CloudWeGo

微服务框架 Kitex 的设计、实践及开源

字节跳动服务框架 杨芮

主讲人

目前主要负责字节跳动 Golang 微服务框架的设计开发。
QCon 2021 明星讲师。

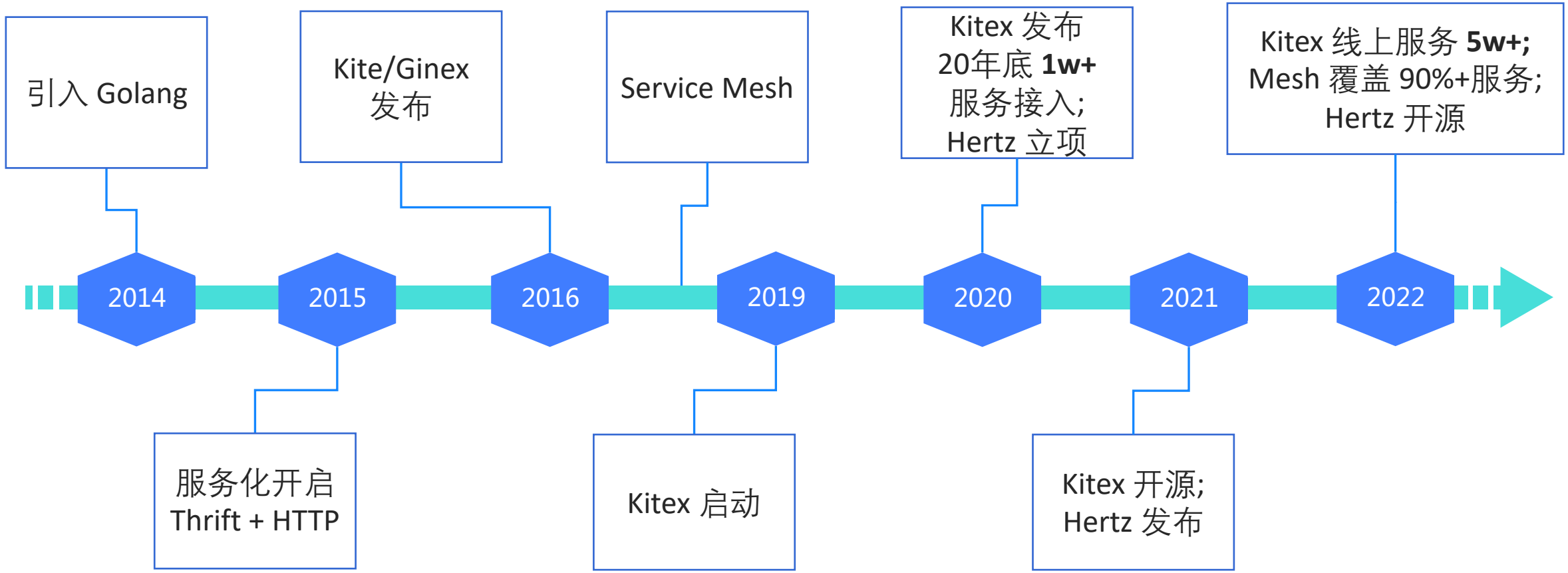
CONTENT

目录

01 字节 Go 微服务框架的演进

字节微服务及 Golang 框架演进

Golang RPC 框架: Kite -> Kitex
Golang HTTP 框架: Ginex -> Hertz



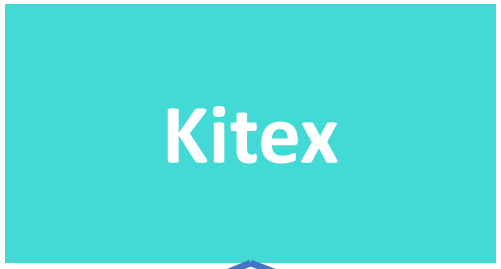
旧框架 Kite 的问题

- 能力扩展受限
  - Kite 强耦合 Thrift、无法横向扩展协议
  - 无法优雅支持其他特性，如新的交互方式
- 维护成本高
  - 生成代码逻辑重，难改造
  - 多仓库维护
- 用户使用体验不佳
  - IDL 定义约束
  - 代码生成工具依赖 Apache Thrift
- 性能优化难
  - 受限于与 Thrift 的深度耦合，网络模型还是编解码层面，优化成本高
- 网络库优化诉求
  - 感知连接状态 & 连接多路复用

业务场景复杂，需求也会多样化
接入服务及调用量逐年增长
不做好准备，后面会越来越难维护

Kitex 框架及其核心依赖

字节接入线上服务 5w+
峰值 QPS 上亿



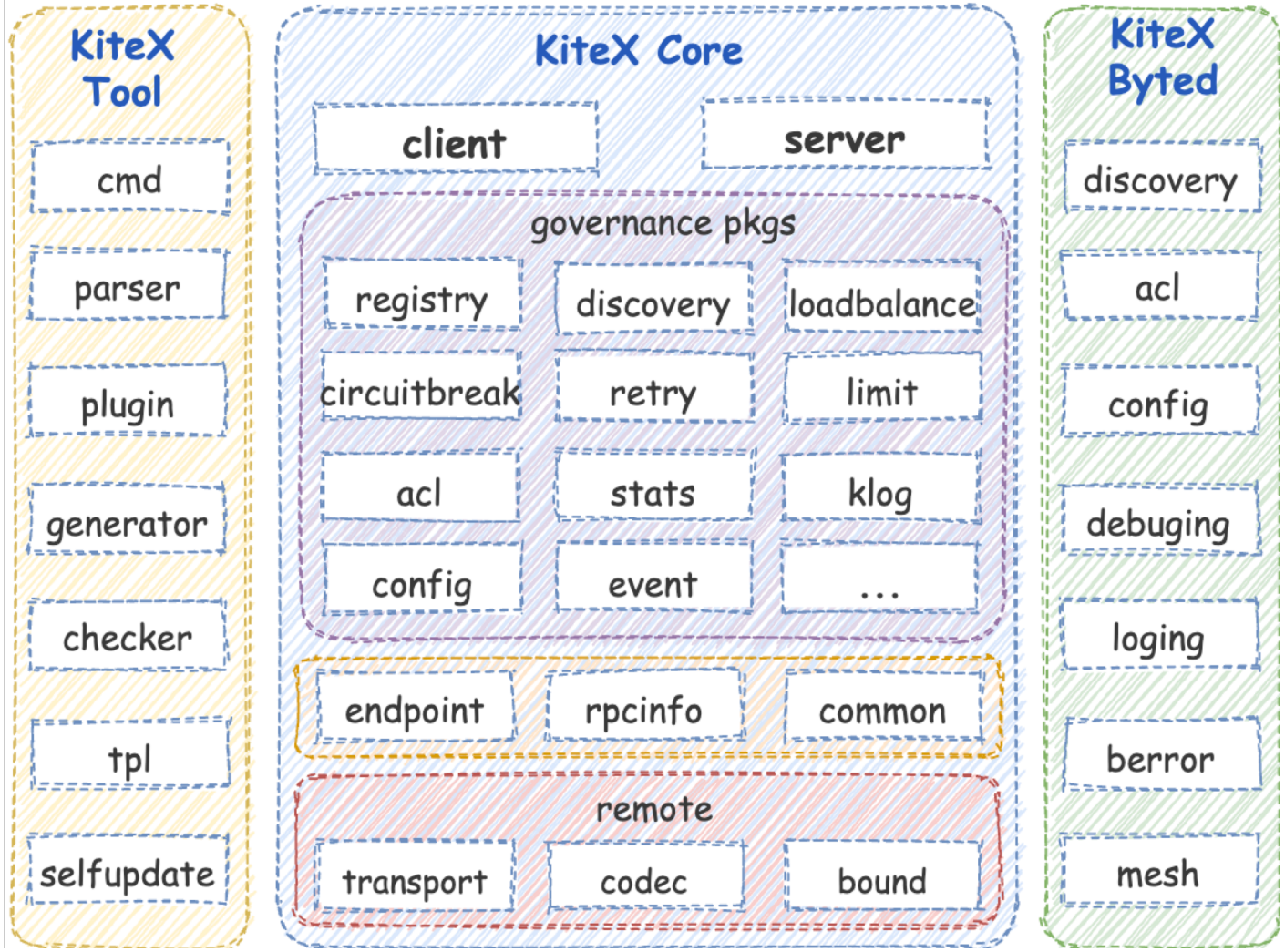
- ✓ 面向 RPC 场景
- ✓ 可感知连接状态
- ✓ 提供高性能的 LinkBuffer 支持 NoCopy 的读写接口
- ✓ 支持内存池、协程池

