Volo 新能力

面向性能和易用性的框架迭代

王杰

Jie Wang

目录 | Contents

01

Volo 简介

致力于 Rust 微服务生态的高性能框架

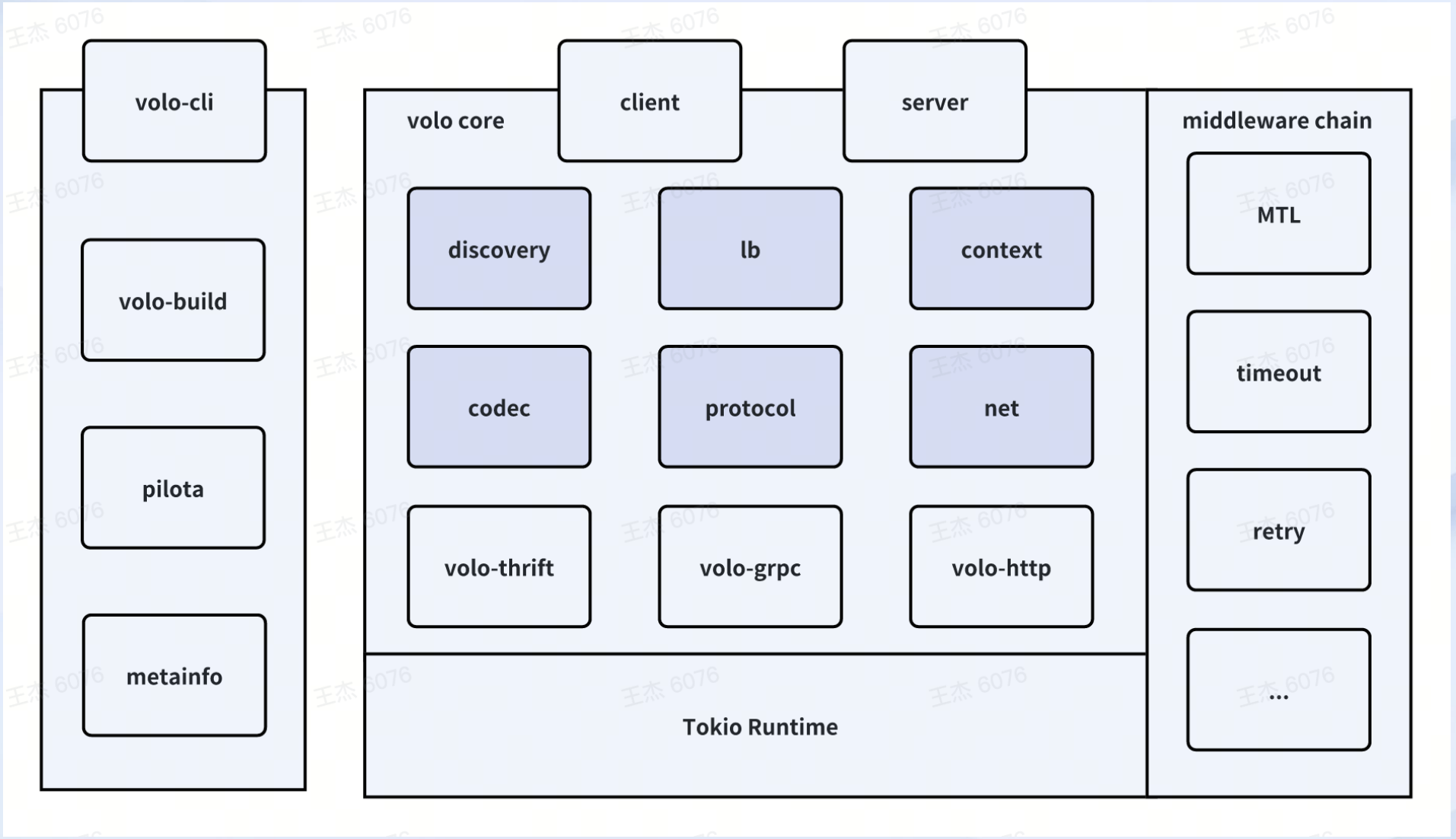
Volo 目标 & 现状

目标

- 极致性能
- 易用性
- 可扩展性

核心抽象

- codec/protocol/net
- sd/lb
- context



