gRPC 사용하기

Introduction to gRPC

2020.12.22. 석사과정 이요셉

Part 1. gRPC 개념 톺아보기

1.1. gRPC 소개

Why gRPC?

- gRPC는 프로세스간 통신 기술 중 하나임

- 기존 기술인 Restful 방식의 문제(분산 어플리케이션 간의 연결성 문제)
  - 부피가 크고 비효율적(바이너리가 아닌 텍스트 형식을 사용하기 때문)
  - 에러가 쉽게 발생함

- gRPC 프레임워크의 역할과 범위
  - 엄격한 서비스 규격 확인
  - 데이터 직렬화
  - 네트워크 통신, 인증
  - 접근 제어
  - 관찰 가능성(Observability)

1.1. gRPC 소개

gRPC 장점

- 프로세스간 통신의 효율성: 프로토콜 버터 기반 바이너리 프로토콜을 사용하기 때문(텍스트 기반이 아님), HTTP/2 의 적용
- 간단 명확한 서비스 인터페이스와 스키마: IDL로 프로토콜 버퍼를 사용하기 때문
- 엄격한 타입 점검 형식
- 폴리글랏: 여러 프로그래밍 언어와 작동하도록 설계됨
- 이중 스트리밍: 서버-클라이언트 각각에 스트리밍을 지원함
- 유용한 내장 기능 지원
- 클라우드 네이티브 생태계와 통합
- 성숙하고 널리 채택됨: 도커, 시스코, 스퀘어, 리프트, 넷플릭스 등