- ✓ Thrift IDL 解析和代码生成器
- ✓ 支持完善的 Thrift IDL 语法和语义检查
- ✓ 支持插件机制，可根据需求自定义生成代码

02 Kitex 框架设计概述和功能特性

框架设计的关键要素

Kitex 框架设计概述 – 模块划分

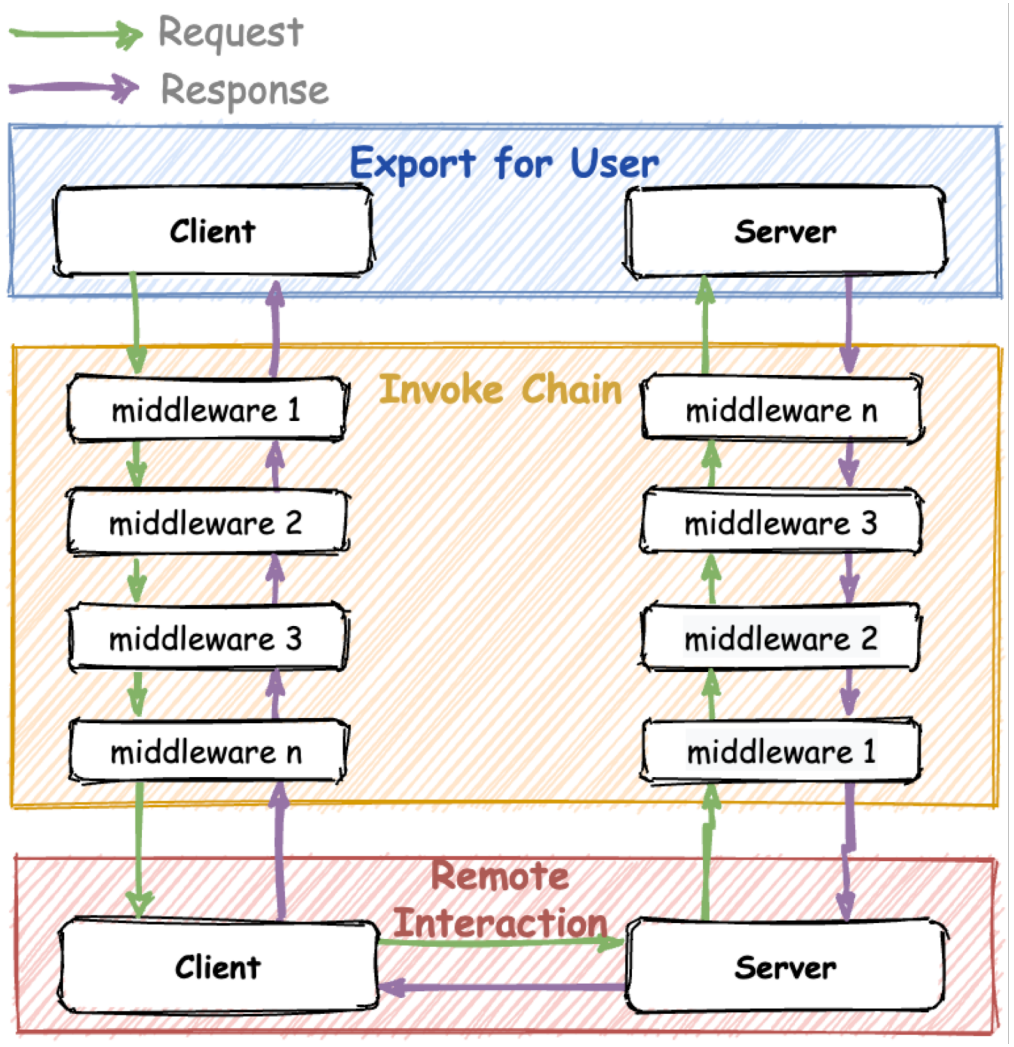


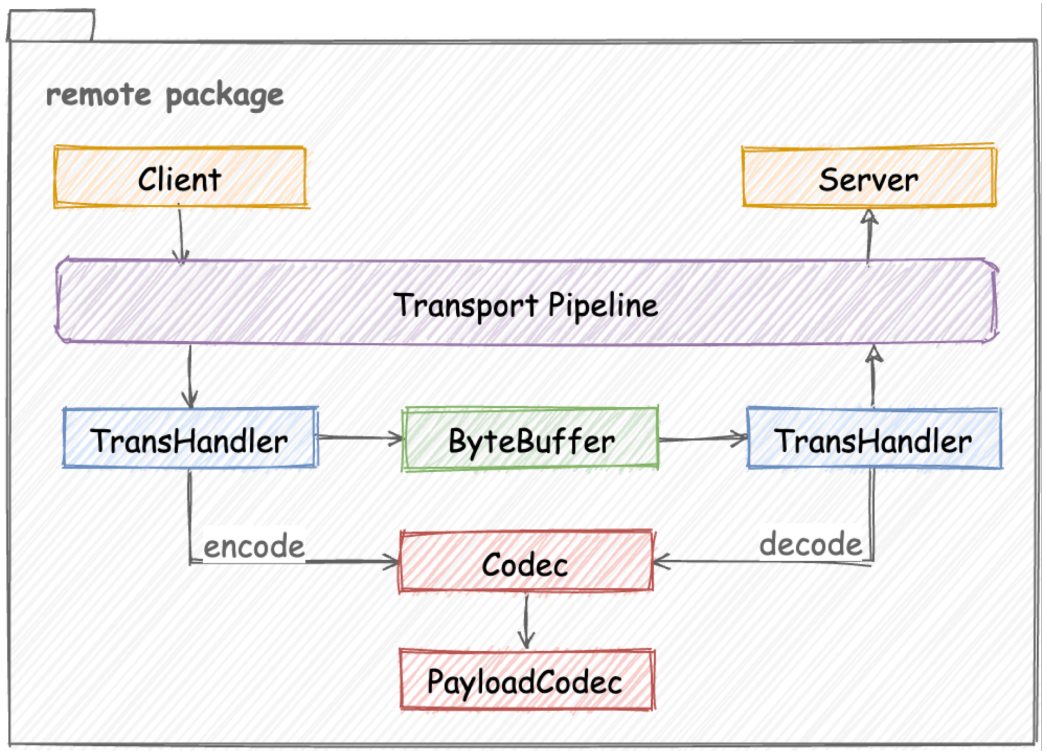
- Kitex 核心
  API 定义、核心逻辑实现以及对各 API 的默认扩展

