.getName());  
  
    @Override  
    public void unaryHello(HelloRequest request, StreamObserver<HelloResponse> responseObserver) {  
        logger.info(msg: "[트럼프] 요청 받음 : " + request.getRequest());  
  
        // 1초 동안 비즈니스 로직 처리 후에 응답한다고 가정  
        try { Thread.sleep(millis: 1000); } catch (InterruptedException e) { e.printStackTrace(); }  
  
        HelloResponse helloResponse =  
            HelloResponse.newBuilder()  
                .setResponse("입금했다 미친XX야!!")  
                .build();  
  
        // 응답 시작  
        responseObserver.onNext(helloResponse);  
        // 응답 시작 후 1초 후에 응답 완료된다고 가정  
        try { Thread.sleep(millis: 1000); } catch (InterruptedException e) { e.printStackTrace(); }  
        // 응답 완료  
        responseObserver.onCompleted();  
    }  
}
```

# gRPC 서버 구성

ServerService  
unaryHello()

Server  
addService(ServerService)

ServerRunner  
main()

# HelloGrpcServer.java

```
public class HelloGrpcServer {  
  
    private final Logger logger = Logger.getLogger(HelloGrpcServer.class.getName());  
  
    private final int port;  
    private final Server server;  
  
    public HelloGrpcServer(int port, BindableService service) {  
        this.port = port;  
        this.server = ServerBuilder.forPort(port)  
            .addService(service)  
            .build();  
    }  
  
    public void start() throws IOException, InterruptedException {  
        this.server.start();  
        logger.info(msg: "트럼프가 " + port + "포트에서 리스닝 중..");  
        this.server.awaitTermination();  
    }  
  
    public void shutdown() {  
        System.err.println("트럼프 서버 종료...");  
        server.shutdown();  
        System.err.println("트럼프 서버 종료 완료");  
    }  
}
```

# HelloGrpcServer.java

```
public class HelloGrpcServer {  
  
    private final Logger logger = Logger.getLogger(HelloGrpcServer.class.getName());  
  
    private final int port;  
    private final Server server;  
  
    public HelloGrpcServer(int port, BindableService service) {  
        this.port = port;  
        this.server = ServerBuilder.forPort(port)  
            .addService(service)  
            .build();  
    }  
  
    public void start() throws IOException, InterruptedException {  
        this.server.start();  
        logger.info(msg: "트럼프가 " + port + "포트에서 리스닝 중..");  
        this.server.awaitTermination();  
    }  
  
    public void shutdown() {  
        System.err.println("트럼프 서버 종료...");  
        server.shutdown();  
        System.err.println("트럼프 서버 종료 완료");  
    }  
}
```

# HelloGrpcServer.java

```
public class HelloGrpcServer {  
  
    private final Logger logger = Logger.getLogger(HelloGrpcServer.class.getName());  
  
    private final int port;  
    private final Server server;  
  
    public HelloGrpcServer(int port, BindableService service) {  
        this.port = port;  
        this.server = ServerBuilder.forPort(port)  
            .addService(service)  
            .build();  
    }  
  
    public void start() throws IOException, InterruptedException {  
        this.server.start();  
        logger.info(msg: "트럼프가 " + port + "포트에서 리스닝 중..");  
        this.server.awaitTermination();  
    }  
  
    public void shutdown() {  
        System.err.println("트럼프 서버 종료...");  
        server.shutdown();  
        System.err.println("트럼프 서버 종료 완료");  
    }  
}
```

# HelloGrpcServer.java

```
public class HelloGrpcServer {  
  
    private final Logger logger = Logger.getLogger(HelloGrpcServer.class.getName());  
  
    private final int port;  
    private final Server server;  
  
    public HelloGrpcServer(int port, BindableService service) {  
        this.port = port;  
        this.server = ServerBuilder.forPort(port)  
            .addService(service)  
            .build();  
    }  
  
    public void start() throws IOException, InterruptedException {  
        this.server.start();  
        logger.info(msg: "트럼프가 " + port + "포트에서 리스닝 중..");  
        this.server.awaitTermination();  
    }  
  
    public void shutdown() {  
        System.err.println("트럼프 서버 종료...");  
        server.shutdown();  
        System.err.println("트럼프 서버 종료 완료");  
    }  
}
```

# HelloGrpcServer.java

```
public class HelloGrpcServer {  
  
    private final Logger logger = Logger.getLogger(HelloGrpcServer.class.getName());  
  
    private final int port;  
    private final Server server;  
  
    public HelloGrpcServer(int port, BindableService service) {  
        this.port = port;  
        this.server = ServerBuilder.forPort(port)  
            .addService(service)  
            .build();  
    }  
  
    public void start() throws IOException, InterruptedException {  
        this.server.start();  
        logger.info(msg:"트럼프가 " + port + "포트에서 리스닝 중..");  
        this.server.awaitTermination();  
    }  
  
    public void shutdown() {  
        System.err.println("트럼프 서버 종료...");  
        server.shutdown();  
        System.err.println("트럼프 서버 종료 완료");  
    }  
}
```

# HelloGrpcServer.java

```
public class HelloGrpcServer {  
  
    private final Logger logger = Logger.getLogger(HelloGrpcServer.class.getName());  
  
    private final int port;  
    private final Server server;  
  
    public HelloGrpcServer(int port, BindableService service) {  
        this.port = port;  
        this.server = ServerBuilder.forPort(port)  
            .addService(service)  
            .build();  
    }  
  
    public void start() throws IOException, InterruptedException {  
        this.server.start();  
        logger.info(msg: "트럼프가 " + port + "포트에서 리스닝 중..");  
        this.server.awaitTermination();  
    }  
  
    public void shutdown() {  
        System.err.println("트럼프 서버 종료...");  
        server.shutdown();  
        System.err.println("트럼프 서버 종료 완료");  
    }  
}
```

# gRPC 서버 구성

ServerService  
unaryHello()

Server  
addService(ServerService)

ServerRunner  
main()

# HelloGrpcServerRunner.java

```
public class HelloGrpcServerRunner {  
    public static void main(String[] args) throws IOException, InterruptedException {  
        final int port = 54321;  
        final BindableService helloService = new HelloGrpcServerService();  
  
        HelloGrpcServer server = new HelloGrpcServer(port, helloService);  
  
        server.start();  
  
        Runtime.getRuntime().addShutdownHook(  
            new Thread(() -> server.shutdown())  
        );  
    }  
}
```

# HelloGrpcServerRunner.java

```
public class HelloGrpcServerRunner {  
    public static void main(String[] args) throws IOException, InterruptedException {  
        final int port = 54321;  
        final BindableService helloService = new HelloGrpcServerService();  
  
