以grpc为例

微服务框架协议扩展实践

晁岳攀 (鸟窝)

目录 / CONTENTS

个人介绍

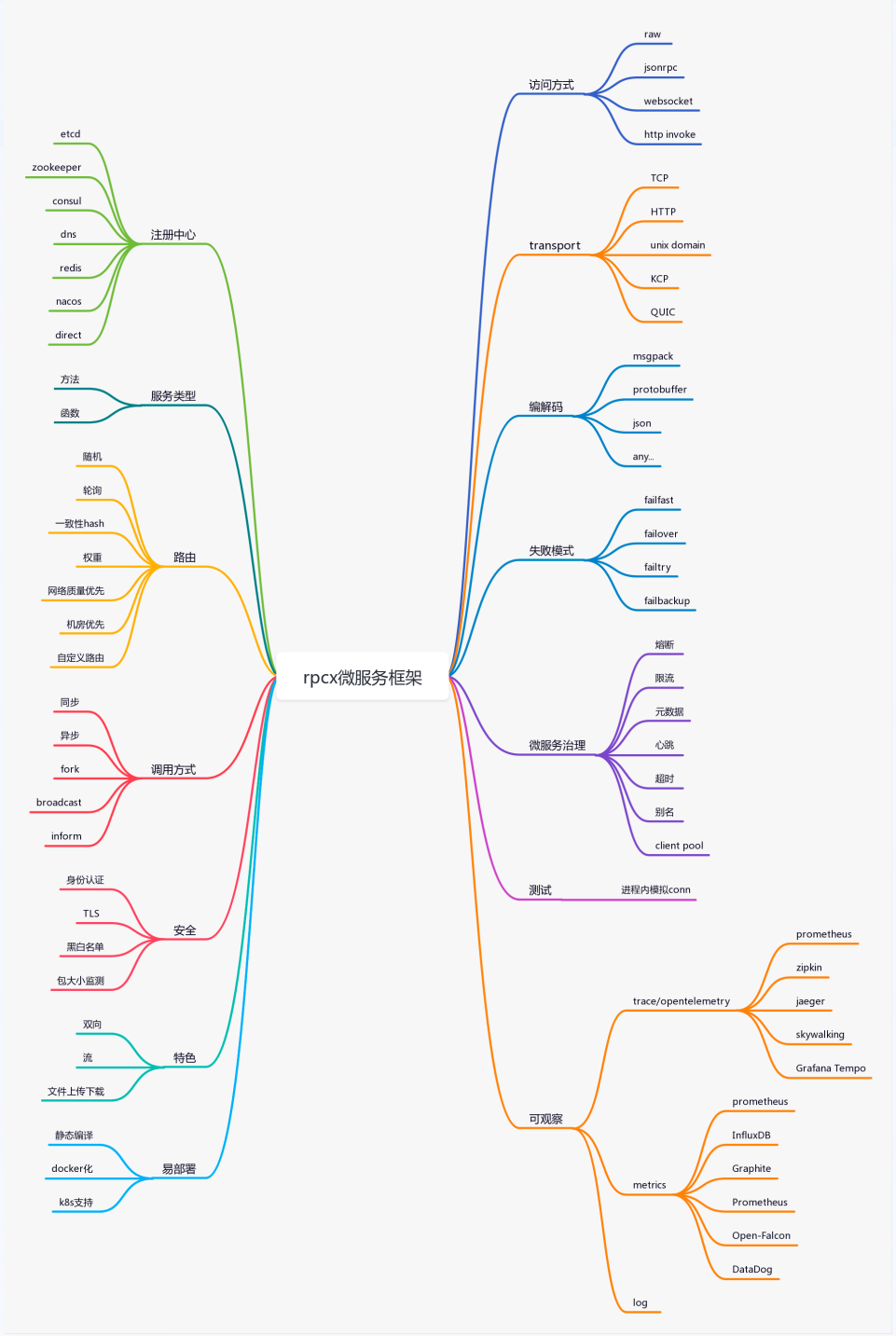
- 鸟窝， https://colobu.com
- Go微服务框架 rpcx的作者
- 20多年的编程开发经验
- Go语言的布道师，GopherChina大会讲师
- 极客时间专栏《Go并发编程实战课》作者
- 国内第一本原创Scala图书作者，版权输出到中国台湾地区

