RPC设计

目标

支持PAI在线预测服务，覆盖以下场景：

1. 低频在线预测调用；
2. 高频在线预测调用；
3. IO 型在线预测调用；

要求具备以下特性：

- [ ] 多语言在线预测调用；
  - [x] C++ Server端
  - [ ] C++ Client端
  - [ ] Java Server端
  - [ ] Java Client端
  - [x] Python Server端
  - [ ] Pyhton Client端
- [x] 复杂网络结构穿透；
- [x] 用户认证；
- [x] 轻客户端，应用方代码级别依赖尽可能低，尝试成本尽可能低；
- [x] 支持 http 协议，http 协议是支持网关穿透最好的协议，也对认证支持较好；
- [x] 支持长连接，以保证在 TCP 链接上的性能；
- [ ] 支持多种压缩模式，以适配不同的 client-server 端 IO 压力配比；
  - [ ] zlib压缩
  - [ ] gzip压缩
  - [ ] lz4压缩
  - [ ] snappy压缩

为何不用现有方案

ERPC

ERPC 是飞天和盘古等设施使用的内部通信方案，基于 protobuf 的 RPC 接口。ERPC 是非常优秀的集群内通信方案，即 service 和 client 由同一个团队开发，能够整体协调技术选型，从而保证pb版本的兼容性，以及easy等底层库的二进制兼容性。

由于 ERPC 使用私有协议，无法满足 EAS 中穿透复杂网络的需求。并且EAS使用 protobuf v2编码，在某些场景下序列化效率较低； 而EAS 业务方在调用端也未必方便使用 pb 库。

restful api

由于底层基于 http 协议，restful api具有非常良好的网络穿透性， 不过默认使用的 json 与 xml 编码冗余数据较多，通信效率过低，不适用于高吞吐的场景。

gRPC

gRPC 是 Google 基于 HTTP/2 协议开发的 RPC 框架，但 HTTP/2 协议要求客户端使用较新的 http 通信库，并最好使用 gRPC 的SDK。 gRPC 的 SDK 对调用方的开发环境侵入比较深，可能会要求调用方大规模升级开发环境，这是 EAS 目前所不希望的。此外，EAS为了兼容原PAI在线预测服务，使用基于消息签名的认证机制，导致应用层与通信层底层信令耦合，该机制在gRPC上难以实现。

此外，gRPC 大量集成了 Google 自家技术，比如 bazel 构建工具、protobuf 序列化、snappy 压缩算法等，而 EAS 希望调用方在技术选型上能够有更大空间，比如尝试使用 facebook 最近开源的 zstd 压缩算法，使用 scons 或者 cmake 等构建工具

技术目标

1. 基于http协议，并开放协议细节，保证任意语言、任意团队均可调用；
2. http协议设计保证能以最小代价穿透复杂网络结构，比如仅需一个简单的nginx反向代理；
3. 支持几种典型的通信模式，比如request-reply，push-pull以及batch方式通信；

详细设计

Wire Protocol

wire protocol定义框架收发哪些二进制数据，以及如何收发。EAS RPC 主要处理三种二进制数据：

1. 消息（message），是EAS RPC 中的主要数据，每次 RPC 调用的请求以及响应均表示为消息（request message/response message）；
2. 信令（command），是EAS RPC 控制通信过程需要的二进制数据，主用用于身份认证、信道控制、同步操作等作用；在整个通信过程中，应尽可能避免信令的使用，并且信令对 client 端来说不是必须的；为了可读性，信令以字符串的形式发送。
3. 数据帧（frame），是EAS RPC 中的额外数据，frame 用于编码裸的二进制数据，无需经过消息的序列化与反序列化；