        HelloGrpcServer server = new HelloGrpcServer(port, helloService);  
  
        server.start();  
  
        Runtime.getRuntime().addShutdownHook(  
            new Thread(() -> server.shutdown())  
        );  
    }  
}
```

# HelloGrpcServerRunner.java

```
public class HelloGrpcServerRunner {  
    public static void main(String[] args) throws IOException, InterruptedException {  
        final int port = 54321;  
        final BindableService helloService = new HelloGrpcServerService();  
  
        HelloGrpcServer server = new HelloGrpcServer(port, helloService);  
  
        server.start();  
  
        Runtime.getRuntime().addShutdownHook(  
            new Thread(() -> server.shutdown())  
        );  
    }  
}
```

# HelloGrpcServerRunner.java

```
public class HelloGrpcServerRunner {  
    public static void main(String[] args) throws IOException, InterruptedException {  
        final int port = 54321;  
        final BindableService helloService = new HelloGrpcServerService();  
  
        HelloGrpcServer server = new HelloGrpcServer(port, helloService);  
  
        server.start();  
  
        Runtime.getRuntime().addShutdownHook(  
            new Thread(() -> server.shutdown())  
        );  
    }  
}
```

# HelloGrpcServerRunner 실행

```
Run [HelloGrpcServerRunner]
[Run] "C:\Program Files\RedHat\java-1.8.0-openjdk-1.8.0.111-3\bin\java" ...
[Up] 4월 22, 2017 9:46:44 오전 homo.efficio.springcamp2017.grpc.hello.HelloGrpcServer start
[Down] 정보: 트럼프가 54321포트에서 리스닝 중..
[Pause] [Stop] [Minimize] [Close]
```

# gRPC Hello World



build.gradle

서버랑 완전 같음

gRPC 기반 코드 생성

gRPC Dependency 해결

gRPC 클라이언트 작성

# gRPC 클라이언트 구성

Client

`sendAsyncUnaryMessage()`

# gRPC 클라이언트 구성

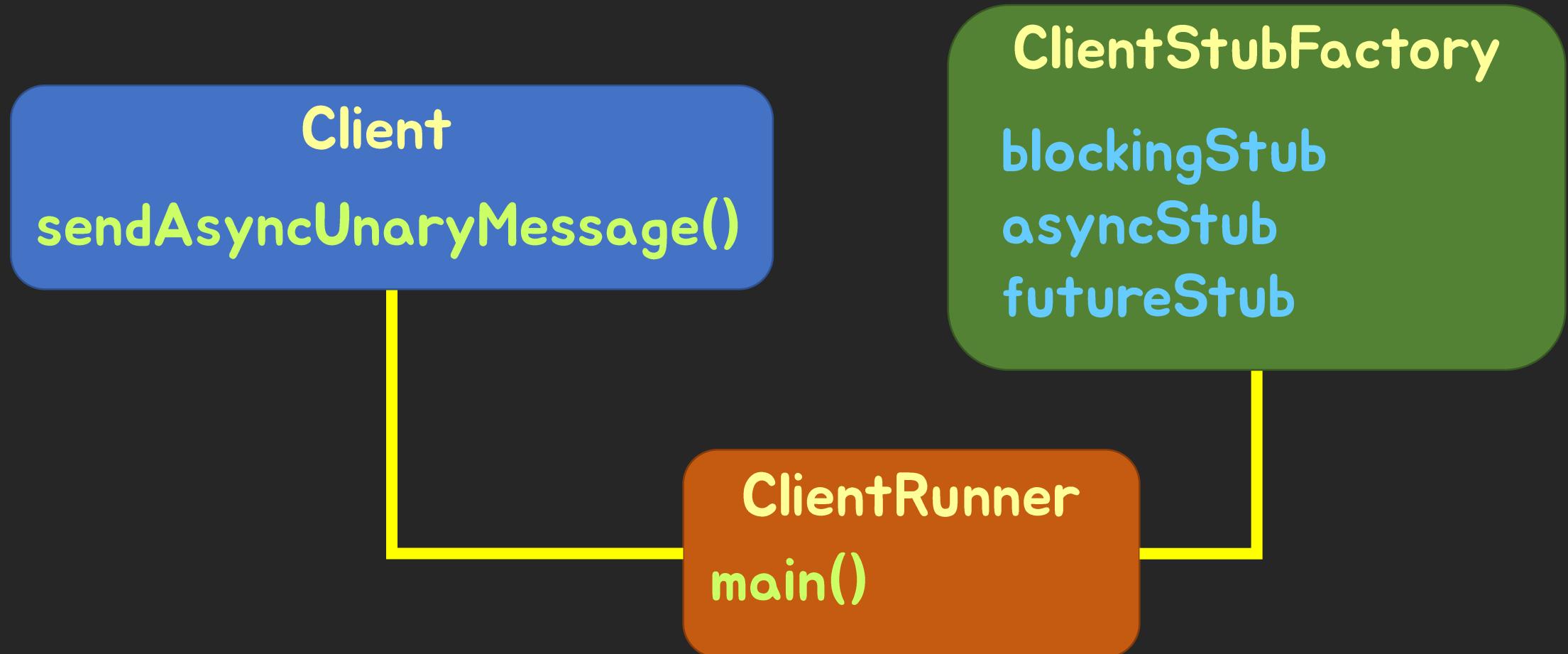
Client

`sendAsyncUnaryMessage()`

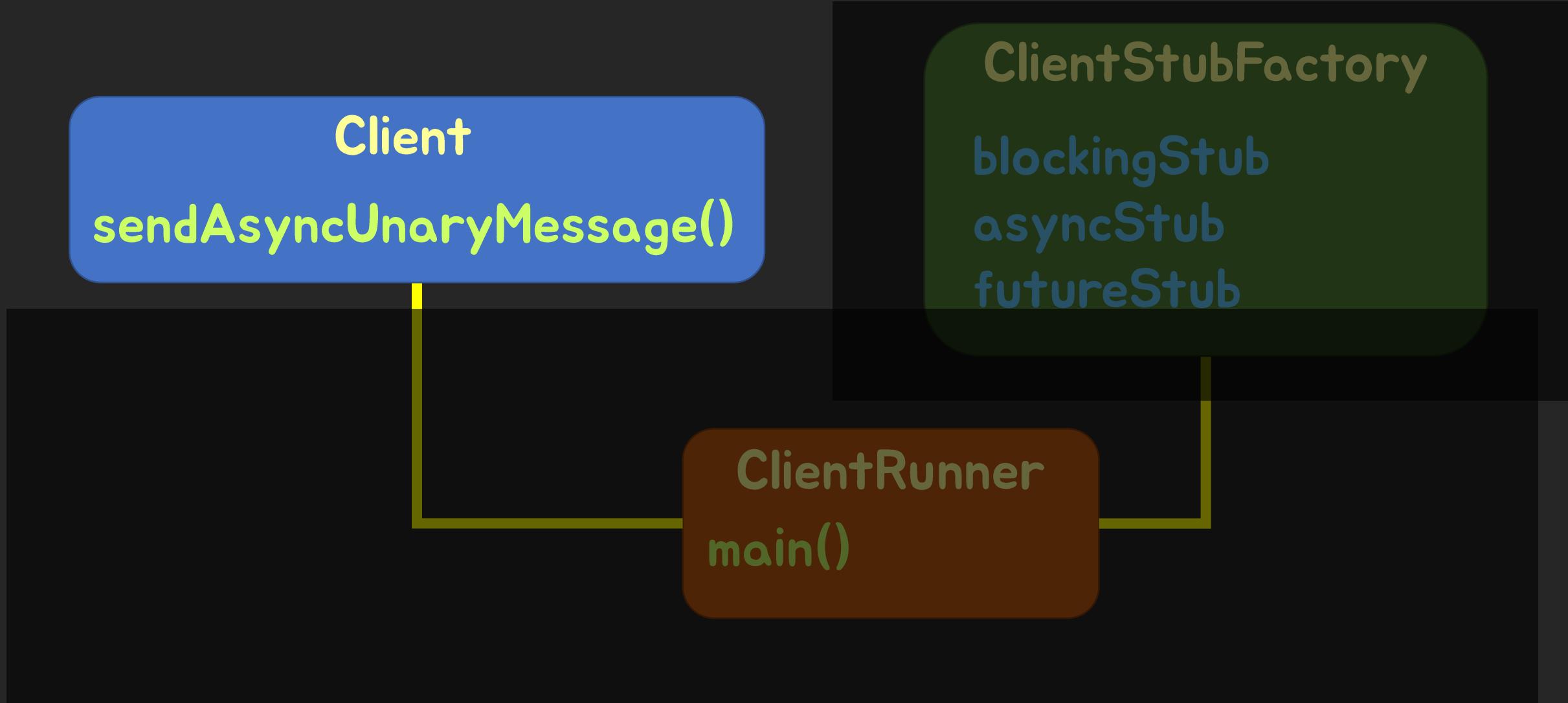
ClientStubFactory

`blockingStub`  
`asyncStub`  
`futureStub`

# gRPC 클라이언트 구성



# gRPC 클라이언트 구성



# HelloGrpcClient.java - 1/2

