gRPC and Protocol Buffers

CapitalGo 2018

JUNE 22, 2018
Arlington, Virginia

Presenter: Doug Fort

Decipher Technology Studios

doug.fort@deciphernow.com
@dougfort on twitter

sample code at: https://github.com/dougfort/mystore
slides at: https://github.com/dougfort/grpc-talk-2018

Code Generation

Uber's Prototool:
Your Swiss Army Knife for Protocol Buffers
https://github.com/uber/prototool

Decipher Fabric
https://github.com/DecipherNow/gm-fabric-go

HTTP/2 and gRPC

https://blog.netsil.com/http-2-and-grpc-the-next-generation-of-microservices-interactions-aff4ffa6faed

HTTP2 introduces fully asynchronous, multiplexing of requests by introducing concept of streams. Client and servers can both initiate multiple streams on a single underlying TCP connection.

gRPC Server

```go
func main() {
    const serverAddress = "localhost:1111"

    var listener net.Listener
    var grpcServer *grpc.Server
    var err error

    if listener, err = net.Listen("tcp", serverAddress); err != nil {
        log.Fatalf("net.Listen failled: %v", err)
    }

    grpcServer = grpc.NewServer()

    log.Printf("Server starts: listening on %s", serverAddress)
    if err := grpcServer.Serve(listener); err != nil {
        log.Printf("grpcServer.Serve ended with %s", err)
    }
}
```

Options

Dave Cheney

The acme of foolishness

RESOURCES FOR NEW GO PROGRAMMERS UNOFFICIAL ARM TARBALLS FOR GO TRAINING ABO

Functional options for friendly APIs

What follows is the text of my presentation, Functional options for friendly APIs that I gave at dotGo this year. It has been edited slightly for readability.

https://dave.cheney.net/2014/10/17/functional-options-for-friendly-apis

Server options

```
func NewServer(opt ...ServerOption) *Server
```

https://godoc.org/google.golang.org/grpc

```
func StatsHandler(h stats.Handler) ServerOption
func StreamInterceptor(i StreamServerInterceptor) ServerOption
func UnaryInterceptor(i UnaryServerInterceptor) ServerOption
```

Store Example

Download catalog

Order item

select random item

The Catalog

```go
// catalogEntry describes a single item from an imaginary store
type catalogEntry struct {
    itemID      string
    description string
    price       int
    available   int
}

var catalog = []catalogEntry{
    catalogEntry{
        itemID:      "1001",
        description: "abacus",
        price:       1,
        available:   1000000000,
    },

    // ...

    catalogEntry{
        itemID:      "1008",
        description: "Clever Hans",
        price:       8,
        available:   1000000000,
    }
```

Protocol Buffer Definition

```
tree protobuf/
protobuf/
└── mystore.proto

0 directories, 1 file
```

```
import (
    pb "github.com/dougfort/mystore/02/protobuf"
)
```

gRPC Service Definition

https://grpc.io/docs/guides/concepts.html

```
service Store {
    // CatalogStream returns a stream of catalog entries
    rpc CatalogStream(CatalogStreamRequest) returns (stream CatalogResponse) {}

    // OrderItem orders a specified item from the catalog
    rpc OrderItem(OrderItemRequest) returns (OrderResponse) {}
}
```

CatalogStream messages

```
//  rpc CatalogStream(CatalogStreamRequest) returns (stream CatalogResponse) {}


message CatalogStreamRequest {}

message CatalogResponse {
    string itemID = 1;
    string description = 2;
    int32 price = 3;
    int32 available = 4;
}
```

OrderItem messages

```
//  rpc OrderItem(OrderItemRequest) returns (OrderResponse) {}


message OrderItemRequest {
    string itemID = 1;
    string customerName = 2;
}

message OrderResponse {
    bool successful = 1;
}
```

Protocol Buffers Compiler

https://github.com/google/protobuf

```
dougfort@pop-os:~$ protoc --version
libprotoc 3.0.0

sudo apt install protobuf-compiler
```

