

```
package main
import (
    "flag"
    "fmt"
    "gomail/pkg/config"
    "gomail/pkg/db"
    "gomail/pkg/imap"
    "gomail/pkg/mailbox"
    "gomail/pkg/mailbox/auth"
    "gomail/pkg/proto"
    "gomail/pkg/smtp"
    "google.golang.org/grpc"
    "log"
    "net"
    "os"
    "os/signal"
    "syscall"
)
var configFile string
func init() {
    flag.StringVar(&configFile, "config", "config.yaml", "path for config file")
}
func main() {
    flag.Parse()
    mailConfig := config.Load(configFile)
    mongo, err := db.New(mailConfig.Mongo)
    if err != nil {
        log.Fatal(err)
    }
    interceptor := auth.NewAuthInterceptor(mongo, mongo)
    s := mailbox.NewGRPCServer(grpc.StreamInterceptor(interceptor.StreamAuth),
        grpc.UnaryInterceptor(interceptor.UnaryAuth))
    smtpClient := smtp.NewClient(mailConfig.Smtp)
    postman := imap.NewPostMan(mailConfig.Imap.MailServers)
    postman.Start()
    mb := mailbox.NewMailBoxService(postman, smtpClient, mongo, mongo)
    proto.RegisterMailBoxServer(s, mb)
    lis, err := net.Listen("tcp", fmt.Sprintf(":%d", mailConfig.Port))
    if err != nil {
        log.Fatalf("failed to listen: %v", err)
    }
    go func() {
        err := s.Serve(lis)
        if err != nil {
            panic(err)
        }
    }()
    log.Println("server start !")
    sigs := make(chan os.Signal, 2)
    signal.Notify(sigs, os.Interrupt, syscall.SIGTERM)
```

```

    select {
    case <-sigs:
        s.GracefulStop()
        postman.Close()
    }
}
package config
import (
    "gopkg.in/yaml.v3"
    "log"
    "os"
)
func Load(path string) (config Config) {
    data, err := os.ReadFile(path)
    if err != nil {
        log.Fatal(err)
    }
    err = yaml.Unmarshal(data, &config)
    if err != nil {
        log.Fatal(err)
    }
    return
}
package config
import "time"
type Mongo struct {
    Url      string `yaml:"url"`
    Db       string `yaml:"db"`
    User     string `yaml:"user"`
    Password string `yaml:"password"`
    GridPrefix string `yaml:"grid_prefix"`
    Collection string `yaml:"collection"`
}
type Smtplib struct {
    User      string `yaml:"user"`
    Password  string `yaml:"pwd"`
    Host      string `yaml:"host"`
    Port      string `yaml:"port"`
}
type Imap struct {
    MailServers []MailServer `yaml:"mailServers"`
    Network     string        `yaml:"network"`
    Timeout     time.Duration `yaml:"timeout"`
}
type MailServer struct {
    Host      string `yaml:"host"`
    Port      string `yaml:"port"`
    Auth      Auth    `yaml:"auth"`
    Name      string `yaml:"name"`
    Timeout   time.Duration `yaml:"timeout"`
}

```

```

    FlushTime time.Duration `yaml:"flush_time"`
}
type Auth struct {
    User      string `yaml:"user"`
    Password  string `yaml:"pwd"`
}
type Config struct {
    Smtplib Smtplib `yaml:"smtp"`
    Imaplib Imaplib `yaml:"imap"`
    Name    string `yaml:"name"`
    Host    string `yaml:"host"`
    Port    int    `yaml:"port"`
    Mongo   Mongo `yaml:"mongo"`
}
package db
import (
    "errors"
    "gomail/pkg/config"
    "gopkg.in/mgo.v2"
    "gopkg.in/mgo.v2/bson"
    "io"
)
type Client struct {
    DB *mgo.Database
    gridPrefix string
    collection string
}
type WrapObject struct {
    Id interface{} "_id"
    Obj interface{} "Obj"
}
func (client *Client) Upload(filename string, contentType string, stream io.ReadCloser) (string, error) {
    defer func() { _ = stream.Close() }()
    gridFS := client.DB.GridFS(client.gridPrefix)
    file, err := gridFS.Create(filename)
    if err != nil {
        return "", err
    }
    defer func() { _ = file.Close() }()
    file.SetContentType(contentType)
    _, err = io.Copy(file, stream)
    if err != nil {
        return "", err
    }
    id := file.Id().(bson.ObjectId).Hex()
    return id, nil
}
func (client *Client) Download(id string) (File, error) {
    mongoId := bson.ObjectIdHex(id)

```

```

    if !mongoId.Valid() {
        return nil, errors.New("invalid file id")
    }
    gridFS := client.DB.GridFS(client.gridPrefix)
    file, err := gridFS.OpenId(mongoId)
    return file, err
}

func (client *Client) Set(obj interface{}) (string, error) {
    err := client.DB.C(client.collection).Insert(obj)
    return "", err
}

func (client *Client) Get(conditions map[string]interface{}, result interface{}) error {
    return client.DB.C(client.collection).Find(bson.M(conditions)).One(result)
}

func (client *Client) Exist(condition map[string]interface{}) bool {
    n, err := client.DB.C(client.collection).Find(bson.M(condition)).Count()
    if err != nil {
        return false
    }
    return n > 0
}

func (client *Client) Close() {
    client.DB.Session.Close()
}

func New(mongoConfig config.Mongo) (*Client, error) {
    session, err := mgo.Dial(mongoConfig.Url)
    if err != nil {
        return nil, err
    }
    db := session.DB(mongoConfig.Db)
    if mongoConfig.User != "" {
        err = db.Login(mongoConfig.User, mongoConfig.Password)
        if err != nil {
            return nil, err
        }
    }
    client := &Client{
        DB:      db,
        gridPrefix: mongoConfig.GridPrefix,
        collection: mongoConfig.Collection,
    }
    return client, nil
}

package db
import (
    "io"
    "time"
)

type Storage interface {
    Upload(filename string, contentType string, stream io.ReadCloser) (id string, err error)
}

```

```

        Download(id string) (File, error)
    }
    type Session interface {
        Set(obj interface{}) (string, error)
        Get(condition map[string]interface{}, result interface{}) error
        Exist(condition map[string]interface{}) bool
    }
    type File interface {
        io.ReadSeekCloser
        ContentType() string
        Name() string
        MD5() (md5 string)
        UploadDate() time.Time
    }
    package imap
    import (
        "github.com/axgle/mahonia"
        "github.com/emersion/go-imap"
        "github.com/emersion/go-imap/client"
        "github.com/emersion/go-message"
        "gmail/pkg/config"
        "gmail/pkg/proto"
        "gmail/pkg/util/sortlist"
        "io"
        "log"
        "net"
        "strings"
        "sync"
        "time"
    )
    func init() {
        message.CharsetReader = func(charset string, input io.Reader) (reader io.Reader, error) {
            if strings.ToLower(charset) == "gb2312" {
                charset = "GB18030"
            }
            decoder := mahonia.NewDecoder(charset)
            if decoder != nil {
                reader = decoder.NewReader(input)
            } else {
                reader = input
            }
            return
        }
    }
    type Watcher interface {
        Subscribe(serverName, id string, weight int32, ch chan *proto.Mail) (*Subscriber, error)
        UnSubscribe(*Subscriber)
        Start()
    }

```

```

    Close()
    ListServer() []string
}
type Subscriber struct {
    serverName string
    Channel     chan *proto.Mail
    Weight      int32
    ID          string
}
func SubscriberCompare(a, b *Subscriber) int {
    if a.Weight == b.Weight {
        return 0
    }
    if a.Weight > b.Weight {
        return 1
    }
    return -1
}
type Client struct {
    flushTime      time.Duration
    subscriberLimit int
    Host           string
    Port           string
    lock           sync.Mutex
    subscribers     sortlist.SortedList[*Subscriber]
    User           string
    Password       string
    Done           chan error
    mailBox        *client.Client
}
func (cli *Client) Fetch() (chan *imap.Message, *imap.SeqSet) {
    if err := cli.mailBox.Noop(); err != nil {
        cli.Done <- err
        return nil, nil
    }
    seqSet := &imap.SeqSet{}
    ch := make(chan *imap.Message, 100)
    seqids, err := cli.SearchUnseen()
    if err != nil {
        log.Println(cli.User, " fetch unsee error: ", err)
        cli.Done <- err
        close(ch)
        return ch, nil
    }
    if len(seqids) == 0 {
        log.Println(cli.User, " 没有邮件")
        close(ch)
        return ch, nil
    }
    seqSet.AddNum(seqids...)

```

```

    go func() {
        err := cli.mailBox.Fetch(seqSet, []imap.FetchItem{imap.FetchBody + "[]", imap.FetchFlags, imap.FetchUid}, ch)
        if err != nil {
            cli.Done <- err
        }
    }()
    return ch, seqSet
}

func (cli *Client) SearchUnseen() (ids []uint32, err error) {
    criteria := imap.NewSearchCriteria()
    criteria.WithoutFlags = []string{imap.SeenFlag}
    ids, err = cli.mailBox.Search(criteria)
    return
}

func (cli *Client) See(seqSet *imap.SeqSet) {
    cli.Done <- cli.mailBox.Store(seqSet, imap.AddFlags, []interface{}{imap.SeenFlag}, nil)
}

func (cli *Client) addSubscriber(sub *Subscriber) bool {
    cli.lock.Lock()
    defer cli.lock.Unlock()
    if cli.subscribers.Size() >= cli.subscriberLimit {
        return false
    }
    cli.subscribers.Push(sub)
    return true
}

func (cli *Client) unSubscribe(subscriber *Subscriber) {
    cli.lock.Lock()
    cli.subscribers.DeleteItem(subscriber)
    cli.lock.Unlock()
}

func (cli *Client) Login() (err error) {
    err = cli.mailBox.Login(cli.User, cli.Password)
    if err != nil {
        _, _ = cli.mailBox.Select("INBOX", false)
    }
    return
}

func (cli *Client) Reconnect() (err error) {
    cli.mailBox, err = client.DialTLS(net.JoinHostPort(cli.Host, cli.Port), nil)
    if err != nil {
        return
    }
    err = cli.mailBox.Login(cli.User, cli.Password)
    _, _ = cli.mailBox.Select("INBOX", false)
    return
}

func (cli *Client) Close() {
    cli.lock.Lock()

```

```

    _ = cli.mailBox.Close()
    cli.lock.Unlock()
}
func New(imapConfig config.MailServer) (instance *Client, err error) {
    remote := net.JoinHostPort(imapConfig.Host, imapConfig.Port)
    imapClient, err := client.DialTLS(remote, nil)
    if err != nil {
        return
    }
    imapClient.Timeout = imapConfig.Timeout * time.Second
    instance = &Client{
        flushTime:      imapConfig.FlushTime,
        subscriberLimit: 50,
        mailBox:         imapClient,
        Host:            imapConfig.Host,
        Port:            imapConfig.Port,
        User:            imapConfig.Auth.User,
        Password:        imapConfig.Auth.Password,
        Done:            make(chan error, 10),
        subscribers:     sortlist.NewSortedList[*Subscriber](SubscriberCompare, 0),
    }
    err = instance.Login()
    _, _ = instance.mailBox.Select("INBOX", false)
    return
}
package imap
import (
    "errors"
    "github.com/emersion/go-imap"
    "github.com/emersion/go-message/mail"
    "gomail/pkg/config"
    "gomail/pkg/proto"
    "io"
    "log"
    "sync"
    "time"
)
// Postman alive check, subscribe restart client
type Postman struct {
    mailPool map[string]*Client
    lock     *sync.Mutex
}
func (postman *Postman) Subscribe(serverName, id string, weight int32, ch chan *proto.Ma
il) (*Subscriber, error) {
    chooseBox, ok := postman.mailPool[serverName]
    if !ok {
        return nil, errors.New("server is invalid")
    }
    sub := &Subscriber{
        Weight:    weight,

```

```

        ID:      id,
        Channel: ch,
        serverName: serverName,
    }
    if !chooseBox.addSubscriber(sub) {
        return nil, errors.New("up to the max subscribe client")
    }
    log.Println(serverName + " subscribe successfully")
    return sub, nil
}

func (postman *Postman) UnSubscribe(sub *Subscriber) {
    chooseBox, ok := postman.mailPool[sub.serverName]
    if !ok {
        return
    }
    chooseBox.unSubscribe(sub)
    return
}

func (postman *Postman) addClients(accounts []config.MailServer) {
    postman.lock.Lock()
    defer postman.lock.Unlock()
    for _, account := range accounts {
        _, ok := postman.mailPool[account.Name]
        if ok {
            continue
        }
        client, err := New(account)
        if err != nil {
            log.Println(err)
            continue
        }
        postman.mailPool[account.Name] = client
    }
}

func (postman *Postman) Start() {
    for _, cli := range postman.mailPool {
        go func(client *Client) {
            ticker := time.NewTicker(client.flushTime * time.Second)
            defer ticker.Stop()
            for {
                select {
                case <-ticker.C:
                    mailChan, seqSet := client.Fetch()
                    for msg := range mailChan {
                        message, err := postman.openMessage(msg)
                        if err != nil {
                            log.Printf("open message: %s", err)
                            continue
                        }
                    }
                    log.Println("start to push msg , subscribers :", client.subscribers.