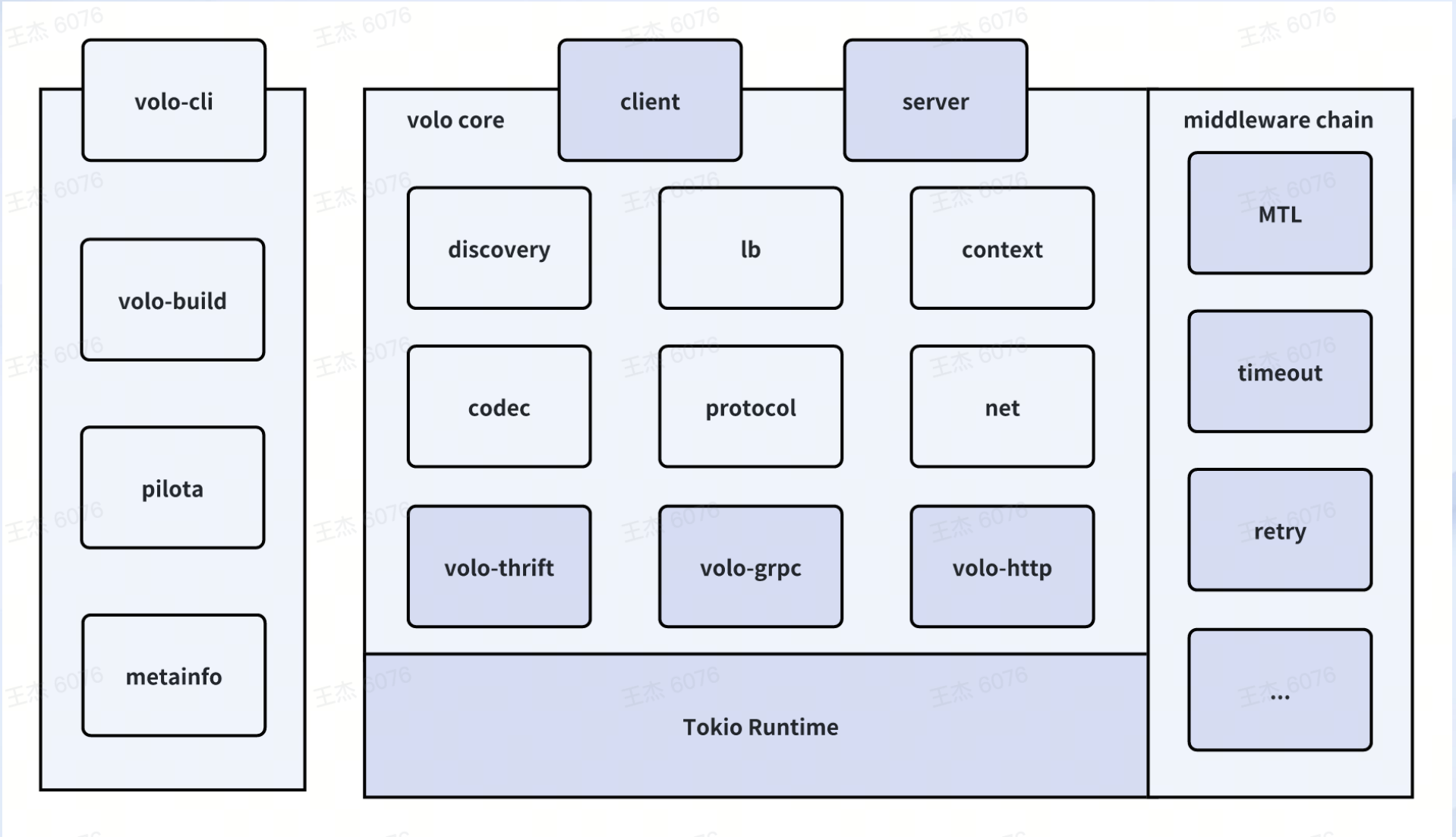
Volo 目标 & 现状

目标

- 极致性能
- 易用性
- 易扩展

实现

- 协议
  - volo-thrift
  - volo-grpc
  - volo-http
- 中间件系统
  - 限流
  - 重试
  - 超时
  - MLT
- 传输层
  - TCP
  - TLS
- Tokio Runtime



