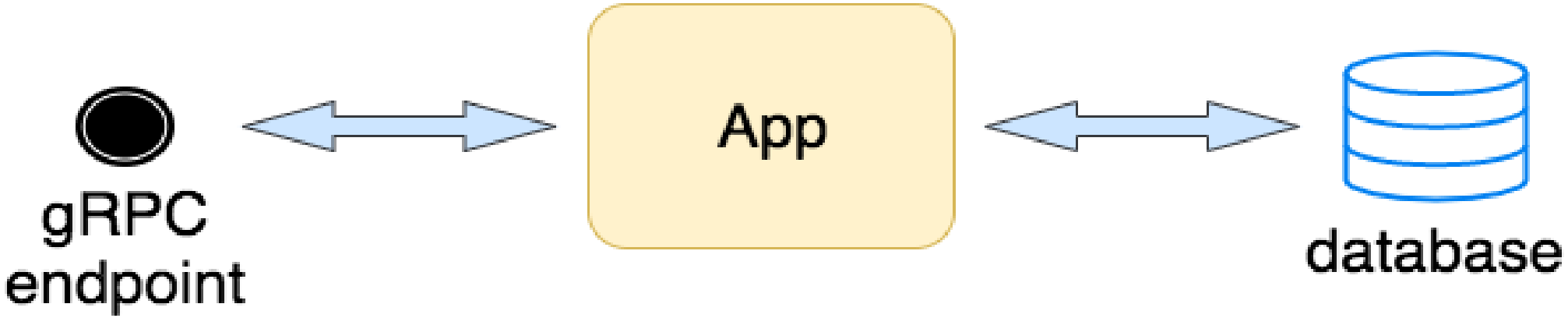
protoc-gen-struct-transformer

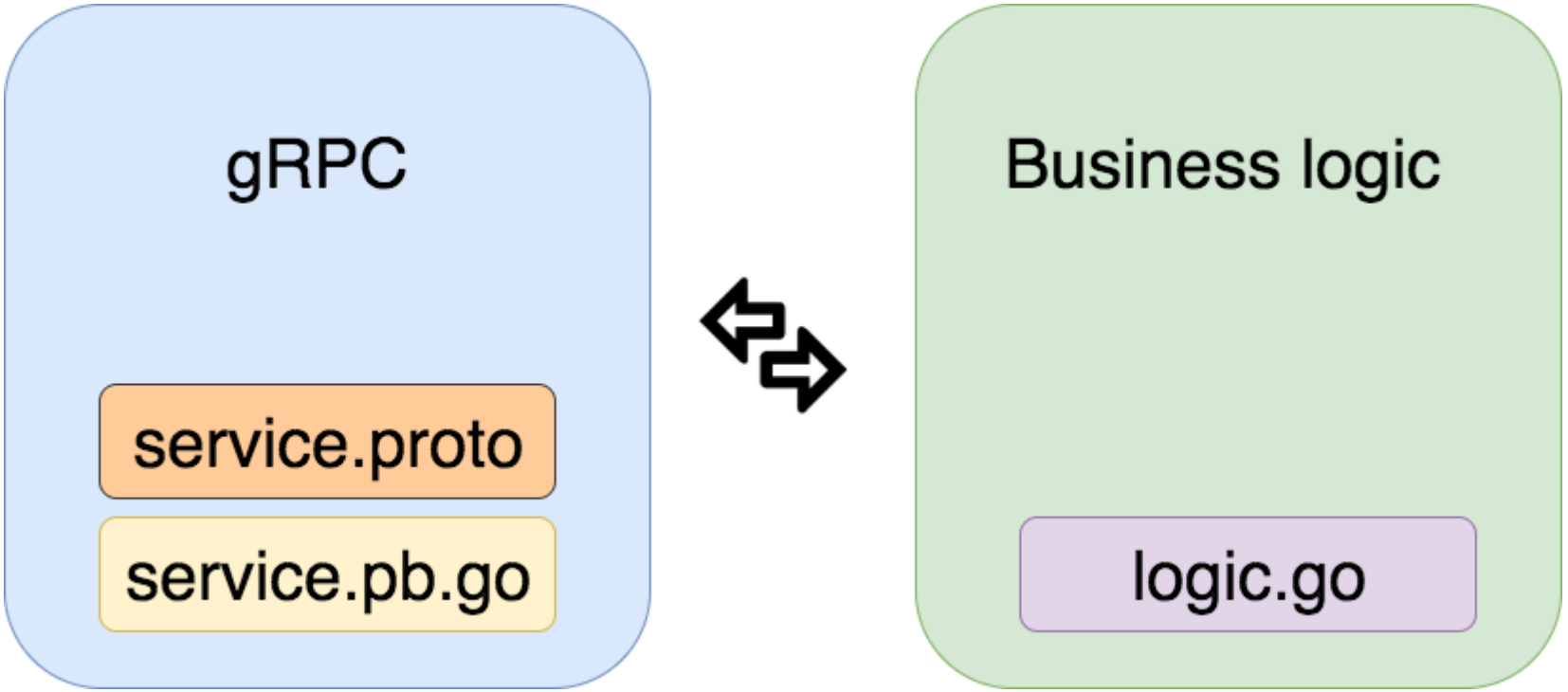
Bridge between auto-generated gRPC structures and structures in your code.