Generating go service code

```
#!/bin/bash

set -euxo pipefail

MYSTOREPATH="$GOPATH/src/github.com/dougfort/mystore/02"

protoc --proto_path=$MYSTOREPATH/protobuf \
    --plugin=$GOPATH/bin/protoc-gen-go \
    --go_out=plugins=grpc:$MYSTOREPATH/protobuf \
    $MYSTOREPATH/protobuf/mystore.proto
```

Generating go service code

```
./go_out.sh
+ MYSTOREPATH=/home/dougfort/go/src/github.com/dougfort/mystore/02
+ protoc --proto_path=/home/dougfort/go/src/github.com/dougfort/mystore/02/protobuf --plugin=/home/dougfort/go/bin/protoc-gen-go --
go_out=plugins=grpc:/home/dougfort/go/src/github.com/dougfort/mystore/02/protobuf /home/dougfort/go/src/github.com/dougfort/mystore/02/protobuf/mystore.proto
```

```
tree protobuf/
protobuf/
├── mystore.pb.go
└── mystore.proto

0 directories, 2 files
```

The generated Server interface

```go
// StoreServer is the server API for Store service.
type StoreServer interface {
    // CatalogStream returns a stream of catalog entries
    CatalogStream(*CatalogStreamRequest, Store_CatalogStreamServer) error
    // OrderItem orders a specified item from the catalog
    OrderItem(context.Context, *OrderItemRequest) (*OrderResponse, error)
}
```

Server State object

```go
type serverState struct {
    sync.Mutex
    catalog map[string]catalogEntry
}

// NewMyStoreServer returns an object that implements the pb.StoreServer interface
func NewMyStoreServer() pb.StoreServer {
    var s serverState
    s.catalog = make(map[string]catalogEntry)
    for _, entry := range catalog {
        s.catalog[entry.itemID] = entry
    }
    return &s
}
```

Registering the server with gRPC

```go
package main

    // ...

    grpcServer = grpc.NewServer()

    pb.RegisterStoreServer(grpcServer, NewMyStoreServer())

    log.Printf("Server starts: listening on %s", serverAddress)
    if err := grpcServer.Serve(listener); err != nil {
        log.Printf("grpcServer.Serve ended with %s", err)
    }
}
```

Stub Methods

```go
// CatalogStream returns a stream of catalog entries
func (s *serverState) CatalogStream(
    request *pb.CatalogStreamRequest,
    stream pb.Store_CatalogStreamServer,
) error {
    return nil
}

// OrderItem orders a specified item from the catalog
func (s *serverState) OrderItem(
    ctx context.Context,
    request *pb.OrderItemRequest,
) (*pb.OrderResponse, error) {
    var response pb.OrderResponse

    return &response, nil
}
```

Server streaming to client

```go
    // CatalogStream returns a stream of catalog entries
func (s *serverState) CatalogStream(
    request *pb.CatalogStreamRequest,
    stream pb.Store_CatalogStreamServer,
) error {
    log.Printf("CatalogStream starts")

    s.Lock()
    defer s.Unlock()

    for key, entry := range s.catalog {
        if err := stream.Send(&pb.CatalogResponse{
            ItemID:      entry.itemID,
            Description: entry.description,
            Price:       int32(entry.price),
            Available:   int32(entry.available),
        }); err != nil {
            return errors.Wrapf(err, "stream.Send %s", key)
        }
    }

    log.Printf("CatalogStream returns normally")
    return nil
}
```

Client Options

```
if conn, err = grpc.Dial(serverAddress, grpc.WithInsecure()); err != nil {
    log.Fatalf("grpc.Dial(%s) failed: %v", serverAddress, err)
}
```

https://godoc.org/google.golang.org/grpc

```
func Dial(target string, opts ...DialOption) (*ClientConn, error)
```

Client Method stubs

```go
type storeClientState struct {
    pb.StoreClient
}

func (s storeClientState) getCatalog() ([]pb.CatalogResponse, error) {
    var catalog []pb.CatalogResponse

    return catalog, nil
}

func (s storeClientState) orderItem(userName string, itemID string) error {
    return nil
}
```