01

rpcx

go生态圈的微服务框架

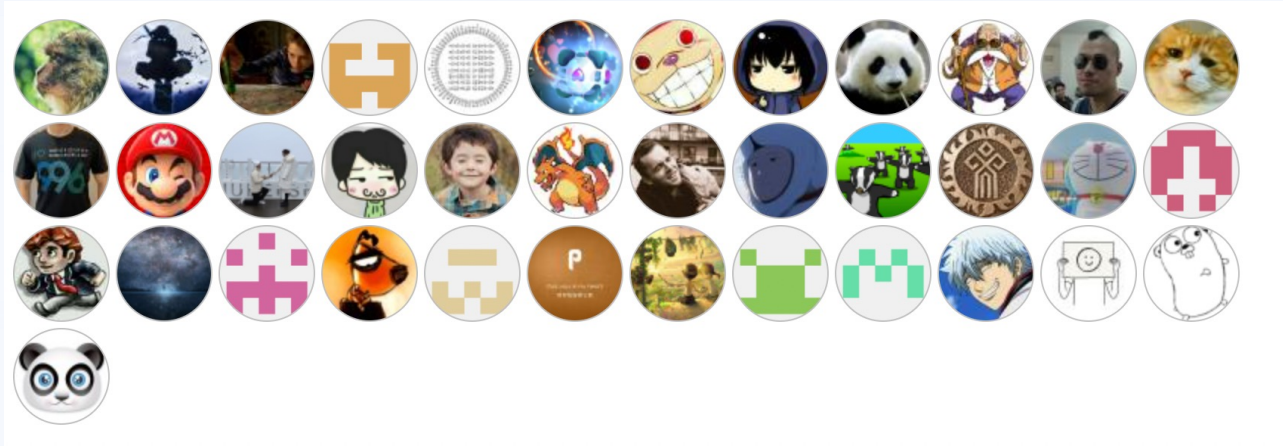
微服务框架



https://processon.com/chart_image/61d2930fe401fd7a53d4300e.png

rpcx简介 历史

第一次提交2017年10月04日



rpcx简介

rpcx设计考量

1 插件化设计

易于扩展，更容易引入新技术

2 Less is More

简单化设计，功能丰富

3 杜绝全家桶

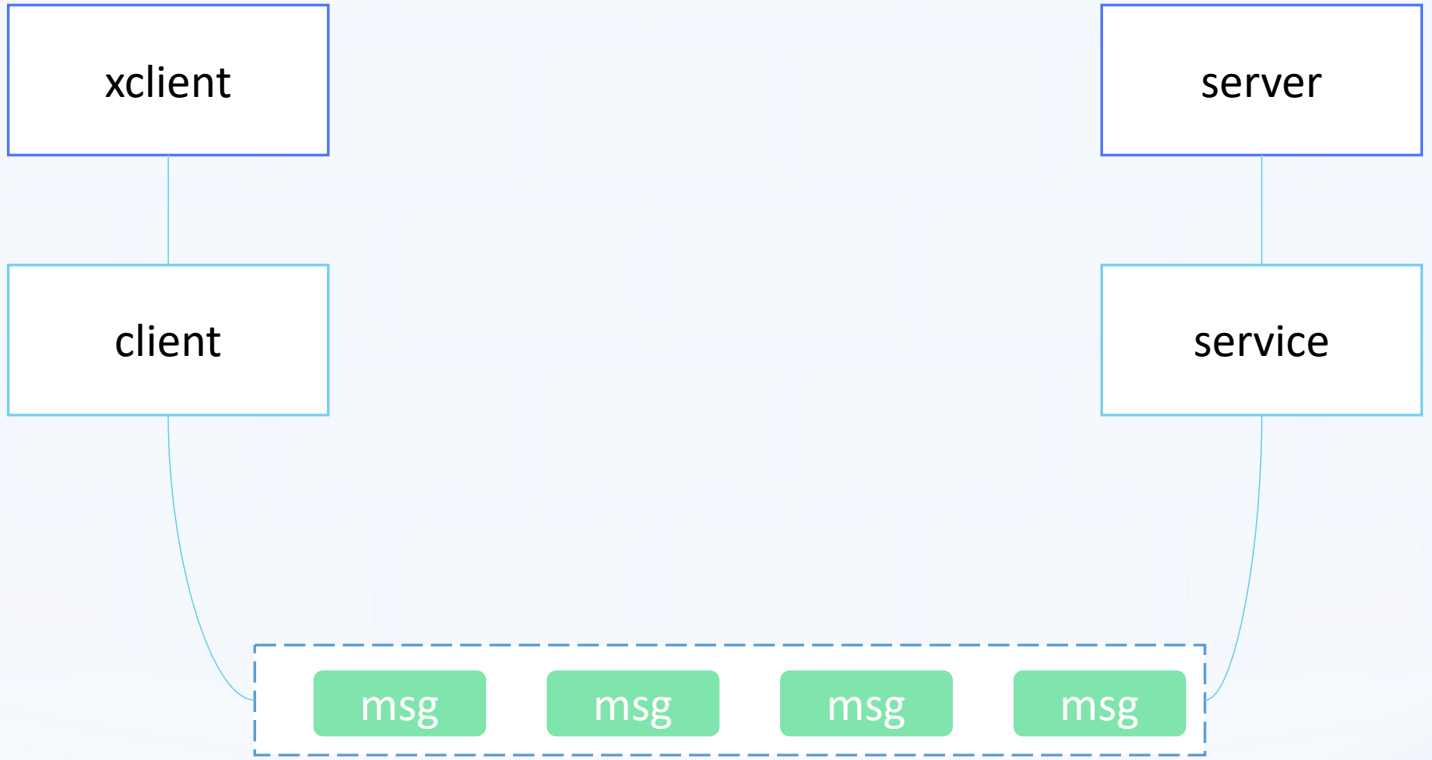
不限定用户的基础设施，便于采纳

4 性能保障

不追求极致的性能，但是相比较而言性能是杠杠的

rpcx简介

transport层和rpc层




grpc
Google官方支持的跨语言的rpc框架

grpc简介

初始发布于2015年2月。2017年5月grpc-go 第一个release

grpc简介

Hacker News new | past | comments | ask | show | jobs | submit logi

▲ Game_Ender on Aug 5, 2018 | parent | context | favorite | on: From Google to the world: the Kubernetes origin st...

Google does not use gRPC internally it, like k8s, it is an open re-implementation of an internal google technology. Also, like k8s & Bazel, it is less advanced and lower performance than their internal technology.

At a high level Google is open sourcing things that have genuine value, but also make integration with their money making services easier.