```
public class HelloGrpcClient {  
  
    private final Logger logger = Logger.getLogger(HelloGrpcClient.class.getName());  
  
    private final HelloSpringCampGrpc.HelloSpringCampBlockingStub blockingStub;  
    private final HelloSpringCampGrpc.HelloSpringCampStub asyncStub;  
    private final HelloSpringCampGrpc.HelloSpringCampFutureStub futureStub;  
  
    public HelloGrpcClient(HelloSpringCampGrpc.HelloSpringCampBlockingStub blockingStub,  
                           HelloSpringCampGrpc.HelloSpringCampStub asyncStub,  
                           HelloSpringCampGrpc.HelloSpringCampFutureStub futureStub) {  
        this.blockingStub = blockingStub;  
        this.asyncStub = asyncStub;  
        this.futureStub = futureStub;  
    }  
  
    public void sendAsyncUnaryMessage() {...}  
}
```

# HelloGrpcClient.java - 1/2

```
public class HelloGrpcClient {  
  
    private final Logger logger = Logger.getLogger(HelloGrpcClient.class.getName());  
  
    private final HelloSpringCampGrpc.HelloSpringCampBlockingStub blockingStub;  
    private final HelloSpringCampGrpc.HelloSpringCampStub asyncStub;  
    private final HelloSpringCampGrpc.HelloSpringCampFutureStub futureStub;  
  
    public HelloGrpcClient(HelloSpringCampGrpc.HelloSpringCampBlockingStub blockingStub,  
                           HelloSpringCampGrpc.HelloSpringCampStub asyncStub,  
                           HelloSpringCampGrpc.HelloSpringCampFutureStub futureStub) {  
        this.blockingStub = blockingStub;  
        this.asyncStub = asyncStub;  
        this.futureStub = futureStub;  
    }  
  
    public void sendAsyncUnaryMessage() {...}  
}
```

# HelloGrpcClient.java - 1/2

```
public class HelloGrpcClient {  
  
    private final Logger logger = Logger.getLogger(HelloGrpcClient.class.getName());  
  
    private final HelloSpringCampGrpc.HelloSpringCampBlockingStub blockingStub;  
    private final HelloSpringCampGrpc.HelloSpringCampStub asyncStub;  
    private final HelloSpringCampGrpc.HelloSpringCampFutureStub futureStub;  
  
    public HelloGrpcClient(HelloSpringCampGrpc.HelloSpringCampBlockingStub blockingStub,  
                           HelloSpringCampGrpc.HelloSpringCampStub asyncStub,  
                           HelloSpringCampGrpc.HelloSpringCampFutureStub futureStub) {  
        this.blockingStub = blockingStub;  
        this.asyncStub = asyncStub;  
        this.futureStub = futureStub;  
    }  
  
    public void sendAsyncUnaryMessage() {...}  
}
```

## HelloGrpcClient.java - 2/2

```
public void sendAsyncUnaryMessage() {
    // 클라이언트 비즈니스 로직 수행 결과를 message에 할당한 clientName으로 request 생성
    String message = "입금해 형 ㅋ";
    // message로 request 생성
    HelloRequest request = HelloRequest.newBuilder().setRequest(message).build();
    // 서버에 보낼 데이터를 담은 request와
    // 비동기 방식으로 서버에서 호출될 콜백 객체도 함께 파라미터로 전달
    logger.info(msg: "[김정은] 요청 전송 : " + message);
    asyncStub.unaryHello(
        request,
        // 서버에 보낼 콜백 객체
        new StreamObserver<HelloResponse>() {...}
    );
    // 서버에서 응답이 올 때까지 기다리지 않고, 호출 결과에 상관없이 다른 작업 수행 가능
    logger.info(msg: "[김정은] : 강 핵이나 쏴야지 ㅋㅋ");
}
```

## HelloGrpcClient.java - 2/2

```
public void sendAsyncUnaryMessage() {
    // 클라이언트 비즈니스 로직 수행 결과를 message에 할당한 clientName으로 request 생성
    String message = "입금해 형 ㅋ";
    // message로 request 생성
    HelloRequest request = HelloRequest.newBuilder().setRequest(message).build();