- Kitex Byted
  与字节内部基础能力集成的扩展实现

- Kitex Tool
  IDL 解析和代码生成器

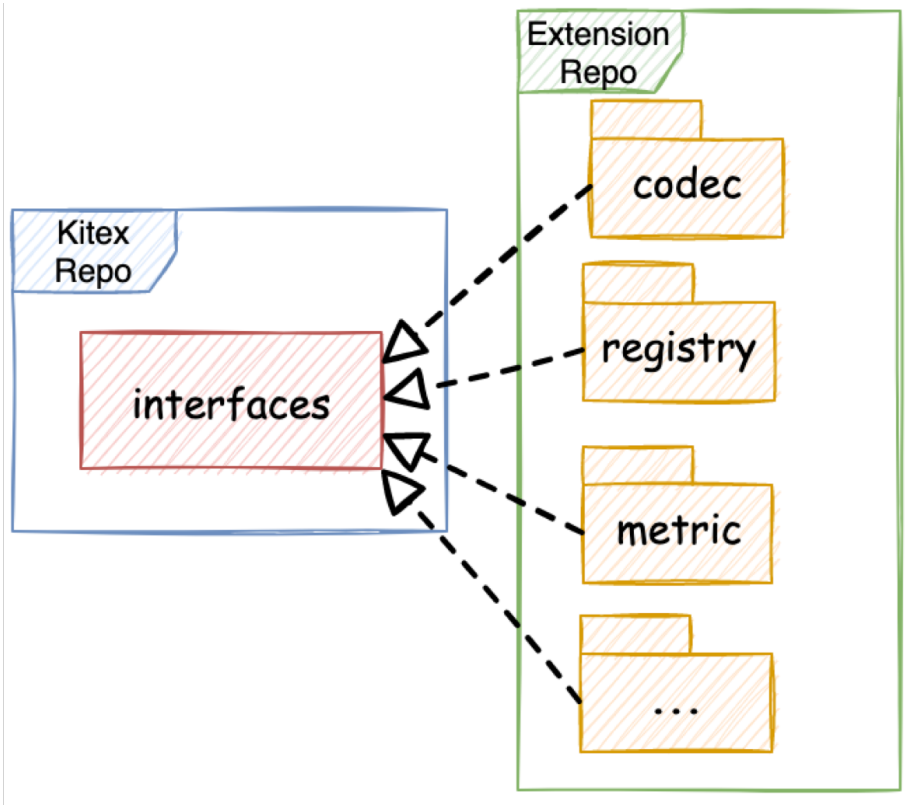
Kitex 框架设计概述 – 模块层级

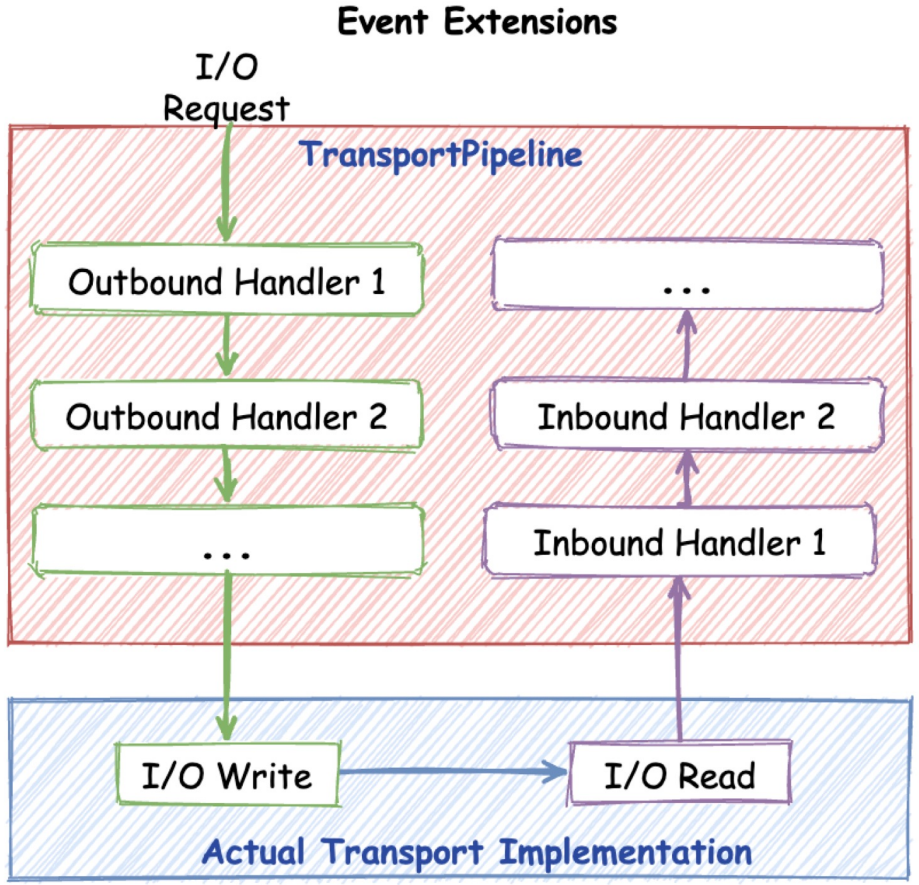




Kitex 框架设计概述 – 扩展性

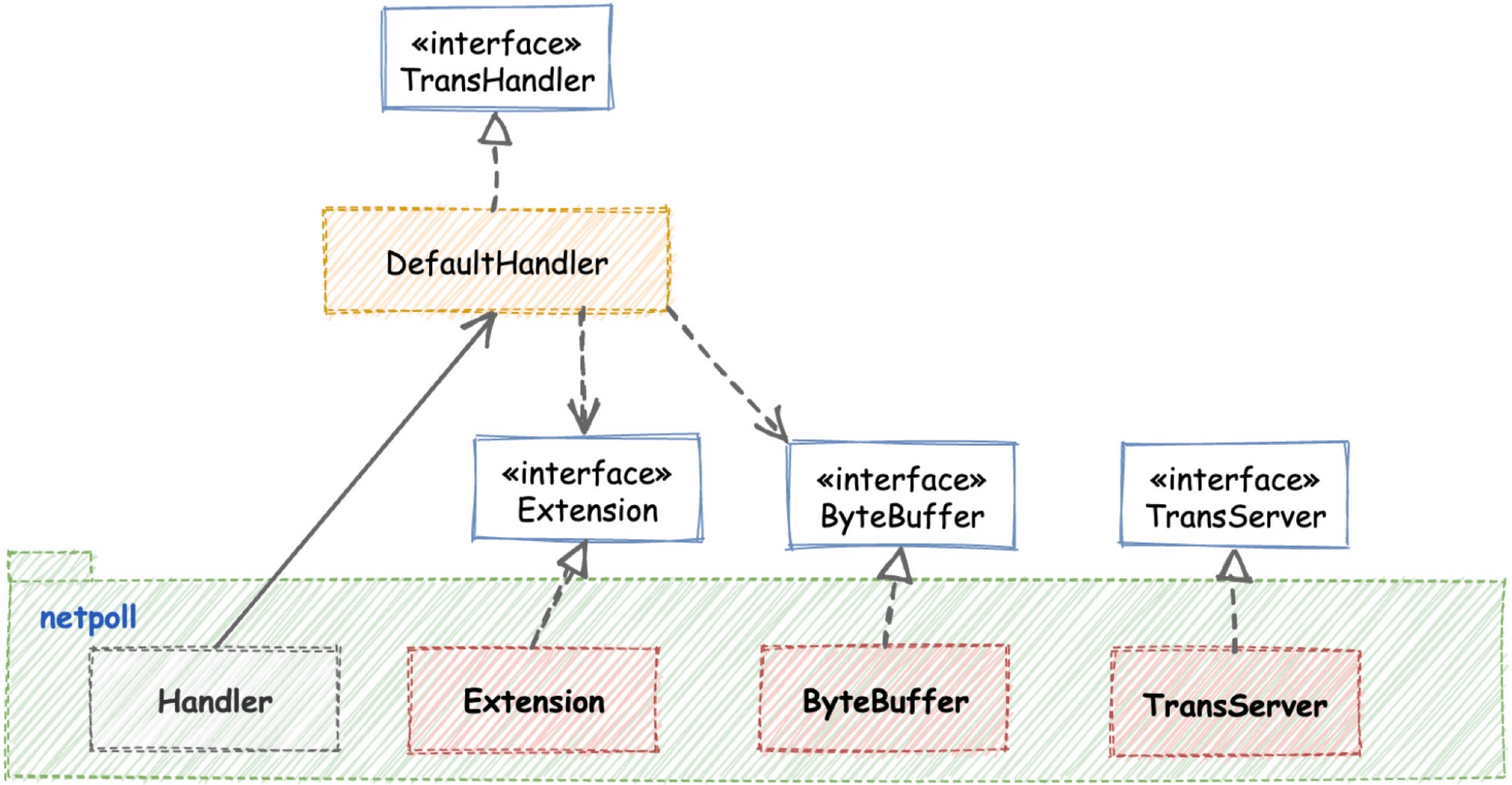
- 协议、通信、服务治理（注册、发现、限流…）、监控、异常处理、元信息处理、诊断…





Kitex 框架设计概述 – 扩展性

- Netpoll 扩展示例



Kitex 框架设计概述 – 易用性

- 无论是 Thrift、Kitex Protobuf、gRPC，一样的使用体验

```
// Thrift Service
service GreetService {
    HelloRequest SayHello(1: HelloResponse req);
}
```

```
// Protobut Service
service GreetService {
    rpc SayHello(HelloRequest) returns (HelloResponse);
}
```

$ kitex -service mydemo greet.thrift

*$ kitex -service mydemo **-type protobuf** greet.proto*

- 实现 handler

```
// GreetServiceImpl implements the last service interface defined in the IDL.
type GreetServiceImpl struct{}

// SayHello implements the GreetServiceImpl interface.
func (s *GreetServiceImpl) SayHello(ctx context.Context, req *demo.HelloRequest) (resp
*demo.HelloResponse, err error) {
    resp = &demo.HelloResponse{Msg: "Hello Kitex!"}
    return
}
```

Kitex 框架设计概述 – 易用性

- RPC直连调用测试代码

```go
func TestGreetService(t *testing.T) {
    var opts []client.Option
    opts = append(opts, client.WithHostPorts(":8888"))
    // 使用 gRPC 协议
    //opts = append(opts, client.WithTransportProtocol(transport.GRPC))
    cli := greetservice.MustNewClient("greetService", opts...)

    resp, err := cli.SayHello(context.Background(), &demo.HelloRequest{Msg: "Hi"})
    if err != nil {
        t.Fatal(err.Error())
    }
    klog.Info(resp.Msg)
}
```

- 编译 & 启动服务

$ ./build.sh
$ ./output/bootstrap.sh

```
2022/05/26 15:03:01.405923 server.go:77: [Info]
KITEX: server listen at addr=[::]:8888
```

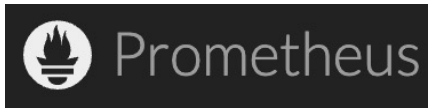
- 测试

$ go test -v -run TestGreetService

```
2022/05/29 15:03:10.006479 client_test.go:20: [Info]
Hello Kitex!
```

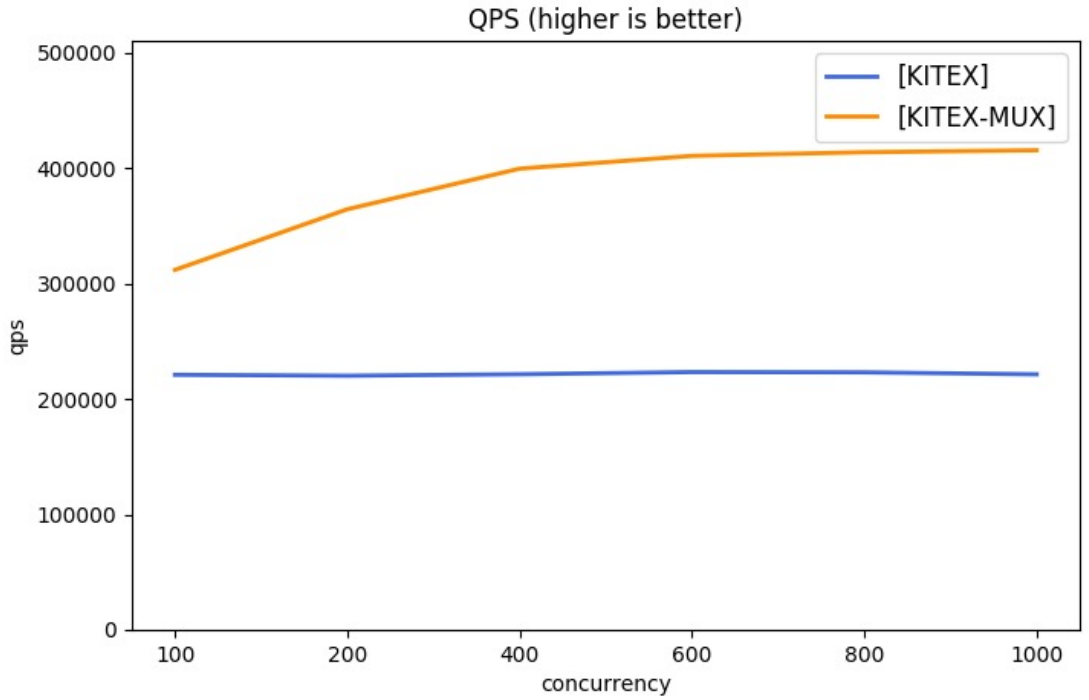
Kitex 框架设计概述 – 功能丰富度

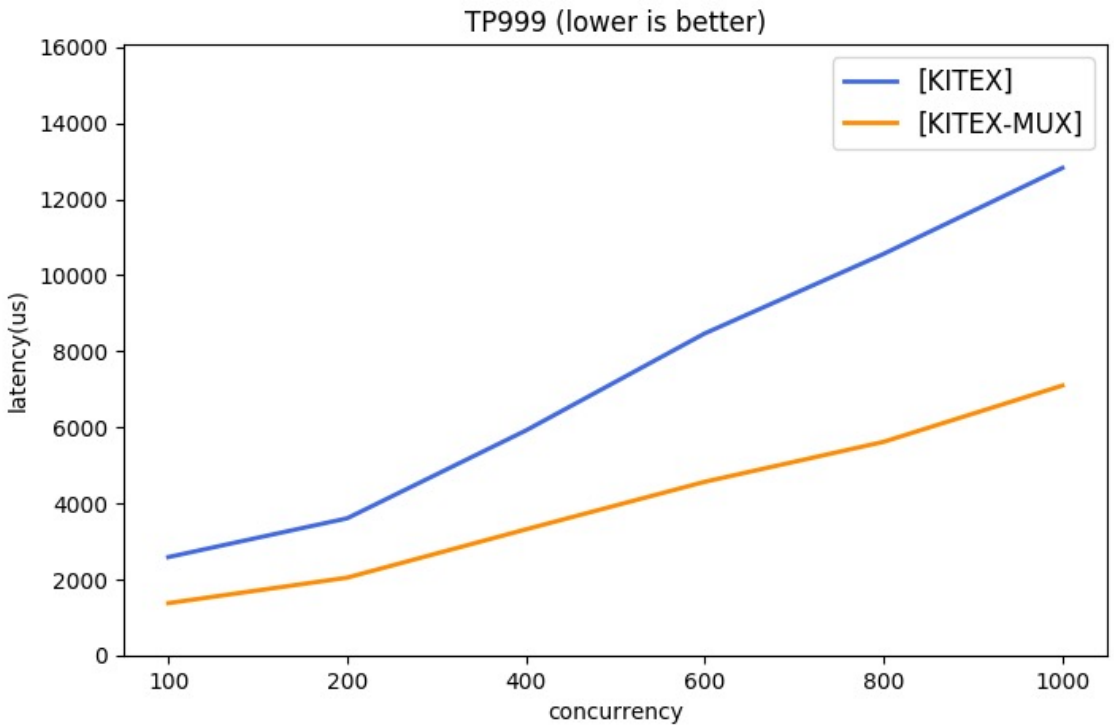
- 内置：通用的内置到框架，也支持扩展定制
  - 如负载均衡、熔断、限流、超时、重试、连接池、监控埋点、日志
- 扩展：非通用的扩展完善
  - 如服务注册发现、指标监控、链路跟踪、配置