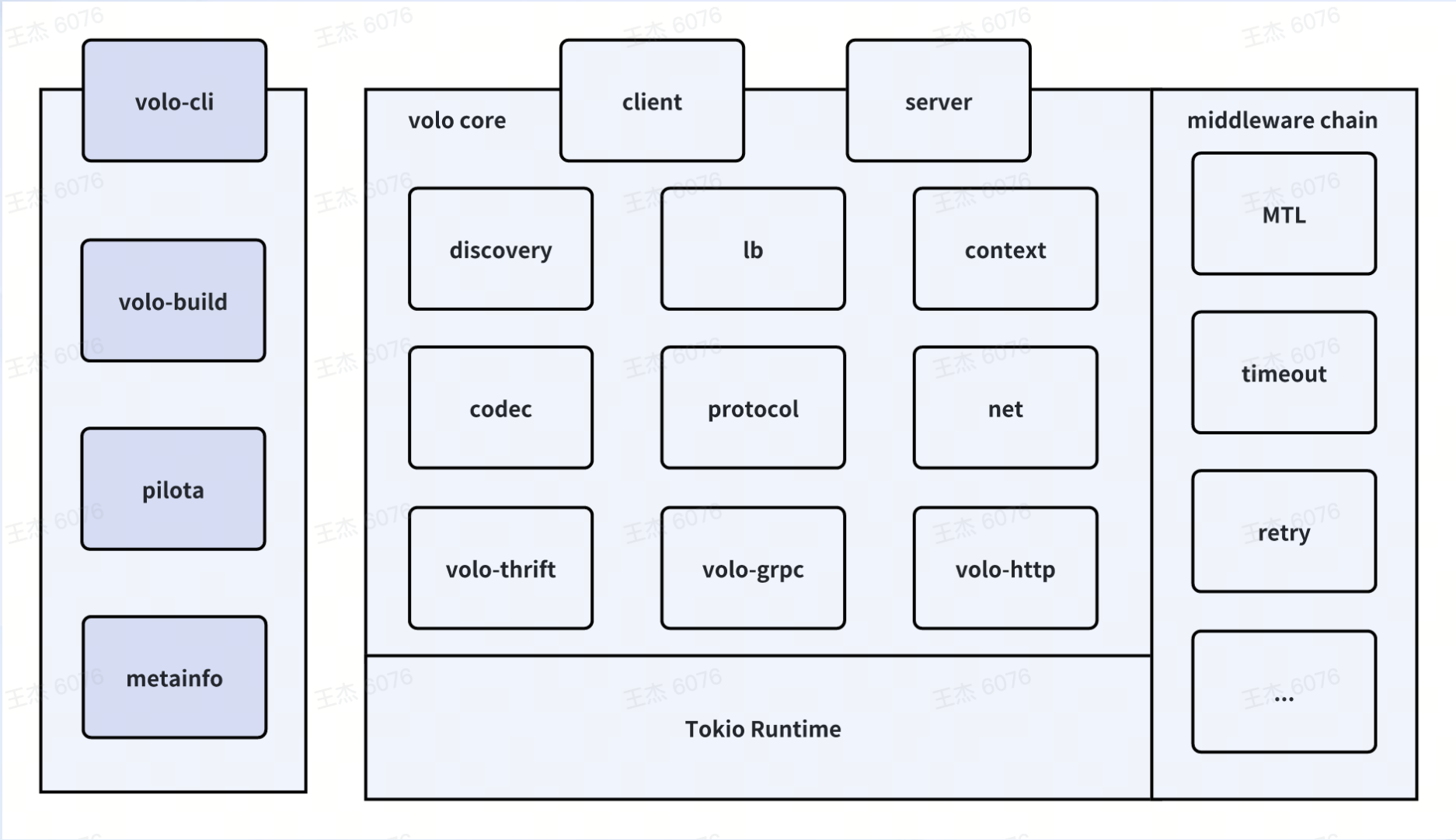
Volo 目标 & 现状

目标

- 极致性能
- 易用性
- 易扩展

重要模块

- pilota
  - thrift
  - protobuf
- metainfo
- volo-cli
  - volo.yml



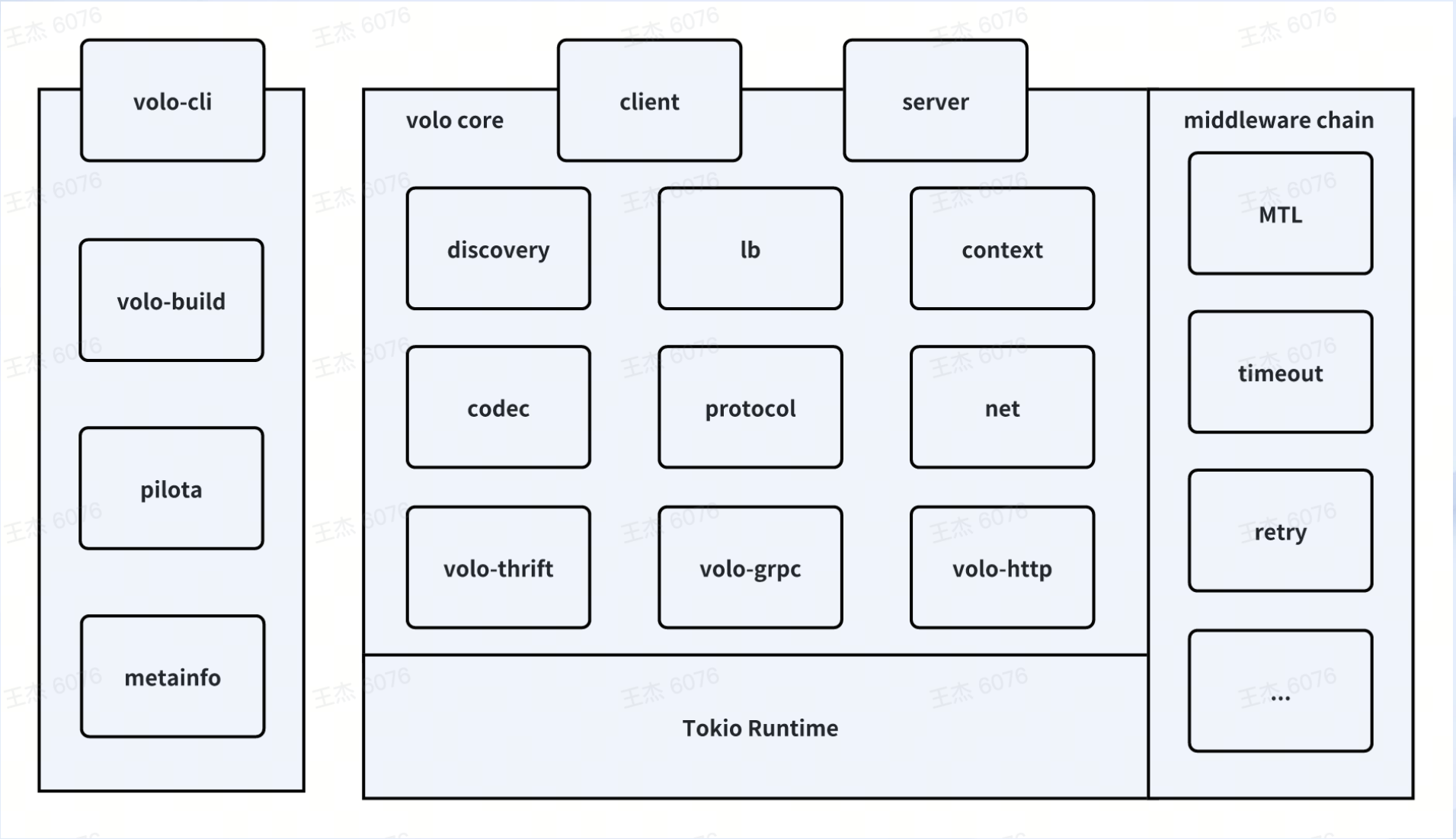
Volo 优势 & 落地

优势

- 统一抽象
  - 协议
  - 治理
  - 传输
- 高性能
- 微服务工程化

落地

- 多业务线
- 多服务
- go -> rust 平均 CPU 收益 40% - 50%



02

RPC 框架迭代

Protobuf Unknown Fields、Protobuf Options、Thrift Field Mask

RPC 框架迭代

Protobuf Unknown Fields

01

- Google Protobuf 协议扩展
- 贴合 Volo-build 现有生态，接入简单