    // 서버에 보낼 데이터를 담은 request와
    // 비동기 방식으로 서버에서 호출될 콜백 객체도 함께 파라미터로 전달
    logger.info(msg: "[김정은] 요청 전송 : " + message);
    asyncStub.unaryHello(
        request,
        // 서버에 보낼 콜백 객체
        new StreamObserver<HelloResponse>() {...}
    );
    // 서버에서 응답이 올 때까지 기다리지 않고, 호출 결과에 상관없이 다른 작업 수행 가능
    logger.info(msg: "[김정은] : 강 핵이나 쏴야지 ㅋㅋ");
}
```

## HelloGrpcClient.java - 2/2

```
public void sendAsyncUnaryMessage() {
    // 클라이언트 비즈니스 로직 수행 결과를 message에 할당한 clientName으로 request 생성
    String message = "입금해 형 ㅋ";
    // message로 request 생성
    HelloRequest request = HelloRequest.newBuilder().setRequest(message).build();

    // 서버에 보낼 데이터를 담은 request와
    // 비동기 방식으로 서버에서 호출될 콜백 객체도 함께 파라미터로 전달
    logger.info(msg: "[김정은] 요청 전송 : " + message);
    asyncStub.unaryHello(
        request,
        // 서버에 보낼 콜백 객체
        new StreamObserver<HelloResponse>() {...}
    );
    // 서버에서 응답이 올 때까지 기다리지 않고, 호출 결과에 상관없이 다른 작업 수행 가능
    logger.info(msg: "[김정은] : 강 핵이나 쏴야지 ㅋㅋ");
}
```

## HelloGrpcClient.java - 2/2

```
public void sendAsyncUnaryMessage() {
    // 클라이언트 비즈니스 로직 수행 결과를 message에 할당한 clientName으로 request 생성
    String message = "입금해 형 ㅋ";
    // message로 request 생성
    HelloRequest request = HelloRequest.newBuilder().setRequest(message).build();

    // 서버에 보낼 데이터를 담은 request와
    // 비동기 방식으로 서버에서 호출될 콜백 객체도 함께 파라미터로 전달
    logger.info(msg: "[김정은] 요청 전송 : " + message);
    asyncStub.unaryHello(
        request,
        // 서버에 보낼 콜백 객체
        new StreamObserver<HelloResponse>() {...}
    );
    // 서버에서 응답이 올 때까지 기다리지 않고, 호출 결과에 상관없이 다른 작업 수행 가능
    logger.info(msg: "[김정은] : 강 핵이나 쏴야지 ㅋㅋ");
}
```

## HelloGrpcClient.java - 2/2

```
public void sendAsyncUnaryMessage() {
    // 클라이언트 비즈니스 로직 수행 결과를 message에 할당한 clientName으로 request 생성
    String message = "입금해 형 ㅋ";
    // message로 request 생성
    HelloRequest request = HelloRequest.newBuilder().setRequest(message).build();
    // 서비스에 보내는 메시지로는 HelloRequest이며, 응답은 HelloResponse로
    // 동시에 콜백으로 예상되는 StreamObserver<HelloResponse>도 함께 파라미터로 전달
    logger.info(msg: "[김정은] 요청 전송 : " + message);
    asyncStub.unaryHello(
        request,
        // 서버에 보낼 콜백 객체
        new StreamObserver<HelloResponse>() {...}
    );
    // 서버에서 응답이 올 때까지 기다리지 않고, 호출 결과에 상관없이 다른 작업 수행 가능
    logger.info(msg: "[김정은] : 양 핵이나 쏴야지 ㅋㅋ");
}
```

### 로컬에서드처럼 호출하는 RPC

## HelloGrpcClient.java - 2/2

```
public void sendAsyncUnaryMessage() {
    // 클라이언트 비즈니스 로직 수행 결과를 message에 할당한 clientName으로 request 생성
    String message = "입금해 형 ㅋ";
    // message로 request 생성
    HelloRequest request = HelloRequest.newBuilder().setRequest(message).build();

    // 서버에 보낼 데이터를 담은 request와
    // 비동기 방식으로 서버에서 호출될 콜백 객체도 함께 파라미터로 전달
    logger.info(msg: "[김정은] 요청 전송 : " + message);
    asyncStub.unaryHello(
        request,
        // 서버에 보낼 콜백 객체
        new StreamObserver<HelloResponse>() {...}
    );
    // 서버에서 응답이 올 때까지 기다리지 않고, 호출 결과에 상관없이 다른 작업 수행 가능
    logger.info(msg: "[김정은] : 강 핵이나 쏴야지 ㅋㅋ");
}
```

## HelloGrpcClient.java - 2/2

```
new StreamObserver<HelloResponse>() {
    @Override
    public void onNext(HelloResponse response) {
        logger.info(msg: "[트럼프로부터의 응답] : " + response.getResponse());
    }

    @Override
    public void onError(Throwable t) {
        logger.log(Level.SEVERE, msg:"Async Unary responseObserver.onError() 호출됨");
    }

    @Override
    public void onCompleted() { logger.info(msg: "[트럼프로부터의 응답 종료]"); }
}
```

## HelloGrpcClient.java - 2/2

```
new StreamObserver<HelloResponse>() {
    @Override
    public void onNext(HelloResponse response) {
        logger.info(msg: "[트럼프로부터의 응답] : " + response.getResponse());
    }

    @Override
    public void onError(Throwable t) {
        logger.log(Level.SEVERE, msg:"Async Unary responseObserver.onError() 호출됨");
    }

    @Override
    public void onCompleted() { logger.info(msg: "[트럼프로부터의 응답 종료]"); }
}
```

## HelloGrpcClient.java - 2/2

```
new StreamObserver<HelloResponse>() {
    @Override
    public void onNext(HelloResponse response) {
        logger.info(msg: "[트럼프로부터의 응답] : " + response.getResponse());
    }

    @Override
    public void onError(Throwable t) {
        logger.log(Level.SEVERE, msg:"Async Unary responseObserver.onError() 호출됨");
    }