Kitex 框架设计概述 – 高性能

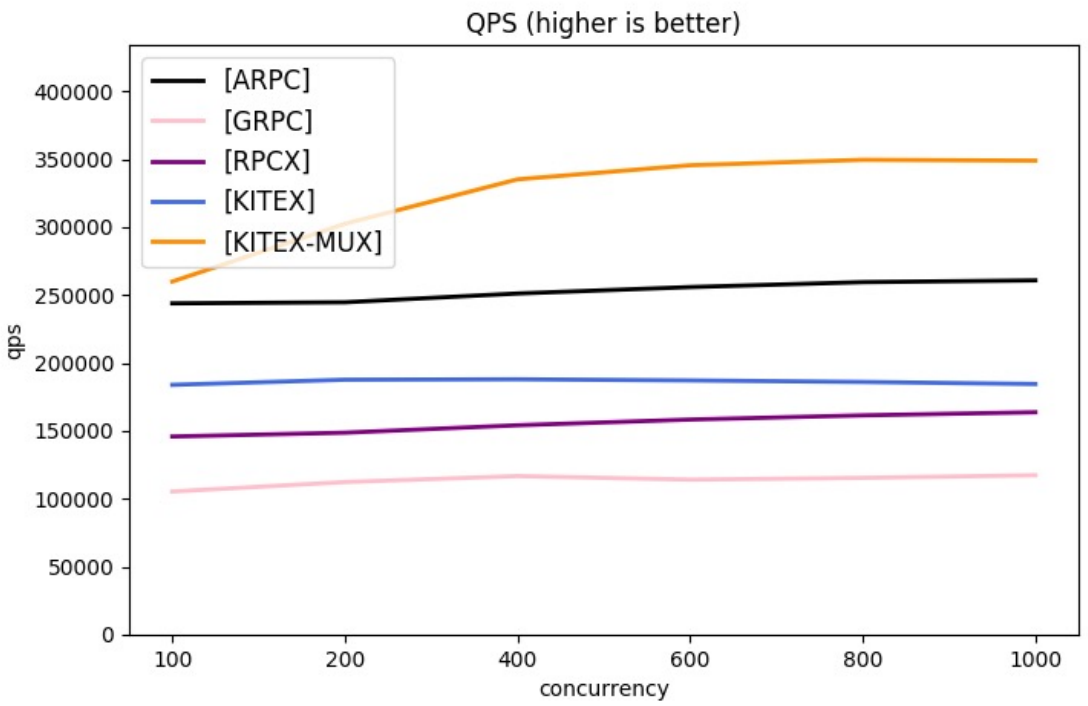
- **Thrift**

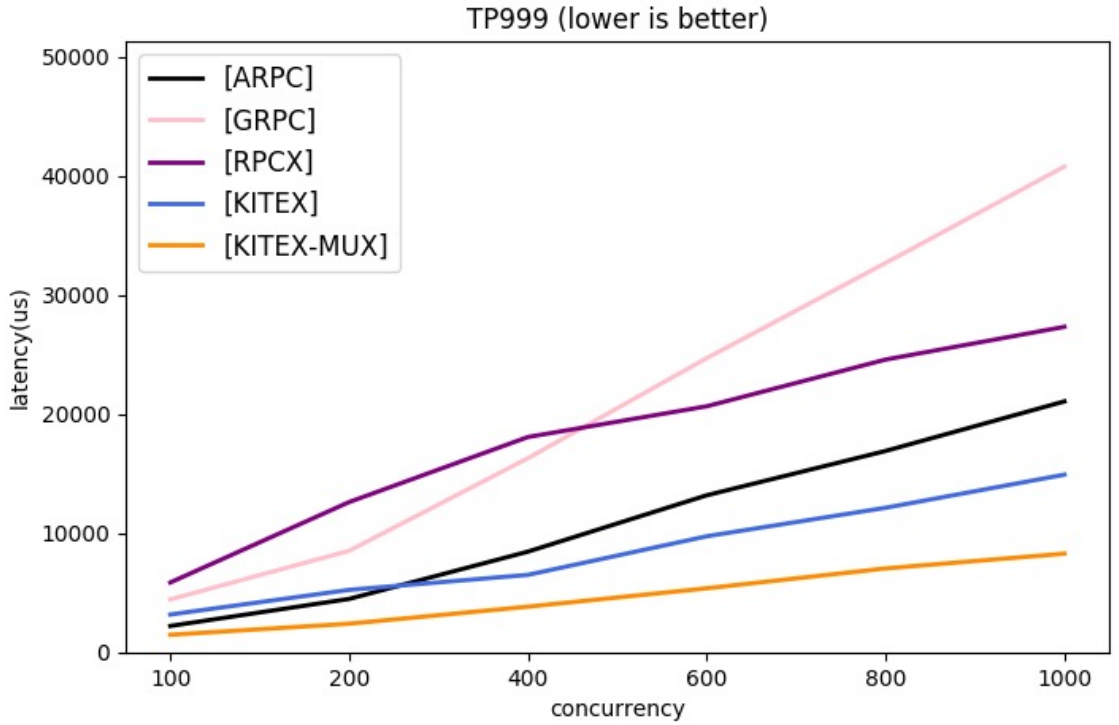




Kitex 框架设计概述 – 高性能

➢ Protobuf





注意：以上是各框架对 Protobuf 支持的压测，但并非使用一个协议，gRPC 协议基于 HTTP2 处理复杂性更高。
后面有对针对 gRPC 协议的压测数据。

Kitex 功能特性 – 协议、通信

多消息协议

- Thrift
- Kitex Protobuf
- gRPC

多应用层传输协议

- TTHeader
- HTTP2

多交互方式

- Ping-Pong/Unary
- Oneway
- Streaming

多种连接类型

- 长连接池
- 连接多路复用
- 短连接

Kitex 功能特性 – Thrift 高性能编解码

➢ Thrift – FastWrite/Read

结合 Netpoll NoCopy API 重新设计的 Thrift 生成代码，减少函数调用、内存拷贝，优化生成代码提升 inline 效果。

	A	B	C	D	E
1	**Benchmark**	**iter**	**time/op**	**bytes/op**	**allocs/op**
2	BenchmarkOldMarshal-4	7830	138 µs ± 3%	25.4KB	19
3	BenchmarkNewMarshal-4	42490	29 µs ± 3%	26.4KB	11
4	**Marshal Delta**		**-78.97%**	**3.87%**	**-42.11%**
5	BenchmarkOldUnmarshal-4	5000	199 µs ± 3%	4720	1360
6	BenchmarkNewUnmarshal-4	13100	94µs ± 5%	4700	1280
7	**Unmarshal Delta**		**-52.93%**	**-0.24%**	**-5.38%**

杨芮 2664

Kitex 功能特性 – Thrift 高性能编解码

➢ Frugal

无需生成编解码代码，基于 JIT 的高性能动态 Thrift 编解码器~

结合 Kitex 在大部分场景性能表现优于生成代码的编解码

	A	B	C	D	E	F	G
1	数据结构	连接类型	并发数	包大小	极限 TPS	延迟 TP99	延迟 TP999
2	复杂	[KITEX-MUX]	100	1024	169131.15(+0.5%)	1.63(-0.6%)	2.13(-0.9%)
3		[KITEX-MUX]	200	1024	187606.33(-0.7%)	2.69(+1.1%)	3.58(+3.8%)
4		[KITEX]	100	1024	167613.9(+20.8%)	1.39(-22.3%)	7.92(+26.9%)
5		[KITEX]	200	1024	165659.22(+18.2%)	2.83(+0.4%)	12.14(-13.3%)
6	简单	[KITEX-MUX]	100	1024	315973.82(+0.2%)	1.07(+0.0%)	1.4(+0.0%)
7		[KITEX-MUX]	200	1024	364755.89(+0.3%)	1.65(+0.6%)	2.11(+1.0%)
8		[KITEX]	100	1024	232710.84(+7.7%)	0.69(-34.3%)	0.99(-66.1%)
9		[KITEX]	200	1024	238802.41(+10.3%)	1.93(-12.7%)	2.75(-16.9%)

Kitex 功能特性 – Protobuf 高性能编解码

➢ Protobuf – FastWrite/Read

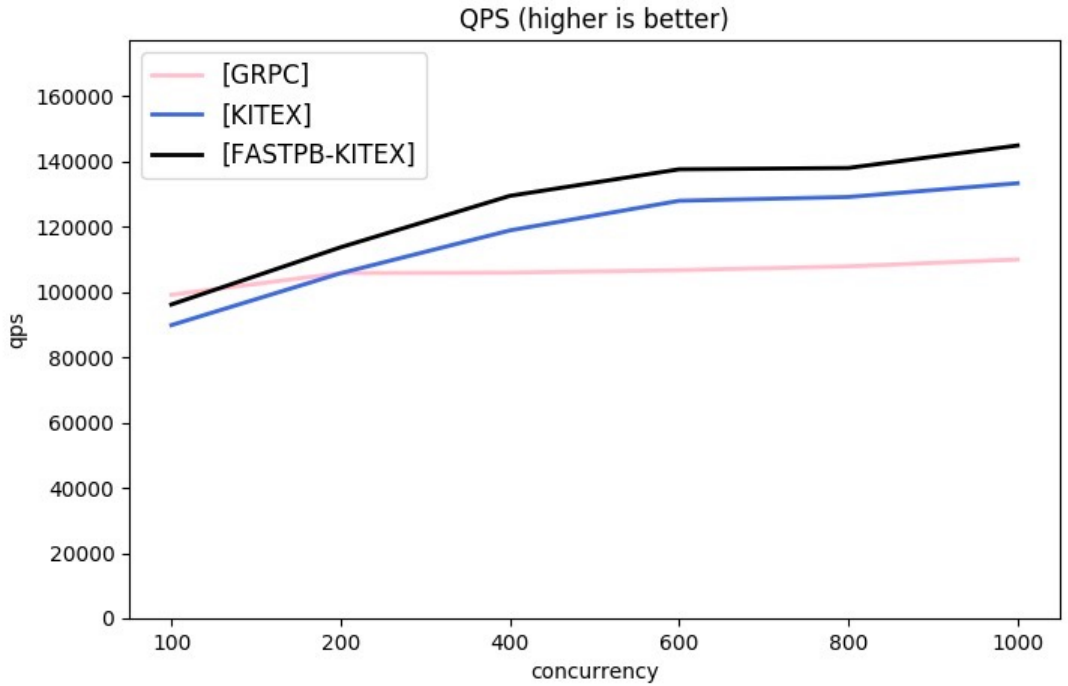
参考 Thrift 的 FastWrite/Read 实现，重新设计了 Protobuf 生成代码，避免反射和拷贝的开销。
目前内置在 Kitex，后续考虑拆分开源。