- 关注性能，全链路零拷贝

Protobuf Options

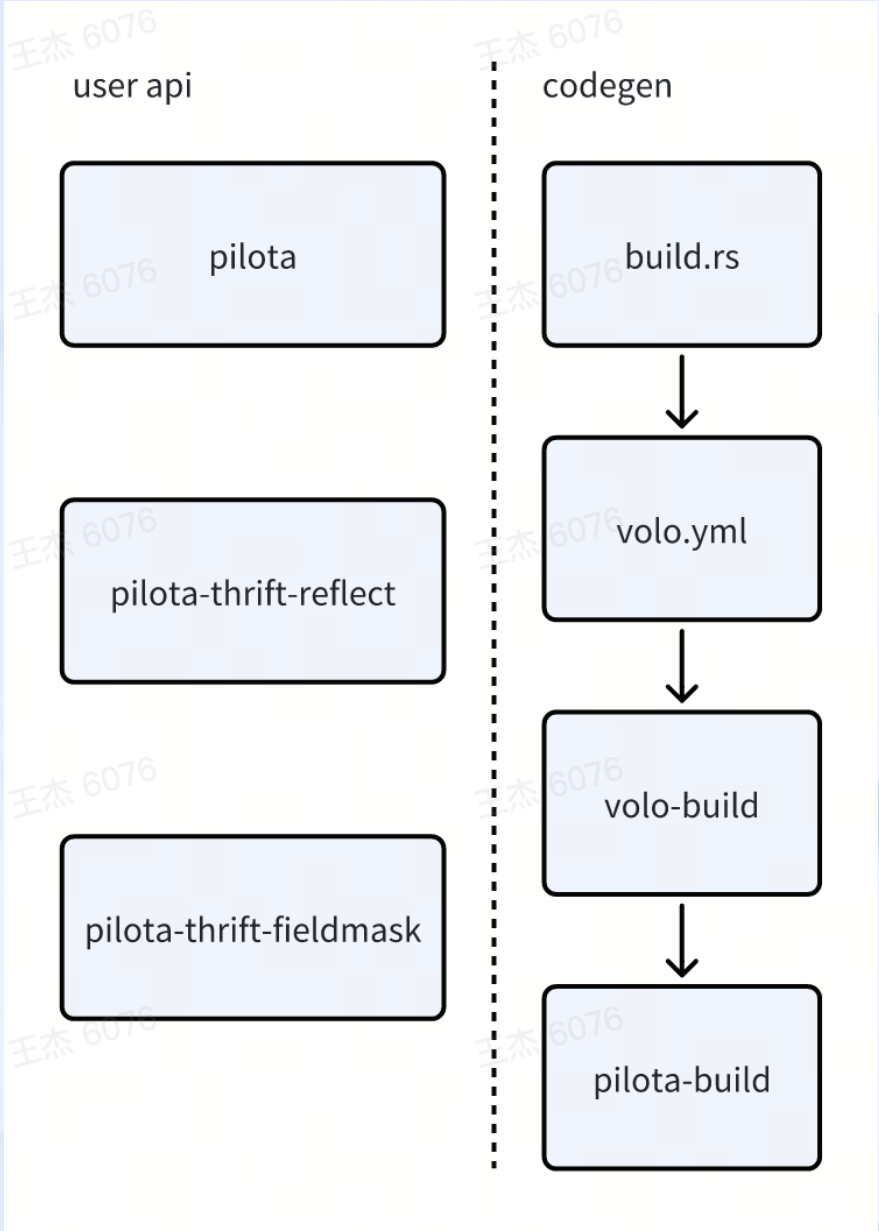
02

- Google Protobuf 协议扩展
- 内置支持了用于控制生成代码的注解，对齐 pilota thrift 的实现
- 关注易用性，提供自定义注解解析和获取的易用方法

Thrift Field Mask

03

- IDL Request 运行时动态裁剪能力
- 对齐 Kitex 生态，支持 Thrift Path 解析器
- 支持用户 API，并在生成代码中自适应裁剪



RPC 框架迭代——PB Unknown Fields

- Google PB Unknown Fields 定义
  - 老版本兼容新版本定义
  - 跨链路传递减少强制同步升级
- Volo 用户接入
  - volo-cli 一键创建服务
  - volo.yml 配置开启
- Volo 接口
  - _unknown_fields 字段
- 注意事项
  - 新增的透传字段放到最后