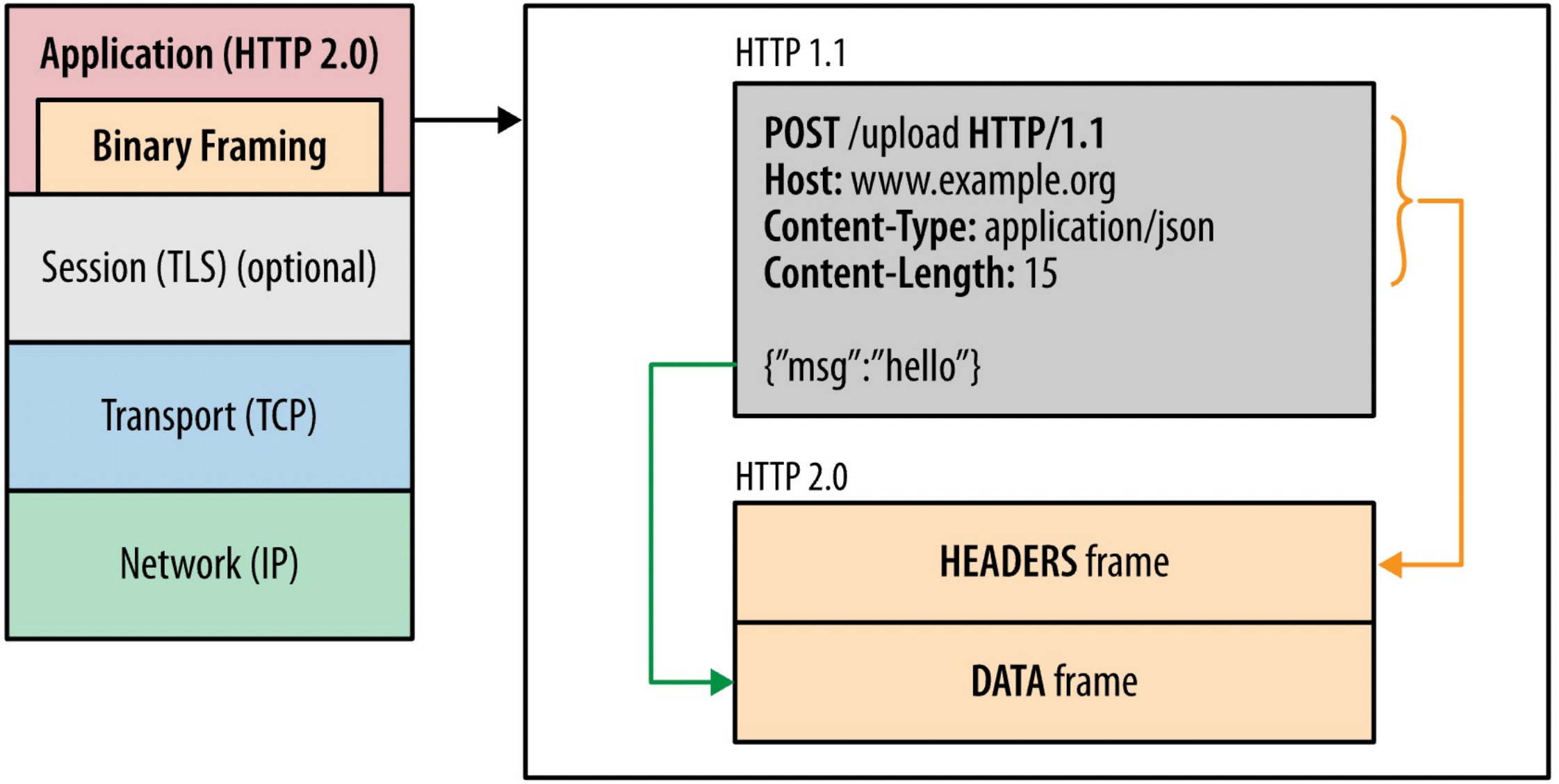
How Google uses gRPC

gRPC is being used for communication in internal production, on Google Cloud Platform, and in public-facing APIs.

For the past 15 years, Google has solved these problems internally with Stubby, an RPC framework that consists of a core RPC layer that can handle internet-scale of tens of billions of requests per second (yes, billions!). Now, this technology is available for anyone as part of the open-source project called gRPC. It's intended to provide the same scalability, performance and functionality that we enjoy at Google to the community at large.