```

```

Size())
        client.subscribers.Each(func(index int, a *Subscriber) {
            log.Println("pushing message !!")
            a.Channel <- message
        })
    }
    if seqSet != nil {
        log.Println("start to see")
        go client.See(seqSet)
        log.Println("saw !")
    }
    case err := <-client.Done: //处理异常需开启协程
        if err != nil {
            log.Println("error happen:", err)
            err = client.Reconnect()
            if err != nil {
                log.Println("retry :" + err.Error())
                return
            } else {
                log.Println("retry success !")
            }
        }
    }
}
}(cli)
}
}

func (postman *Postman) ListServer() []string {
    server := make([]string, len(postman.mailPool))
    i := 0
    for s := range postman.mailPool {
        server[i] = s
        i++
    }
    return server
}

func (postman *Postman) openMessage(msg *imap.Message) (*proto.Mail, error) {
    var section imap.BodySectionName
    mr, err := mail.CreateReader(msg.GetBody(&section))
    if err != nil {
        log.Println("construct message error:", err)
        return nil, err
    }
    email := postman.parseMsg(mr)
    return email, nil
}

func (postman *Postman) parseMsg(mr *mail.Reader) *proto.Mail {
    header := mr.Header
    subject, _ := header.Subject()
    log.Println(subject)
}

```

```

toAddress, _ := header.AddressList("To")
fromAddress, _ := header.AddressList("From")
var attachBody *proto.Body
var text []*proto.Body
for {
    p, err := mr.NextPart()
    if err == io.EOF {
        break
    } else if err != nil {
        log.Fatal(err)
    }
    switch h := p.Header.(type) {
    case *mail.InlineHeader:
        b, _ := io.ReadAll(p.Body)
        t, _, _ := h.ContentType()
        text = append(text, &proto.Body{MainBody: b, ContentType: t})
    case *mail.AttachmentHeader:
        contentType, _, _ := h.ContentType()
        b, _ := io.ReadAll(p.Body)
        attachBody = &proto.Body{ContentType: contentType, MainBody: b}
    }
}
msgStruct := &proto.Mail{
    MessageID: header.Get("Message-Id"),
    Subject:    subject,
    To:         changeAddress2str(toAddress),
    From:       changeAddress2str(fromAddress),
    Text:       text,
    Attachment: attachBody,
}
if len(fromAddress) > 0 {
    msgStruct.From = &proto.Address{Name: fromAddress[0].Name, Address: fromAddress[0].Address}
}
return msgStruct
}

func changeAddress2str(addresses []*mail.Address) (to []*proto.Address) {
    to = make([]*proto.Address, len(addresses))
    for key, address := range addresses {
        to[key] = &proto.Address{
            Name:    address.Name,
            Address: address.Address,
        }
    }
    return
}

func (postman *Postman) Close() {
    for _, cli := range postman.mailPool {
        cli.Close()
    }
}

```

```

}
func NewPostMan(accounts []config.MailServer) Watcher {
    postman := &Postman{
        mailPool: make(map[string]*Client, len(accounts)),
        lock:      &sync.Mutex{},
    }
    postman.addClients(accounts)
    return postman
}
package auth
import (
    "context"
    "errors"
    "gomail/pkg/db"
    "google.golang.org/grpc"
    "google.golang.org/grpc/metadata"
    "log"
    "strings"
)
var (
    AuthenticationNotFound = errors.New("can not found auth information")
    AuthenticationUnknown  = errors.New("auth string is unknown")
    AuthenticationFailed   = errors.New("user not found or wrong password")
    WhiteList = []string{"proto.MailBox/Register", "proto.MailBox/Login"}
)
type Interceptor interface {
    StreamAuth(srv interface{}, ss grpc.ServerStream, info *grpc.StreamServerInfo, handler
grpc.StreamHandler) error
    UnaryAuth(ctx context.Context, req interface{}, info *grpc.UnaryServerInfo, handler gr
pc.UnaryHandler) (interface{}, error)
}
func InWhiteList(url string) bool {
    for _, s := range WhiteList {
        if s == url {
            return true
        }
    }
    return false
}
func NewAuthInterceptor(storage db.Storage, sess db.Session) Interceptor {
    return &defaultInterceptor{registry: storage, sess: sess}
}
type defaultInterceptor struct {
    registry db.Storage
    sess     db.Session
}
type User struct {
    ID        string `bson:"_id"`
    Name      string `bson:"user"`
    Password  string `bson:"password"`

```

```

    Weight    int32    `bson:"weight"`
}
func (d *defaultInterceptor) getUser(token Token) *User {
    res := &User{}
    conditions := map[string]interface{}{}
    switch token.Type() {
    case BearerAuthenticationType:
        conditions["_id"] = token.String()
    case BasicAuthenticationType:
        authStr := token.String()
        strings.Split(authStr, passwordSeparator)
        if len(authStr) != 2 {
            return nil
        }
        conditions["user"] = authStr[0]
        conditions["password"] = authStr[1]
    default:
        return nil
    }
    err := d.sess.Get(conditions, res)
    if err != nil {
        log.Printf("user:%s cannot found because error : %v", token, err)
        return nil
    }
    return res
}

func (d *defaultInterceptor) check(ctx context.Context, method string) error {
    md, ok := metadata.FromIncomingContext(ctx)
    if !ok || len(md["authorization"]) == 0 || md["authorization"][0] == "" {
        return AuthenticationNotFound
    }
    if !InWhiteList(method) {
        tk, err := FromHeaderString(md["authorization"][0])
        if err != nil {
            return err
        }
        if d.getUser(tk) == nil {
            return AuthenticationFailed
        }
    }
    return nil
}

func (d *defaultInterceptor) StreamAuth(srv interface{}, ss grpc.ServerStream, info *grpc.StreamServerInfo, handler grpc.StreamHandler) error {
    if info.IsClientStream {
        if err := d.check(ss.Context(), info.FullMethod); err != nil {
            return err
        }
    }
    return handler(srv, ss)
}

```

```

}
func (d *defaultInterceptor) UnaryAuth(ctx context.Context, req interface{}, info *grpc.Unary
yServerInfo, handler grpc.UnaryHandler) (interface{}, error) {
    if err := d.check(ctx, info.FullMethod); err != nil {
        return nil, err
    }
    return handler(ctx, req)
}
package auth
import (
    "encoding/base64"
    "golang.org/x/oauth2"
    "strings"
    "time"
)
const (
    BasicAuthenticationType = "Basic"
    BearerAuthenticationType = "Bearer"
)
const passwordSeparator = ":"
type Token interface {
    oauth2.TokenSource
    Type() string
    String() string
}
type BasicToken struct {
    User    string
    Password string
}
func (b BasicToken) Token() (*oauth2.Token, error) {
    return &oauth2.Token{
        AccessToken: base64.URLEncoding.EncodeToString([]byte(b.String())),
        TokenType:   "basic",
        RefreshToken: "",
        Expiry:      time.Now().Add(time.Hour * 24),
    }, nil
}
func (b BasicToken) Type() string {
    return BasicAuthenticationType
}
func (b BasicToken) String() string {
    return b.User + ":" + b.Password
}
func NewBasicToken(user, pass string) Token {
    return BasicToken{
        User:    user,
        Password: pass,
    }
}
type BearerToken struct {

```

```

    ID string
}
func (b BearerToken) Token() (*oauth2.Token, error) {
    return &oauth2.Token{
        AccessToken: b.ID,
        TokenType:   "basic",
        RefreshToken: "",
        Expiry:      time.Now().Add(time.Hour * 24),
    }, nil
}
func (b BearerToken) Type() string {
    return BearerAuthenticationTyp
}
func (b BearerToken) String() string {
    return b.ID
}
func NewBearerToken(id string) Token {
    return BearerToken{
        ID: id,
    }
}
func FromHeaderString(authStr string) (Token, error) {
    if strings.HasPrefix(authStr, BasicAuthenticationType) {
        authStr = strings.TrimLeft(authStr[len(BasicAuthenticationType):], " ")
        userPass, err := base64.URLEncoding.DecodeString(authStr)
        if err != nil {
            return BasicToken{}, AuthenticationUnknown
        }
        str := strings.Split(string(userPass), passwordSeparator)
        if len(str) != 2 {
            return BasicToken{}, AuthenticationUnknown
        }
        return BasicToken{
            User:      str[0],
            Password: str[1],
        }, nil
    }
    if strings.HasPrefix(authStr, BearerAuthenticationTyp) {
        authStr = strings.TrimLeft(authStr[len(BearerAuthenticationTyp):], " ")
        if len(authStr) == 0 {
            return BasicToken{}, AuthenticationUnknown
        }
        return NewBearerToken(authStr), nil
    }
    return BasicToken{}, AuthenticationUnknown
}
package mailbox
import (
    "google.golang.org/grpc"
)

```

```

func NewGRPCServer(opts ...grpc.ServerOption) *grpc.Server {
    s := grpc.NewServer(opts...)
    return s
}

package mailbox
import (
    "context"
    "github.com/golang/protobuf/ptypes/empty"
    "gomail/pkg/db"
    "gomail/pkg/imap"
    "gomail/pkg/mailbox/auth"
    "gomail/pkg/proto"
    "gomail/pkg/smtp"
    "google.golang.org/grpc/codes"
    "google.golang.org/grpc/metadata"
    "google.golang.org/grpc/status"
    "io"
    "log"
    "sync"
)

type DefaultMailBoxService struct {
    proto.UnimplementedMailBoxServer
    Watcher  imap.Watcher
    Registry db.Storage
    Session  db.Session
    Tool     smtp.Tool
    lock     sync.Mutex
}

func (s *DefaultMailBoxService) Send(_ context.Context, t *proto.MailTask) (*proto.SendMailResponse, error) {
    task := smtp.MailTask{
        From:      AddressString(t.From),
        To:         AddressStrings(t.To),
        Cc:         AddressStrings(t.Cc),
        Bcc:        AddressStrings(t.Bcc),
        Subject:    t.Subject,
        ReplyId:    t.ReplyId,
        Body:       t.Text.MainBody,
        ContentType: t.Text.ContentType,
    }
    if t.Attachment != nil && t.Attachment.WithAttachment {
        file, err := s.Registry.Download(t.Attachment.AttachmentID)
        if err != nil {
            return nil, status.Errorf(codes.Internal, "error happen %v", err)
        }
        task.Attachment = smtp.Attachment{
            File:      file,
            WithFile:  true,
        }
    }
}

```

```

    msgID, err := s.Tool.Send(task)
    if err != nil {
        return nil, status.Errorf(codes.Internal, "error happen %v", err)
    }
    return &proto.SendMailResponse{MsgID: msgID}, nil
}

func (s *DefaultMailBoxService) ListServer(context.Context, *empty.Empty) (*proto.ServerList, error) {
    resp := &proto.ServerList{}
    for _, name := range s.Watcher.ListServer() {
        resp.Items = append(resp.Items, &proto.Server{Name: name})
    }
    return resp, nil
}

func (s *DefaultMailBoxService) Upload(us proto.MailBox_UploadServer) error {
    uf, err := us.Recv()
    if err != nil {
        return status.Errorf(codes.InvalidArgument, "error happen %v", err)
    }
    errChan := make(chan error, 1)
    defer close(errChan)
    pr, pw := io.Pipe()
    go func() {
        defer func() { _ = pw.Close() }()
        for {
            uf, err := us.Recv()
            if err != nil {
                errChan <- err
                return
            }
            _, err = pw.Write(uf.GetContent())
            if err != nil {
                errChan <- err
                return
            }
        }
    }()
    id, err := s.Registry.Upload(uf.GetName(), uf.GetContentType(), pr)
    if err != nil {
        return err
    }
    err = <-errChan
    if err != nil {
        return err
    }
    return us.SendAndClose(&proto.UploadResponse{FileID: id})
}

func (s *DefaultMailBoxService) Watch(server *proto.Server, ws proto.MailBox_WatchServer) error {
    md, ok := metadata.FromIncomingContext(ws.Context())

```

```

    if !ok {
        return status.Error(codes.Unknown, "header not found")
    }
    temp := md.Get("UserID")
    if len(temp) == 0 {
        return status.Error(codes.Unknown, "user not found")
    }
    id := temp[0]
    u := &auth.User{}
    err := s.Session.Get(map[string]interface{}{"_id": id}, u)
    if err != nil {
        return err
    }
    done := make(chan error)
    msgChan := make(chan *proto.Mail, 50)
    sub, err := s.Watcher.Subscribe(ser.GetName(), id, u.Weight, msgChan)
    if err != nil {
        return err
    }
    defer func() {
        s.Watcher.UnSubscribe(sub)
        close(msgChan)
    }()
    for {
        select {
        case msg := <-msgChan:
            {
                err := ws.Send(msg)
                if err != nil {
                    return err
                }
            }
        case err := <-done:
            {
                if err != nil {
                    log.Println(err, "client clean up !")
                    return err
                }
            }
        }
    }
}

func (s *DefaultMailBoxService) Register(_ context.Context, u *proto.User) (*proto.UserResponse, error) {
    s.lock.Lock()
    defer s.lock.Unlock()
    if s.Session.Exist(map[string]interface{}{"name": u.Name, "password": u.Password}) {
        return nil, status.Error(codes.AlreadyExists, "user existed")
    }
    id, err := s.Session.Set(&auth.User{Password: u.Password, Name: u.Name, Weight: u.W