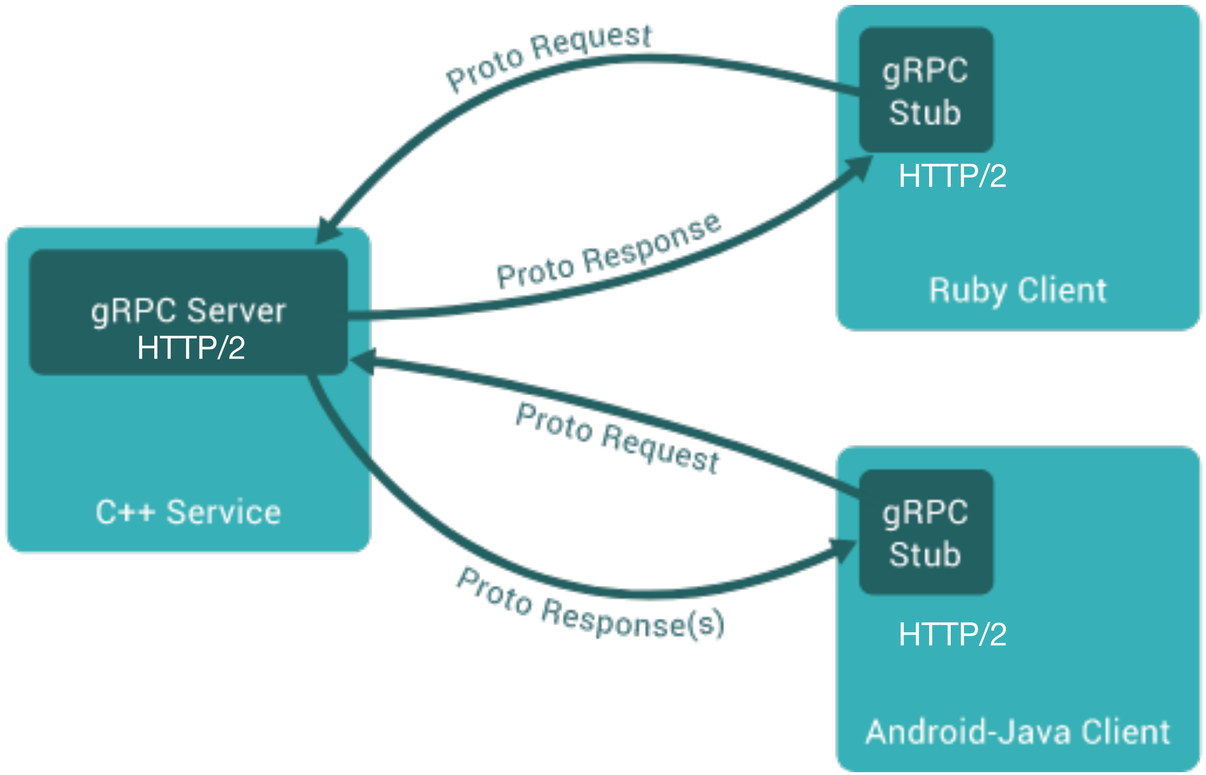
1.1. gRPC 소개

gRPC 단점

- 외부 서비스 부적합
  - 대부분 외부 사용가에게 gRPC는 새롭기 때문
- 서비스 정의의 급격한 변경에 따른 개발 프로세스 복잡성
  - 스키마 수정은 실제 프로젝트에서 빈번히 발생하지만 gRPC의 경우 이 경우 코드를 다시 생성해야만 하는 문제가 있음
  - 전체 개발 수명 주기를 복잡하게 할 수 있음
- 상대적으로 적은 생태계
  - REST나 HTTP에 비해 상대적으로 작음
  - 브라우저, 모바일 환경에서 gRPC의 지원이 여전히 초기 단계임

1.1. gRPC의 구조

Protocol Buffers를 활용 - 메시지의 효율적인 직렬화



https://grpc.io/docs/what-is-grpc/introduction/

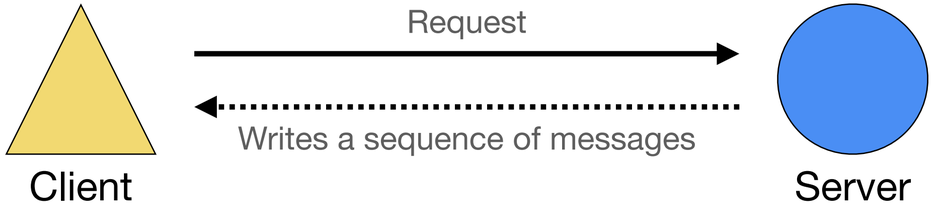
1.2. gRPC의 4가지 통신 방법

Streaming의 적용에 따른 4가지 조합의 통신 방법이 있음

1. Unary RPC(Simple RPC)

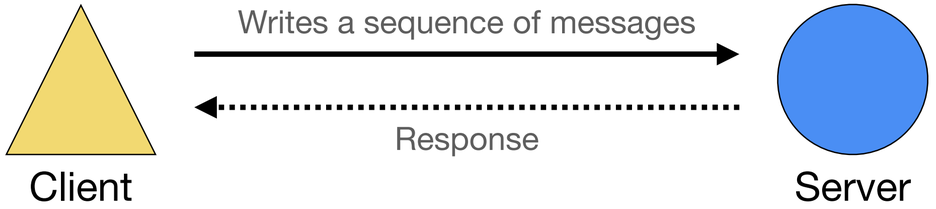
요청-응답의 가장 간단한 형태

2. Server-side streaming RPC



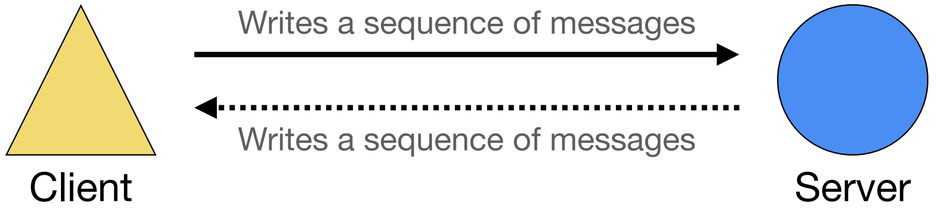
서버 메시지 전송이 끝나고 관련된 상태와
메타 데이터를 클라이언트가 수신하면 종료