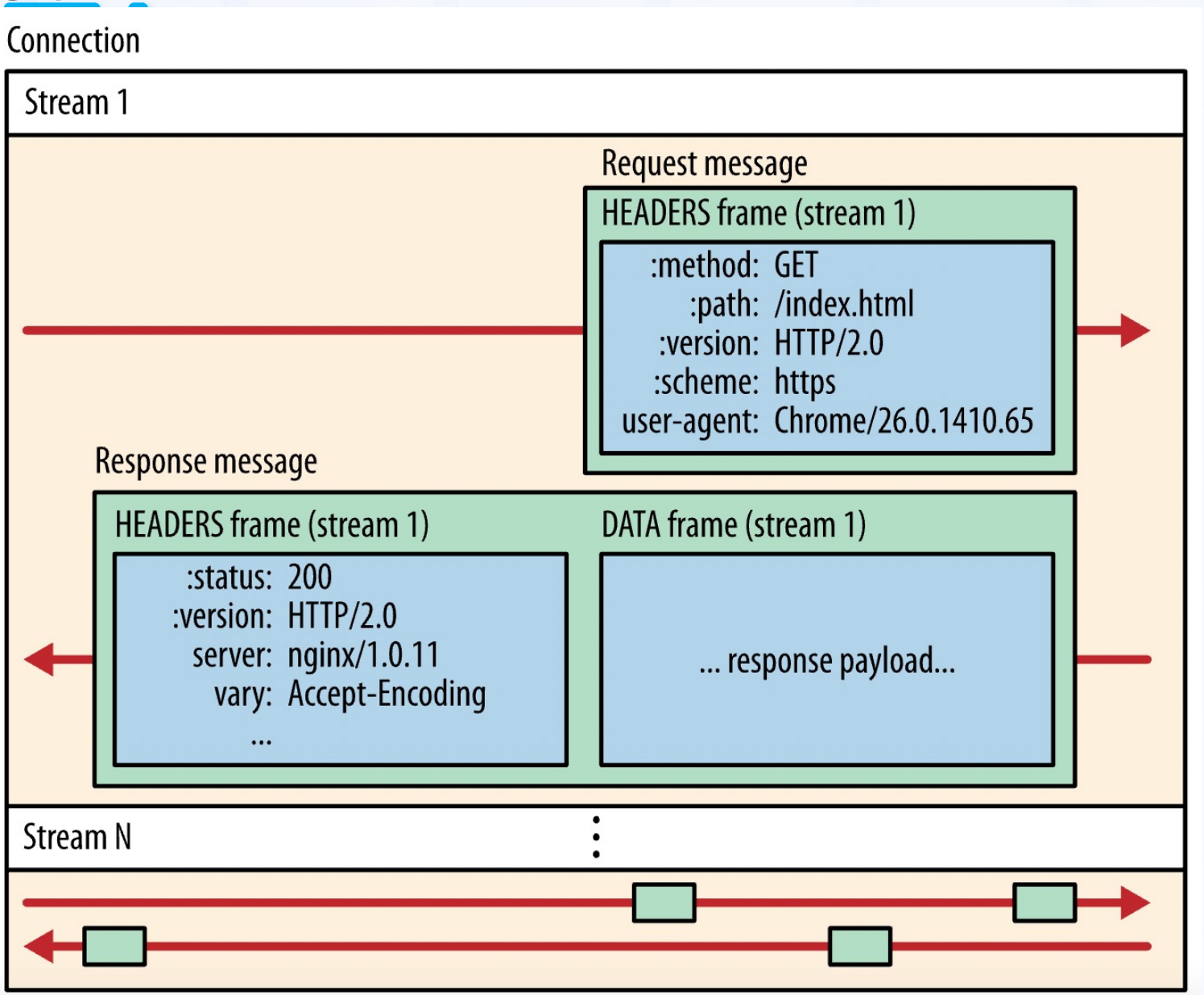
grpc简介

基于 HTTP 2.0



grpc简介

基于 HTTP 2.0



grpc简介 基于 HTTP 2.0

➢Request → Request-Headers *Length-Prefixed-Message EOS

- Request-Headers are delivered as HTTP2 headers in HEADERS + CONTINUATION frames.
- The repeated sequence of Length-Prefixed-Message items is delivered in DATA frames
- For requests, EOS (end-of-stream) is indicated by the presence of the END_STREAM flag

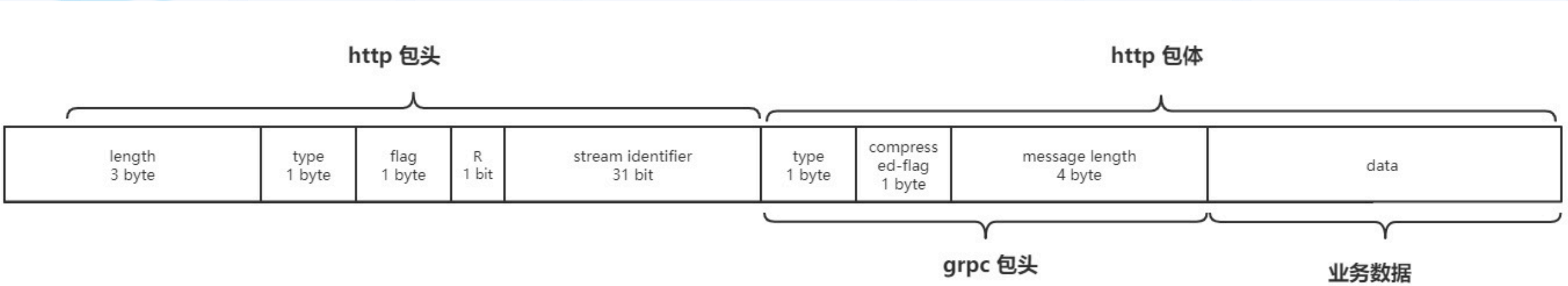
on the last received DATA frame.