Step 1 volo init 创建项目

```
volo init --includes=idl volo.example.item idl/volo_example_item.proto
```

Step 2 编辑 volo.yml 开启 keep_unknown_fields

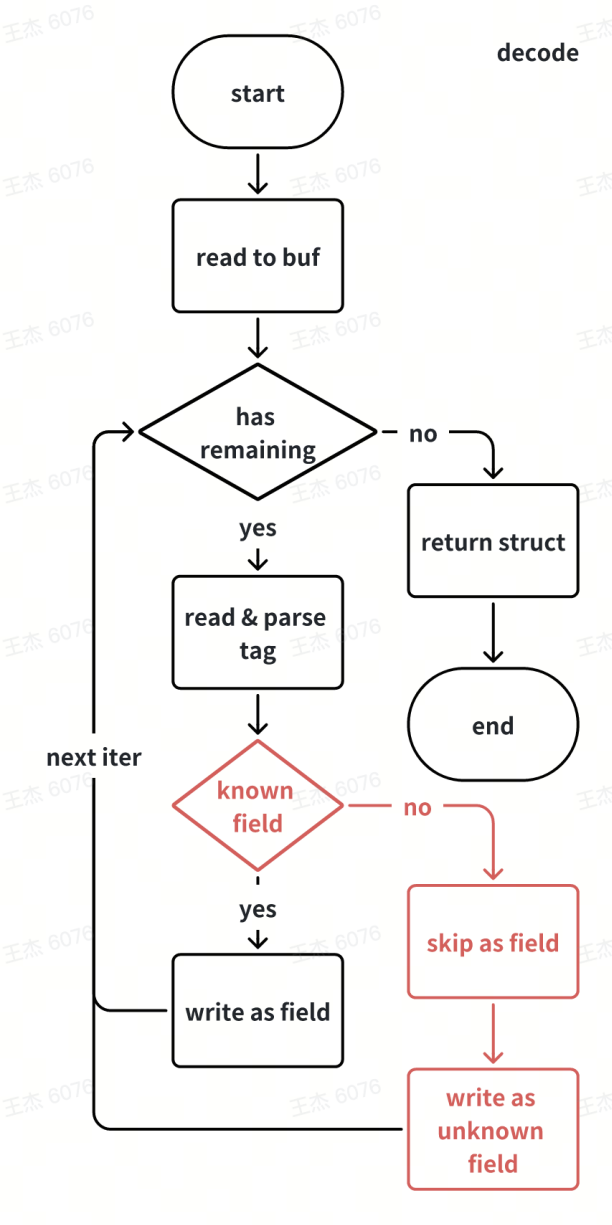
```yaml
# Please refer to https://www.cloudwego.io/docs/volo/guide/config/
entries:
  default:
    filename: volo_gen.rs
    protocol: protobuf
    services:
    - idl:
        source: local
        path: ../idl/volo_example_item.proto
        includes:
        - ../idl
      codegen_option:
        keep_unknown_fields: true
```

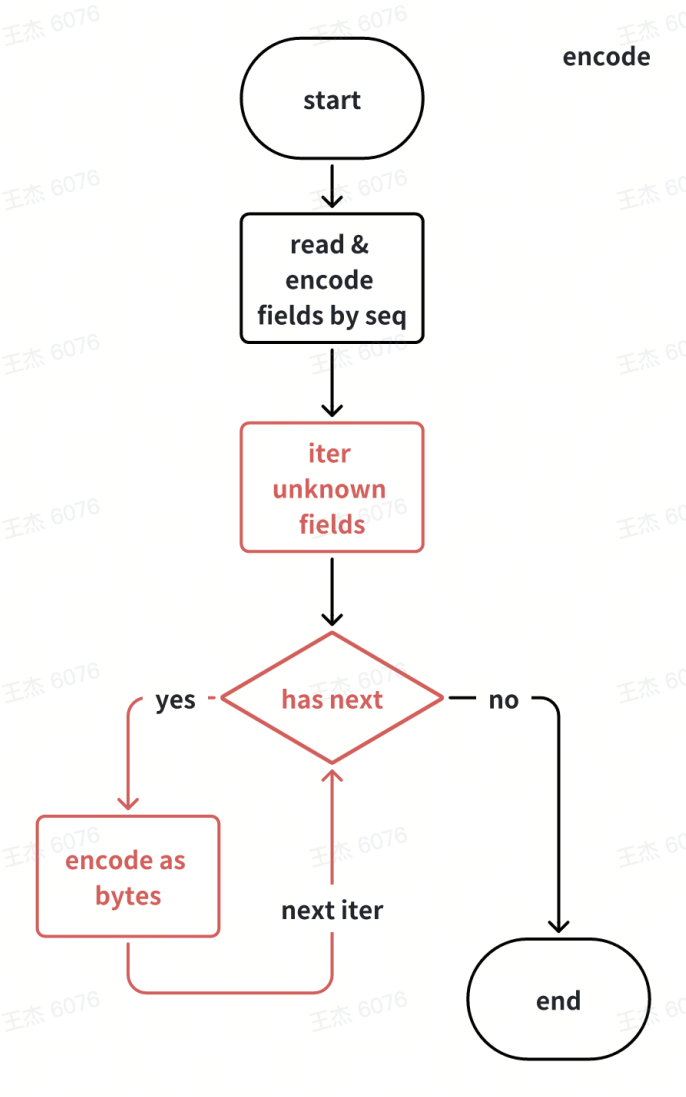
RPC 框架迭代——PB Unknown Fields

```rust
pub mod volo_gen {
    pub mod volo {
        pub mod example {
            pub mod item {
                #[derive(Debug, Default, Clone, PartialEq)]
                5 implementations
                pub struct Item {
                    pub id: i64,

                    pub title: ::pilota::FastStr,

                    pub content: ::pilota::FastStr,

                    pub extra: ::pilota::AHashMap<::pilota::FastStr, ::pilota::FastStr>,

                    pub _unknown_fields: ::pilota::BytesVec,
                }
```

```rust
#[derive(
    PartialOrd, Hash, Eq, Ord, Debug, Clone, PartialEq, serde::Serialize, serde::Deserialize,
)]
11 implementations
pub struct BytesVec {
    pub list: VecDeque<Bytes>,
    pub size: usize,
}
```