3. Client-side streaming RPC



클라이언트 메시지 전송이 끝나고 관련된 상태와
메타 데이터를 서버가 수신하면 종료

4. Bidirectional streaming RPC



어느 메시지가 먼저 도착할 지 보장하지 않음을 주의

Part 2. gRPC 통신 실습

2. 실습 개요

#1. Hello world

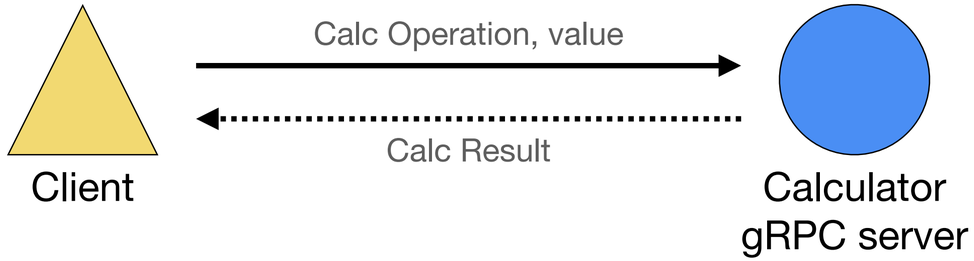
- gRPC 공식 홈페이지에 있는 Hello World 예제를 실시함
- 이를 통해 Protoc, Go 등 환경이 잘 설정되어 있는지 확인함

https://grpc.io/docs/languages/go/quickstart/

2. 실습 개요

#2. Calculator

- 실습을 위해 gRPC를 활용한 계산기를 구현함
- 클라이언트는 사칙연산과 값을 계산기에 전송하고 그 결과를 전송받음
- rpc를 통한 통신이기 때문에 클라이언트의 코드는 함수를 통해 처리하는 것과 다름 없게 표시됨
- gRPC의 모든 서비스를 구현하는 서버 단 코드와 클라이언트 단 코드가 모두 포함되어 있음



https://github.com/lecture4u/gRPC-introduction

2. 실습 개요

protocol buffers를 통해 통신 정의하기: @/calc/calc/calc.proto

파일 참고

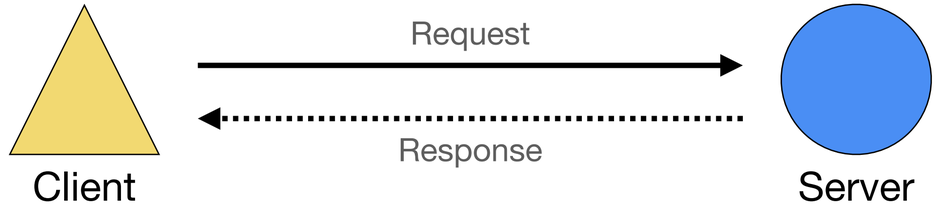
2. 실습 개요

Proto file을 컴파일하기

```
$ protoc --go_out=. --go_opt=paths=source_relative \
    --go-grpc_out=. --go-grpc_opt=paths=source_relative \
    calc/calc.proto
```

2.1. Unary RPC (simple RPC)

Unary RPC (simple RPC): CalcOperation( )



클라이언트 코드

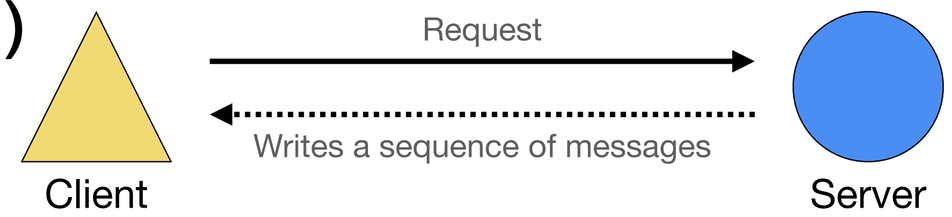
```
func calcOperation(ctx context.Context, c
pb.CalculatorClient) *pb.CalcResponse {
   r, err := c.CalcOperation(ctx, &pb.CalcRequest{
      Op:  pb.Operation_add,
      Val: 10,
   })

   if err != nil {
      log.Fatalf("could not calc: %v", err)
   }
   return r
}
```

