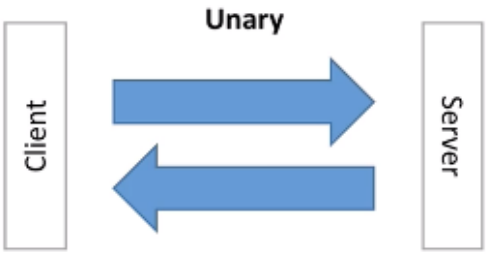
4 - [Hands-On] gRPC Unary

What's a Unary API?

- Unary RPC calls are the basic Request/Response that everyone is familiar with
- The client will send one message to the server and will receive one response from the server
- Unary RPC calls will be the most common for your APIs.
    - Unary calls are very well suited when your data is small
    - Start with Unary when writing APIs and use streaming API if performance is an issue



- In gRPC Unary Calls are defined using Protocol Buffers
- For each RPC call we have to define a "Request" message and a "Response" message.

Greet API Definition

- Hands On: Let's define a Unary "Greet" API.
- Our message is Greeting and contains first_name & last_name string field
- It will take a GreetRequest that contains a Greeting
- It will return a GreetResponse that contains a result string

/src/greet/greetpb/greet.proto

```
syntax = "proto3";

package greet;
option go_package = "greetpb";

message Greeting {
    string first_name = 1;
    string last_name = 2;
}

```

```
message GreetRequest {
    Greeting greeting = 1;
}

message GreetResponse {
    string result = 1;
}

service GreetService {
    // Unary
    rpc Greet(GreetRequest) returns (GreetResponse) {};
}
```

Unary API Server Implementation

- Hands-on:
- We'll implement a Unary Greet RPC
- We'll hook our new GreetService to our Server
- We'll start our Server

/src/greet/greet_server/server.go

```
package main

import (
    "context"
    "fmt"
    "log"
    "net"

    "github.com/gRPC-go-microservices/src/greet/greetpb"
    "google.golang.org/grpc"
)

type server struct{}

func (*server) Greet(ctx context.Context, req *greetpb.GreetRequest)
(*greetpb.GreetResponse, error) {
    // extract information from request
    firstName := req.GetGreeting().GetFirstName()
    // form a response
    result := "Hello " + firstName
    res := &greetpb.GreetResponse{
        Result: result,
    }
    return res, nil
}

func main() {
    fmt.Println("Hello, I'm a server")

    // listener
    // 50051 is the default port number for gRPC
```

```
    lis, err := net.Listen("tcp", "0.0.0.0:50051")
    if err != nil {
        log.Fatalf("Failed to listen: %v", err)
    }

    // create grpc server
    s := grpc.NewServer()
    greetpb.RegisterGreetServiceServer(s, &server{})

    // bind the server to the port
    if err := s.Serve(lis); err != nil {
        log.Fatalf("failed to serve: %v", err)
    }
}
```

Unary API Client Implementation

- Hands-on:
- We'll implement a client call for our Unary RPC
- We'll test it against our server that is running!

/src/greet/greet_client/client.go

```
package main

import (
    "context"
    "fmt"
    "log"

    "github.com/gRPC-go-microservices/src/greet/greetpb"
    "google.golang.org/grpc"
)

func main() {
    fmt.Println("Hello I'm a client")

    // by default gRPC has SSL, for now, without this
    conn, err := grpc.Dial("localhost:50051", grpc.WithInsecure())
    if err != nil {
        log.Fatalf("could not connect: %v", err)
    }

    // defer means defer this statement at the very end of this function
    defer conn.Close()

    // create a new client
    c := greetpb.NewGreetServiceClient(conn)
    doUnary(c)
}

func doUnary(c greetpb.GreetServiceClient) {
    req := &greetpb.GreetRequest{
        Greeting: &greetpb.Greeting{
```

```
            FirstName: "Jieqiong",
            LastName:  "Yu",
        },
    }
    res, err := c.Greet(context.Background(), req)

    if err != nil {
        log.Fatalf("error while calling Greet RPC: %v", err)
    }

    log.Printf("Response from Greet: %v", res.Result)
}
```

[Solution] Sum API

/src/calculator