RPC 框架迭代——PB Unknown Fields





RPC 框架迭代——PB Unknown Fields

演进方向：短路优化

- 优化思路：当读完所有已知字段后，后续所有的bytes 一次性载入 unknown fields 列表中
- 优化障碍：根据 pb 官方定义，repeated 字段的元素类型为 string、bytes 或者 message，是不被要求连续出现的，而标量类型也可以标注为非 packed 编码方式，这样也会允许不连续出现。
- 使用建议：把新增透传字段放在最后，序列号递增

RPC 框架迭代——PB Options

- PB Options 用途
  - pilota 注解：提供编解码控制能力
  - 用户自定义注解：业务相关数据
- Volo 用户接入
  - volo-cli 一键创建服务
  - volo.yml 配置开启
  - proto idl 中使用注解
- Volo 接口
  - exts_{filename}_mod
  - ::pilota::pb::extension::*
- 注意事项
  - 自定义注解需要规避 google 和 pilota 定义的注解 tag id
  - 自定义注解可能需要使用 touch_all 配置

Step 1 volo init 创建项目

```
volo init --includes=idl volo.example.item idl/volo_example_item.proto
```

Step 2 编辑 volo.yml 开启 with_descriptor

```yaml
# Please refer to https://www.cloudwego.io/docs/volo/guide/config/
entries:
  default:
    filename: volo_gen.rs
    protocol: protobuf
    with_descriptor: true
    services:
    - idl:
        source: local
        path: ../idl/volo_example_item.proto
        includes:
        - ../idl
```

RPC 框架迭代——PB Options

PB IDL

```
import "pilota.proto";                                          step 1
import "google/protobuf/descriptor.proto";

extend google.protobuf.MessageOptions {                         step 2
    optional string alias = 1111;
    optional float f32 = 1112;
}

message Item {
    int64 id = 1;
    string title = 2;
    string content = 3;

    map<string, string> extra = 10;

    option (alias) = "MyItem";                                  step 3
    option (f32) = 1.23;
}

message GetItemRequest {
    int64 id = 1;
}

message GetItemResponse {
    Item item = 1;
}

service ItemService {
    option (pilota.rust_wrapper_arc_all)=true;                  step 3
    rpc GetItem(GetItemRequest) returns (GetItemResponse);
}
```

RPC 框架迭代——PB Options

pilota.rust_wrapper_arc_all = true

```rust
impl volo_gen::volo::example::item::ItemService for S {
    async fn get_item(
        &self,
        _req: ::volo_grpc::Request<std::sync::Arc<volo_gen::volo::example::item::GetItemRequest>>,
    ) -> ::std::result::Result<
        ::volo_grpc::Response<std::sync::Arc<volo_gen::volo::example::item::GetItemResponse>>,
        ::volo_grpc::Status,
    > {
        ::std::result::Result::Ok(::volo_grpc::Response::new(message: Default::default()))
    }
}
```

RPC 框架迭代——PB Options

pilota.proto

级别	注解名(pilota.xxx)	类型	作用
service	rust_wrapper_arc_all	bool	标注是否在生成代码中对所有的方法中的 request 和 response 结构体使用 Arc
message	serde_attribute_message	string	为消息添加 serde 的任意 attribute
	name_message	string	指定生成代码中消息的名字
field	rust_wrapper_arc	bool	标注对某个字段是否使用 Arc
	serde_attribute_field	string	为字段添加 serde 的任意 attribute
	name_field	string	指定生成代码中字段的名字
	rust_type	string	指定生成代码中字段的类型
	optional_repeated	bool	指定 repeated 字段为 optional，会生成 Option<Vec>
enum	serde_attribute_enum	string	为 enum 添加 serde 的任意 attribute
	name_enum	string	指定生成代码中 enum 的名字
enumvalue	serde_attribute_enum_value	string	为 enum value 添加 serde 的任意 attribute