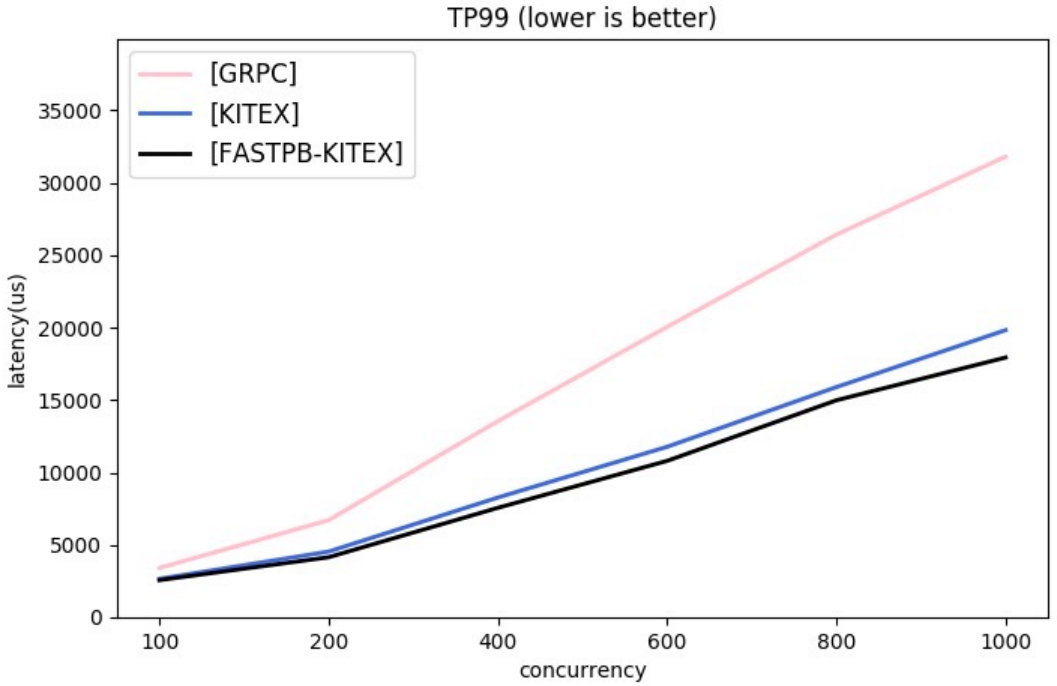
- Protobuf Benchmark 官方对比测试
  - FastWrite: (ns/op) ↓41.5% ，(B/op) ↓4.5%
  - FastWrite: (ns/op) ↓67.8% ，(B/op) ↓83.9%

Kitex 功能特性 – Protobuf 高性能编解码

➢ Protobuf – FastWrite/Read

- gRPC 协议 Unary 请求压测对比





Kitex 也在针对低并发场景优化性能

Kitex 功能特性 – Thrift Validator

➢ thrift-gen-validator

thriftgo 工具的插件，根据 Thrift IDL 中定义的注解描述约束给对应的 struct 生成 IsValid() error 方法，校验值的合法性。

$ kitex --thrift-plugin validator -service a.b.c youridl.thrift

```
enum MapKey {
    A, B, C, D, E, F
}

struct DemoTestRequest{
    1: required string Message (vt.max_size = "30", vt.prefix = "Debug")
    2: i32 ID (vt.gt = "1000", vt.le = "10000")
    3: list<double> Values (vt.elem.gt = "0.25")
    4: map<MapKey, binary> KeyValues (vt.key.defined_only = "true", vt.min_size = "1")
}

struct DemoTestResponse{
    1: required string Message (vt.in = "Debug", vt.in = "Info", vt.in = "Warn", vt.in = "Error")
}
```

Kitex 功能特性 – 泛化调用

- **网关场景**

- **网关泛化调用** - 没有生成代码，没有业务数据结构

开源的一些泛化方式

Kitex 泛化方式：基于 idl 中的注解映射

Kitex 功能特性 – 泛化调用

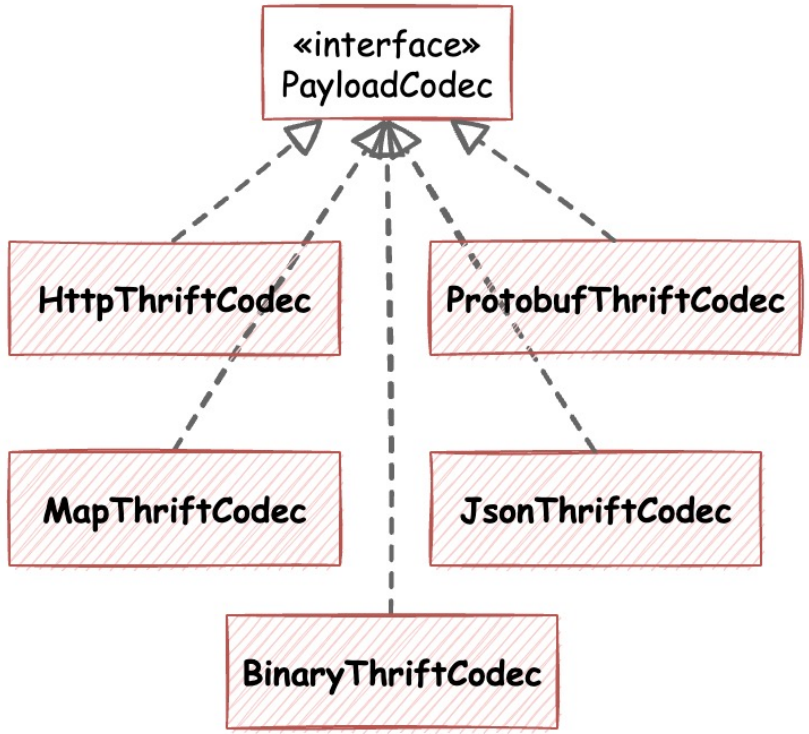
- HTTP -> Thrift 泛化调用

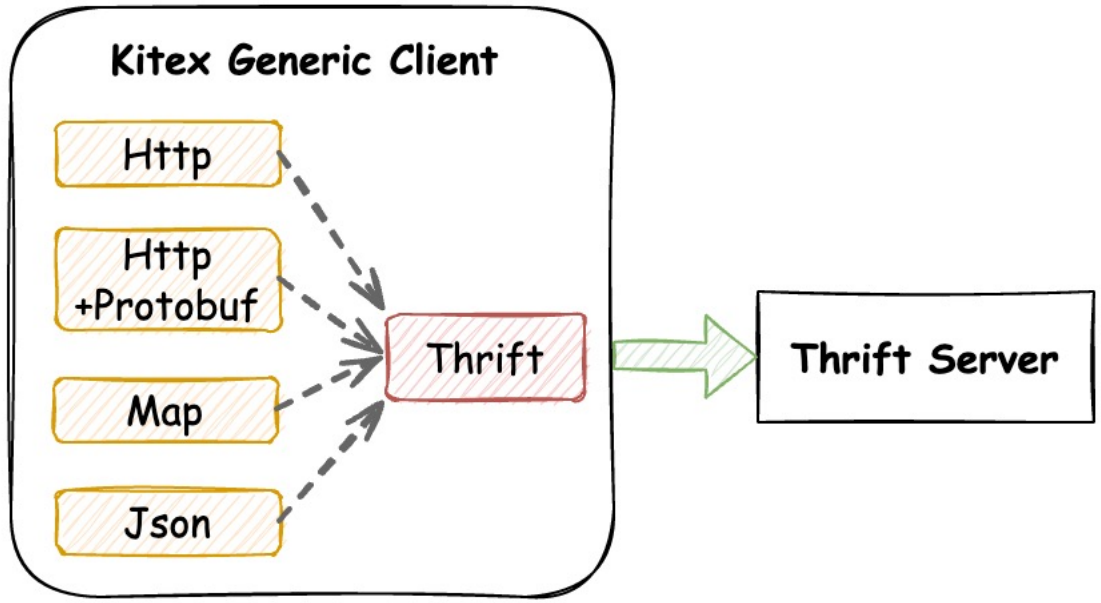
```
struct BizRequest {
    1: optional string text(api.body = 'text')
    3: optional i32 token(api.header = 'token')
    6: optional list<string> req_items(api.query = 'req_items')
    7: optional i32 api_version(api.path = 'version')
    8: optional i64 uid(api.path = 'uid')
}
struct RspItem {
    1: optional i64 item_id
    2: optional string text
}
struct BizResp {
    1: optional map<i64, RspItem> rsp_items
    2: optional i32 http_code(api.http_code
}
service BizService {
    BizResp mock(1: BizReq req)(api.post =
}
```

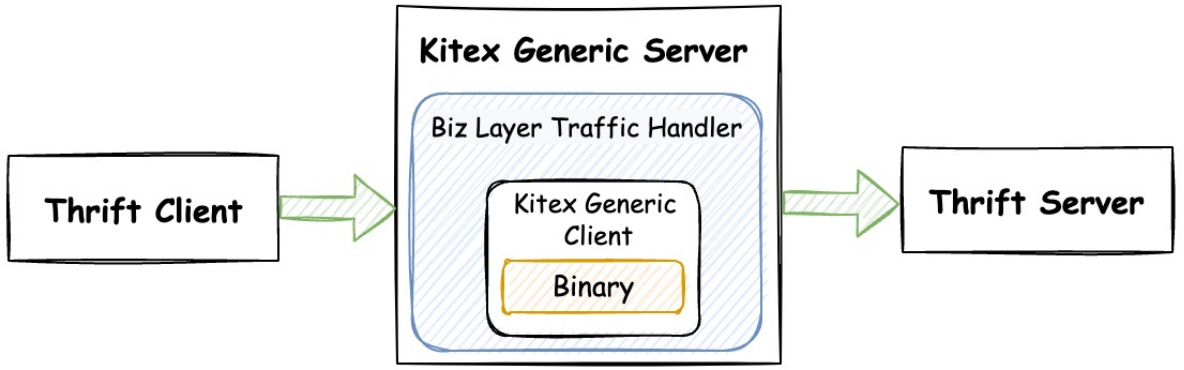
```
// 初始化 Client
p, err := generic.NewThriftFileProvider("idl_path")
g, err := generic.HTTPThriftGeneric(p)
cli, _ := genericclient.NewClient("serviceName", g, opts...)

// 调用 "mock" 方法
resp, err := cli.GenericCall(ctx, "mock", generic.FromHTTPRequest(httpReq))
```