➢Response → (Response-Headers *Length-Prefixed-Message Trailers) / Trailers-Only

- Response-Headers & Trailers-Only are each delivered in a single HTTP2 HEADERS frame block.
- For responses end-of-stream is indicated by the presence of the END_STREAM flag

on the last received HEADERS frame that carries Trailers.

grpc简介 基于 HTTP 2.0




```
HyperText Transfer Protocol 2
  Stream: SETTINGS, Stream ID: 0, Length 0
      Length: 0
      Type: SETTINGS (4)
    > Flags: 0x01, ACK
      0... .... .... .... .... .... .... .... = Reserved: 0x0
      .000 0000 0000 0000 0000 0000 0000 0000 = Stream Identifier: 0
  Stream: HEADERS, Stream ID: 1, Length 67, POST /pb.Greeter/SayHello
      Length: 67
      Type: HEADERS (1)
    > Flags: 0x04, End Headers
      0... .... .... .... .... .... .... .... = Reserved: 0x0
      .000 0000 0000 0000 0000 0000 0000 0001 = Stream Identifier: 1
      [Pad Length: 0]
      Header Block Fragment: 8386458f62b8d7c5614a92d8c6e1fac65a283f418aa0e41d13
      [Header Length: 189]
      [Header Count: 7]
    > Header: :method: POST
    > Header: :scheme: http
    > Header: :path: /pb.Greeter/SayHello
    > Header: :authority: localhost:8972
    > Header: content-type: application/grpc
    > Header: user-agent: grpc-go/1.43.0
    > Header: te: trailers
```

```
HyperText Transfer Protocol 2
  Stream: DATA, Stream ID: 1, Length 12
      Length: 12
      Type: DATA (0)
    > Flags: 0x01, End Stream
      0... .... .... .... .... .... .... .... = Reserved: 0x0
      .000 0000 0000 0000 0000 0000 0000 0001 = Stream Identifier: 1
      [Pad Length: 0]
      DATA payload (12 bytes)
GRPC Message: /pb.Greeter/SayHello, Request
    Compressed Flag: Not Compressed (0)
    Message Length: 7
    Message Data: 7 bytes
Protocol Buffers: /pb.Greeter/SayHello,request
  Message: <UNKNOWN> Message Type
      [Message Name: <UNKNOWN> Message Type]
    > Field(1):
```

grpc简介

基于 HTTP 2.0

8	7.288167	::1	::1	HTTP2	100	Magic
9	7.288180	::1	::1	HTTP2	85	SETTINGS[0]
10	7.288191	::1	::1	TCP	76	8972 → 54168 [ACK] Seq=1 Ack=25 Win=407744 Len=0 TSval=2698632602 TSecr=1083462436
11	7.288198	::1	::1	TCP	76	8972 → 54168 [ACK] Seq=1 Ack=34 Win=407744 Len=0 TSval=2698632602 TSecr=1083462436
12	7.293345	::1	::1	HTTP2	91	SETTINGS[0]
13	7.293399	::1	::1	TCP	76	54168 → 8972 [ACK] Seq=34 Ack=16 Win=407744 Len=0 TSval=1083462441 TSecr=2698632607
14	7.293991	::1	::1	HTTP2	85	SETTINGS[0]
15	7.294015	::1	::1	TCP	76	54168 → 8972 [ACK] Seq=34 Ack=25 Win=407744 Len=0 TSval=1083462441 TSecr=2698632607
16	7.294467	::1	::1	HTTP2	161	SETTINGS[0], HEADERS[1]: POST /pb.Greeter/SayHello
17	7.294507	::1	::1	TCP	76	8972 → 54168 [ACK] Seq=25 Ack=119 Win=407680 Len=0 TSval=2698632608 TSecr=1083462442
18	7.294620	::1	::1	GRPC	97	DATA[1] (GRPC) (PROTOBUF)
19	7.294665	::1	::1	TCP	76	8972 → 54168 [ACK] Seq=25 Ack=140 Win=407616 Len=0 TSval=2698632608 TSecr=1083462442
20	7.295196	::1	::1	HTTP2	106	WINDOW_UPDATE[0], PING[0]
21	7.295340	::1	::1	TCP	76	54168 → 8972 [ACK] Seq=140 Ack=55 Win=407744 Len=0 TSval=1083462442 TSecr=2698632608
22	7.295365	::1	::1	HTTP2	93	PING[0]
23	7.295411	::1	::1	TCP	76	8972 → 54168 [ACK] Seq=55 Ack=157 Win=407616 Len=0 TSval=2698632608 TSecr=1083462442
24	7.296926	::1	::1	GRPC	159	HEADERS[1]: 200 OK, DATA[1] (GRPC) (PROTOBUF), HEADERS[1]
25	7.297035	::1	::1	TCP	76	54168 → 8972 [ACK] Seq=157 Ack=138 Win=407616 Len=0 TSval=1083462444 TSecr=2698632610
26	7.297562	::1	::1	HTTP2	106	WINDOW_UPDATE[0], PING[0]
27	7.297623	::1	::1	TCP	76	8972 → 54168 [ACK] Seq=138 Ack=187 Win=407552 Len=0 TSval=2698632610 TSecr=1083462444
28	7.297642	::1	::1	HTTP2	93	PING[0]
29	7.297692	::1	::1	TCP	76	54168 → 8972 [ACK] Seq=187 Ack=155 Win=407616 Len=0 TSval=1083462444 TSecr=2698632610
30	7.298876	::1	::1	TCP	76	54168 → 8972 [FIN, ACK] Seq=187 Ack=155 Win=407616 Len=0 TSval=1083462445 TSecr=2698632610
31	7.298928	::1	::1	TCP	76	8972 → 54168 [ACK] Seq=155 Ack=188 Win=407552 Len=0 TSval=2698632611 TSecr=1083462445
32	7.298954	::1	::1	TCP	76	8972 → 54168 [FIN, ACK] Seq=155 Ack=188 Win=407552 Len=0 TSval=2698632611 TSecr=1083462445
33	7.299007	::1	::1	TCP	76	54168 → 8972 [ACK] Seq=188 Ack=156 Win=407616 Len=0 TSval=1083462445 TSecr=2698632611
34	9.428484	127.0.0.1	127.0.0.1	TCP	44	54090 → 1080 [ACK] Seq=1 Ack=1 Win=6366 Len=0