```

```

eight}))
    if err != nil {
        return nil, status.Errorf(codes.Internal, "error when saving user %v", err)
    }
    return &proto.UserResponse{
        ID: id,
        Name: u.Name,
    }, nil
}

func (s *DefaultMailBoxService) Login(_ context.Context, u *proto.User) (*proto.UserResponse, error) {
    var user = &auth.User{}
    if err := s.Session.Get(map[string]interface{}{"name": u.Name, "password": u.Password}, u); err != nil {
        return nil, status.Error(codes.PermissionDenied, err.Error())
    }
    return &proto.UserResponse{
        ID: user.ID,
        Name: u.Name,
    }, nil
}

func NewMailBoxService(watcher imap.Watcher, client smtp.Tool, storage db.Storage, session db.Session) *DefaultMailBoxService {
    return &DefaultMailBoxService{
        Watcher: watcher,
        Tool: client,
        Registry: storage,
        Session: session,
        lock: sync.Mutex{},
    }
}

package mailbox
import (
    "gomail/pkg/proto"
    "mime"
    "strings"
    "unicode/utf8"
)

func AddressStrings(as []*proto.Address) []string {
    res := make([]string, len(as))
    for i, a := range as {
        res[i] = AddressString(a)
    }
    return res
}

func AddressString(a *proto.Address) string {
    // Format address local@domain
    at := strings.LastIndex(a.Address, "@")
    var local, domain string
    if at < 0 {

```

```

        local = a.Address
    } else {
        local, domain = a.Address[:at], a.Address[at+1:]
    }
    quoteLocal := false
    for i, r := range local {
        if isAtext(r, false, false) {
            continue
        }
        if r == '.' {
            if i > 0 && local[i-1] != '.' && i < len(local)-1 {
                continue
            }
        }
        quoteLocal = true
        break
    }
    if quoteLocal {
        local = quoteString(local)
    }
    s := "<" + local + "@" + domain + ">"
    if a.Name == "" {
        return s
    }
    // If every character is printable ASCII, quoting is simple.
    allPrintable := true
    for _, r := range a.Name {
        // isWSP here should actually be isFWS,
        // but we don't support folding yet.
        if !isVchar(r) && !isWSP(r) || isMultibyte(r) {
            allPrintable = false
            break
        }
    }
    if allPrintable {
        return quoteString(a.Name) + " " + s
    }
    if strings.ContainsAny(a.Name, "\"#$%&'(),.;<>@[^{}|]~") {
        return mime.BEncoding.Encode("utf-8", a.Name) + " " + s
    }
    return mime.QEncoding.Encode("utf-8", a.Name) + " " + s
}

func isAtext(r rune, dot, permissive bool) bool {
    switch r {
    case '.':
        return dot
    // RFC 5322 3.2.3. specials
    case '(', ')', '[', ']', ';', '@', '\\', ':':
        return permissive
    case '<', '>', '"', '\':

```

```

        return false
    }
    return isVchar(r)
}
// isQtext reports whether r is an RFC 5322 qtext character.
func isQtext(r rune) bool {
    // Printable US-ASCII, excluding backslash or quote.
    if r == '\\' || r == '"' {
        return false
    }
    return isVchar(r)
}
// quoteString renders a string as an RFC 5322 quoted-string.
func quoteString(s string) string {
    var buf strings.Builder
    buf.WriteByte('"')
    for _, r := range s {
        if isQtext(r) || isWSP(r) {
            buf.WriteRune(r)
        } else if isVchar(r) {
            buf.WriteByte('\\')
            buf.WriteRune(r)
        }
    }
    buf.WriteByte('"')
    return buf.String()
}
func isVchar(r rune) bool {
    // Visible (printing) characters.
    return '!' <= r && r <= '~' || isMultibyte(r)
}
func isMultibyte(r rune) bool {
    return r >= utf8.RuneSelf
}
func isWSP(r rune) bool {
    return r == ' ' || r == '\t'
}
package proto
import (
    empty "github.com/golang/protobuf/ptypes/empty"
    protoreflect "google.golang.org/protobuf/reflect/protoreflect"
    protoimpl "google.golang.org/protobuf/runtime/protoimpl"
    reflect "reflect"
    sync "sync"
)
const (
    _ = protoimpl.EnforceVersion(20 - protoimpl.MinVersion)
    _ = protoimpl.EnforceVersion(protoimpl.MaxVersion - 20)
)
type Mail struct {

```

```

state      protoimpl.MessageState
sizeCache  protoimpl.SizeCache
unknownFields protoimpl.UnknownFields
MessageID  string      `protobuf:"bytes,1,opt,name=MessageID,proto3" json:"MessageID,
omitempty"` // Unique ID number for this person.
Subject    string      `protobuf:"bytes,2,opt,name=Subject,proto3" json:"Subject,omitemp
ty"`
To          []*Address `protobuf:"bytes,3,rep,name=To,proto3" json:"To,omitempty"`
From        *Address  `protobuf:"bytes,4,opt,name=From,proto3" json:"From,omitempt
y"`
Text        []*Body   `protobuf:"bytes,5,rep,name=Text,proto3" json:"Text,omitempty"`
Attachment *Body    `protobuf:"bytes,6,opt,name=Attachment,proto3" json:"Attachme
nt,omitempty"`
}
func (x *Mail) Reset() {
    *x = Mail{}
    if protoimpl.UnsafeEnabled {
        mi := &file_mail_proto_msgTypes[0]
        ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
        ms.StoreMessageInfo(mi)
    }
}
func (x *Mail) String() string {
    return protoimpl.X.MessageStringOf(x)
}
func (*Mail) ProtoMessage() {}
func (x *Mail) ProtoReflect() protoreflect.Message {
    mi := &file_mail_proto_msgTypes[0]
    if protoimpl.UnsafeEnabled && x != nil {
        ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
        if ms.LoadMessageInfo() == nil {
            ms.StoreMessageInfo(mi)
        }
        return ms
    }
    return mi.MessageOf(x)
}
func (*Mail) Descriptor() ([]byte, []int) {
    return file_mail_proto_rawDescGZIP(), []int{0}
}
func (x *Mail) GetMessageID() string {
    if x != nil {
        return x.MessageID
    }
    return ""
}
func (x *Mail) GetSubject() string {
    if x != nil {
        return x.Subject
    }
}

```

```

    return ""
}
func (x *Mail) GetTo() []*Address {
    if x != nil {
        return x.To
    }
    return nil
}
func (x *Mail) GetFrom() *Address {
    if x != nil {
        return x.From
    }
    return nil
}
func (x *Mail) GetText() []*Body {
    if x != nil {
        return x.Text
    }
    return nil
}
func (x *Mail) GetAttachment() *Body {
    if x != nil {
        return x.Attachment
    }
    return nil
}
type MailTask struct {
    state          protoimpl.MessageState
    sizeCache      protoimpl.SizeCache
    unknownFields  protoimpl.UnknownFields
    From           *Address          `protobuf:"bytes,1,opt,name=From,proto3" json:"From,omitempty"`
    To             []*Address         `protobuf:"bytes,2,rep,name=To,proto3" json:"To,omitempty"`
    Cc             []*Address         `protobuf:"bytes,3,rep,name=Cc,proto3" json:"Cc,omitempty"`
    Bcc            []*Address         `protobuf:"bytes,4,rep,name=Bcc,proto3" json:"Bcc,omitempty"`
    Subject        string             `protobuf:"bytes,5,opt,name=Subject,proto3" json:"Subject,omitempty"`
    ReplyId        string             `protobuf:"bytes,6,opt,name=ReplyId,proto3" json:"ReplyId,omitempty"`
    Text           *Body             `protobuf:"bytes,7,opt,name=Text,proto3" json:"Text,omitempty"`
    Attachment     *AttachmentRequest `protobuf:"bytes,8,opt,name=Attachment,proto3" json:"Attachment,omitempty"`
}
func (x *MailTask) Reset() {
    *x = MailTask{}
    if protoimpl.UnsafeEnabled {

```

```

        mi := &file_mail_proto_msgTypes[1]
        ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
        ms.StoreMessageInfo(mi)
    }
}
func (x *MailTask) String() string {
    return protoimpl.X.MessageStringOf(x)
}
func (*MailTask) ProtoMessage() {}
func (x *MailTask) ProtoReflect() protoreflect.Message {
    mi := &file_mail_proto_msgTypes[1]
    if protoimpl.UnsafeEnabled && x != nil {
        ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
        if ms.LoadMessageInfo() == nil {
            ms.StoreMessageInfo(mi)
        }
        return ms
    }
    return mi.MessageOf(x)
}
func (*MailTask) Descriptor() ([]byte, []int) {
    return file_mail_proto_rawDescGZIP(), []int{1}
}
func (x *MailTask) GetFrom() *Address {
    if x != nil {
        return x.From
    }
    return nil
}
func (x *MailTask) GetTo() []*Address {
    if x != nil {
        return x.To
    }
    return nil
}
func (x *MailTask) GetCc() []*Address {
    if x != nil {
        return x.Cc
    }
    return nil
}
func (x *MailTask) GetBcc() []*Address {
    if x != nil {
        return x.Bcc
    }
    return nil
}
func (x *MailTask) GetSubject() string {
    if x != nil {
        return x.Subject
    }

```



```

    }
    return ""
}
func (x *MailTask) GetReplyId() string {
    if x != nil {
        return x.ReplyId
    }
    return ""
}
func (x *MailTask) GetText() *Body {
    if x != nil {
        return x.Text
    }
    return nil
}
func (x *MailTask) GetAttachment() *AttachmentRequest {
    if x != nil {
        return x.Attachment
    }
    return nil
}
type AttachmentRequest struct {
    state          protoimpl.MessageState
    sizeCache      protoimpl.SizeCache
    unknownFields  protoimpl.UnknownFields
    WithAttachment bool    `protobuf:"varint,1,opt,name=WithAttachment,proto3" json:"WithAttachment,omitempty"`
    AttachmentID   string `protobuf:"bytes,2,opt,name=AttachmentID,proto3" json:"AttachmentID,omitempty"`
}
func (x *AttachmentRequest) Reset() {
    *x = AttachmentRequest{}
    if protoimpl.UnsafeEnabled {
        mi := &file_mail_proto_msgTypes[2]
        ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
        ms.StoreMessageInfo(mi)
    }
}
func (x *AttachmentRequest) String() string {
    return protoimpl.X.MessageStringOf(x)
}
func (*AttachmentRequest) ProtoMessage() {}
func (x *AttachmentRequest) ProtoReflect() protoreflect.Message {
    mi := &file_mail_proto_msgTypes[2]
    if protoimpl.UnsafeEnabled && x != nil {
        ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
        if ms.LoadMessageInfo() == nil {
            ms.StoreMessageInfo(mi)
        }
    }
    return ms
}

```

```

    }
    return mi.MessageOf(x)
}
func (*AttachmentRequest) Descriptor() ([]byte, []int) {
    return file_mail_proto_rawDescGZIP(), []int{2}
}
func (x *AttachmentRequest) GetWithAttachment() bool {
    if x != nil {
        return x.WithAttachment
    }
    return false
}
func (x *AttachmentRequest) GetAttachmentID() string {
    if x != nil {
        return x.AttachmentID
    }
    return ""
}
type Body struct {
    state          protoimpl.MessageState
    sizeCache      protoimpl.SizeCache
    unknownFields  protoimpl.UnknownFields
    ContentType     string `protobuf:"bytes,1,opt,name=contentType,proto3" json:"contentType,omit
empty"`
    MainBody       []byte `protobuf:"bytes,2,opt,name=mainBody,proto3" json:"mainBody,omit
empty"`
}
func (x *Body) Reset() {
    *x = Body{}
    if protoimpl.UnsafeEnabled {
        mi := &file_mail_proto_msgTypes[3]
        ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
        ms.StoreMessageInfo(mi)
    }
}
func (x *Body) String() string {
    return protoimpl.X.MessageStringOf(x)
}
func (*Body) ProtoMessage() {}
func (x *Body) ProtoReflect() protoreflect.Message {
    mi := &file_mail_proto_msgTypes[3]
    if protoimpl.UnsafeEnabled && x != nil {
        ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
        if ms.LoadMessageInfo() == nil {
            ms.StoreMessageInfo(mi)
        }
        return ms
    }
    return mi.MessageOf(x)
}

```

```

func (*Body) Descriptor() ([]byte, []int) {
    return file_mail_proto_rawDescGZIP(), []int{3}
}
func (x *Body) GetContentType() string {
    if x != nil {
        return x.ContentType
    }
    return ""
}
func (x *Body) GetMainBody() []byte {
    if x != nil {
        return x.MainBody
    }
    return nil
}
type Address struct {
    state          protoimpl.MessageState
    sizeCache      protoimpl.SizeCache
    unknownFields  protoimpl.UnknownFields
    Name           string `protobuf:"bytes,1,opt,name=name,proto3" json:"name,omitempty"`
    Address        string `protobuf:"bytes,2,opt,name=address,proto3" json:"address,omitempty"`
}
func (x *Address) Reset() {
    *x = Address{}
    if protoimpl.UnsafeEnabled {
        mi := &file_mail_proto_msgTypes[4]
        ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
        ms.StoreMessageInfo(mi)
    }
}
func (x *Address) String() string {
    return protoimpl.X.MessageStringOf(x)
}
func (*Address) ProtoMessage() {}
func (x *Address) ProtoReflect() protoreflect.Message {
    mi := &file_mail_proto_msgTypes[4]
    if protoimpl.UnsafeEnabled && x != nil {
        ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
        if ms.LoadMessageInfo() == nil {
            ms.StoreMessageInfo(mi)
        }
        return ms
    }
    return mi.MessageOf(x)
}
func (*Address) Descriptor() ([]byte, []int) {
    return file_mail_proto_rawDescGZIP(), []int{4}
}
func (x *Address) GetName() string {
    if x != nil {