Kitex 功能特性 – 泛化调用

- 泛化调用相关的扩展实现







有需求也会考虑 Protobuf 泛化调用

Kitex 功能特性 – 可视化-细粒度监控埋点

```
// 服务端开始处理业务handler
stats.Record(ctx, rpcinfo, stats.ServerHandleStart, nil)
// 服务端完成处理业务handler
stats.Record(ctx, rpcinfo, stats.ServerHandleFinish, nil)
// 发送包大小
rpcinfo.AsMutableRPCStats(ri.Stats()).SetS
// 接收包大小
rpcinfo.AsMutableRPCStats(ri.Stats()).SetR
```

```
// 框架内置的 events.
var (
    RPCStart  = newEvent(rpcStart, LevelBase)
    RPCFinish = newEvent(rpcFinish, LevelBase)

    ServerHandleStart  = newEvent(serverHandleStart, LevelDetailed)
    ServerHandleFinish = newEvent(serverHandleFinish, LevelDetailed)
    ClientConnStart    = newEvent(clientConnStart, LevelDetailed)
    ClientConnFinish   = newEvent(clientConnFinish, LevelDetailed)
    ReadStart          = newEvent(readStart, LevelDetailed)
    ReadFinish         = newEvent(readFinish, LevelDetailed)
    WaitReadStart      = newEvent(waitReadStart, LevelDetailed)
    WaitReadFinish     = newEvent(waitReadFinish, LevelDetailed)
    WriteStart         = newEvent(writeStart, LevelDetailed)
    WriteFinish        = newEvent(writeFinish, LevelDetailed)
)

// 用户自定义 event
stats.DefineNewEvent("yourEvent", LevelDetailed)
```

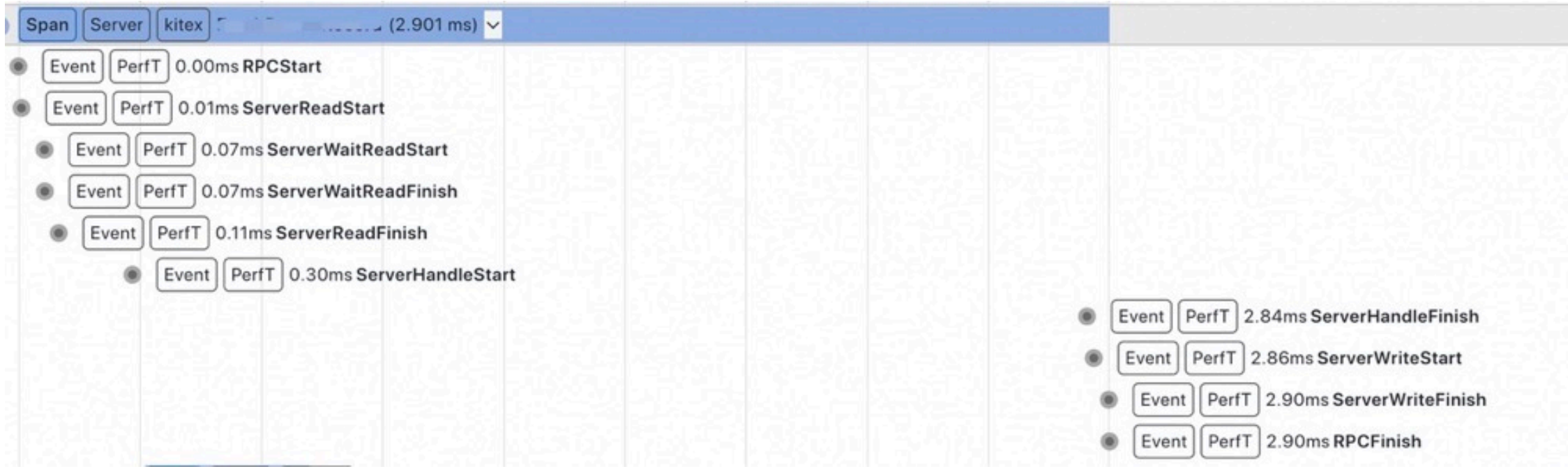
Kitex 功能特性 – 可视化-细粒度监控埋点

- 结合链路跟踪展示详细 RPC 调用信息

```
// Tracer is executed at the start and finish of an RPC.
type Tracer interface {
        Start(ctx context.Context) context.Context
        Finish(ctx context.Context)
}

// 获取埋点信息
rpcinfo.GetRPCInfo(ctx).Stats()

// Server 添加 Tracer
server.WithTracer(yourTracerImpl)
// Client 添加 Tracer
client.WithTracer(yourTracerImpl)
```

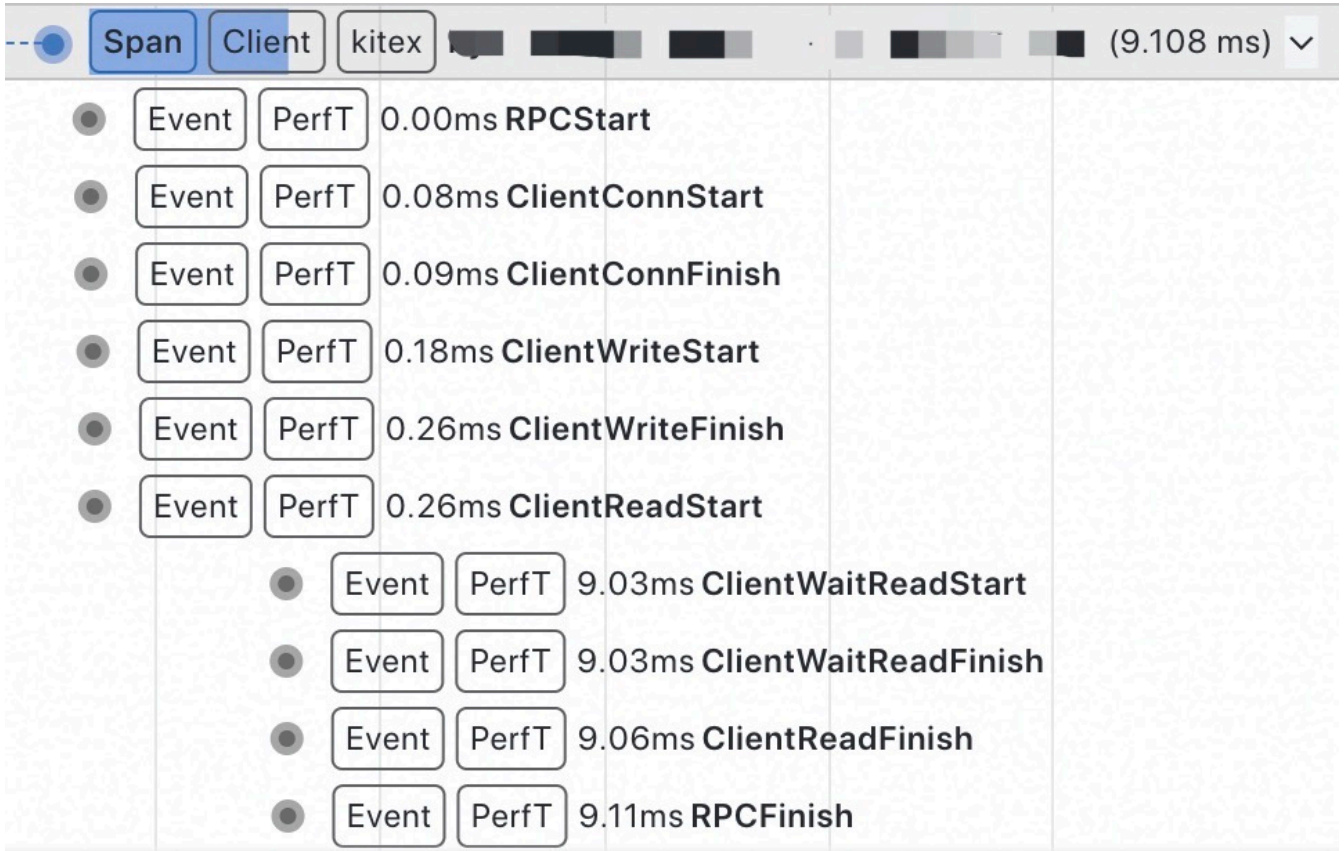
Kitex 功能特性 – 可视化-细粒度监控埋点

- 服务端 Tracer 展示



Kitex 功能特性 – 可视化-细粒度监控埋点

- 调用端 Tracer 展示



Kitex 功能特性 – 可视化-服务信息查询

- 框架 Debug 运行时状态查询 & pprof

/kitex

Types of check info available:

client_info
conn_pool
dependencies
goversion
instances
lb_cache
server_info

Descriptions:

client_info:	Service info and all config info in client side, include dynamic config change events. You can use query param [destpsm, queryinfo] to filter show info. Eg: 'http://ip:port/kitex/client_info?destpsm=xxx&queryinfo=consul'. The optional [queryinfo] include: 'config', 'remote_config', 'options', 'events', 'service_info', 'circuit'
conn_pool:	Conn pool info, just for client side. You can use query param [destpsm, addr] to filter show info. Eg: 'http://ip:port/kitex/conn_pool?destpsm=xxx&addr=xxx', addr is 'ip:port'.
dependencies:	Dependencies info of current repository.
goversion:	Go version.
instances:	Discovery instances, just for client side. You can use query param [destpsm] to filter show info. Eg: 'http://ip:port/kitex/instances?destpsm=xxx'.
lb_cache:	Global instances cache for discovery, the instances are split with route info, just for client side.
server_info:	Service info and all config info in server side, include dynamic config change events. You can use query param [queryinfo] to filter show info. Eg: 'http://ip:port/kitex/server_info?queryinfo=remote_config'. The optional [queryinfo] include: 'config', 'remote_config', 'options', 'events', 'service_info','