서버 코드

```
func (s *server) CalcOperation(ctx context.Context, in
*pb.CalcRequest) (*pb.CalcResponse, error) {
   log.Printf("received: %s%s%v", in.GetOp().String(), " ,
", in.GetVal())
   processCalculation(s.c, in)

   return &pb.CalcResponse{
      Error:  true,
      Result: s.c.data,
   }, nil
}
```

2.2. Client-side streaming RPC

Server-side streaming RPC: getCurrentOperationsResults( )



클라이언트 코드

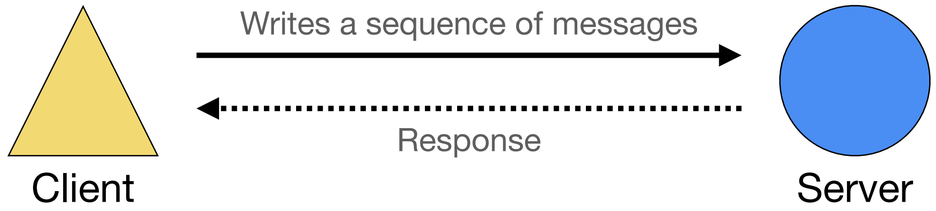
```
func getCurrentOperationResults(c pb.CalculatorClient) {
   ctx, cancel := context.WithTimeout(context.Background(),
10*time.Second)
   defer cancel()
   stream, err := c.GetCurrentOperationResults(ctx,
&pb.CalcRequest{})
   if err != nil {
      log.Fatalf("could not start streaming: %v", err)
   }
   log.Printf("Current operation results")
   for {
      r, err := stream.Recv()
      if err == io.EOF {
         break
      }
      if err != nil {
         log.Fatalf("got invalid value: %v", err)
      }
      log.Printf("result: %v | %v", r.GetResult(), r.GetError())
   }
}
```

서버 코드

```
func (s *server) GetCurrentOperationResults(in
*pb.CalcRequest, stream
pb.Calculator_GetCurrentOperationResultsServer) error {
   for _, r := range s.c.record {
      if err := stream.Send(&pb.CalcResponse{
         Error:  true,
         Result: r.Val,
      }); err != nil {
         return err
      }
   }
   return nil
}
```

2.3. Server-side streaming RPC

Client-side streaming RPC: CalcOperations( )



클라이언트 코드

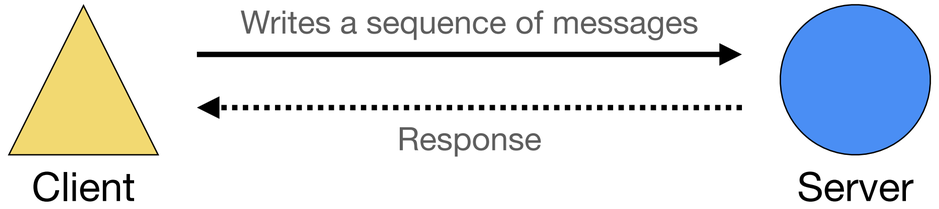
```
func calcOperations(c pb.CalculatorClient) *pb.CalcResponse {
    stream, ctxErr := c.CalcOperations(context.Background())
    if ctxErr != nil {
        log.Fatalf("could not stream: %v", ctxErr)
    }

    var ops []*pb.CalcRequest
    for i := 1; i < 11; i++ {
        ops = append(ops, &pb.CalcRequest{
            Op:  pb.Operation_add,
            Val: int64(i),
        })
    }

    //send
    for _, op := range ops {
        if err := stream.Send(op); err != nil {
            log.Fatalf("could not send: %v", err)
        }
    }

    response, err := stream.CloseAndRecv()
    if err != nil {
        log.Fatalf("could not get response from the server: %v", err)
    }
    return response
}
```