```

```

        return x.Name
    }
    return ""
}
func (x *Address) GetAddress() string {
    if x != nil {
        return x.Address
    }
    return ""
}
type SendMailResponse struct {
    state          protoimpl.MessageState
    sizeCache      protoimpl.SizeCache
    unknownFields  protoimpl.UnknownFields
    MsgID string `protobuf:"bytes,1,opt,name=MsgID,proto3" json:"MsgID,omitempty"` // Unique ID number for this person.
}
func (x *SendMailResponse) Reset() {
    *x = SendMailResponse{}
    if protoimpl.UnsafeEnabled {
        mi := &file_mail_proto_msgTypes[5]
        ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
        ms.StoreMessageInfo(mi)
    }
}
func (x *SendMailResponse) String() string {
    return protoimpl.X.MessageStringOf(x)
}
func (*SendMailResponse) ProtoMessage() {}
func (x *SendMailResponse) ProtoReflect() protoreflect.Message {
    mi := &file_mail_proto_msgTypes[5]
    if protoimpl.UnsafeEnabled && x != nil {
        ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
        if ms.LoadMessageInfo() == nil {
            ms.StoreMessageInfo(mi)
        }
        return ms
    }
    return mi.MessageOf(x)
}
func (*SendMailResponse) Descriptor() ([]byte, []int) {
    return file_mail_proto_rawDescGZIP(), []int{5}
}
func (x *SendMailResponse) GetMsgID() string {
    if x != nil {
        return x.MsgID
    }
    return ""
}
type Server struct {

```

```

    state          protoimpl.MessageState
    sizeCache      protoimpl.SizeCache
    unknownFields  protoimpl.UnknownFields
    Name string `protobuf:"bytes,1,opt,name=Name,proto3" json:"Name,omitempty"`
}
func (x *Server) Reset() {
    *x = Server{}
    if protoimpl.UnsafeEnabled {
        mi := &file_mail_proto_msgTypes[6]
        ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
        ms.StoreMessageInfo(mi)
    }
}
func (x *Server) String() string {
    return protoimpl.X.MessageStringOf(x)
}
func (*Server) ProtoMessage() {}
func (x *Server) ProtoReflect() protoreflect.Message {
    mi := &file_mail_proto_msgTypes[6]
    if protoimpl.UnsafeEnabled && x != nil {
        ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
        if ms.LoadMessageInfo() == nil {
            ms.StoreMessageInfo(mi)
        }
        return ms
    }
    return mi.MessageOf(x)
}
func (*Server) Descriptor() ([]byte, []int) {
    return file_mail_proto_rawDescGZIP(), []int{6}
}
func (x *Server) GetName() string {
    if x != nil {
        return x.Name
    }
    return ""
}
type UploadFile struct {
    state          protoimpl.MessageState
    sizeCache      protoimpl.SizeCache
    unknownFields  protoimpl.UnknownFields
    Name          string `protobuf:"bytes,1,opt,name=Name,proto3" json:"Name,omitempty"`
    ContentType    string `protobuf:"bytes,2,opt,name=ContentType,proto3" json:"ContentType,omitempty"`
    Content        []byte `protobuf:"bytes,3,opt,name=Content,proto3" json:"Content,omitempty"`
}
func (x *UploadFile) Reset() {
    *x = UploadFile{}
    if protoimpl.UnsafeEnabled {

```

```

        mi := &file_mail_proto_msgTypes[7]
        ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
        ms.StoreMessageInfo(mi)
    }
}
func (x *UploadFile) String() string {
    return protoimpl.X.MessageStringOf(x)
}
func (*UploadFile) ProtoMessage() {}
func (x *UploadFile) ProtoReflect() protoreflect.Message {
    mi := &file_mail_proto_msgTypes[7]
    if protoimpl.UnsafeEnabled && x != nil {
        ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
        if ms.LoadMessageInfo() == nil {
            ms.StoreMessageInfo(mi)
        }
        return ms
    }
    return mi.MessageOf(x)
}
// Deprecated: Use UploadFile.ProtoReflect.Descriptor instead.
func (*UploadFile) Descriptor() ([]byte, []int) {
    return file_mail_proto_rawDescGZIP(), []int{7}
}
func (x *UploadFile) GetName() string {
    if x != nil {
        return x.Name
    }
    return ""
}
func (x *UploadFile) GetContentType() string {
    if x != nil {
        return x.ContentType
    }
    return ""
}
func (x *UploadFile) GetContent() []byte {
    if x != nil {
        return x.Content
    }
    return nil
}
type UploadResponse struct {
    state          protoimpl.MessageState
    sizeCache      protoimpl.SizeCache
    unknownFields  protoimpl.UnknownFields
    FileID string `protobuf:"bytes,1,opt,name=FileID,proto3" json:"FileID,omitempty"`
}
func (x *UploadResponse) Reset() {
    *x = UploadResponse{}
}

```

```

    if protoimpl.UnsafeEnabled {
        mi := &file_mail_proto_msgTypes[8]
        ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
        ms.StoreMessageInfo(mi)
    }
}

func (x *UploadResponse) String() string {
    return protoimpl.X.MessageStringOf(x)
}

func (*UploadResponse) ProtoMessage() {}

func (x *UploadResponse) ProtoReflect() protoreflect.Message {
    mi := &file_mail_proto_msgTypes[8]
    if protoimpl.UnsafeEnabled && x != nil {
        ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
        if ms.LoadMessageInfo() == nil {
            ms.StoreMessageInfo(mi)
        }
        return ms
    }
    return mi.MessageOf(x)
}

// Deprecated: Use UploadResponse.ProtoReflect.Descriptor instead.
func (*UploadResponse) Descriptor() ([]byte, []int) {
    return file_mail_proto_rawDescGZIP(), []int{8}
}

func (x *UploadResponse) GetFileID() string {
    if x != nil {
        return x.FileID
    }
    return ""
}

type User struct {
    state          protoimpl.MessageState
    sizeCache      protoimpl.SizeCache
    unknownFields  protoimpl.UnknownFields
    Name           string `protobuf:"bytes,1,opt,name=Name,proto3" json:"Name,omitempty"`
    Password       string `protobuf:"bytes,2,opt,name=Password,proto3" json:"Password,omitempty"`
    Weight         int32  `protobuf:"varint,3,opt,name=Weight,proto3" json:"Weight,omitempty"`
}

func (x *User) Reset() {
    *x = User{}
    if protoimpl.UnsafeEnabled {
        mi := &file_mail_proto_msgTypes[9]
        ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
        ms.StoreMessageInfo(mi)
    }
}

func (x *User) String() string {
    return protoimpl.X.MessageStringOf(x)
}

```

```

}
func (*User) ProtoMessage() {}
func (x *User) ProtoReflect() protoreflect.Message {
    mi := &file_mail_proto_msgTypes[9]
    if protoimpl.UnsafeEnabled && x != nil {
        ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
        if ms.LoadMessageInfo() == nil {
            ms.StoreMessageInfo(mi)
        }
        return ms
    }
    return mi.MessageOf(x)
}
// Deprecated: Use User.ProtoReflect.Descriptor instead.
func (*User) Descriptor() ([]byte, []int) {
    return file_mail_proto_rawDescGZIP(), []int{9}
}
func (x *User) GetName() string {
    if x != nil {
        return x.Name
    }
    return ""
}
func (x *User) GetPassword() string {
    if x != nil {
        return x.Password
    }
    return ""
}
func (x *User) GetWeight() int32 {
    if x != nil {
        return x.Weight
    }
    return 0
}
type UserResponse struct {
    state          protoimpl.MessageState
    sizeCache      protoimpl.SizeCache
    unknownFields  protoimpl.UnknownFields
    ID             string `protobuf:"bytes,1,opt,name=ID,proto3" json:"ID,omitempty"`
    Name           string `protobuf:"bytes,2,opt,name=Name,proto3" json:"Name,omitempty"`
}
func (x *UserResponse) Reset() {
    *x = UserResponse{}
    if protoimpl.UnsafeEnabled {
        mi := &file_mail_proto_msgTypes[10]
        ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
        ms.StoreMessageInfo(mi)
    }
}

```

```

func (x *UserResponse) String() string {
    return protoimpl.X.MessageStringOf(x)
}
func (*UserResponse) ProtoMessage() {}
func (x *UserResponse) ProtoReflect() protoreflect.Message {
    mi := &file_mail_proto_msgTypes[10]
    if protoimpl.UnsafeEnabled && x != nil {
        ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
        if ms.LoadMessageInfo() == nil {
            ms.StoreMessageInfo(mi)
        }
        return ms
    }
    return mi.MessageOf(x)
}
// Deprecated: Use UserResponse.ProtoReflect.Descriptor instead.
func (*UserResponse) Descriptor() ([]byte, []int) {
    return file_mail_proto_rawDescGZIP(), []int{10}
}
func (x *UserResponse) GetID() string {
    if x != nil {
        return x.ID
    }
    return ""
}
func (x *UserResponse) GetName() string {
    if x != nil {
        return x.Name
    }
    return ""
}
type ServerList struct {
    state          protoimpl.MessageState
    sizeCache      protoimpl.SizeCache
    unknownFields  protoimpl.UnknownFields
    Items []*Server `protobuf:"bytes,1,rep,name=Items,proto3" json:"Items,omitempty"`
}
func (x *ServerList) Reset() {
    *x = ServerList{}
    if protoimpl.UnsafeEnabled {
        mi := &file_mail_proto_msgTypes[11]
        ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
        ms.StoreMessageInfo(mi)
    }
}
func (x *ServerList) String() string {
    return protoimpl.X.MessageStringOf(x)
}
func (*ServerList) ProtoMessage() {}
func (x *ServerList) ProtoReflect() protoreflect.Message {

```

```

    mi := &file_mail_proto_msgTypes[11]
    if protoimpl.UnsafeEnabled && x != nil {
        ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
        if ms.LoadMessageInfo() == nil {
            ms.StoreMessageInfo(mi)
        }
        return ms
    }
    return mi.MessageOf(x)
}
// Deprecated: Use ServerList.ProtoReflect.Descriptor instead.
func (*ServerList) Descriptor() ([]byte, []int) {
    return file_mail_proto_rawDescGZIP(), []int{11}
}
func (x *ServerList) GetItems() []*Server {
    if x != nil {
        return x.Items
    }
    return nil
}
var File_mail_proto protoreflect.FileDescriptor
var file_mail_proto_rawDesc = []byte{
    0x0a, 0x0a, 0x6d, 0x61, 0x69, 0x6c, 0x2e, 0x70, 0x72, 0x6f, 0x74, 0x6f, 0x12, 0x05,
    0x70, 0x72,
    0x6f, 0x74, 0x6f, 0x1a, 0x1b, 0x67, 0x6f, 0x67, 0x6c, 0x65, 0x2f, 0x70, 0x72,
    0x6f, 0x74,
    0x6f, 0x62, 0x75, 0x66, 0x2f, 0x65, 0x6d, 0x70, 0x74, 0x79, 0x2e, 0x70, 0x72, 0x6f,
    0x74, 0x6f,
    0x22, 0xd0, 0x01, 0x0a, 0x04, 0x4d, 0x61, 0x69, 0x6c, 0x12, 0x1c, 0x0a, 0x09, 0x4d,
    0x65, 0x73,
    0x73, 0x61, 0x67, 0x65, 0x49, 0x44, 0x18, 0x01, 0x20, 0x01, 0x28, 0x09, 0x52, 0x0
    9, 0x4d, 0x65,
    0x73, 0x73, 0x61, 0x67, 0x65, 0x49, 0x44, 0x12, 0x18, 0x0a, 0x07, 0x53, 0x75, 0x6
    2, 0x6a, 0x65,
    0x63, 0x74, 0x18, 0x02, 0x20, 0x01, 0x28, 0x09, 0x52, 0x07, 0x53, 0x75, 0x62, 0x6
    a, 0x65, 0x63,
    0x74, 0x12, 0x1e, 0x0a, 0x02, 0x54, 0x6f, 0x18, 0x03, 0x20, 0x03, 0x28, 0x0b, 0x32,
    0x0e, 0x2e,
    0x70, 0x72, 0x6f, 0x74, 0x6f, 0x2e, 0x41, 0x64, 0x64, 0x72, 0x65, 0x73, 0x73, 0x52,
    0x02, 0x54,
    0x6f, 0x12, 0x22, 0x0a, 0x04, 0x46, 0x72, 0x6f, 0x6d, 0x18, 0x04, 0x20, 0x01, 0x28,
    0x0b, 0x32,
    0x0e, 0x2e, 0x70, 0x72, 0x6f, 0x74, 0x6f, 0x2e, 0x41, 0x64, 0x64, 0x72, 0x65, 0x73,
    0x73, 0x52,
    0x04, 0x46, 0x72, 0x6f, 0x6d, 0x12, 0x1f, 0x0a, 0x04, 0x54, 0x65, 0x78, 0x74, 0x18,
    0x05, 0x20,
    0x03, 0x28, 0x0b, 0x32, 0x0b, 0x2e, 0x70, 0x72, 0x6f, 0x74, 0x6f, 0x2e, 0x42, 0x6f,
    0x64, 0x79,
    0x52, 0x04, 0x54, 0x65, 0x78, 0x74, 0x12, 0x2b, 0x0a, 0x0a, 0x41, 0x74, 0x74, 0x6
    1, 0x63, 0x68,