03

如何集成

集成的方案、实现以及工具

如何集成

方案抉择

方案一

扩展rpcx协议，以实现grpc协议

方案二

封装grpc,包装成rpcx

Life is full of trade-offs, the programming world is, too.

如何集成

方案二：包装grpc client/server

```
type GreeterService struct {
    helloworld.UnimplementedGreeterServer
}
```

grpc 服务

```
func (*GreeterService) SayHello(ctx context.Context, req *helloworld.HelloRequest) (*helloworld.HelloReply, error) {
    reply := &helloworld.HelloReply{Message: "hello " + req.GetName()}
    return reply, nil
}
```

rpcx 服务

```
func (*GreeterService) Greet(ctx context.Context, req *helloworld.HelloRequest, reply *helloworld.HelloReply) error {
    *reply = helloworld.HelloReply{Message: "hello " + req.Name}
    return nil
}
```

如何集成

方案二：包装grpc client/server

grpc client调用

```
// grpc client visits
conn, err := grpc.Dial(lis.Addr().String(), grpc.WithInsecure(), grpc.WithTimeout(time.Second))
assert.NoError(t, err)

defer conn.Close()
c := helloworld.NewGreeterClient(conn)

ctx, cancel := context.WithTimeout(context.Background(), time.Second)
defer cancel()
r, err := c.SayHello(ctx, &helloworld.HelloRequest{Name: "smallnest"})
```

rpcx client调用

```
// rpcx client
d, _ := client.NewPeer2PeerDiscovery("grpc@"+lis.Addr().String(), "")
opt := client.DefaultOption
xclient := client.NewXClient("helloworld.Greeter", client.Failtry, client.RandomSelect, d, opt)
defer xclient.Close()

argv := &helloworld.HelloRequest{
    Name: "smallnest",
}
reply := &helloworld.HelloReply{}
err = xclient.Call(context.Background(), "SayHello", argv, reply)
```

如何集成

server端支持

```
type GrpcServerPlugin struct {
    mu         sync.RWMutex
    l          net.Listener
    grpcServer *grpc.Server

    closed bool
}

// NewGrpcServerPlugin creates a new grpc server.
func NewGrpcServerPlugin() *GrpcServerPlugin {
    s := &GrpcServerPlugin{}
    s.grpcServer = grpc.NewServer()
    return s
}

// MuxMatch splits grpc Listener.
func (s *GrpcServerPlugin) MuxMatch(m cmux.CMux) {
    s.mu.Lock()
    defer s.mu.Unlock()

    s.l = m.MatchWithWriters(cmux.HTTP2MatchHeaderFieldSendSettings("content-type", "application/grpc"))
}
```