http://ip:18888/kitex/runtime

http://ip:18888/debug/pprof

Kitex 功能特性 – 信息透传

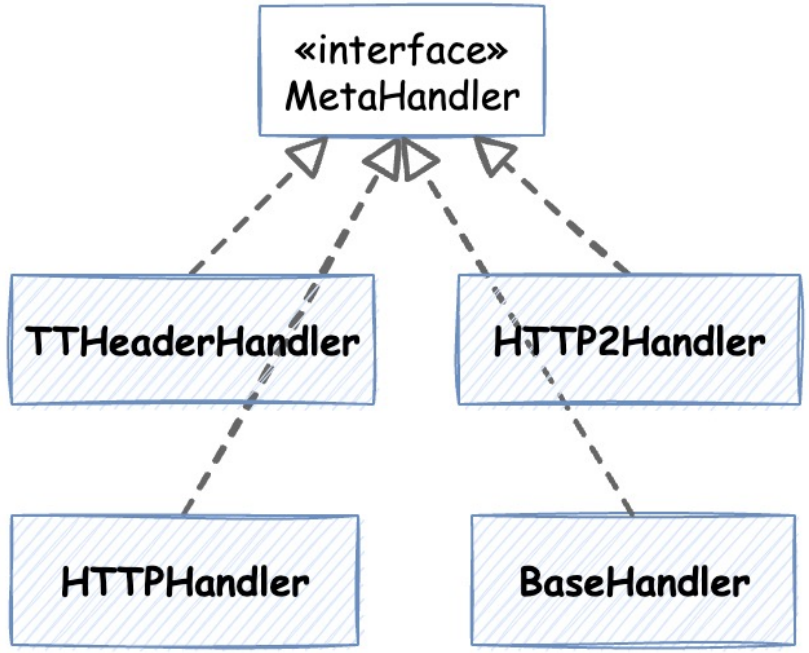
- Metainfo
  - Golang 组件通用的元信息透传基础库
  - github.com/bytedance/gopkg/cloud/metainfo

```
/* 上游透传给下游 */
// 上游透传给直接下游，不做链路透传
ctx = metainfo.WithValue(ctx, "Key1", "Value1")
// 链路透传，该 k-v 会持续透传给下游
ctx = metainfo.WithPersistentValue(ctx, "Key2",
"Value2")

/* 下游获取上游的透传 */
// 获取直接上游的透传信息
value, exist = metainfo.GetValue(ctx, "Key1")
// 获取链路透传信息
value, exist = metainfo.GetPersistentValue(ctx, "Key1")
```

Kitex 功能特性 – 信息透传

- MetaHandler

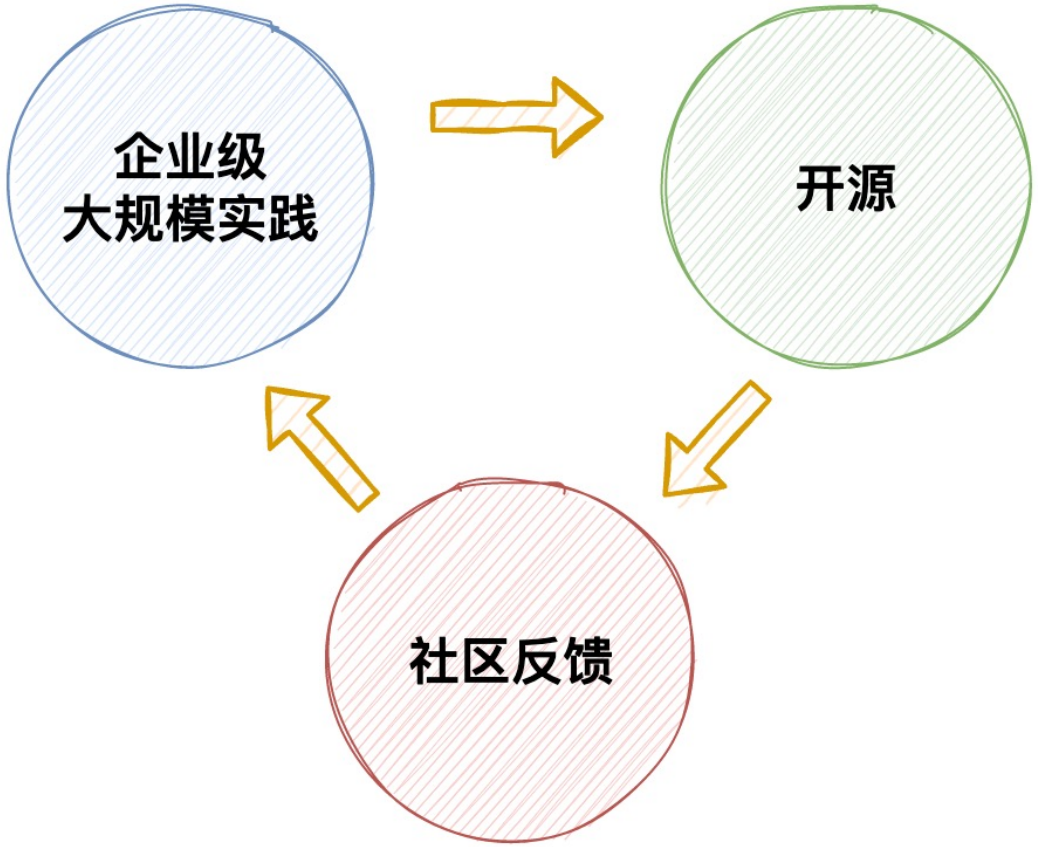
```go
// MetaHandler reads or writes metadata through certain protocol.
type MetaHandler interface {
    WriteMeta(ctx context.Context, msg Message) (context.Context, error)
    ReadMeta(ctx context.Context, msg Message) (context.Context, error)
}

// StreamingMetaHandler reads or writes metadata through streaming
header(http2 header)
type StreamingMetaHandler interface {
    // writes metadata before create a stream
    OnConnectStream(ctx context.Context) (context.Context, error)
    // reads metadata before read first message from stream
    OnReadStream(ctx context.Context) (context.Context, error)
}
```



03 Kitex 开源和在字节内的实践

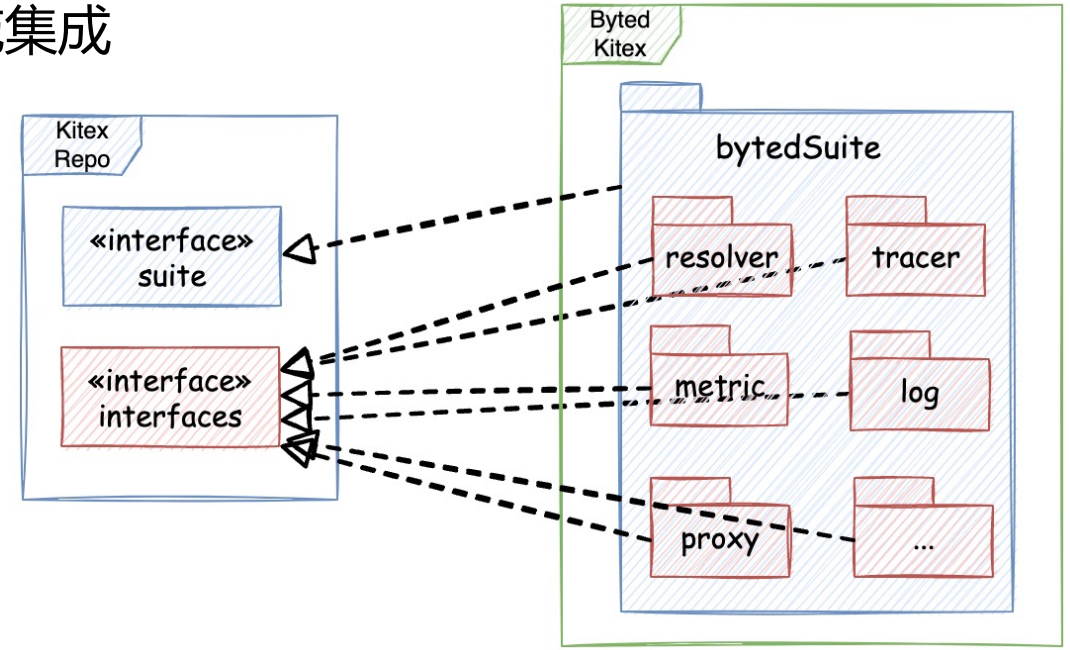
Kitex 开源

Kitex 不是为了开源而实现的，但它的实现是面向开源的~



Kitex 字节内部的实践

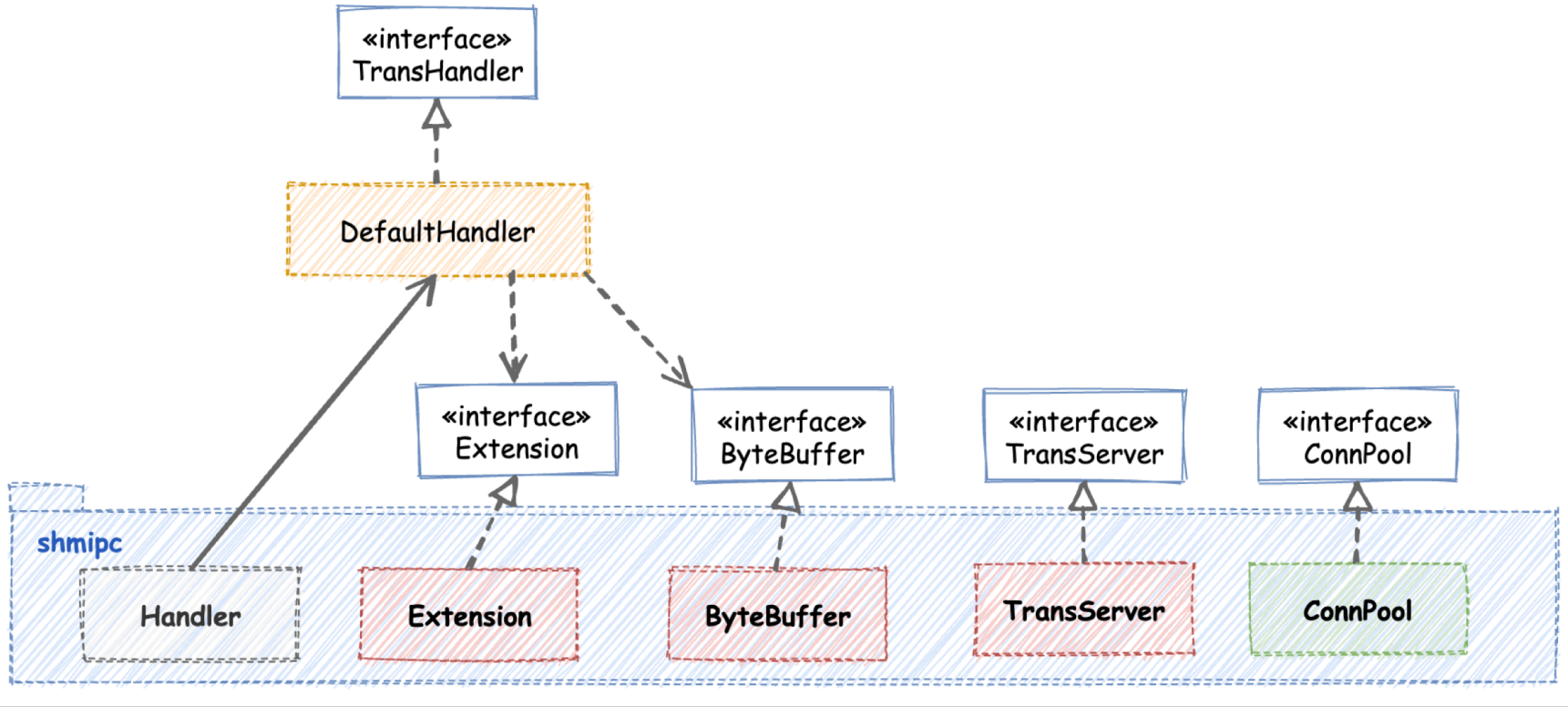
- 如何与内部基础设施集成




```
// Client 初始化 option
options = append(options, byted.ClientSuiteWithConfig(serviceInfo(), config))

// Server 初始话 option
options = append(options, byted.ServerSuiteWithConfig(serviceInfo(), config))
```