```

0x6d, 0x65, 0x6e, 0x74, 0x18, 0x06, 0x20, 0x01, 0x28, 0x0b, 0x32, 0x0b, 0x2e, 0x70, 0x72, 0x6f,
 0x74, 0x6f, 0x2e, 0x42, 0x6f, 0x64, 0x79, 0x52, 0x0a, 0x41, 0x74, 0x74, 0x61, 0x63, 0x68, 0x6d,
 0x65, 0x6e, 0x74, 0x22, 0x9f, 0x02, 0x0a, 0x08, 0x4d, 0x61, 0x69, 0x6c, 0x54, 0x61, 0x73, 0x6b,
 0x12, 0x22, 0x0a, 0x04, 0x46, 0x72, 0x6f, 0x6d, 0x18, 0x01, 0x20, 0x01, 0x28, 0x0b, 0x32, 0x0e,
 0x2e, 0x70, 0x72, 0x6f, 0x74, 0x6f, 0x2e, 0x41, 0x64, 0x64, 0x72, 0x65, 0x73, 0x73, 0x52, 0x04,
 0x46, 0x72, 0x6f, 0x6d, 0x12, 0x1e, 0x0a, 0x02, 0x54, 0x6f, 0x18, 0x02, 0x20, 0x03, 0x28, 0x0b,
 0x32, 0x0e, 0x2e, 0x70, 0x72, 0x6f, 0x74, 0x6f, 0x2e, 0x41, 0x64, 0x64, 0x72, 0x65, 0x73, 0x73,
 0x52, 0x02, 0x54, 0x6f, 0x12, 0x1e, 0x0a, 0x02, 0x43, 0x63, 0x18, 0x03, 0x20, 0x03, 0x28, 0x0b,
 0x32, 0x0e, 0x2e, 0x70, 0x72, 0x6f, 0x74, 0x6f, 0x2e, 0x41, 0x64, 0x64, 0x72, 0x65, 0x73, 0x73,
 0x52, 0x02, 0x43, 0x63, 0x12, 0x20, 0x0a, 0x03, 0x42, 0x63, 0x63, 0x18, 0x04, 0x20, 0x03, 0x28,
 0x0b, 0x32, 0x0e, 0x2e, 0x70, 0x72, 0x6f, 0x74, 0x6f, 0x2e, 0x41, 0x64, 0x64, 0x72, 0x65, 0x73,
 0x73, 0x52, 0x03, 0x42, 0x63, 0x63, 0x12, 0x18, 0x0a, 0x07, 0x53, 0x75, 0x62, 0x6a, 0x65, 0x63,
 0x74, 0x18, 0x05, 0x20, 0x01, 0x28, 0x09, 0x52, 0x07, 0x53, 0x75, 0x62, 0x6a, 0x65, 0x63, 0x74,
 0x12, 0x18, 0x0a, 0x07, 0x52, 0x65, 0x70, 0x6c, 0x79, 0x49, 0x64, 0x18, 0x06, 0x20, 0x01, 0x28,
 0x09, 0x52, 0x07, 0x52, 0x65, 0x70, 0x6c, 0x79, 0x49, 0x64, 0x12, 0x1f, 0x0a, 0x04, 0x54, 0x65,
 0x78, 0x74, 0x18, 0x07, 0x20, 0x01, 0x28, 0x0b, 0x32, 0x0b, 0x2e, 0x70, 0x72, 0x6f, 0x74, 0x6f,
 0x2e, 0x42, 0x6f, 0x64, 0x79, 0x52, 0x04, 0x54, 0x65, 0x78, 0x74, 0x12, 0x38, 0x0a, 0x0a, 0x41,
 0x74, 0x74, 0x61, 0x63, 0x68, 0x6d, 0x65, 0x6e, 0x74, 0x18, 0x08, 0x20, 0x01, 0x28, 0x0b, 0x32,
 0x18, 0x2e, 0x70, 0x72, 0x6f, 0x74, 0x6f, 0x2e, 0x41, 0x74, 0x74, 0x61, 0x63, 0x68, 0x6d, 0x65,
 0x6e, 0x74, 0x52, 0x65, 0x71, 0x75, 0x65, 0x73, 0x74, 0x52, 0x0a, 0x41, 0x74, 0x74, 0x61, 0x63,
 0x68, 0x6d, 0x65, 0x6e, 0x74, 0x22, 0x5f, 0x0a, 0x11, 0x41, 0x74, 0x74, 0x61, 0x63, 0x68, 0x6d,
 0x65, 0x6e, 0x74, 0x52, 0x65, 0x71, 0x75, 0x65, 0x73, 0x74, 0x12, 0x26, 0x0a, 0x0e, 0x57, 0x69,
 0x74, 0x68, 0x41, 0x74, 0x74, 0x61, 0x63, 0x68, 0x6d, 0x65, 0x6e, 0x74, 0x18, 0x01, 0x20, 0x01,
 0x28, 0x08, 0x52, 0x0e, 0x57, 0x69, 0x74, 0x68, 0x41, 0x74, 0x74, 0x61, 0x63, 0x68, 0x6d, 0x65,
 0x6e, 0x74, 0x12, 0x22, 0x0a, 0x0c, 0x41, 0x74, 0x74, 0x61, 0x63, 0x68, 0x6d, 0x65, 0x6e, 0x74,

0x49, 0x44, 0x18, 0x02, 0x20, 0x01, 0x28, 0x09, 0x52, 0x0c, 0x41, 0x74, 0x74, 0x61, 0x63, 0x68,
 0x6d, 0x65, 0x6e, 0x74, 0x49, 0x44, 0x22, 0x44, 0x0a, 0x04, 0x42, 0x6f, 0x64, 0x79, 0x12, 0x20,
 0x0a, 0x0b, 0x63, 0x6f, 0x6e, 0x74, 0x65, 0x6e, 0x74, 0x54, 0x79, 0x70, 0x65, 0x18, 0x01, 0x20,
 0x01, 0x28, 0x09, 0x52, 0x0b, 0x63, 0x6f, 0x6e, 0x74, 0x65, 0x6e, 0x74, 0x54, 0x79, 0x70, 0x65,
 0x12, 0x1a, 0x0a, 0x08, 0x6d, 0x61, 0x69, 0x6e, 0x42, 0x6f, 0x64, 0x79, 0x18, 0x02, 0x20, 0x01,
 0x28, 0x0c, 0x52, 0x08, 0x6d, 0x61, 0x69, 0x6e, 0x42, 0x6f, 0x64, 0x79, 0x22, 0x37, 0x0a, 0x07,
 0x41, 0x64, 0x64, 0x72, 0x65, 0x73, 0x73, 0x12, 0x12, 0x0a, 0x04, 0x6e, 0x61, 0x6d, 0x65, 0x18,
 0x01, 0x20, 0x01, 0x28, 0x09, 0x52, 0x04, 0x6e, 0x61, 0x6d, 0x65, 0x12, 0x18, 0x0a, 0x07, 0x61,
 0x64, 0x64, 0x72, 0x65, 0x73, 0x73, 0x18, 0x02, 0x20, 0x01, 0x28, 0x09, 0x52, 0x07, 0x61, 0x64,
 0x64, 0x72, 0x65, 0x73, 0x73, 0x22, 0x28, 0x0a, 0x10, 0x53, 0x65, 0x6e, 0x64, 0x4d, 0x61, 0x69,
 0x6c, 0x52, 0x65, 0x73, 0x70, 0x6f, 0x6e, 0x73, 0x65, 0x12, 0x14, 0x0a, 0x05, 0x4d, 0x73, 0x67,
 0x49, 0x44, 0x18, 0x01, 0x20, 0x01, 0x28, 0x09, 0x52, 0x05, 0x4d, 0x73, 0x67, 0x49, 0x44, 0x22,
 0x1c, 0x0a, 0x06, 0x53, 0x65, 0x72, 0x76, 0x65, 0x72, 0x12, 0x12, 0x0a, 0x04, 0x4e, 0x61, 0x6d,
 0x65, 0x18, 0x01, 0x20, 0x01, 0x28, 0x09, 0x52, 0x04, 0x4e, 0x61, 0x6d, 0x65, 0x22, 0x5c, 0x0a,
 0x0a, 0x55, 0x70, 0x6c, 0x6f, 0x61, 0x64, 0x46, 0x69, 0x6c, 0x65, 0x12, 0x12, 0x0a, 0x04, 0x4e,
 0x61, 0x6d, 0x65, 0x18, 0x01, 0x20, 0x01, 0x28, 0x09, 0x52, 0x04, 0x4e, 0x61, 0x6d, 0x65, 0x12,
 0x20, 0x0a, 0x0b, 0x43, 0x6f, 0x6e, 0x74, 0x65, 0x6e, 0x74, 0x54, 0x79, 0x70, 0x65, 0x18, 0x02,
 0x20, 0x01, 0x28, 0x09, 0x52, 0x0b, 0x43, 0x6f, 0x6e, 0x74, 0x65, 0x6e, 0x74, 0x54, 0x79, 0x70,
 0x65, 0x12, 0x18, 0x0a, 0x07, 0x43, 0x6f, 0x6e, 0x74, 0x65, 0x6e, 0x74, 0x18, 0x03, 0x20, 0x01,
 0x28, 0x0c, 0x52, 0x07, 0x43, 0x6f, 0x6e, 0x74, 0x65, 0x6e, 0x74, 0x22, 0x28, 0x0a, 0x0e, 0x55,
 0x70, 0x6c, 0x6f, 0x61, 0x64, 0x52, 0x65, 0x73, 0x70, 0x6f, 0x6e, 0x73, 0x65, 0x12, 0x16, 0x0a,
 0x06, 0x46, 0x69, 0x6c, 0x65, 0x49, 0x44, 0x18, 0x01, 0x20, 0x01, 0x28, 0x09, 0x52, 0x06, 0x46,
 0x69, 0x6c, 0x65, 0x49, 0x44, 0x22, 0x4e, 0x0a, 0x04, 0x55, 0x73, 0x65, 0x72, 0x12, 0x12, 0x0a,
 0x04, 0x4e, 0x61, 0x6d, 0x65, 0x18, 0x01, 0x20, 0x01, 0x28, 0x09, 0x52, 0x04, 0x4e, 0x61, 0x6d,
 0x65, 0x12, 0x1a, 0x0a, 0x08, 0x50, 0x61, 0x73, 0x73, 0x77, 0x6f, 0x72, 0x64, 0x18, 0x02, 0x20,