Client Streaming from Server

```
    if psc, err = s.CatalogStream(ctx, &pb.CatalogStreamRequest{}); err != nil {
        return nil, errors.Wrap(err, "client.CatalogStream")
    }

CATALOG_LOOP:
    for {
        var cr *pb.CatalogResponse

        if cr, err = psc.Recv(); err != nil {
            if err == io.EOF {
                break CATALOG_LOOP
            }
            return nil, errors.Wrap(err, "psc.Recv()")
        }

        catalog = append(catalog, *cr)
    }
```

Run the server and client

```
dougs-mbp:02 dougfort$ mystore-server
2018/06/17 16:19:36 Server starts: listening on localhost:1111
```

```
dougs-mbp:02 dougfort$ mystore-client
2018/06/17 16:19:54 found 8 catalog entries
2018/06/17 16:19:54 ordering (1002) pile of rocks at $2; 1000000000 available
2018/06/17 16:19:54 order successful
```

```
2018/06/17 16:19:54 CatalogStream starts
2018/06/17 16:19:54 CatalogStream returns normally
2018/06/17 16:19:54 OrderItem: 1002
2018/06/17 16:19:54 OrderItem: 1002 successful
```

HTTP Gateway Proxy

```
syntax = "proto3";

package protobuf;

import "google/api/annotations.proto";

// Interface exported by the server.
service Store {
    // CatalogStream returns a stream of catalog entries
    rpc CatalogStream(CatalogStreamRequest) returns (stream CatalogResponse) {
        option (google.api.http) = {
            get: "/acme/services/catalog"
        };
    }

    // OrderItem orders a specified item from the catalog
    rpc OrderItem(OrderItemRequest) returns (OrderResponse) {
        option (google.api.http) = {
            post: "/acme/services/order/{itemID}/{customerName}"
        };
    }
}
```

Generating Gateway Code

```
#!/bin/bash

set -euxo pipefail

MYSTOREPATH="$GOPATH/src/github.com/dougfort/mystore/03"

protoc -I $GOPATH/src/github.com/grpc-ecosystem/grpc-gateway/third_party/googleapis \
    --proto_path=$MYSTOREPATH/protobuf \
    --plugin=$GOPATH/bin/protoc-gen-grpc-gateway \
    --grpc-gateway_out=logtostderr=true:$MYSTOREPATH/protobuf \
    $MYSTOREPATH/protobuf/mystore.proto
```

Generating Gateway Code

```
tree protobuf/
protobuf/
├── mystore.pb.go
├── mystore.pb.gw.go
└── mystore.proto

0 directories, 3 files
```

Gateway Proxy Server

```go
mux := runtime.NewServeMux()
ctx := context.Background()

log.Printf("Proxying to %s", serverAddress)
err = pb.RegisterStoreHandlerFromEndpoint(
    ctx,
    mux,
    serverAddress,
    []grpc.DialOption{grpc.WithInsecure()},
)
if err != nil {
    log.Fatalf("RegisterStoreHandlerFromEndpoint failed: %s", err)
}

log.Printf("listening to %s", proxyAddress)
if listener, err = net.Listen("tcp", proxyAddress); err != nil {
    log.Fatalf("net.Listen failed: %s", err)
}

http.Serve(listener, mux)
```

HTTP Get

```
dougs-mbp:~ dougfort$ curl localhost:8080/acme/services/catalog
{"result":{"itemID":"1003","description":"sliderule","price":3,"available":1000000000}}
{"result":{"itemID":"1004","description":"Curta","price":4,"available":1000000000}}
{"result":{"itemID":"1005","description":"IBM 360","price":1000,"available":1000000000}}
{"result":{"itemID":"1006","description":"Qubit","price":6,"available":1000000000}}
{"result":{"itemID":"1007","description":"human being","price":7,"available":1000000000}}
{"result":{"itemID":"1008","description":"Clever Hans","price":8,"available":1000000000}}
{"result":{"itemID":"1001","description":"abacus","price":1,"available":1000000000}}
{"result":{"itemID":"1002","description":"pile of rocks","price":2,"available":1000000000}}
```

HTTP Post

```
dougs-mbp:~ dougfort$ curl -X POST localhost:8080/acme/services/order/1004/dougfort
{"successful":true}
```