RPC 框架迭代——PB Options

custom options

volo-gen > src > lib.rs > ...

```rust
mod tests {
    #[test]
    ▶ Run Test | ⚙ Debug
    fn test_volo_gen_custom_options() {
        volo::example::item::file_descriptor_volo_example_item() &'static FileDescriptor
            .messages() impl Iterator<Item = MessageDescriptor>
            .for_each(|m: MessageDescriptor| {
                // the name is same with the idl definition
                if m.name() == "Item" {
                    let opt: &MessageOptions = m.proto().options.as_ref().unwrap();
                    if let Ok(f32_opt: f32) = volo::example::item::exts_volo_example_item::f32.get(opt) {
                        println!("f32_opt: {:?}", f32_opt);
                        assert_eq!(f32_opt, 1.23);
                    }
                    if let Ok(alias: FastStr) = volo::example::item::exts_volo_example_item::alias.get(opt) {
                        println!("alias: {:?}", alias);
                        assert_eq!(alias, "MyItem");
                    }
                }
            });
    }
}
```

Problems Output Debug Console Ports **Terminal** GitLens

```
---- tests::test_volo_gen_custom_options stdout ----
f32_opt: 1.23
alias: "MyItem"
```

RPC 框架迭代——PB Options

PB File Descriptor & Extension Mod

```rust
pub mod volo_gen {
    pub mod volo {
        pub mod example {
            pub mod item {
                pub fn file_descriptor_volo_example_item()
                -> &'static ::pilota::pb::reflect::FileDescriptor {
                    &*FILE_DESCRIPTOR_VOLO_EXAMPLE_ITEM
                }
                pub mod exts_volo_example_item {
                    pub const alias: ::pilota::pb::extension::CustomExtField<
                        ::pilota::pb::descriptor::MessageOptions,
                        ::pilota::pb::extension::StrOptionValueExtractor,
                    > = ::pilota::pb::extension::CustomExtField::new(1111);
                    pub const f32: ::pilota::pb::extension::CustomExtField<
                        ::pilota::pb::descriptor::MessageOptions,
                        ::pilota::pb::extension::FloatOptionValueExtractor,
                    > = ::pilota::pb::extension::CustomExtField::new(1112);
                }
```

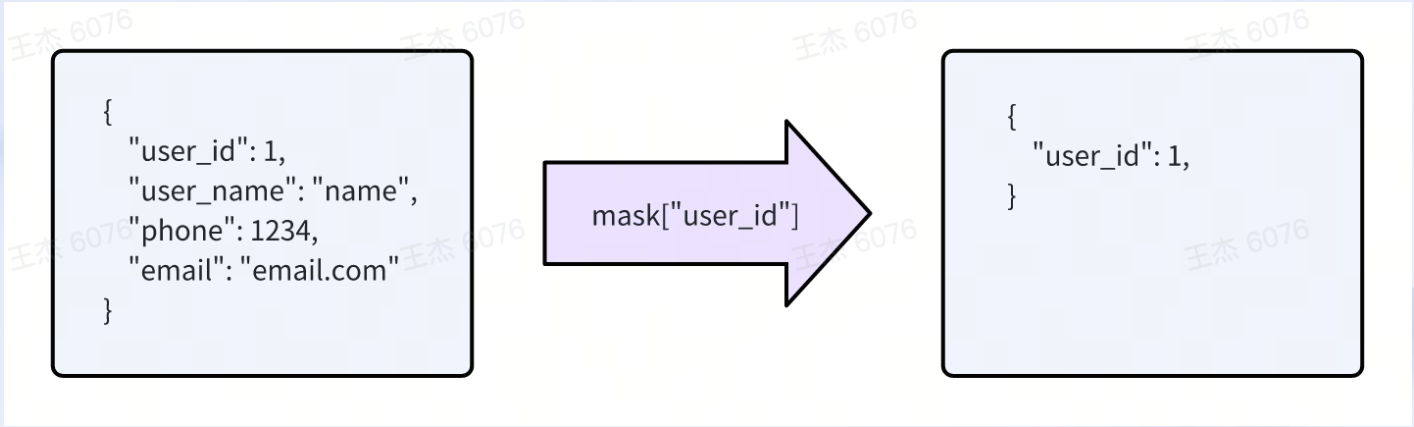
RPC 框架迭代——PB Options

演进方向：pb custom options 的性能和可扩展性提升

- pb descriptor 在生成代码中提供更多易用方法
- pb custom options 的使用有些情况需要开启 touch_all，这样会带来 build 阶段的性能损失，也带来了更大的代码体积，后续会沿用按需生成的方式来处理注解
- 在 pilota 中提供更多 hook 点
  - 目前我们仅支持在 rir 层提供 plugin 能力，所以用户仅能在这一层获取自定义注解，控制生成代码的行为

RPC 框架迭代——Thrift Field Mask

- Thrift Field Mask
  - 减少传输带宽和编解码开销
  - gateway/proxy 服务统一 idl
  - 字段权限控制
- Volo 用户接入
  - volo-cli 一键创建服务
  - volo.yml 配置开启
- Volo 接口
  - ::pilota_thrift_reflect::*
  - ::pilota_thrift_fieldmask::*
- 注意事项
  - 使用全局缓存，减少 path 计算开销对主路径的影响



Step 1 volo init 创建项目

```
volo init volo.example.item idl/volo_example_item.thrift
```