0x01, 0x28, 0x09, 0x52, 0x08, 0x50, 0x61, 0x73, 0x73, 0x77, 0x6f, 0x72, 0x64, 0x1
 2, 0x16, 0x0a,
 0x06, 0x57, 0x65, 0x69, 0x67, 0x68, 0x74, 0x18, 0x03, 0x20, 0x01, 0x28, 0x05, 0x5
 2, 0x06, 0x57,
 0x65, 0x69, 0x67, 0x68, 0x74, 0x22, 0x32, 0x0a, 0x0c, 0x55, 0x73, 0x65, 0x72, 0x5
 2, 0x65, 0x73,
 0x70, 0x6f, 0x6e, 0x73, 0x65, 0x12, 0x0e, 0x0a, 0x02, 0x49, 0x44, 0x18, 0x01, 0x20,
 0x01, 0x28,
 0x09, 0x52, 0x02, 0x49, 0x44, 0x12, 0x12, 0x0a, 0x04, 0x4e, 0x61, 0x6d, 0x65, 0x1
 8, 0x02, 0x20,
 0x01, 0x28, 0x09, 0x52, 0x04, 0x4e, 0x61, 0x6d, 0x65, 0x22, 0x31, 0x0a, 0x0a, 0x5
 3, 0x65, 0x72,
 0x76, 0x65, 0x72, 0x4c, 0x69, 0x73, 0x74, 0x12, 0x23, 0x0a, 0x05, 0x49, 0x74, 0x6
 5, 0x6d, 0x73,
 0x18, 0x01, 0x20, 0x03, 0x28, 0x0b, 0x32, 0x0d, 0x2e, 0x70, 0x72, 0x6f, 0x74, 0x6f,
 0x2e, 0x53,
 0x65, 0x72, 0x76, 0x65, 0x72, 0x52, 0x05, 0x49, 0x74, 0x65, 0x6d, 0x73, 0x32, 0xb
 6, 0x02, 0x0a,
 0x07, 0x4d, 0x61, 0x69, 0x6c, 0x42, 0x6f, 0x78, 0x12, 0x32, 0x0a, 0x04, 0x53, 0x65,
 0x6e, 0x64,
 0x12, 0x0f, 0x2e, 0x70, 0x72, 0x6f, 0x74, 0x6f, 0x2e, 0x4d, 0x61, 0x69, 0x6c, 0x54,
 0x61, 0x73,
 0x6b, 0x1a, 0x17, 0x2e, 0x70, 0x72, 0x6f, 0x74, 0x6f, 0x2e, 0x53, 0x65, 0x6e, 0x64,
 0x4d, 0x61,
 0x69, 0x6c, 0x52, 0x65, 0x73, 0x70, 0x6f, 0x6e, 0x73, 0x65, 0x22, 0x00, 0x12, 0x39,
 0x0a, 0x0a,
 0x4c, 0x69, 0x73, 0x74, 0x53, 0x65, 0x72, 0x76, 0x65, 0x72, 0x12, 0x16, 0x2e, 0x6
 7, 0x6f, 0x6f,
 0x67, 0x6c, 0x65, 0x2e, 0x70, 0x72, 0x6f, 0x74, 0x6f, 0x62, 0x75, 0x66, 0x2e, 0x45,
 0x6d, 0x70,
 0x74, 0x79, 0x1a, 0x11, 0x2e, 0x70, 0x72, 0x6f, 0x74, 0x6f, 0x2e, 0x53, 0x65, 0x72,
 0x76, 0x65,
 0x72, 0x4c, 0x69, 0x73, 0x74, 0x22, 0x00, 0x12, 0x36, 0x0a, 0x06, 0x55, 0x70, 0x6
 c, 0x6f, 0x61,
 0x64, 0x12, 0x11, 0x2e, 0x70, 0x72, 0x6f, 0x74, 0x6f, 0x2e, 0x55, 0x70, 0x6c, 0x6f,
 0x61, 0x64,
 0x46, 0x69, 0x6c, 0x65, 0x1a, 0x15, 0x2e, 0x70, 0x72, 0x6f, 0x74, 0x6f, 0x2e, 0x55,
 0x70, 0x6c,
 0x6f, 0x61, 0x64, 0x52, 0x65, 0x73, 0x70, 0x6f, 0x6e, 0x73, 0x65, 0x22, 0x00, 0x28,
 0x01, 0x12,
 0x27, 0x0a, 0x05, 0x57, 0x61, 0x74, 0x63, 0x68, 0x12, 0x0d, 0x2e, 0x70, 0x72, 0x6f,
 0x74, 0x6f,
 0x2e, 0x53, 0x65, 0x72, 0x76, 0x65, 0x72, 0x1a, 0x0b, 0x2e, 0x70, 0x72, 0x6f, 0x74,
 0x6f, 0x2e,
 0x4d, 0x61, 0x69, 0x6c, 0x22, 0x00, 0x30, 0x01, 0x12, 0x2e, 0x0a, 0x08, 0x52, 0x6
 5, 0x67, 0x69,
 0x73, 0x74, 0x65, 0x72, 0x12, 0x0b, 0x2e, 0x70, 0x72, 0x6f, 0x74, 0x6f, 0x2e, 0x55,
 0x73, 0x65,
 0x72, 0x1a, 0x13, 0x2e, 0x70, 0x72, 0x6f, 0x74, 0x6f, 0x2e, 0x55, 0x73, 0x65, 0x72,
 0x52, 0x65,

```

    0x73, 0x70, 0x6f, 0x6e, 0x73, 0x65, 0x22, 0x00, 0x12, 0x2b, 0x0a, 0x05, 0x4c, 0x6f,
    0x67, 0x69,
    0x6e, 0x12, 0x0b, 0x2e, 0x70, 0x72, 0x6f, 0x74, 0x6f, 0x2e, 0x55, 0x73, 0x65, 0x72,
    0x1a, 0x13,
    0x2e, 0x70, 0x72, 0x6f, 0x74, 0x6f, 0x2e, 0x55, 0x73, 0x65, 0x72, 0x52, 0x65, 0x73,
    0x70, 0x6f,
    0x6e, 0x73, 0x65, 0x22, 0x00, 0x42, 0x09, 0x5a, 0x07, 0x2e, 0x2f, 0x70, 0x72, 0x6f,
    0x74, 0x6f,
    0x62, 0x06, 0x70, 0x72, 0x6f, 0x74, 0x6f, 0x33,
}
var (
    file_mail_proto_rawDescOnce sync.Once
    file_mail_proto_rawDescData = file_mail_proto_rawDesc
)
func file_mail_proto_rawDescGZIP() []byte {
    file_mail_proto_rawDescOnce.Do(func() {
        file_mail_proto_rawDescData = protoimpl.X.CompressGZIP(file_mail_proto_rawDescData)
    })
    return file_mail_proto_rawDescData
}
var file_mail_proto_msgTypes = make([]protoimpl.MessageInfo, 12)
var file_mail_proto_goTypes = []interface{}{
    (*Mail)(nil),           // 0: proto.Mail
    (*MailTask)(nil),       // 1: proto.MailTask
    (*AttachmentRequest)(nil), // 2: proto.AttachmentRequest
    (*Body)(nil),           // 3: proto.Body
    (*Address)(nil),        // 4: proto.Address
    (*SendMailResponse)(nil), // 5: proto.SendMailResponse
    (*Server)(nil),         // 6: proto.Server
    (*UploadFile)(nil),     // 7: proto.UploadFile
    (*UploadResponse)(nil), // 8: proto.UploadResponse
    (*User)(nil),           // 9: proto.User
    (*UserResponse)(nil),   // 10: proto.UserResponse
    (*ServerList)(nil),     // 11: proto.ServerList
    (*empty.Empty)(nil),    // 12: google.protobuf.Empty
}
var file_mail_proto_depIdxs = []int32{
    4, // 0: proto.Mail.To:type_name -> proto.Address
    4, // 1: proto.Mail.From:type_name -> proto.Address
    3, // 2: proto.Mail.Text:type_name -> proto.Body
    3, // 3: proto.Mail.Attachment:type_name -> proto.Body
    4, // 4: proto.MailTask.From:type_name -> proto.Address
    4, // 5: proto.MailTask.To:type_name -> proto.Address
    4, // 6: proto.MailTask.Cc:type_name -> proto.Address
    4, // 7: proto.MailTask.Bcc:type_name -> proto.Address
    3, // 8: proto.MailTask.Text:type_name -> proto.Body
    2, // 9: proto.MailTask.Attachment:type_name -> proto.AttachmentRequest
    6, // 10: proto.ServerList.Items:type_name -> proto.Server
    1, // 11: proto.MailBox.Send:input_type -> proto.MailTask
}

```

```

12, // 12: proto.MailBox.ListServer:input_type -> google.protobuf.Empty
7, // 13: proto.MailBox.Upload:input_type -> proto.UploadFile
6, // 14: proto.MailBox.Watch:input_type -> proto.Server
9, // 15: proto.MailBox.Register:input_type -> proto.User
9, // 16: proto.MailBox.Login:input_type -> proto.User
5, // 17: proto.MailBox.Send:output_type -> proto.SendMailResponse
11, // 18: proto.MailBox.ListServer:output_type -> proto.ServerList
8, // 19: proto.MailBox.Upload:output_type -> proto.UploadResponse
0, // 20: proto.MailBox.Watch:output_type -> proto.Mail
10, // 21: proto.MailBox.Register:output_type -> proto.UserResponse
10, // 22: proto.MailBox.Login:output_type -> proto.UserResponse
17, // [17:23] is the sub-list for method output_type
11, // [11:17] is the sub-list for method input_type
11, // [11:11] is the sub-list for extension type_name
11, // [11:11] is the sub-list for extension extendee
0, // [0:11] is the sub-list for field type_name
}
func init() { file_mail_proto_init() }
func file_mail_proto_init() {
    if File_mail_proto != nil {
        return
    }
    if !protoimpl.UnsafeEnabled {
        file_mail_proto_msgTypes[0].Exporter = func(v interface{}, i int) interface{} {
            switch v := v.(*Mail); i {
                case 0:
                    return &v.state
                case 1:
                    return &v.sizeCache
                case 2:
                    return &v.unknownFields
                default:
                    return nil
            }
        }
        file_mail_proto_msgTypes[1].Exporter = func(v interface{}, i int) interface{} {
            switch v := v.(*MailTask); i {
                case 0:
                    return &v.state
                case 1:
                    return &v.sizeCache
                case 2:
                    return &v.unknownFields
                default:
                    return nil
            }
        }
        file_mail_proto_msgTypes[2].Exporter = func(v interface{}, i int) interface{} {
            switch v := v.(*AttachmentRequest); i {
                case 0:

```

```

        return &v.state
    case 1:
        return &v.sizeCache
    case 2:
        return &v.unknownFields
    default:
        return nil
    }
}
file_mail_proto_msgTypes[3].Exporter = func(v interface{}, i int) interface{} {
    switch v := v.(*Body); i {
    case 0:
        return &v.state
    case 1:
        return &v.sizeCache
    case 2:
        return &v.unknownFields
    default:
        return nil
    }
}
file_mail_proto_msgTypes[4].Exporter = func(v interface{}, i int) interface{} {
    switch v := v.(*Address); i {
    case 0:
        return &v.state
    case 1:
        return &v.sizeCache
    case 2:
        return &v.unknownFields
    default:
        return nil
    }
}
file_mail_proto_msgTypes[5].Exporter = func(v interface{}, i int) interface{} {
    switch v := v.(*SendMailResponse); i {
    case 0:
        return &v.state
    case 1:
        return &v.sizeCache
    case 2:
        return &v.unknownFields
    default:
        return nil
    }
}
file_mail_proto_msgTypes[6].Exporter = func(v interface{}, i int) interface{} {
    switch v := v.(*Server); i {
    case 0:
        return &v.state
    case 1:

```

```

        return &v.sizeCache
    case 2:
        return &v.unknownFields
    default:
        return nil
    }
}
file_mail_proto_msgTypes[7].Exporter = func(v interface{}, i int) interface{} {
    switch v := v.(*UploadFile); i {
    case 0:
        return &v.state
    case 1:
        return &v.sizeCache
    case 2:
        return &v.unknownFields
    default:
        return nil
    }
}
file_mail_proto_msgTypes[8].Exporter = func(v interface{}, i int) interface{} {
    switch v := v.(*UploadResponse); i {
    case 0:
        return &v.state
    case 1:
        return &v.sizeCache
    case 2:
        return &v.unknownFields
    default:
        return nil
    }
}
file_mail_proto_msgTypes[9].Exporter = func(v interface{}, i int) interface{} {
    switch v := v.(*User); i {
    case 0:
        return &v.state
    case 1:
        return &v.sizeCache
    case 2:
        return &v.unknownFields
    default:
        return nil
    }
}
file_mail_proto_msgTypes[10].Exporter = func(v interface{}, i int) interface{} {
    switch v := v.(*UserResponse); i {
    case 0:
        return &v.state
    case 1:
        return &v.sizeCache
    case 2:

```

```

        return &v.unknownFields
    default:
        return nil
    }
}
file_mail_proto_msgTypes[11].Exporter = func(v interface{}, i int) interface{} {
    switch v := v.(*ServerList); i {
    case 0:
        return &v.state
    case 1:
        return &v.sizeCache
    case 2:
        return &v.unknownFields
    default:
        return nil
    }
}
}
type x struct{}
out := protoimpl.TypeBuilder{
    File: protoimpl.DescBuilder{
        GoPackagePath: reflect.TypeOf(x{}).PkgPath(),
        RawDescriptor: file_mail_proto_rawDesc,
        NumEnums:      0,
        NumMessages:    12,
        NumExtensions: 0,
        NumServices:    1,
    },
    GoTypes:          file_mail_proto_goTypes,
    DependencyIndexes: file_mail_proto_depIdxs,
    MessageInfos:     file_mail_proto_msgTypes,
}.Build()
File_mail_proto = out.File
file_mail_proto_rawDesc = nil
file_mail_proto_goTypes = nil
file_mail_proto_depIdxs = nil
}
// Code generated by protoc-gen-go-grpc. DO NOT EDIT.
package proto
import (
    context "context"
    empty "github.com/golang/protobuf/ptypes/empty"
    grpc "google.golang.org/grpc"
    codes "google.golang.org/grpc/codes"
    status "google.golang.org/grpc/status"
)
const _ = grpc.SupportPackageIsVersion7
type MailBoxClient interface {
    Send(ctx context.Context, in *MailTask, opts ...grpc.CallOption) (*SendMailResponse, error)

```

```

    ListServer(ctx context.Context, in *empty.Empty, opts ...grpc.CallOption) (*ServerList, error)
    Upload(ctx context.Context, opts ...grpc.CallOption) (MailBox_UploadClient, error)
    Watch(ctx context.Context, in *Server, opts ...grpc.CallOption) (MailBox_WatchClient, error)
    Register(ctx context.Context, in *User, opts ...grpc.CallOption) (*UserResponse, error)
    Login(ctx context.Context, in *User, opts ...grpc.CallOption) (*UserResponse, error)
}
type mailBoxClient struct {
    cc grpc.ClientConnInterface
}
func NewMailBoxClient(cc grpc.ClientConnInterface) MailBoxClient {
    return &mailBoxClient{cc}
}
func (c *mailBoxClient) Send(ctx context.Context, in *MailTask, opts ...grpc.CallOption) (*SendMailResponse, error) {
    out := new(SendMailResponse)
    err := c.cc.Invoke(ctx, "/proto.MailBox/Send", in, out, opts...)
    if err != nil {
        return nil, err
    }
    return out, nil
}
func (c *mailBoxClient) ListServer(ctx context.Context, in *empty.Empty, opts ...grpc.CallOption) (*ServerList, error) {
    out := new(ServerList)
    err := c.cc.Invoke(ctx, "/proto.MailBox/ListServer", in, out, opts...)
    if err != nil {
        return nil, err
    }
    return out, nil
}
func (c *mailBoxClient) Upload(ctx context.Context, opts ...grpc.CallOption) (MailBox_UploadClient, error) {
    stream, err := c.cc.NewStream(ctx, &MailBox_ServiceDesc.Streams[0], "/proto.MailBox/Upload", opts...)
    if err != nil {
        return nil, err
    }
    x := &mailBoxUploadClient{stream}
    return x, nil
}
type MailBox_UploadClient interface {
    Send(*UploadFile) error
    CloseAndRecv() (*UploadResponse, error)
    grpc.ClientStream
}
type mailBoxUploadClient struct {
    grpc.ClientStream
}

