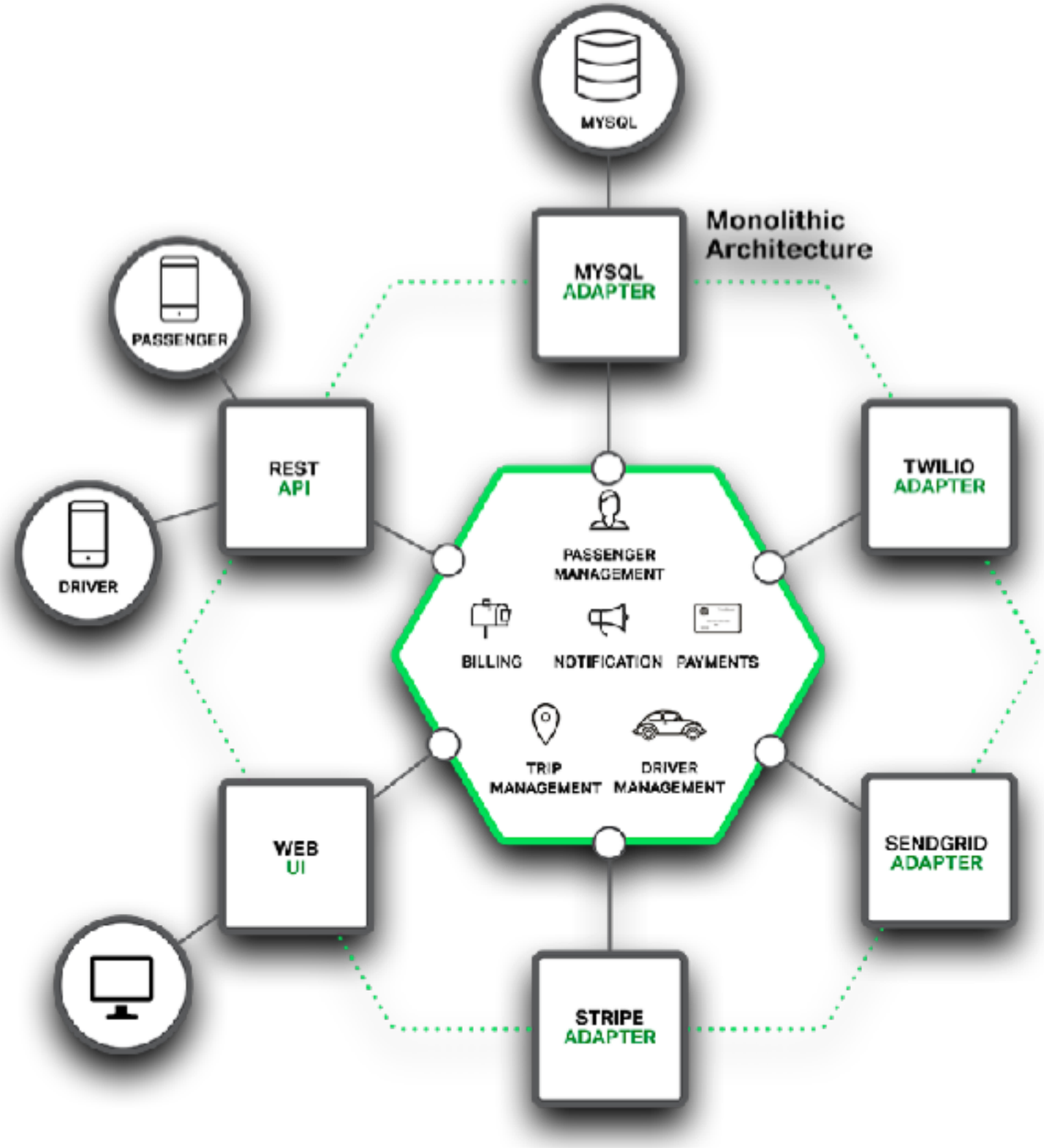
gRPC在美图微服务中的应用

meitu

提纲

- 背景
- 设计的选型考量点
- 微服务框架的设计和实现
- gRPC的一些经验

单体应用



注：上图来源于网上

微服务架构

收益

- 解耦
- 独立部署，让持续部署成为可能
- 有利于水平扩展
- 容错性
- 减少重复开发

带来问题

- 维护成本
- 链路变长，排查难度加大
- 服务的SLA

meitu

meitu微服务面临的一些挑战

- 差异化的业务架构模型

- 多语言化的架构形态

- 性能和可用性要求

当前团队的现状

- Dubbo
- Yar
- Avro
- gRPC