    @Override
    public void onCompleted() { logger.info(msg: "[트럼프로부터의 응답 종료]"); }
}
```

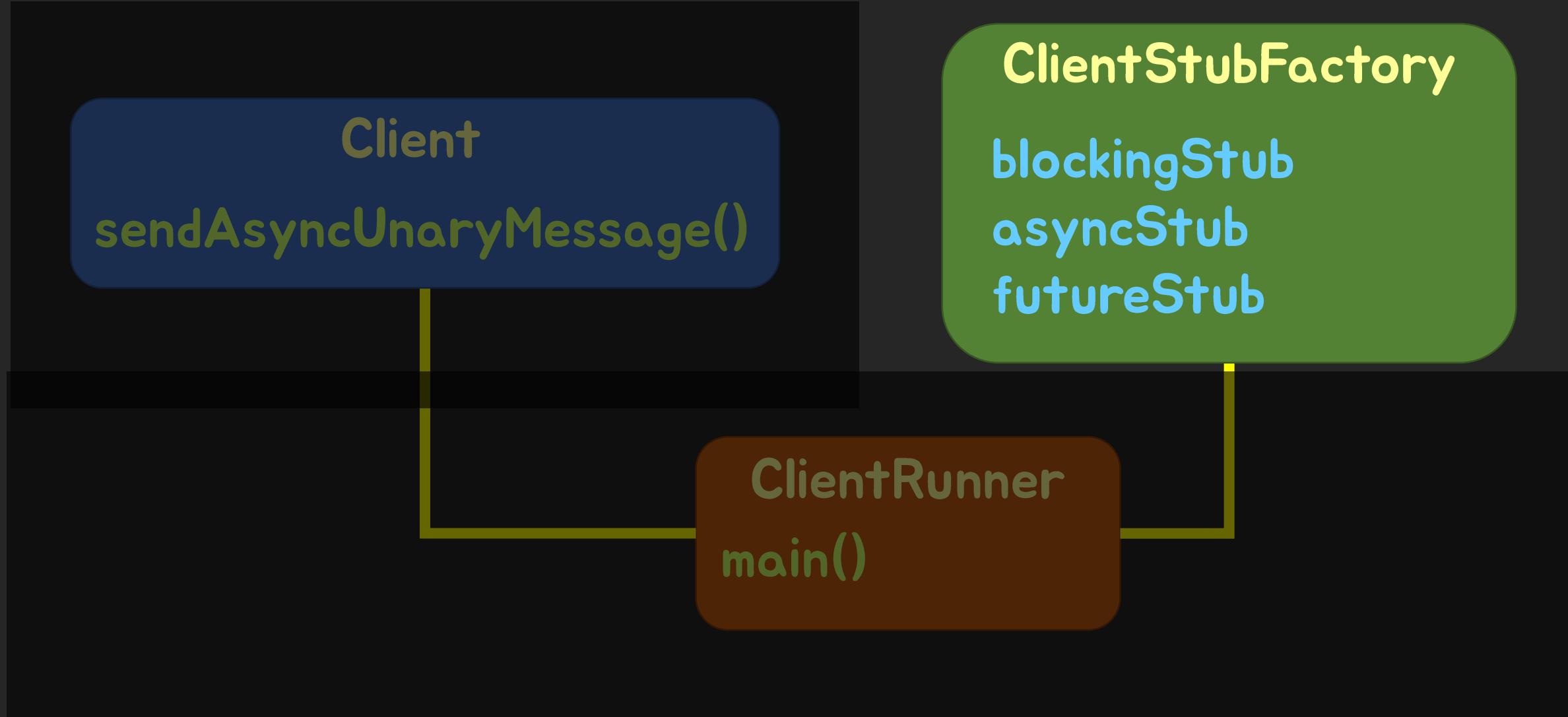
## HelloGrpcClient.java - 2/2

```
new StreamObserver<HelloResponse>() {
    @Override
    public void onNext(HelloResponse response) {
        logger.info(msg: "[트럼프로부터의 응답] : " + response.getResponse());
    }

    @Override
    public void onError(Throwable t) {
        logger.log(Level.SEVERE, msg:"Async Unary responseObserver.onError() 호출됨");
    }

    @Override
    public void onCompleted() { logger.info(msg: "[트럼프로부터의 응답 종료]"); }
}
```

# gRPC 클라이언트 구성



# HelloGrpcClientStubFactory.java - 1/2

↳ HelloGrpcClientStubFactory.java ×

```
13  public class HelloGrpcClientStubFactory {  
14  
15      private final Logger logger = Logger.getLogger(HelloGrpcClientStubFactory.class.getName());  
16  
17      private final ManagedChannel channel;  
18      private final HelloSpringCampGrpc.HelloSpringCampBlockingStub blockingStub;  
19      private final HelloSpringCampGrpc.HelloSpringCampStub asyncStub;  
20      private final HelloSpringCampGrpc.HelloSpringCampFutureStub futureStub;  
21  
22      public HelloGrpcClientStubFactory(String host, int port) {  
23          this.channel = ManagedChannelBuilder.forAddress(host, port)  
24              .usePlaintext(true)  
25              .build();  
26          this.blockingStub = HelloSpringCampGrpc.newBlockingStub(channel);  
27          this.asyncStub = HelloSpringCampGrpc.newStub(channel);  
28          this.futureStub = HelloSpringCampGrpc.newFutureStub(channel);  
29      }  
30  }
```

# HelloGrpcClientStubFactory.java - 1/2

1 HelloGrpcClientStubFactory.java ×

```
13 public class HelloGrpcClientStubFactory {  
14  
15     private final Logger logger = Logger.getLogger(HelloGrpcClientStubFactory.class.getName());  
16  
17     private final ManagedChannel channel;  
18     private final HelloSpringCampGrpc.HelloSpringCampBlockingStub blockingStub;  
19     private final HelloSpringCampGrpc.HelloSpringCampStub asyncStub;  
20     private final HelloSpringCampGrpc.HelloSpringCampFutureStub futureStub;  
21  
22     public HelloGrpcClientStubFactory(String host, int port) {  
23         this.channel = ManagedChannelBuilder.forAddress(host, port)  
24             .usePlaintext(true)  
25             .build();  
26         this.blockingStub = HelloSpringCampGrpc.newBlockingStub(channel);  
27         this.asyncStub = HelloSpringCampGrpc.newStub(channel);  
28         this.futureStub = HelloSpringCampGrpc.newFutureStub(channel);  
29     }  
30 }
```

# HelloGrpcClientStubFactory.java - 1/2

1 HelloGrpcClientStubFactory.java \*

```
13 public class HelloGrpcClientStubFactory {  
14  
15     private final Logger logger = Logger.getLogger(HelloGrpcClientStubFactory.class.getName());  
16  
17     private final ManagedChannel channel;  
18     private final HelloSpringCampGrpc.HelloSpringCampBlockingStub blockingStub;  
19     private final HelloSpringCampGrpc.HelloSpringCampStub asyncStub;  
20     private final HelloSpringCampGrpc.HelloSpringCampFutureStub futureStub;  
21  
22     public HelloGrpcClientStubFactory(String host, int port) {  
23         this.channel = ManagedChannelBuilder.forAddress(host, port)  
24             .usePlaintext(true)  
25             .build();  
26         this.blockingStub = HelloSpringCampGrpc.newBlockingStub(channel);  
27         this.asyncStub = HelloSpringCampGrpc.newStub(channel);  
28         this.futureStub = HelloSpringCampGrpc.newFutureStub(channel);  
29     }  
30 }
```

# HelloGrpcClientStubFactory.java - 1/2

1 HelloGrpcClientStubFactory.java ×