如何集成

server端支持

```go
// RegisterService registers grpc service by this method.
func (s *GrpcServerPlugin) RegisterService(registerFunc func(grpcServer *grpc.Server)) {
    registerFunc(s.grpcServer)
}

// Start stats the grpc server.
func (s *GrpcServerPlugin) Start() error {
    for {
        if s.closed {
            return nil
        }
        s.mu.RLock()
        l := s.l
        s.mu.RUnlock()
        if l != nil {
            break
        }
        time.Sleep(time.Second) // wait rpcx server starts
    }

    s.mu.RLock()
    l := s.l
    s.mu.RUnlock()

    if err := s.grpcServer.Serve(l); err != cmux.ErrListenerClosed {
        return err
    }

    return nil
}
```

如何集成

server端支持

```
s := server.NewServer()
gs := NewGrpcServerPlugin()
s.Plugins.Add(gs)

greetService := &GreeterService{}

// register rpcx service
err := s.Register(greetService, "")
assert.NoError(t, err)
// register grpc service
gs.RegisterService(func(grpcServer *grpc.Server) {
    helloworld.RegisterGreeterServer(grpcServer, greetService)
})

// must start rpcx server
go s.Serve("tcp", "127.0.0.1:0")
defer s.Close()
time.Sleep(100 * time.Millisecond)

// and starts grpc server
go func() {
    err := gs.Start()
    assert.NoError(t, err)
}()
```

如何集成

client支持

```go
// GenerateClient generates an new grpc client.
func (c *GrpcClientPlugin) GenerateClient(k, servicePath, serviceMethod string) (client client.RPCClient, err error) {
    _, addr := splitNetworkAndAddress(k)

    conn, err := grpc.Dial(addr, c.dialOpts...)
    if err != nil {
        return nil, err
    }

    c.clientMap[servicePath] = &GrpcClient{
        client:     conn,
        callOpts:   c.callOpts,
        remoteAddr: addr,
    }
    return c.clientMap[servicePath], err
}
```

```go
// FindCachedClient gets a cached client if exist.
func (c *GrpcClientPlugin) FindCachedClient(k, servicePath, serviceMethod string) client.RPCClient {
    c.clientMapMu.RLock()
    defer c.clientMapMu.RUnlock()
    client, ok := c.clientMap[servicePath]
    if !ok {
        return nil
    }

    return client
}
```

如何集成

client支持

```go
// Call invoke the grpc sevice.
func (c *GrpcClient) Call(ctx context.Context, servicePath, serviceMethod string, argv interface{}, reply interface{}) error {
    cc := c.client
    if cc == nil {
        return ErrClientNotRegistered
    }

    err := grpc.Invoke(ctx, fmt.Sprintf("/%s/%s", servicePath, serviceMethod), argv, reply, cc, c.callOpts...)
    return err
}
```

如何集成

client支持

```go
// Go calls the grpc servics asynchronizously.
func (c *GrpcClient) Go(ctx context.Context, servicePath, serviceMethod string, args interface{},
    if done == nil {
        done = make(chan *client.Call, 10)
    }

    call := new(client.Call)
    call.ServicePath = servicePath
    call.ServiceMethod = serviceMethod
    meta := ctx.Value(share.ReqMetaDataKey)
    if meta != nil { // copy meta in context to meta in requests
        call.Metadata = meta.(map[string]string)
    }

    if _, ok := ctx.(*share.Context); !ok {
        ctx = share.NewContext(ctx)
    }

    call.Args = args
    call.Reply = reply
    call.Done = done

    go func() {
        err := c.Call(ctx, servicePath, serviceMethod, args, reply)
        call.Error = err
        close(call.Done)
    }()

    return call
}
```

如何集成

client支持

```
// register CacheClientBuilder
gcp := NewGrpcClientPlugin([]grpc.DialOption{grpc.WithInsecure()}, nil)
client.RegisterCacheClientBuilder("grpc", gcp)

// rpcx client
d, _ := client.NewPeer2PeerDiscovery("grpc@"+lis.Addr().String(), "")
opt := client.DefaultOption
xclient := client.NewXClient("helloworld.Greeter", client.Failtry, client.RandomSelect, d, opt)
defer xclient.Close()

argv := &helloworld.HelloRequest{
    Name: "smallnest",
}
reply := &helloworld.HelloReply{}
err = xclient.Call(context.Background(), "SayHello", argv, reply)
```

总结

- 基于rpcx可扩展的设计，易于支持其它rpc框架
- 已经支持了grpc、 brpc(厂内)等框架
- 看起来还可以支持kitex
- 包装而不是重实现
- 依然保持简单
- 可以通过工具生成代码