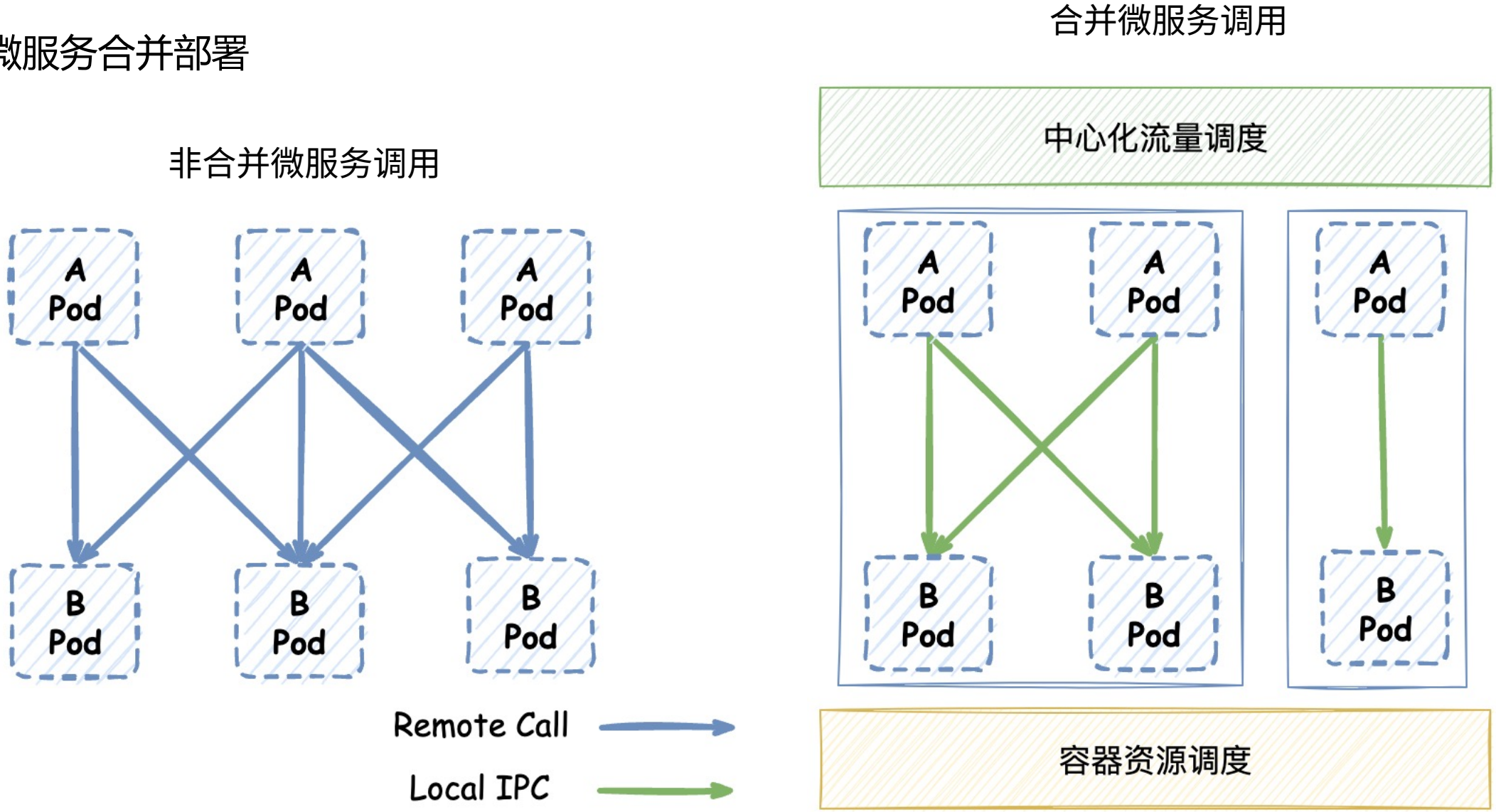
Kitex 字节内部的实践

- ShmIPC-应用在 ServiceMesh 和 IPC 场景
  - 根据环境变量开启，用户无需感知



Kitex 字节内部的实践

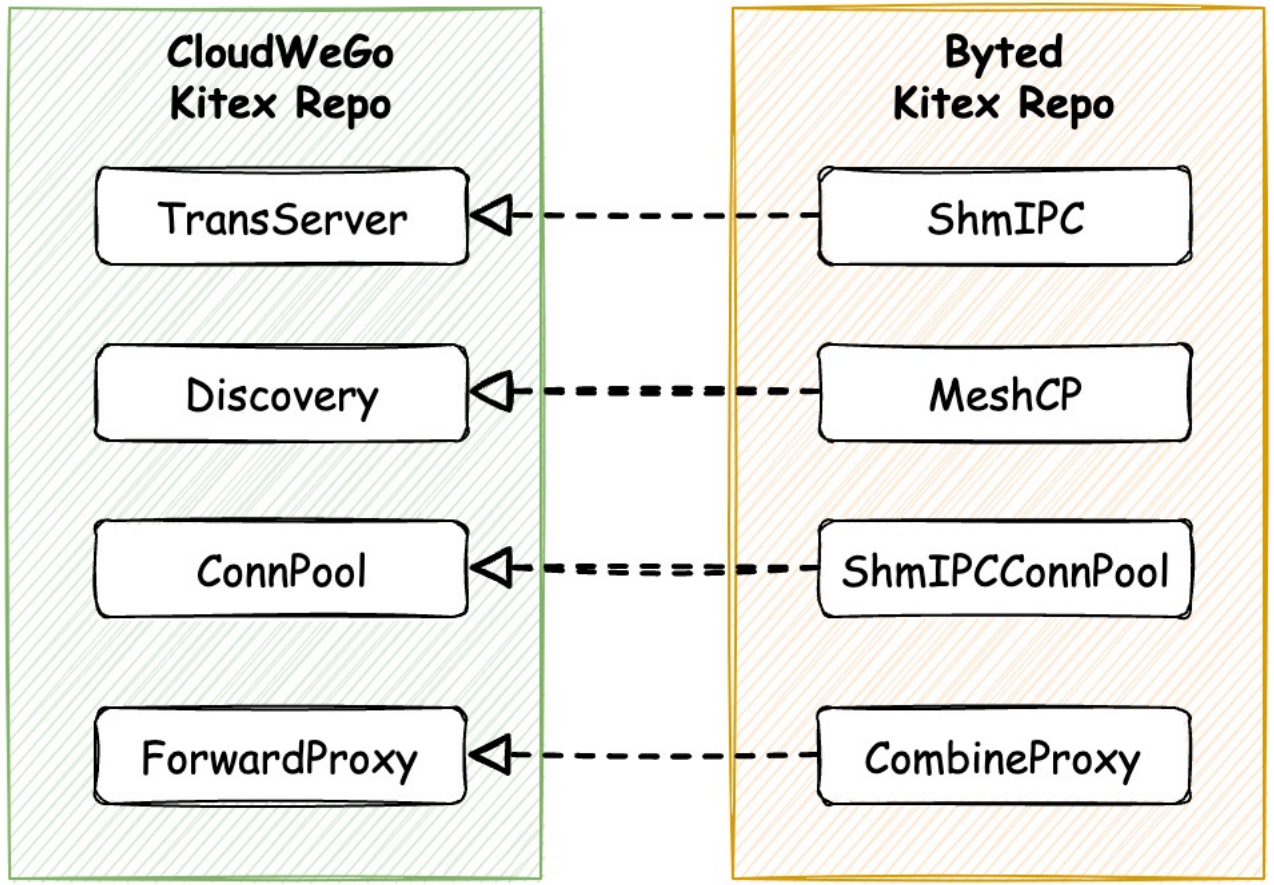
- 微服务合并部署



Kitex 字节内部的实践

- 微服务合并部署

- 初始化条件判断 - 合并模式
- 监听多个地址
- 单独的服务发现
- 单独的连接池策略（ShmIPC）
- 多通信方式：Shm、UDS、TC



04 总结和展望

总结

- 字节 Golang 框架的演进和遇到的问题
- 框架设计的关键要素：扩展性、易用性、功能丰富度、高性能
- Kitex 框架设计概述，并介绍了部分特有的功能特性

展望

- 与社区同学共建，持续丰富社区生态
- 结合工程实践，为微服务开发者提供更多便利
- 完善好 BDThrift 生态，同时也会优化 Protobuf/gRPC
- 更多特性支持或开源，ShmIPC、Proxyless、 Quic、Protobuf 泛化…

CloudWeGo 开源项目

www.cloudwego.io

服务框架

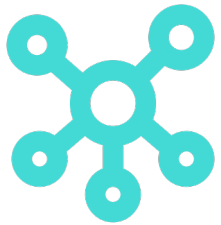
Kitex

Hertz

Rust 框架

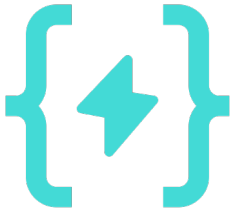
...

基础库



Netpoll

Thriftgo



Sonic

...

Frugal

THANKS

得物技术沙龙Golang反馈问卷（填写完可以抽奖）

扫一扫

CloudWeGo 公众号

CloudWeGo