```
13 public class HelloGrpcClientStubFactory {  
14  
15     private final Logger logger = Logger.getLogger(HelloGrpcClientStubFactory.class.getName());  
16  
17     private final ManagedChannel channel;  
18     private final HelloSpringCampGrpc.HelloSpringCampBlockingStub blockingStub;  
19     private final HelloSpringCampGrpc.HelloSpringCampStub asyncStub;  
20     private final HelloSpringCampGrpc.HelloSpringCampFutureStub futureStub;  
21  
22     public HelloGrpcClientStubFactory(String host, int port) {  
23         this.channel = ManagedChannelBuilder.forAddress(host, port)  
24             .usePlaintext(true)  
25             .build();  
26         this.blockingStub = HelloSpringCampGrpc.newBlockingStub(channel);  
27         this.asyncStub = HelloSpringCampGrpc.newStub(channel);  
28         this.futureStub = HelloSpringCampGrpc.newFutureStub(channel);  
29     }  
~~
```

# HelloGrpcClientStubFactory.java - 2/2

1: HelloGrpcClientStubFactory.java \*

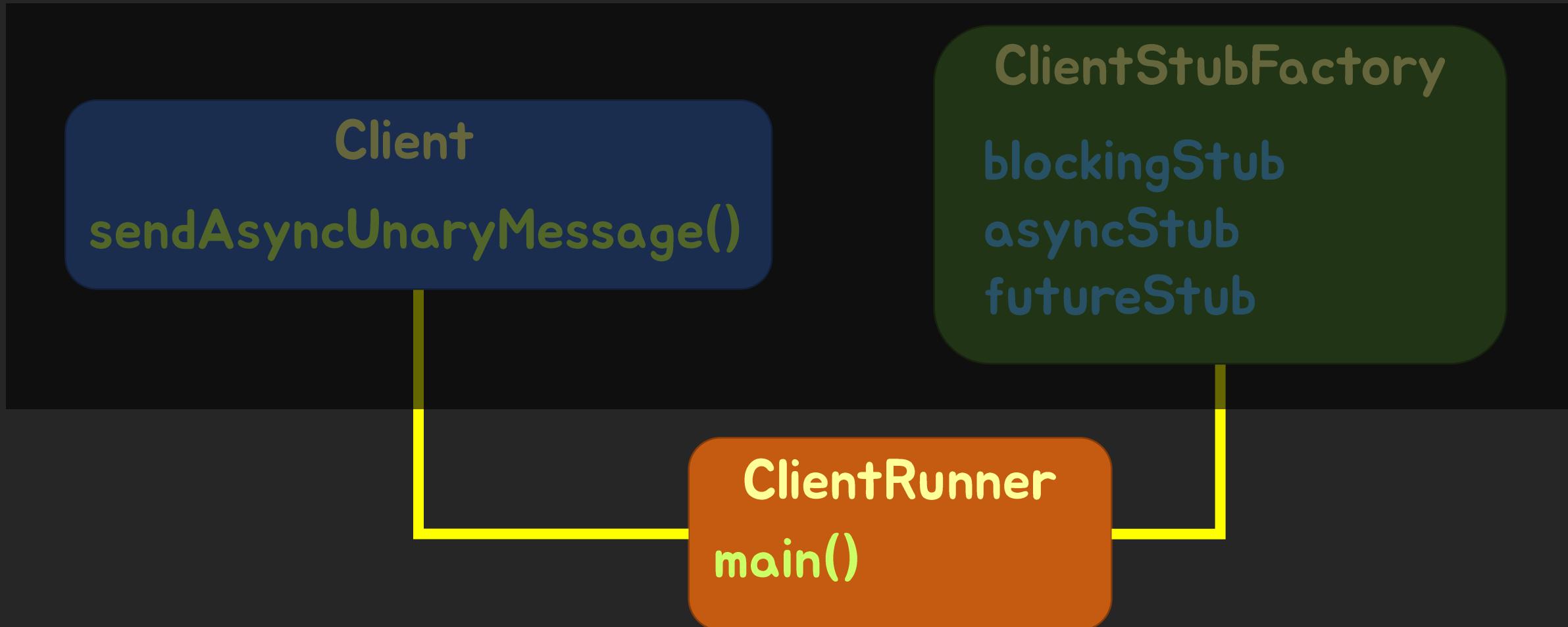
```
30
31     public void shutdownChannel() throws InterruptedException {
32         logger.info( msg: "gRPC Channel shutdown..." );
33         this.channel.shutdown().awaitTermination( timeout: 2, TimeUnit.SECONDS );
34     }
35
36     public HelloSpringCampGrpc.HelloSpringCampBlockingStub getBlockingStub() {
37         return blockingStub;
38     }
39
40     public HelloSpringCampGrpc.HelloSpringCampStub getAsyncStub() {
41         return asyncStub;
42     }
43
44     public HelloSpringCampGrpc.HelloSpringCampFutureStub getFutureStub() {
45         return futureStub;
46     }
47 }
```

# HelloGrpcClientStubFactory.java - 2/2

1: HelloGrpcClientStubFactory.java \*

```
30
31     public void shutdownChannel() throws InterruptedException {
32         logger.info( msg: "gRPC Channel shutdown..." );
33         this.channel.shutdown().awaitTermination( timeout: 2, TimeUnit.SECONDS );
34     }
35
36     public HelloSpringCampGrpc.HelloSpringCampBlockingStub getBlockingStub() {
37         return blockingStub;
38     }
39
40     public HelloSpringCampGrpc.HelloSpringCampStub getAsyncStub() {
41         return asyncStub;
42     }
43
44     public HelloSpringCampGrpc.HelloSpringCampFutureStub getFutureStub() {
45         return futureStub;
46     }
47 }
```

# gRPC 클라이언트 구성



# HelloGrpcClientRunner.java

```
public class HelloGrpcClientRunner {  
    public static void main(String[] args) throws InterruptedException {  
        String host = "localhost";  
        int port = 54321;  
  
        HelloGrpcClientStubFactory clientStubFactory =  
            new HelloGrpcClientStubFactory(host, port);  
  
        HelloGrpcClient grpcClient =  
            new HelloGrpcClient(clientStubFactory.getBlockingStub(),  
                clientStubFactory.getAsyncStub(),  
                clientStubFactory.getFutureStub());  
  
        grpcClient.sendAsyncUnaryMessage();  
        Thread.sleep( millis: 3000 );  
        clientStubFactory.shutdownChannel();  
    }  
}
```

# HelloGrpcClientRunner.java