```

```

func (x *mailBoxUploadClient) Send(m *UploadFile) error {
    return x.ClientStream.SendMsg(m)
}

func (x *mailBoxUploadClient) CloseAndRecv() (*UploadResponse, error) {
    if err := x.ClientStream.CloseSend(); err != nil {
        return nil, err
    }
    m := new(UploadResponse)
    if err := x.ClientStream.RecvMsg(m); err != nil {
        return nil, err
    }
    return m, nil
}

func (c *mailBoxClient) Watch(ctx context.Context, in *Server, opts ...grpc.CallOption) (MailBox_WatchClient, error) {
    stream, err := c.cc.NewStream(ctx, &MailBox_ServiceDesc.Streams[1], "/proto.MailBox/Watch", opts...)
    if err != nil {
        return nil, err
    }
    x := &mailBoxWatchClient{stream}
    if err := x.ClientStream.SendMsg(in); err != nil {
        return nil, err
    }
    if err := x.ClientStream.CloseSend(); err != nil {
        return nil, err
    }
    return x, nil
}

type MailBox_WatchClient interface {
    Recv() (*Mail, error)
    grpc.ClientStream
}

type mailBoxWatchClient struct {
    grpc.ClientStream
}

func (x *mailBoxWatchClient) Recv() (*Mail, error) {
    m := new(Mail)
    if err := x.ClientStream.RecvMsg(m); err != nil {
        return nil, err
    }
    return m, nil
}

func (c *mailBoxClient) Register(ctx context.Context, in *User, opts ...grpc.CallOption) (*UserResponse, error) {
    out := new(UserResponse)
    err := c.cc.Invoke(ctx, "/proto.MailBox/Register", in, out, opts...)
    if err != nil {
        return nil, err
    }
}

```

```

    return out, nil
}
func (c *mailBoxClient) Login(ctx context.Context, in *User, opts ...grpc.CallOption) (*UserResponse, error) {
    out := new(UserResponse)
    err := c.cc.Invoke(ctx, "/proto.MailBox/Login", in, out, opts...)
    if err != nil {
        return nil, err
    }
    return out, nil
}
type MailBoxServer interface {
    Send(context.Context, *MailTask) (*SendMailResponse, error)
    ListServer(context.Context, *empty.Empty) (*ServerList, error)
    Upload(MailBox_UploadServer) error
    Watch(*Server, MailBox_WatchServer) error
    Register(context.Context, *User) (*UserResponse, error)
    Login(context.Context, *User) (*UserResponse, error)
    mustEmbedUnimplementedMailBoxServer()
}
type UnimplementedMailBoxServer struct {
}
func (UnimplementedMailBoxServer) Send(context.Context, *MailTask) (*SendMailResponse, error) {
    return nil, status.Errorf(codes.Unimplemented, "method Send not implemented")
}
func (UnimplementedMailBoxServer) ListServer(context.Context, *empty.Empty) (*ServerList, error) {
    return nil, status.Errorf(codes.Unimplemented, "method ListServer not implemented")
}
func (UnimplementedMailBoxServer) Upload(MailBox_UploadServer) error {
    return status.Errorf(codes.Unimplemented, "method Upload not implemented")
}
func (UnimplementedMailBoxServer) Watch(*Server, MailBox_WatchServer) error {
    return status.Errorf(codes.Unimplemented, "method Watch not implemented")
}
func (UnimplementedMailBoxServer) Register(context.Context, *User) (*UserResponse, error) {
    return nil, status.Errorf(codes.Unimplemented, "method Register not implemented")
}
func (UnimplementedMailBoxServer) Login(context.Context, *User) (*UserResponse, error) {
    return nil, status.Errorf(codes.Unimplemented, "method Login not implemented")
}
func (UnimplementedMailBoxServer) mustEmbedUnimplementedMailBoxServer() {}
type UnsafeMailBoxServer interface {
    mustEmbedUnimplementedMailBoxServer()
}
func RegisterMailBoxServer(s grpc.ServiceRegistrar, srv MailBoxServer) {
    s.RegisterService(&MailBox_ServiceDesc, srv)
}

```

```

}
func _MailBox_Send_Handler(srv interface{}, ctx context.Context, dec func(interface{}) error, interceptor grpc.UnaryServerInterceptor) (interface{}, error) {
    in := new(MailTask)
    if err := dec(in); err != nil {
        return nil, err
    }
    if interceptor == nil {
        return srv.(MailBoxServer).Send(ctx, in)
    }
    info := &grpc.UnaryServerInfo{
        Server:    srv,
        FullMethod: "/proto.MailBox/Send",
    }
    handler := func(ctx context.Context, req interface{}) (interface{}, error) {
        return srv.(MailBoxServer).Send(ctx, req.(*MailTask))
    }
    return interceptor(ctx, in, info, handler)
}

func _MailBox_ListServer_Handler(srv interface{}, ctx context.Context, dec func(interface{}) error, interceptor grpc.UnaryServerInterceptor) (interface{}, error) {
    in := new(empty.Empty)
    if err := dec(in); err != nil {
        return nil, err
    }
    if interceptor == nil {
        return srv.(MailBoxServer).ListServer(ctx, in)
    }
    info := &grpc.UnaryServerInfo{
        Server:    srv,
        FullMethod: "/proto.MailBox/ListServer",
    }
    handler := func(ctx context.Context, req interface{}) (interface{}, error) {
        return srv.(MailBoxServer).ListServer(ctx, req.(*empty.Empty))
    }
    return interceptor(ctx, in, info, handler)
}

func _MailBox_Upload_Handler(srv interface{}, stream grpc.ServerStream) error {
    return srv.(MailBoxServer).Upload(&mailBoxUploadServer{stream})
}

type MailBox_UploadServer interface {
    SendAndClose(*UploadResponse) error
    Recv() (*UploadFile, error)
    grpc.ServerStream
}

type mailBoxUploadServer struct {
    grpc.ServerStream
}

func (x *mailBoxUploadServer) SendAndClose(m *UploadResponse) error {
    return x.ServerStream.SendMsg(m)
}

```

```

}
func (x *mailBoxUploadServer) Recv() (*UploadFile, error) {
    m := new(UploadFile)
    if err := x.ServerStream.RecvMsg(m); err != nil {
        return nil, err
    }
    return m, nil
}
func _MailBox_Watch_Handler(srv interface{}, stream grpc.ServerStream) error {
    m := new(Server)
    if err := stream.RecvMsg(m); err != nil {
        return err
    }
    return srv.(MailBoxServer).Watch(m, &mailBoxWatchServer{stream})
}
type MailBox_WatchServer interface {
    Send(*Mail) error
    grpc.ServerStream
}
type mailBoxWatchServer struct {
    grpc.ServerStream
}
func (x *mailBoxWatchServer) Send(m *Mail) error {
    return x.ServerStream.SendMsg(m)
}
func _MailBox_Register_Handler(srv interface{}, ctx context.Context, dec func(interface{}) error, interceptor grpc.UnaryServerInterceptor) (interface{}, error) {
    in := new(User)
    if err := dec(in); err != nil {
        return nil, err
    }
    if interceptor == nil {
        return srv.(MailBoxServer).Register(ctx, in)
    }
    info := &grpc.UnaryServerInfo{
        Server:    srv,
        FullMethod: "/proto.MailBox/Register",
    }
    handler := func(ctx context.Context, req interface{}) (interface{}, error) {
        return srv.(MailBoxServer).Register(ctx, req.(*User))
    }
    return interceptor(ctx, in, info, handler)
}
func _MailBox_Login_Handler(srv interface{}, ctx context.Context, dec func(interface{}) error, interceptor grpc.UnaryServerInterceptor) (interface{}, error) {
    in := new(User)
    if err := dec(in); err != nil {
        return nil, err
    }
    if interceptor == nil {

```

```

        return srv.(MailBoxServer).Login(ctx, in)
    }
    info := &grpc.UnaryServerInfo{
        Server:      srv,
        FullMethod: "/proto.MailBox/Login",
    }
    handler := func(ctx context.Context, req interface{}) (interface{}, error) {
        return srv.(MailBoxServer).Login(ctx, req.(*User))
    }
    return interceptor(ctx, in, info, handler)
}
var MailBox_ServiceDesc = grpc.ServiceDesc{
    ServiceName: "proto.MailBox",
    HandlerType: (*MailBoxServer)(nil),
    Methods: []grpc.MethodDesc{
        {
            MethodName: "Send",
            Handler:    _MailBox_Send_Handler,
        },
        {
            MethodName: "ListServer",
            Handler:    _MailBox_ListServer_Handler,
        },
        {
            MethodName: "Register",
            Handler:    _MailBox_Register_Handler,
        },
        {
            MethodName: "Login",
            Handler:    _MailBox_Login_Handler,
        },
    },
    Streams: []grpc.StreamDesc{
        {
            StreamName:    "Upload",
            Handler:        _MailBox_Upload_Handler,
            ClientStreams: true,
        },
        {
            StreamName:    "Watch",
            Handler:        _MailBox_Watch_Handler,
            ServerStreams: true,
        },
    },
    Metadata: "mail.proto",
}
package smtp
import (
    "bytes"
    "crypto/sha256"

```

```

"encoding/base64"
"errors"
"fmt"
. "gmail/pkg/config"
"gmail/pkg/db"
"gmail/pkg/util/random"
"io"
"net"
"net/smtp"
"strings"
"time"
)
const (
    SplitLine      = "\r\n"
    Boundary       = "GoBoundary"
    BoundarySign   = "--"
    DefaultEncoding = "base64"
)
type MailTask struct {
    MessageId string
    From       string    `json:"from"`
    To         []string `json:"to"`
    Cc         []string `json:"cc"`
    Bcc        []string `json:"bcc"`
    Subject    string   `json:"subject"`
    ReplyId    string   `json:"reply_id"`
    Body       []byte   `json:"body"`
    ContentType string   `json:"content_type"`
    Attachment Attachment `json:"attachment"`
}
type Attachment struct {
    db.File
    WithFile bool `json:"with_file"`
}
type Tool interface {
    Send(task MailTask) (string, error)
}
type MailTool struct {
    buf *bytes.Buffer
    Host string
    Auth smtp.Auth
    Port string
}
func (c *MailTool) generatorMessageId() string {
    randomByte, _ := random.Alpha(uint64(32))
    hash := sha256.New()
    hash.Write(randomByte)
    randomStr := base64.StdEncoding.EncodeToString(hash.Sum(nil))
    randomStr = strings.ReplaceAll(randomStr, "=", "")
    randomStr = strings.ReplaceAll(randomStr, "/", "")

```

```

    randomStr = strings.ReplaceAll(randomStr, "+", "")
    return fmt.Sprintf("<%s@%s>", randomStr, c.Host)
}

func (c *MailTool) writeHeader(Header map[string]string) {
    header := ""
    for key, value := range Header {
        header += key + ":" + value + SplitLine
    }
    c.buf.WriteString(header)
    c.WriteSplitLine()
}

func (c *MailTool) writeFile(reader io.Reader) {
    file, err := io.ReadAll(reader)
    if err != nil {
        panic(err.Error())
    }
    payload := make([]byte, base64.StdEncoding.EncodedLen(len(file)))
    base64.StdEncoding.Encode(payload, file)
    for index, line := 0, len(payload); index < line; index++ {
        c.buf.WriteByte(payload[index])
        if (index+1)%76 == 0 {
            c.buf.WriteString(SplitLine)
        }
    }
}

func (c *MailTool) WriteSplitLine() {
    c.buf.WriteString(SplitLine)
}

func (c *MailTool) WriteBody(body []byte) {
    c.buf.WriteString(SplitLine)
    c.buf.Write(body)
    c.buf.WriteString(SplitLine)
}

func (c *MailTool) buildHeader(task MailTask) map[string]string {
    Header := make(map[string]string)
    Header["From"] = task.From
    Header["To"] = strings.Join(task.To, ";")
    Header["Cc"] = strings.Join(task.Cc, ";")
    Header["Bcc"] = strings.Join(task.Bcc, ";")
    Header["Subject"] = task.Subject
    Header["Message-Id"] = task.MessageId
    Header["In-Reply-To"] = task.ReplyId
    Header["References"] = task.ReplyId
    Header["Content-Type"] = "multipart/mixed;boundary=" + Boundary
    Header["Mime-Version"] = "1.0"
    Header["Date"] = time.Now().String()
    return Header
}

func (c *MailTool) writeContentType(contentType string) {
    c.buf.WriteString("Content-Type:" + contentType)
}