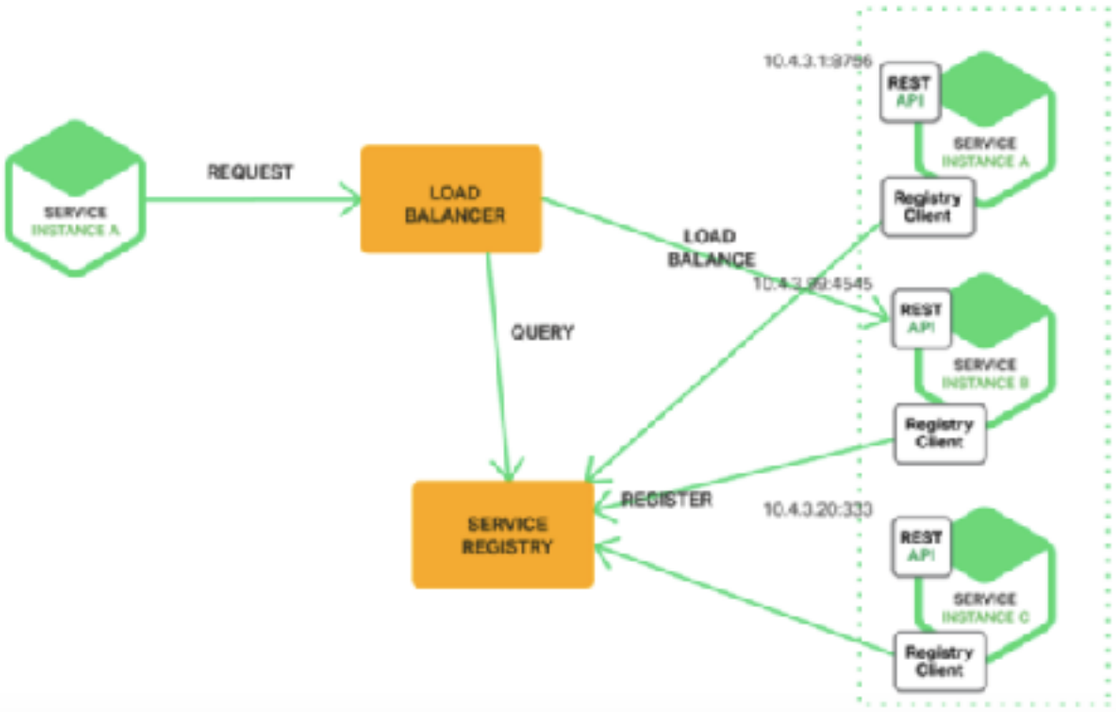
微服务架构考虑的关键点

- 扩展的灵活性
- 多语言异构系统支撑
- 服务可用性的保障
- 服务治理和管控
- 减少接入的成本
- 性能损耗尽可能小

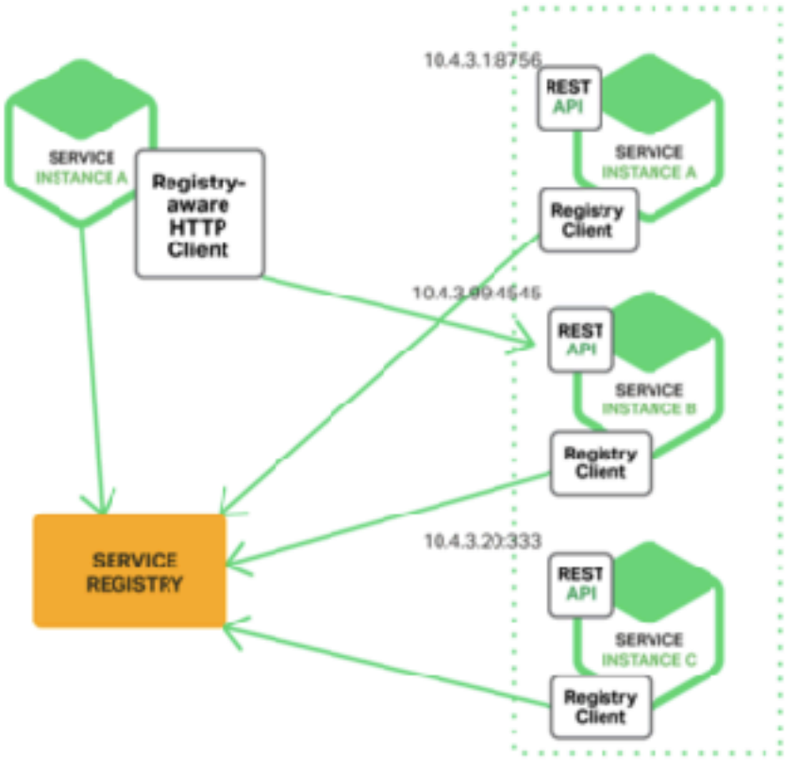
meitu

微服务架构的两种模式

Server-side discovery



Client-side discovery



注：上图来源于网上

中心化和非中心化

	中心化	非中心化
优点	1. 更好的治理—授权，管控，流控 2. 升级方便 3. 兼容性强—协议的兼容	1. 性能更高 2. 可靠性更高
缺点	1.增加可用性保障难度 2.容量瓶颈	1.每个语言都要实现 2.升级困难