서버 코드

```
func (s *server) CalcOperations(stream
pb.Calculator_CalcOperationsServer) error {
   for {
      o, err := stream.Recv()
      if err == io.EOF {
         return stream.SendAndClose(&pb.CalcResponse{
            Error:  true,
            Result: s.c.data,
         })
      }
      if err != nil {
         return err
      }
      processCalculation(s.c, o)
   }
}
```

2.3. Client-side streaming RPC

Client-side streaming RPC: CalcJointOperation( )



클라이언트 코드

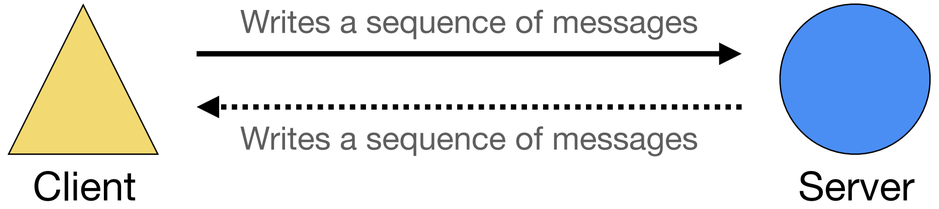
```
func calcJointOperation(c pb.CalculatorClient) {
   ctx, cancel := context.WithTimeout(context.Background(),
10*time.Second)
   defer cancel()
   stream, err := c.CalcJointOperation(ctx, &pb.CalcRequest{
      Op:  pb.Operation_add,
      Val: 100,
   })
   if err != nil {
      log.Fatalf("could not start streaming: %v", err)
   }
   log.Printf("Joint operation results")
   for {
      r, err := stream.Recv()
      if err == io.EOF {
         break
      }
      if err != nil {
         log.Fatalf("got invalid value: %v", err)
      }
      log.Printf("result: %v | %v", r.GetResult(), r.GetError())
   }
}
```

서버 코드

```
func (s *server) CalcJointOperation(in *pb.CalcRequest,
stream pb.Calculator_CalcJointOperationServer) error {
   log.Printf("received: %s%s%v", in.GetOp().String(), " ,
", in.GetVal())
   processCalculation(s.c, in)

   for _, d := range decreaseUnit {
      <-time.After(time.Millisecond * 500)
      processCalculation(s.c, &pb.CalcRequest{
         Op:  pb.Operation_sub,
         Val: int64(d),
      })

      stream.Send(&pb.CalcResponse{
         Error:  true,
         Result: s.c.data,
      })
   }
   return nil
}
```

2.4. Bidirectional streaming RPC

Bidirectional streaming RPC: CalcJointOperations( )



클라이언트 코드

```go
func calcJointOperations(c pb.CalculatorClient) {
	ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second)
	defer cancel()
	stream, err := c.CalcJointOperations(ctx)
	if err != nil {
		log.Fatalf("could not start streaming: %v", err)
	}

	var ops []*pb.CalcRequest
	for i := 0; i < 5; i++ {
		ops = append(ops, &pb.CalcRequest{
			Op:  pb.Operation_add,
			Val: 500,
		})
	}

	waitCh := make(chan struct{})
	go func() {
		for {
			r, err := stream.Recv()
			if err == io.EOF {
				log.Printf("receive done")
				close(waitCh)
				return
			}
			if err != nil {
				log.Fatalf("got invalid value: %v", err)
			}
			log.Printf("Result: %v | %v", r.GetResult(), r.GetError())
		}
	}()

	for _, op := range ops {
		<-time.After(time.Millisecond * 100)
		if err := stream.Send(op); err != nil {
			log.Fatalf("could not send: %v", err)
		}
	}
	stream.CloseSend()
	<-waitCh
}
```