```
public class HelloGrpcClientRunner {  
    public static void main(String[] args) throws InterruptedException {  
        String host = "localhost";  
        int port = 54321;  
  
        HelloGrpcClientStubFactory clientStubFactory =  
            new HelloGrpcClientStubFactory(host, port);  
  
        HelloGrpcClient grpcClient =  
            new HelloGrpcClient(clientStubFactory.getBlockingStub(),  
                clientStubFactory.getAsyncStub(),  
                clientStubFactory.getFutureStub());  
  
        grpcClient.sendAsyncUnaryMessage();  
        Thread.sleep( millis: 3000 );  
        clientStubFactory.shutdownChannel();  
    }  
}
```

# HelloGrpcClientRunner.java

```
public class HelloGrpcClientRunner {  
    public static void main(String[] args) throws InterruptedException {  
        String host = "localhost";  
        int port = 54321;  
  
        HelloGrpcClientStubFactory clientStubFactory =  
            new HelloGrpcClientStubFactory(host, port);  
  
        HelloGrpcClient grpcClient =  
            new HelloGrpcClient(clientStubFactory.getBlockingStub(),  
                clientStubFactory.getAsyncStub(),  
                clientStubFactory.getFutureStub());  
  
        grpcClient.sendAsyncUnaryMessage();  
        Thread.sleep( millis: 3000 );  
        clientStubFactory.shutdownChannel();  
    }  
}
```

# HelloGrpcClientRunner.java

```
public class HelloGrpcClientRunner {  
  
    public static void main(String[] args) throws InterruptedException {  
  
        String host = "localhost";  
        int port = 54321;  
  
        HelloGrpcClientStubFactory clientStubFactory =  
            new HelloGrpcClientStubFactory(host, port);  
  
        HelloGrpcClient grpcClient =  
            new HelloGrpcClient(clientStubFactory.getBlockingStub(),  
                clientStubFactory.getAsyncStub(),  
                clientStubFactory.getFutureStub());  
  
        grpcClient.sendAsyncUnaryMessage();  
        Thread.sleep( millis: 3000 );  
        clientStubFactory.shutdownChannel();  
    }  
}
```

# HelloGrpcClientRunner.java

```
public class HelloGrpcClientRunner {  
    public static void main(String[] args) throws InterruptedException {  
        String host = "localhost";  
        int port = 54321;  
  
        HelloGrpcClientStubFactory clientStubFactory =  
            new HelloGrpcClientStubFactory(host, port);  
  
        HelloGrpcClient grpcClient =  
            new HelloGrpcClient(clientStubFactory.getBlockingStub(),  
                clientStubFactory.getAsyncStub(),  
                clientStubFactory.getFutureStub());  
        grpcClient.sendAsyncUnaryMessage();  
        Thread.sleep( millis: 3000 );  
        clientStubFactory.shutdownChannel();  
    }  
}
```

예제라서 요청 후 채널을 닫았지만,  
실무에서는 채널을 닫지 않고 재사용 가능

# HelloGrpcClientRunner 실행

```
Run [ ] HelloGrpcClientRunner
▶ "C:\Program Files\RedHat\java-1.8.0-openjdk-1.8.0.111-3\bin\java" ...
4월 22, 2017 10:21:20 오전 homo.efficio.springcamp2017.grpc.hello.HelloGrpcClient sendAsyncUnaryMessage
정보: [김정은] 요청 전송 : 입금해 형 ㅋ
4월 22, 2017 10:21:20 오전 homo.efficio.springcamp2017.grpc.hello.HelloGrpcClient sendAsyncUnaryMessage
정보: [김정은] : 강 핵이나 싸야지 ㅋㅋ
4월 22, 2017 10:21:22 오전 homo.efficio.springcamp2017.grpc.hello.HelloGrpcClient$1 onNext
정보: [트럼프로부터의 응답] : 입금했다 미친xx0야!!
4월 22, 2017 10:21:23 오전 homo.efficio.springcamp2017.grpc.hello.HelloGrpcClient$1 onCompleted
정보: [트럼프로부터의 응답 종료]
4월 22, 2017 10:21:24 오전 homo.efficio.springcamp2017.grpc.hello.HelloGrpcClientStubFactory shutdownChannel
정보: gRPC Channel shutdown...
Process finished with exit code 0
```

# HelloGrpcClientRunner 실행

```
Run [ ] HelloGrpcClientRunner
▶ "C:\Program Files\RedHat\java-1.8.0-openjdk-1.8.0.111-3\bin\java" ...
■ 4월 22, 2017 10:21:20 오전 homo.efficio.springcamp2017.grpc.hello.HelloGrpcClient sendAsyncUnaryMessage
 정보: [김정은] 요청 전송 : 입금해 형 ㅋ
■ 4월 22, 2017 10:21:20 오전 homo.efficio.springcamp2017.grpc.hello.HelloGrpcClient sendAsyncUnaryMessage
 정보: [김정은] : 감 핵이나 쌩야지 ㅋㅋ
■ 4월 22, 2017 10:21:22 오전 homo.efficio.springcamp2017.grpc.hello.HelloGrpcClient$1 onNext
 정보: [트럼프로부터의 응답] : 입금했다 미친xx0야!!
■ 4월 22, 2017 10:21:23 오전 homo.efficio.springcamp2017.grpc.hello.HelloGrpcClient$1 onCompleted
 정보: [트럼프로부터의 응답 종료]
■ 4월 22, 2017 10:21:24 오전 homo.efficio.springcamp2017.grpc.hello.HelloGrpcClientStubFactory shutdownChannel
 정보: gRPC Channel shutdown...
Process finished with exit code 0
```

# HelloGrpcClientRunner 실행

Run □ HelloGrpcClientRunner

```
▶   "C:\Program Files\RedHat\java-1.8.0-openjdk-1.8.0.111-3\bin\java" ...
■  4월 22, 2017 10:21:20 오전 homo.efficio.springcamp2017.grpc.hello.HelloGrpcClient sendAsyncUnaryMessage
정보: [김정은] 요청 전송 : 입금해 헝 ㅋ
■  4월 22, 2017 10:21:20 오전 homo.efficio.springcamp2017.grpc.hello.HelloGrpcClient sendAsyncUnaryMessage
정보: [김정은] : 강 핵이나 싸야지 ㅋㅋ
■  4월 22, 2017 10:21:22 오전 homo.efficio.springcamp2017.grpc.hello.HelloGrpcClient$1 onNext
정보: [트럼프로부터의 응답] : 입금했다 미친xx0야!!
■  4월 22, 2017 10:21:23 오전 homo.efficio.springcamp2017.grpc.hello.HelloGrpcClient$1 onCompleted
정보: [트럼프로부터의 응답 종료]
■  4월 22, 2017 10:21:24 오전 homo.efficio.springcamp2017.grpc.hello.HelloGrpcClientStubFactory shutdownChannel
정보: gRPC Channel shutdown...
```

Process finished with exit code 0

# HelloGrpcClientRunner 실행

Run □ HelloGrpcClientRunner

