Amit Himani Follow
Jan 11 · 3 min read · Listen

Save

Optimizing for Latency and Throughput: gRPC and Spring Boot

What is gRPC:

gRPC is a modern, high-performance, and lightweight open source framework for building scalable, distributed systems. It is a highly efficient and low-latency framework, designed to work over a variety of transport layers, such as TCP, HTTP/2, or even unencrypted UDP.

gRPC supports multiple programming languages, such as C++, Java, Python, Go, and Ruby, so it can be used to build a wide range of applications and services.

Open in app ↗ Sign up Sign In

Spring Boot is a popular open source framework for building web applications and microservices using the Spring framework. Some key features and benefits of Spring Boot: Auto-configuration, Embedded Servers, Standalone application, Ease of integration with other OSS projects like spring cloud, Spring data, Apache camel, Kafka etc.

Here is an example of a gRPC service implemented using Spring Boot:

Step 1: Start by adding the gRPC and Spring Boot dependencies to your build file:

```
dependencies {
    implementation 'io.grpc:grpc-netty-shaded:1.32.1'
    implementation 'io.grpc:grpc-protobuf:1.32.1'
    implementation 'io.grpc:grpc-stub:1.32.1'
    implementation 'org.springframework.boot:spring-boot-starter-grpc'
}
```

Step 2: define your gRPC service con 2 tocol Buffers. For example, let's say you have a service named employeeService with a single method named info() that takes a `EmployeeRequest` message and returns a `EmployeeResponse` message. Your `.proto` file might look like below

```
syntax = "proto3";

service EmployeeService {
  rpc greet (EmployeeRequest) returns (EmployeeResponse);
}

message EmployeeRequest {
  string name = 1;
}

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

Step 3: After this, use the `protoc` compiler to generate the gRPC service and message classes for your service. This could be jar file which can be used as dependency in service and client project

Step 4: Create the implementation of service, this will contain the actual logic to handle the incoming requests.

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

Step 5: Finally, in your Spring Boot application, add the `@EnableGRpcServer` annotation to enable the gRPC server.

```
@SpringBootApplication
@EnableGRpcServer
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Step 6: Run your spring boot application, and it will start the gRPC server on port 6565 by default.

PS: This is just a basic example of how to implement a gRPC service using Spring Boot, there are many other considerations and configurations that may be necessary depending on the specific requirements of your application.

References:

https://grpc.io/docs/languages/java/basics/

https://yidongnan.github.io/grpc-spring-boot-starter/en/

Scalability Spring Boot Grpc System Design Interview