서버 코드

```go
func (s *server) CalcJointOperations(stream pb.Calculator_CalcJointOperationsServer) error {
	for {
		r, err := stream.Recv()
		if err == io.EOF {
			return nil
		}
		if err != nil {
			return err
		}

		log.Printf("received: %s%s%v", r.GetOp().String(), " , ", r.GetVal())

		processCalculation(s.c, r)
		go func() {
			for _, d := range decreaseUnit {
				<-time.After(time.Millisecond)
				s.m.Lock()
				processCalculation(s.c, &pb.CalcRequest{
					Op:  pb.Operation_sub,
					Val: int64(d),
				})
				result := s.c.data
				s.m.Unlock()
				stream.Send(&pb.CalcResponse{
					Error:  true,
					Result: result,
				})
			}
		}()
	}
	return nil
}
```

Part 2. PLUM Node와의 통신 실습

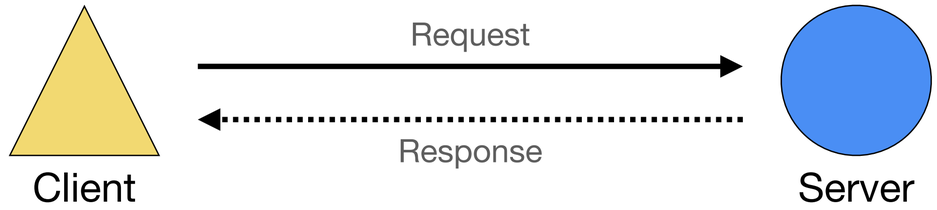
2. 실습 개요

Proto file을 컴파일하기

```
$ protoc --go_out=. --go_opt=paths=source_relative \
    --go-grpc_out=. --go-grpc_opt=paths=source_relative \
    plum/plum.proto
```

2. 실습 개요

노드의 상태를 한 번 조회하기: getPeerState( )

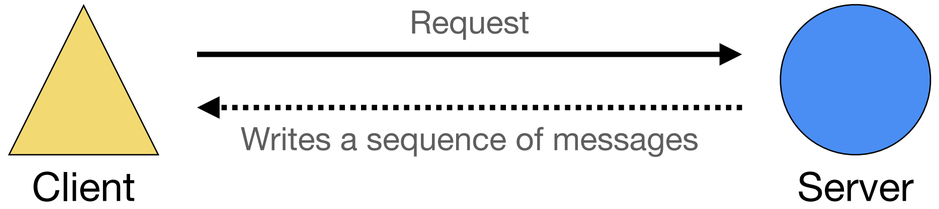


클라이언트 코드

```
func handleGetPeerState(client plum.FarmerClient) {

  ctx, cancel := context.WithTimeout(context.Background(), time.Second*10)
  defer cancel()

  r, err := client.GetPeerState(ctx, &plum.Empty{})
  if err != nil {
    log.Printf("could not get the state of peer: %v", err)
  }
  log.Println(formatPeerState(r))
}
```

2. 실습 개요

노드의 상태를 몇 초간 조회하기: getPeerStateStream( )



클라이언트 코드

```
func handleGetPeerStateStream(client plum.FarmerClient) {
  ctx, cancel := context.WithTimeout(context.Background(), time.Second*10)
  defer cancel()

  stream, err := client.GetPeerStateStream(ctx, &plum.Empty{})
  if err != nil {
    log.Fatalf("could not get peer state by streaming: %v", err)
  }

  for {
    ps, err := stream.Recv()
    if err == io.EOF {
      break
    }
    if err != nil {
      log.Fatalf("colud not get peer state by streaming: %v", err)
    }
    log.Println(formatPeerState(ps))
  }
}
```

감사합니다

QnA