Step 2 编辑 volo.yml 开启 with_descriptor 和 with_field_mask

```yaml
entries:
  thrift:
    filename: thrift_gen.rs
    protocol: thrift
    split_generated_files: true
    with_descriptor: true
    with_field_mask: true
```

RPC 框架迭代——Thrift Field Mask

Thrift Field Mask

- 构建：thrift path + thrift descriptor
- 黑白名单模式
- 全局 cache

```rust
static FIELD_MASK_CACHE: LazyLock<AHashMap<FastStr, FieldMask>> = LazyLock::new(|| {
    let mut cache: AHashMap<FastStr, FieldMask> = AHashMap::new();
    let base_desc: TypeDescriptor = psm::SetUserSexRequest::get_descriptor().type_descriptor();
    cache.insert(
        k: "SetUserSexRequest".into(),
        v: FieldMaskBuilder::new(
            &base_desc,
            paths: &[
                "$.ID",
                "$.B",
                "$.C",
            ],
        ) FieldMaskBuilder
        .build() Result<FieldMask, FieldMaskError>
        .unwrap(),
    );
    cache.insert(
        k: "SetUserSexRequestBlack".into(),
        v: FieldMaskBuilder::new(
            &base_desc,
            paths: &[
                "$.test_double[1, 2, 3]",
                "$.base.LogID",
                "$.base.Caller",
                "$.base.Extra{\"key1\", \"key3\"}",
            ],
        ) FieldMaskBuilder
        .with_options(opts: Options::new().with_black_list_mode(true)) FieldMaskBuilder
        .build() Result<FieldMask, FieldMaskError>
        .unwrap(),
    );
    cache
});
```

RPC 框架迭代——Thrift Field Mask

[Thrift Path 定义]

ThriftPath
$
. fieldname
[index , index ...]
{" key "," key "...}
{ id , id ...}
*

```
struct SetUserSexRequest {
    1: i64 ID
    2: string user_name
    3: string B
    4: string C
    5: string D
    6: string user_nick
    7: string F
    10: binary G,
    11: string type,
    12: binary abstract,
    13: list<double> test_double,
    14: UserInfoResponse r,

    255: base.Base base,
}
```

```
paths: &[
    "$.ID",
    "$.B",
    "$.C",
    "$.D",
    "$.user_nick",
    "$.F",
    "$.G",
    "$.user_name",
    "$.test_double[1, 2, 3]",
    "$.type",
    "$.abstract",
    "$.base.LogID",
    "$.base.Caller",
    "$.base.Extra{\"key1\", \"key3\"}",
],
```

RPC 框架迭代——Thrift Field Mask

Thrift Descriptor

```rust
impl SetUserSexRequest {
    pub fn get_descriptor()
    -> &'static ::pilota_thrift_reflect::thrift_reflection::StructDescriptor {
        let file_descriptor = get_file_descriptor();
        file_descriptor
            .find_struct_by_name("SetUserSexRequest")
            .unwrap()
    }
```

RPC 框架迭代——Thrift Field Mask

Thrift Field Mask - 动态裁剪

- 创建 rpc request 时，将对应的 fieldmask 设置到结构体中
- 生成代码中进行动态裁剪

```rust
// default mode
let mut req_clone: SetUserSexRequest = req.clone();
req_clone.set_field_mask(FIELD_MASK_CACHE["SetUserSexRequest"].clone());
let res: Result<OkResponse, ClientError> = client.set_user_sex(req_clone).await;
match res {
    Ok(res: OkResponse) => {
        tracing::info!("{:?}", res);
    }
    Err(e: ClientError) => {
        tracing::error!("{:?}", e);
    }
}

// black list mode
req.set_field_mask(FIELD_MASK_CACHE["SetUserSexRequestBlack"].clone());
let res: Result<OkResponse, ClientError> = client.set_user_sex(req).await;
match res {
    Ok(res: OkResponse) => {
        tracing::info!("{:?}", res);
    }
    Err(e: ClientError) => {
        tracing::error!("{:?}", e);
    }
}
```

RPC 框架迭代——Thrift Field Mask

Thrift Field Mask API

- path_in_mask
- get_path
- for_each_child

后续演进

- fieldmask 序列化与传递

03

HTTP 框架迭代

HTTP/2、连接池、Trace

HTTP 框架迭代

新功能：HTTP/2

- 通过 feature 控制协议版本
- 默认 feature 为 http1

01

客户端的易用性与性能

- 关注性能，为客户端支持连接池
- 优化 Client 封装方式，使用 BoxService 去除复杂泛型带来的编程和调试负担

02

易用性：Trace

- 面向可观测性，提供了 SpanProvider

03

04

Rust 生态与展望

volo、sonic-rs 等 rust 生态的演进方向

Rust 生态更新

sonic-rs 0.5

- 多架构支持，ARM 支持
- 新增 feature，例如 sort_keys， utf8-lossy
- 提升 object 类型查询性能

01

faststr

- 实现 GDB/LLDB 调试插件

02

linkedbytes

- 支持 io slice
- 支持 concat 接口获取完整 bytes

03

Rust 生态展望

2025 年后续规划

RPC 框架

- 协议扩展：Thrift MultiService、Thrift Streaming
- 性能：Shmipc
- 易用性：pilota-build 错误重构；完善服务发现和服务治理组件；trace 优化

01

Rust 生态

- faststr 大小优化
- …

02

THANKS