Evgeny Khabarov
Bold Commerce, Winnipeg, Canada

Example app



Two parts of the app



gRPC part

```
// service.proto                                  |    // service.pb.go
option go_package = "pb";                         |    package pb
                                                  |
message Order {                                   |    type Order struct {
  int64 id = 1;                                   |      Id     int64  `protobuf:...`
  string number = 2;                              |      Number string `protobuf:...`
  string status = 3;                              |      Status string `protobuf:...`
}                                                 |    }
                                                  |
message Request {                                 |    type Request struct {
  int64 id = 1;                                   |      Id int64 `protobuf:...`
}                                                 |    }
                                                  |
message Response {                                |    type Response struct {
  Order order = 1;                                |      Order *Order `protobuf:...`
}                                                 |    }
                                                  |
service Order{                                    |    type OrderServer interface {
  rpc Get (Request) returns (Response);           |      Get(context.Context, Request) (*Response, error)
}                                                 |    }
```

```
protoc --go_out=plugin=grpc:. ./service.proto
```

logic.go

```
package business_logic

type MyOrder struct {
    ID     int // Order message from service.pb.go has int64 type
    Number string
    Status string
}

type OrderService interface {
    GetOrder(ctx context.Context, id int) (MyOrder, error)
}
```

Link between the parts

```
package grpc_implementation

type server struct {
    svc OrderService
}

// implementation of OrderServer interface from service.pb.go

func (s *server) Get(ctx context.Context, req pb.Request) (*pb.Response, error) {
    o, err := s.svc.GetOrder(ctx, int(req.Id)) // "o" will have a type of business_logic.MyOrder
    if err != nil {
        return nil, err
    }

    return &pb.Response{
        Order: &pb.Order{ // "Order" here is of type *pb.Order
            Id:     int64(o.ID),
            Number: o.Number,
            Status: o.Status,
        },
    }
}
```

Functions

```
package transform

func MyOrderToOrder(mo business_logic.MyOrder) pb.Order {
    return pb.Order{
        Id:     int64(mo.ID),
        Number: mo.Number,
        Status: mo.Status,
    }
}

func OrderToMyOrder(o pb.Order) business_logic.MyOrder {
    return business_logic.MyOrder{
        ID:     int(o.Id),
        Number: o.Number,
        Status: o.Status,
    }
}
```

But why we need it?

- pb.Order != business_logic.MyOrder
- Transforming one structure into another has to be done **field by field**, unless both structures have **identical** underlying types
- Sometimes we have ***pb.Order** or **[]pb.Order** or even **[]*pb.Order**, rather than just **pb.Order**

What if...

```
type MyRealOrder struct {
    ID                int        `json:"id"`
    ShopId            string     `json:"shop_number"`
    Subtotal          float64    `json:"subtotal"`
    SubtotalTax       float64    `json:"subtotal_tax"`
    ShippingSubtotal  float64    `json:"shipping_subtotal"`
    ShippingTax       float64    `json:"shipping_tax"`
    Total             float64    `json:"total"`
    TotalTax          float64    `json:"total_tax"`
    PaymentMethod     string     `json:"payment_method"`
    ShippingMethod    string     `json:"shipping_method"`
    OrderNumber       int        `json:"order_number"`
    Discount          float64    `json:"discount"`
    Notes             string     `json:"notes"`
    CreatedAt         time.Time  `json:"created_at"`
    UpdatedAt         *time.Time `json:"updated_at"`
    PlacedAt          time.Time  `json:"placed_at"`
    DeletedAt         *time.Time `json:"deleted_at"`
    Items             []Item     `json:"order_items"`
    BillingAddress    Address    `json:"billing_address"`
    ShippingAddresses []Address  `json:"shipping_addresses"`
}
```

Who wants to write transformations for such structures manually?