```
▶   "C:\Program Files\RedHat\java-1.8.0-openjdk-1.8.0.111-3\bin\java" ...
■  4월 22, 2017 10:21:20 오전 homo.efficio.springcamp2017.grpc.hello.HelloGrpcClient sendAsyncUnaryMessage
정보: [김정은] 요청 전송 : 입금해 헝 ㅋ
■  4월 22, 2017 10:21:20 오전 homo.efficio.springcamp2017.grpc.hello.HelloGrpcClient sendAsyncUnaryMessage
정보: [김정은] : 감 핵이나 쌩야지 ㅋㅋ
■  4월 22, 2017 10:21:22 오전 homo.efficio.springcamp2017.grpc.hello.HelloGrpcClient$1 onNext
정보: [트럼프로부터의 응답] : 입금했다 미친xx0야!!
■  4월 22, 2017 10:21:23 오전 homo.efficio.springcamp2017.grpc.hello.HelloGrpcClient$1 onCompleted
정보: [트럼프로부터의 응답 종료]
■  4월 22, 2017 10:21:24 오전 homo.efficio.springcamp2017.grpc.hello.HelloGrpcClientStubFactory shutdownChannel
정보: gRPC Channel shutdown...
```

Process finished with exit code 0

# HelloGrpcClientRunner 실행

Run □ HelloGrpcClientRunner

```
▶   "C:\Program Files\RedHat\java-1.8.0-openjdk-1.8.0.111-3\bin\java" ...
■  4월 22, 2017 10:21:20 오전 homo.efficio.springcamp2017.grpc.hello.HelloGrpcClient sendAsyncUnaryMessage
정보: [김정은] 요청 전송 : 입금해 헝 ㅋ
■  4월 22, 2017 10:21:20 오전 homo.efficio.springcamp2017.grpc.hello.HelloGrpcClient sendAsyncUnaryMessage
정보: [김정은] : 감 핵이나 쌩야지 ㅋㅋ
■  4월 22, 2017 10:21:22 오전 homo.efficio.springcamp2017.grpc.hello.HelloGrpcClient$1 onNext
정보: [트럼프로부터의 응답] : 입금했다 미친xx0야!!
■  4월 22, 2017 10:21:23 오전 homo.efficio.springcamp2017.grpc.hello.HelloGrpcClient$1 onCompleted
정보: [트럼프로부터의 응답 종료]
■  4월 22, 2017 10:21:24 오전 homo.efficio.springcamp2017.grpc.hello.HelloGrpcClientStubFactory shutdownChannel
정보: gRPC Channel shutdown...
```

Process finished with exit code 0

# HelloGrpcClientRunner 실행

Run □ HelloGrpcClientRunner

```
▶   "C:\Program Files\RedHat\java-1.8.0-openjdk-1.8.0.111-3\bin\java" ...
■  4월 22, 2017 10:21:20 오전 homo.efficio.springcamp2017.grpc.hello.HelloGrpcClient sendAsyncUnaryMessage
정보: [김정은] 요청 전송 : 입금해 헝 ㅋ
■  4월 22, 2017 10:21:20 오전 homo.efficio.springcamp2017.grpc.hello.HelloGrpcClient sendAsyncUnaryMessage
정보: [김정은] : 감 핵이나 쌩야지 ㅋㅋ
■  4월 22, 2017 10:21:22 오전 homo.efficio.springcamp2017.grpc.hello.HelloGrpcClient$1 onNext
정보: [트럼프로부터의 응답] : 입금했다 미친xx0야!!
■  4월 22, 2017 10:21:23 오전 homo.efficio.springcamp2017.grpc.hello.HelloGrpcClient$1 onCompleted
정보: [트럼프로부터의 응답 종료]
■  4월 22, 2017 10:21:24 오전 homo.efficio.springcamp2017.grpc.hello.HelloGrpcClientStubFactory shutdownChannel
정보: gRPC Channel shutdown...
```

Process finished with exit code 0

# HelloGrpcServerRunner 실행

```
Run [HelloGrpcServerRunner]
" C:\Program Files\RedHat\java-1.8.0-openjdk-1.8.0.111-3\bin\java" ...
4월 22, 2017 10:21:06 오전 homo.efficio.springcamp2017.grpc.hello.HelloGrpcServer start
정보: 트럼프가 54321포트에서 리스닝 중..
4월 22, 2017 10:21:21 오전 homo.efficio.springcamp2017.grpc.hello.HelloGrpcServerService unaryHello
정보: [트럼프] 요청 받음 : 입금해 형 키
```

살짝 더 깊이

Thread?

# Thread와 gRPC

- Thread Safe
  - Stub
  - Channel
- Thread NOT Safe
  - StreamObserver

# Thread와 gRPC

- Thread Safe
  - Stub
  - Channel
- Thread NOT Safe
  - StreamObserver 개발자가 ThreadSafe하게 구현해야

# 해 볼시다.

- 4 스트리밍 \* 3 스텝 조합
- 복합 데이터 구조
- 인증, 예외/에러
- 자동 생성 소스 살펴보기
- 비용이 정말로 절감되나 등등..

# 참고 자료

- **RPC**: <https://goo.gl/KZ2cYC>
- **gRPC vs REST 성능**: <https://goo.gl/ZWk5jb>
- **gRPC Gateway**: <https://goo.gl/q9zDys>
- **gRPC Spring Boot Starter**: <https://goo.gl/NKyrpY>
- **Googley Microservices with gRPC**: <https://goo.gl/8Nc4yW>

# 예제 소스 링크

master 브랜치 : 4가지 Streaming 예제

comic 브랜치 : 김정일 트럼프 Async Unary 예제



[https://github.com/HomoEfficio/  
springcamp2017-grpc-java-  
server](https://github.com/HomoEfficio/springcamp2017-grpc-java-server)

[https://github.com/HomoEfficio/  
springcamp2017-grpc-java-  
client](https://github.com/HomoEfficio/springcamp2017-grpc-java-client)



# 정 리

초당 100억 개의 RPC를 처리하는 기술 gRPC!

.proto 파일만 작성하면 반은 자동!

3 가지 Stub, 4 가지 스트리밍

# 정 리

초당 100억 개의 RPC를 처리하는 기술 gRPC!

.proto 파일만 작성하면 반은 자동!

3 가지 Stub, 4 가지 스트리밍

g | 볍다. 싸장님의 사랑을 들통

# Q & A

# Quit & Adios