```

```

}
func (c *MailTool) writeEncoding(encode string) {
    c.buf.WriteString("Content-Transfer-Encoding:" + encode)
}
func (c *MailTool) writeContentDisposition() {
    c.buf.WriteString("Content-Disposition:attachment")
}
func (c *MailTool) writeContentTypeAndName(ty, name string) {
    c.buf.WriteString(fmt.Sprintf("Content-Type:%s;name=\"%s\"", ty, name))
}
func (c *MailTool) writeAttachment(att Attachment) {
    if att.WithFile {
        return
    }
    c.WriteSplitLine()
    c.writeBoundary(false)
    c.WriteSplitLine()
    c.writeEncoding(DefaultEncoding)
    c.WriteSplitLine()
    c.writeContentDisposition()
    c.WriteSplitLine()
    c.writeContentTypeAndName(att.ContentType(), att.Name())
    c.WriteSplitLine()
    c.writeFile(att.File)
    _ = att.Close()
}
func (c *MailTool) writeBoundary(end bool) {
    if end {
        c.buf.WriteString(BoundarySign + Boundary + BoundarySign)
    } else {
        c.buf.WriteString(BoundarySign + Boundary)
    }
}
func (c *MailTool) build(task MailTask) *bytes.Buffer {
    c.writeHeader(c.buildHeader(task))
    c.WriteSplitLine()
    c.writeBoundary(false)
    c.WriteSplitLine()
    c.writeContentType(task.ContentType)
    c.WriteSplitLine()
    c.WriteBody(task.Body)
    c.WriteSplitLine()
    c.writeAttachment(task.Attachment)
    c.WriteSplitLine()
    c.writeBoundary(true)
    return c.buf
}
func (c *MailTool) Send(task MailTask) (messageId string, err error) {
    if task.From == "" {
        err = errors.New("unknown json string")
    }
}

```

```

        return
    }
    messageId = c.generatorMessageId()
    task.MessageId = messageId
    buffer := c.build(task)
    c.reset()
    err = smtp.SendMail(net.JoinHostPort(c.Host, c.Port), c.Auth, task.From, task.To, buffer.
Bytes())
    return
}
func (c *MailTool) reset() {
    c.buf.Reset()
}
func NewClient(smtpConfig Smtip) Tool {
    //auth
    MailSender := &MailTool{
        Port: smtpConfig.Port,
        Host: smtpConfig.Host,
        buf: bytes.NewBuffer(nil),
        Auth: smtp.PlainAuth("", smtpConfig.User, smtpConfig.Password, smtpConfig.Host),
    }
    return MailSender
}
package random
import (
    "crypto/rand"
    "encoding/binary"
    "errors"
    "math"
)
func Uint64Range(start, end uint64) (uint64, error) {
    var val uint64
    var err error
    if start >= end {
        return val, errors.New("start value must be less than end value")
    }
    size := end - start // Get range size
    min := (math.MaxUint64 - size) % size
    for {
        val, err = Uint64()
        if err != nil {
            return val, err
        }
        if val >= min {
            break
        }
    }
    val = val % size
    // End arc4random_uniform
    // Add start to val to shift numbers to correct range.

```

```

    return val + start, nil
}
func Chars(charset string, n uint64) ([]byte, error) {
    if n == 0 {
        return []byte(""), errors.New("requested string length cannot be 0")
    }
    if len(charset) == 0 {
        return []byte(""), errors.New("charset cannot be empty")
    }
    length := uint64(len(charset))
    b := make([]byte, n)
    for i := range b {
        j, err := Uint64Range(0, length)
        if err != nil {
            return []byte(""), err
        }
        b[i] = charset[j]
    }
    return b, nil
}
func Alpha(n uint64) ([]byte, error) {
    charset := "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
    return Chars(charset, n)
}
func AlphaNum(n uint64) ([]byte, error) {
    charset := "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"
    return Chars(charset, n)
}
func Uint8() (uint8, error) {
    var bytes [1]byte
    _, err := rand.Read(bytes[:])
    if err != nil {
        return uint8(0), err
    }
    return bytes[0], nil
}
func Int8() (int8, error) {
    i, err := Uint8()
    if err != nil {
        return int8(0), err
    }
    return int8(i), nil
}
func Uint16() (uint16, error) {
    var bytes [2]byte
    _, err := rand.Read(bytes[:])
    if err != nil {
        return uint16(0), err
    }
}

```

```

        return binary.LittleEndian.Uint16(bytes[:]), nil
    }
    func Int16() (int16, error) {
        i, err := Uint16()
        if err != nil {
            return int16(0), err
        }
        return int16(i), nil
    }
    func Uint32() (uint32, error) {
        var bytes [4]byte
        _, err := rand.Read(bytes[:])
        if err != nil {
            return uint32(0), err
        }
        return binary.LittleEndian.Uint32(bytes[:]), nil
    }
    func Int32() (int32, error) {
        i, err := Uint32()
        if err != nil {
            return int32(0), err
        }
        return int32(i), nil
    }
    func Uint64() (uint64, error) {
        var bytes [8]byte
        _, err := rand.Read(bytes[:])
        if err != nil {
            return uint64(0), err
        }
        return binary.LittleEndian.Uint64(bytes[:]), nil
    }
    func Int64() (int64, error) {
        i, err := Uint64()
        if err != nil {
            return int64(0), err
        }
        return int64(i), nil
    }
    package sortlist
    const (
        DefaultLoadFactor = 1000
    )
    package sortlist
    import (
        "fmt"
        "sort"
    )
    type Compare[T comparable] func(a, b T) int
    var IntCompare Compare[int] = func(a, b int) int {

```

```

    if a > b {
        return 1
    }
    if a < b {
        return -1
    }
    return 0
}
func BisectRight[T comparable](l []T, c Compare[T], target T) int {
    return sort.Search(len(l), func(i int) bool {
        return c(l[i], target) > 0
    })
}
type SortedList[T comparable] struct {
    offset int
    load   int
    maxes  []T
    lists  [][]T
    indexes []int //index sum tree
    size   int
    c      Compare[T]
}
func (l *SortedList[T]) Push(a T) {
    l.size++
    if len(l.maxes) == 0 {
        l.maxes = append(l.maxes, a)
        l.lists = append(l.lists, []T{a})
        return
    }
    pos := BisectLeft(l.maxes, l.c, a)
    if pos > 0 && l.maxes[pos-1] == a {
        pos--
    }
    if pos == len(l.maxes) {
        pos--
        l.maxes[pos] = a
        l.lists[pos] = append(l.lists[pos], a)
    } else {
        l.lists[pos] = InSort(l.lists[pos], l.c, a)
    }
    l.fresh(pos)
}
func (l *SortedList[T]) DeleteItem(a T) bool {
    if l.size == 0 {
        return false
    }
    pos := BisectLeft[T](l.maxes, l.c, a)
    if pos == len(l.maxes) {
        return false
    }

```

```

var removed bool
l.lists[pos], removed = RemoveSort(l.lists[pos], l.c, a)
if !removed {
    return removed
}
l.size--
if len(l.lists[pos]) == 0 {
    // delete maxes at pos
    copy(l.maxes[pos:], l.maxes[pos+1:])
    l.maxes = l.maxes[:len(l.maxes)-1]
    // delete lists at pos
    copy(l.lists[pos:], l.lists[pos+1:])
    l.lists = l.lists[:len(l.lists)-1]
    l.resetIndex()
} else {
    l.maxes[pos] = l.lists[pos][len(l.lists[pos])-1]
    l.updateIndex(pos, -1)
}
return removed
}
func (l *SortedList[T]) Delete(index int) {
    if index >= l.size {
        return
    }
    var pos, in int
    if index == 0 {
        pos, in = 0, 0
    } else if index == l.size-1 {
        pos = len(l.lists) - 1
        in = len(l.lists[pos]) - 1
    } else {
        if len(l.indexes) == 0 {
            l.buildIndex()
        }
        pos, in = l.findPos(index)
    }
    l.size--
    l.lists[pos] = Remove(l.lists[pos], in)
    if len(l.lists[pos]) == 0 {
        // delete maxes at pos
        l.maxes = Remove(l.maxes, pos)
        // delete lists at pos
        copy(l.lists[pos:], l.lists[pos+1:])
        l.lists = l.lists[:len(l.lists)-1]
        l.resetIndex()
    } else {
        l.maxes[pos] = l.lists[pos][len(l.lists[pos])-1]
        l.updateIndex(pos, -1)
    }
}

```

```

func (l *SortedList[T]) Values() []T {
    res := make([]T, l.Size())
    i := 0
    l.Each(func(_ int, a T) {
        res[i] = a
        i++
    })
    return res
}

func (l *SortedList[T]) At(index int) (item T, found bool) {
    if index >= l.size {
        return
    }
    if index < len(l.lists[0]) {
        return l.lists[0][index], true
    }
    if index == l.size-1 {
        return l.maxes[len(l.maxes)-1], true
    }
    if len(l.indexes) == 0 {
        l.buildIndex()
    }
    pos, in := l.findPos(index)
    return l.lists[pos][in], true
}

func (l *SortedList[T]) Each(f ForEach[T]) {
    i := 0
    for _, list := range l.lists {
        for _, j := range list {
            f(i, j)
            i++
        }
    }
}

func (l *SortedList[T]) Has(a T) bool {
    if l.size == 0 {
        return false
    }
    pos := BisectLeft(l.maxes, l.c, a)
    if pos == len(l.maxes) {
        return false
    }
    index := BisectLeft(l.lists[pos], l.c, a)
    return l.lists[pos][index] == a
}

func (l *SortedList[T]) Index(a T) (int, bool) {
    if l.size == 0 {
        return 0, false
    }
    pos := BisectLeft(l.maxes, l.c, a)

```

```

    if pos == len(l.maxes) {
        return l.size, false
    }
    if a == l.lists[0][0] {
        return 0, true
    }
    if a == l.maxes[0] {
        return len(l.lists[0]) - 1, true
    }
    if a == l.maxes[len(l.maxes)-1] {
        return l.size - 1, true
    }
    index := BisectLeft(l.lists[pos], l.c, a)
    exist := index < len(l.lists[pos]) && l.lists[pos][index] == a
    return l.locate(pos, index), exist
}
func (l *SortedList[T]) Empty() bool {
    return l.size == 0
}
func (l *SortedList[T]) Size() int {
    return l.size
}
func (l *SortedList[T]) Len() int {
    return l.size
}
func (l *SortedList[T]) Clear() {
    l.resetIndex()
    l.lists = [][]T{}
    l.maxes = []T{}
    l.size = 0
}
func (l *SortedList[T]) Top() (item T, ok bool) {
    if l.size == 0 {
        return
    }
    return l.maxes[len(l.maxes)-1], true
}
func (l *SortedList[T]) Bottom() (item T, ok bool) {
    if l.size == 0 {
        return
    }
    return l.lists[0][0], true
}
func (l *SortedList[T]) fresh(pos int) {
    var zeroValue T
    listPosLen := len(l.lists[pos])
    if listPosLen > l.load {
        halfLen := listPosLen >> 1
        half := append([]T{}, l.lists[pos][halfLen:]...)
        l.lists[pos] = l.lists[pos][:halfLen]
    }
}

```

```

    l.lists = append(l.lists, nil)
    copy(l.lists[pos+2:], l.lists[pos+1:])
    l.lists[pos+1] = half
    // update max
    l.maxes[pos] = l.lists[pos][halfLen-1]
    l.maxes = append(l.maxes, zeroValue)
    copy(l.maxes[pos+2:], l.maxes[pos+1:])
    l.maxes[pos+1] = l.lists[pos+1][len(l.lists[pos+1])-1]
    l.resetIndex()
  } else {
    l.maxes[pos] = l.lists[pos][listPosLen-1]
    l.updateIndex(pos, 1)
  }
}
// 重建索引
func (l *SortedList[T]) buildIndex() {
  n := len(l.lists)
  rowLens := roundUpOf2((n + 1) / 2)
  l.offset = rowLens*2 - 1
  indexLens := l.offset + n
  indexes := make([]int, indexLens)
  for i, list := range l.lists { // fill row0
    indexes[len(indexes)-n+i] = len(list)
  }
  last := indexLens - n - rowLens
  for rowLens > 0 {
    for i := 0; i < rowLens; i++ {
      if (last+i)*2+1 >= indexLens {
        break
      }
      if (last+i)*2+2 >= indexLens {
        indexes[last+i] = indexes[(last+i)*2+1]
        break
      }
      indexes[last+i] = indexes[(last+i)*2+1] + indexes[(last+i)*2+2]
    }
    rowLens >= 1
    last -= rowLens
  }
  l.indexes = indexes
}
func (l *SortedList[T]) updateIndex(pos, incr int) {
  if len(l.indexes) > 0 {
    child := l.offset + pos
    for child > 0 {
      l.indexes[child] += incr
      child = (child - 1) >> 1
    }
    l.indexes[0] += 1
  }
}

```

```

}
func (l *SortedList[T]) findPos(index int) (int, int) {
    if index < len(l.lists[0]) {
        return 0, index
    }
    pos := 0
    child := 1
    lenIndex := len(l.indexes)
    for child < lenIndex {
        indexChild := l.indexes[child]
        if index < indexChild {
            pos = child
        } else {
            index -= indexChild
            pos = child + 1
        }
        child = (pos << 1) + 1
    }
    return pos - l.offset, index
}
func (l *SortedList[T]) locate(pos, index int) int {
    if len(l.indexes) == 0 {
        l.buildIndex()
    }
    total := 0
    pos += l.offset
    for pos > 0 {
        if pos&1 == 0 {
            total += l.indexes[pos-1]
        }
        pos = (pos - 1) >> 1
    }
    return total + index
}
func (l *SortedList[T]) resetIndex() {
    l.indexes = []int{}
    l.offset = 0
}
func roundUpOf2(a int) int {
    i := 1
    for ; i < a; i <=< 1 {
    }
    return i
}
func NewSortedList[T comparable](c Compare[T], loadFactor int) SortedList[T] {
    if loadFactor <= 0 {
        loadFactor = DefaultLoadFactor
    }
    return SortedList[T]{load: loadFactor, c: c}
}

```