EAS RPC 要求底层实现必须按照以下逻辑处理数据：

1. 每个request message 都必须是 client 发送给 server，server 必须发送与之对应的 response message；
2. 信令只能 client 主动向 server 发送，server 可以响应 client，但不能不主动向 client 发送信令；
3. 数据帧只能 client 主动向 server 发送，server 可以响应 数据帧；

wire protocol 是 server 向 client 承诺的一组通信行为，在不同的网络协议下可以有不同的实现。

信令设计

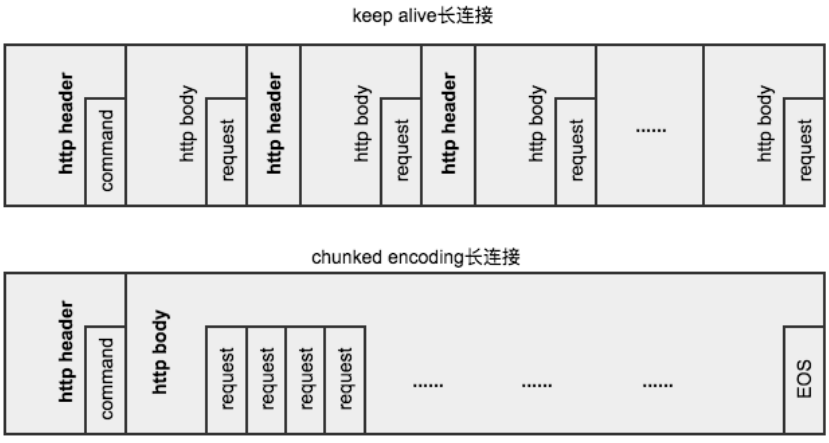
EAS Protocol 应该支持以下信令：

信令	含义	取值
eas.wire.compression	如何压缩 message 和 frame	nil/gzip/snappy/lzo/[lz4]
eas.wire.pattern	通信模式	[req-rep]/pub-sub/push-pull
eas.wire.pipeline	启动/停止类似 redis 的 pipeline 模式	start/end
eas.wire.frame	是否启用 frame 支持	on/[off]
eas.protobuf.use_json	是否以 json格式序列化和反序列化	on/[off]
eas.query.timeout	每次请求的超时时间	整型 [单位：ms]

HTTP/1.1实现 EAS Wire Protocol

EAS RPC 支持使用http协议进行通信：

1. 所有信令必须在 http 请求头中发送，以 http 头的 key-value 对方式表示；
2. 支持两种长连接模式：
   i. keep-alive 模式，即在 http 头中设定 `keep-alive` 为 `1`，每次请求后不关闭链接，直接发送下一次请求；下次请求中，http 头可省略已经发送过，并且生效了的信令；
   ii. chunked encoding 模式，即在 http 头中设置 `encoding` 为 `chunked`，并且以每个 chunk 为一次请求；若需要发送信令，则需要先发送 `EOS` 关闭本次请求，重新发送 http 头，在头中包含需要发送的信令，然后重新以chunk方式发送请求；
3. 支持压缩，压缩在 http 头的 `content-encoding` 字段中配置；



EAS RPC支持三种 HTTP/1.1客户端：

1. curl等简单的 http 库，只需设置认证，并以 post 方式提交请求数据即可
2. 用户自己使用基于 netty 的高性能http 客户端，只需在 http 头中设置正确的信令，然后以 keep-alive方式连续请求即可；
3. EAS 提供的官方 RPC 客户端，主要以 chunked encoding方式实现，充分利用底层 wire protocol 的各种特性。

TCP实现 EAS Wire Protocol

TCP上的 wire protocol 实现以兼容 zeromq 为主要目标

HTTP/2实现 EAS Wire Protocol

HTTP/2上的 wire protocol 以兼容 grpc 为主要目标

消息编码

本节讨论信令、消息和数据帧如何编码

信令编码

信令以纯文本方式发送，并以 utf8编码，信令只支持常规 ASCII 字符，不支持不可见字符以及中文字符。信令按照如下格式编码：

```
<key> [value]
```

即每个信令总是带一个值，比如：

```
eas.wire.compression zlib
```

消息编码

消息编码分两步进行：对象序列化与消息压缩。前者是指通过 json、protobuf 或者 hession 等对象序列化技术，将 OO 语言中的对象序列化成二进制字节流，以便进行传输。后者是指对序列化后的字节流进行压缩，以便降低传输时的带宽开销。

EAS RPC框架为上层服务提供消息压缩，支持主流的压缩算法；为了不限制业务方技术选型，EAS RPC本身并不提供序列化方案，支持业务方根据需要灵活选型。

功能设计

Batching功能 <https://lark.alipay.com/pai/eas/gbuon3>

HTTP Tracing功能 <https://lark.alipay.com/pai/eas/cuhdeh>