Better way

One more plugin: step #1

```
option go_package = "pb";

import "options/annotations.proto"; // adds an ability to use options

option (transformer.go_repo_package) = "logic"; // package with business_logic structures
option (transformer.go_protobuf_package) = "pb"; // package with protobuf structures
option (transformer.go_models_file_path) = "business_logic.go"; // path to business_logic source

message Order {
  option (transformer.go_struct) = "MyOrder"; // structure name in business_logic package

  int64 id = 1;
  string number = 2;
  string status = 3;
}
message Request {
  int64 id = 1;
}
message Response {
  Order order = 1;
}
service Order{
  rpc Get (Request) returns (Response);
}
```

One more plugin: step #2

```
protoc \
  --go_out=Moptions/annotations.proto=path/to/protoc-gen-struct-transformer/options,plugin=grpc:. \
  --struct-transformer_out=package=transform:. \
  ./service.proto
```

Result

```
package transform

// pb.Order => business_logic.MyOrder
func PbToMyOrderPtr(src *pb.Order, opts ...TransformParam) *business_logic.MyOrder {...}
func PbToMyOrderPtrList(src []*pb.Order, opts ...TransformParam) []*business_logic.MyOrder {...}
func PbToMyOrderPtrVal(src *pb.Order, opts ...TransformParam) business_logic.MyOrder {...}
func PbToMyOrderPtrValList(src []*pb.Order, opts ...TransformParam) []business_logic.MyOrder {...}
func PbToMyOrderList(src []*pb.Order, opts ...TransformParam) []business_logic.MyOrder {...}
func PbToMyOrder(src pb.Order, opts ...TransformParam) business_logic.MyOrder {...}
func PbToMyOrderValPtr(src pb.Order, opts ...TransformParam) *business_logic.MyOrder {...}
func PbToMyOrderValList(src []pb.Order, opts ...TransformParam) []business_logic.MyOrder {...}

// business_logic.MyOrder => pb.Order
func MyOrderToPbPtr(src *business_logic.MyOrder, opts ...TransformParam) *pb.Order {...}
func MyOrderToPbPtrList(src []*business_logic.MyOrder, opts ...TransformParam) []*pb.Order {...}
func MyOrderToPbPtrVal(src *business_logic.MyOrder, opts ...TransformParam) pb.Order {...}
func MyOrderToPbValPtrList(src []business_logic.MyOrder, opts ...TransformParam) []*pb.Order {...}
func MyOrderToPbList(src []business_logic.MyOrder, opts ...TransformParam) []*pb.Order {...}
func MyOrderToPb(src business_logic.MyOrder, opts ...TransformParam) pb.Order {...}
func MyOrderToPbValPtr(src business_logic.MyOrder, opts ...TransformParam) *pb.Order {...}
func MyOrderToPbValList(src []business_logic.MyOrder, opts ...TransformParam) []pb.Order {...}
```

One more plugin: step #3

```
package grpc_implementation

type server struct {
    svc OrderService
}

// implementation of OrderServer interface from service.pb.go

func (s *server) Get(ctx context.Context, req pb.Request) (*pb.Response, error) {
    o, err := s.svc.GetOrder(ctx, req.Id) // "o" will have a type of business_logic.MyOrder
    if err != nil {
        return nil, err
    }

    return &pb.Response{                                |
        Order: transform.MyOrderToPbValPtr(o),          |  Order: &pb.Order{
                                                        |    Id:     int64(o.ID),
                                                        |    Number: o.Number,
                                                        |    Status: o.Status,
    }
}
```

Questions?

Try it!

github.com/bold-commerce/protoc-gen-struct-transformer (https://github.com/bold-commerce/protoc-gen-struct-transformer)

Feel free to open a PR

Thank you

Evgeny Khabarov
Bold Commerce, Winnipeg, Canada
@eekhabarov (http://twitter.com/eekhabarov)