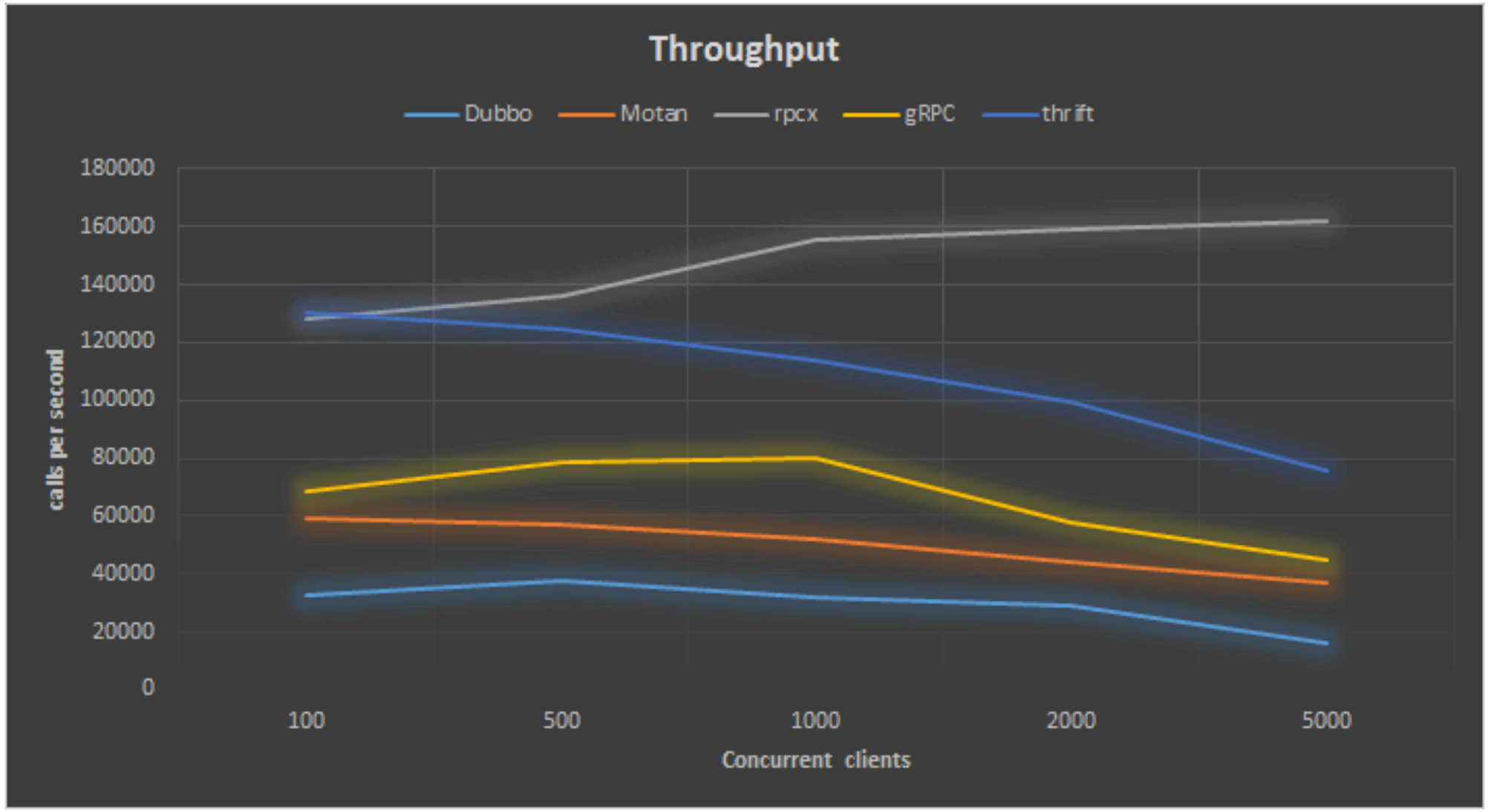
RPC选型

- 功能需求的满足度
- 多语言的支持
- 性能和稳定性
- 社区活跃度、成熟度

框架对比

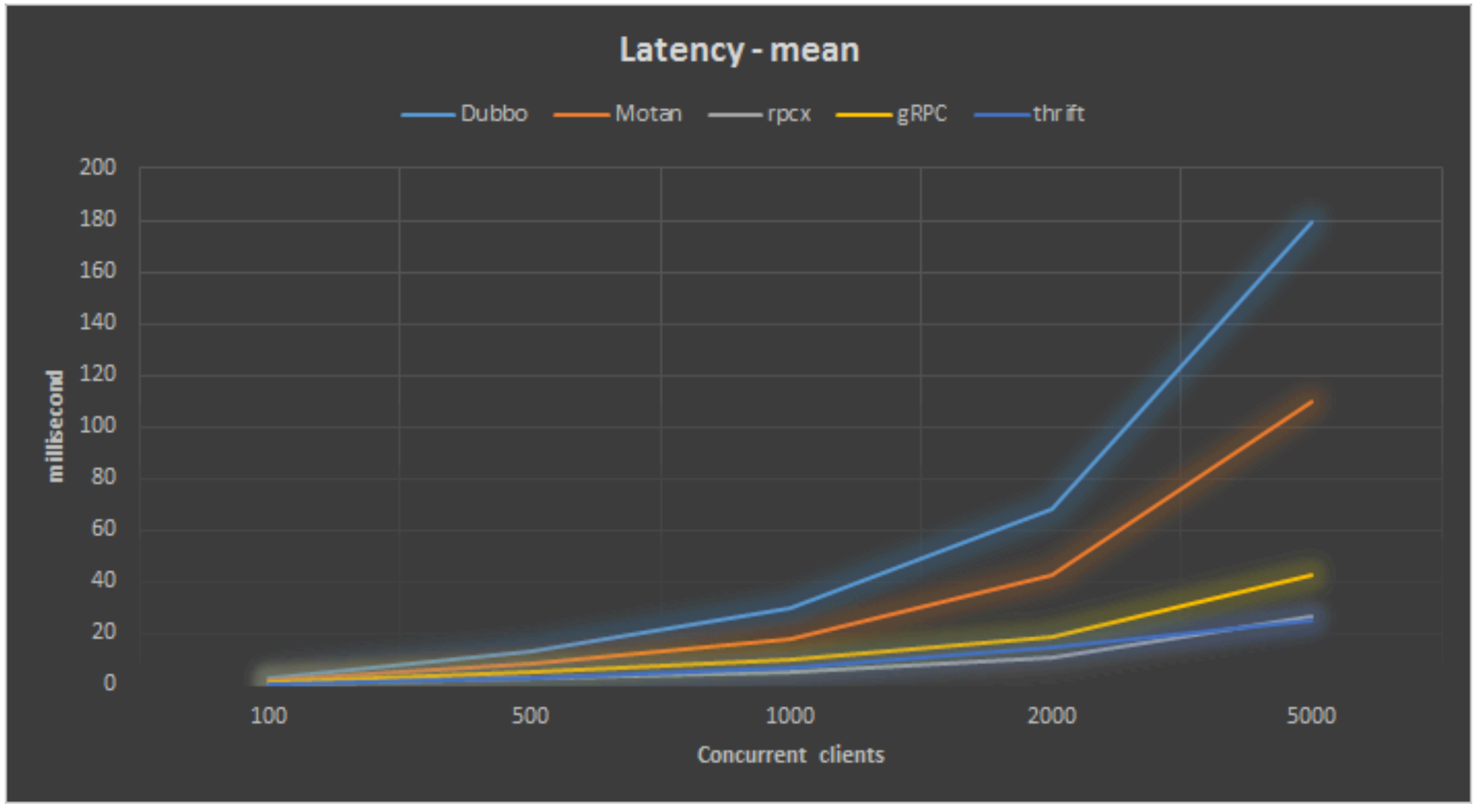
	Dubbo / Dubbox	Motan	gRPC	Thrift
开发语言	Java	Java	跨语言	跨语言
分布式(服务治理)	√	√	×	×
多序列化框架支持	√	√ (当前支持Hessian2、Json,可扩展)	√	√
多种注册中心	√	√	×	×
管理中心	√	√	×	×
跨编程语言	×	× (支持php client和C server)	√	√

性能对比



注：上图来源于网上

性能对比



注：上图来源于网上

why gRPC

- 跨语言
- 性能可以接受
- 协议简单，基于协议做扩展
- 使用简单、支持IDL的生成代码
- 社区活跃、生态完善

gRPC

What is gRPC?

gRPC stands for **g**RPC **R**emote **P**rocedure **C**alls.

A high performance, open source, general purpose standards-based, feature-rich RPC framework.

Open sourced version of Stubby RPC used in Google.

Actively developed and production-ready, current version is 1.0.1.

Google Cloud Platform

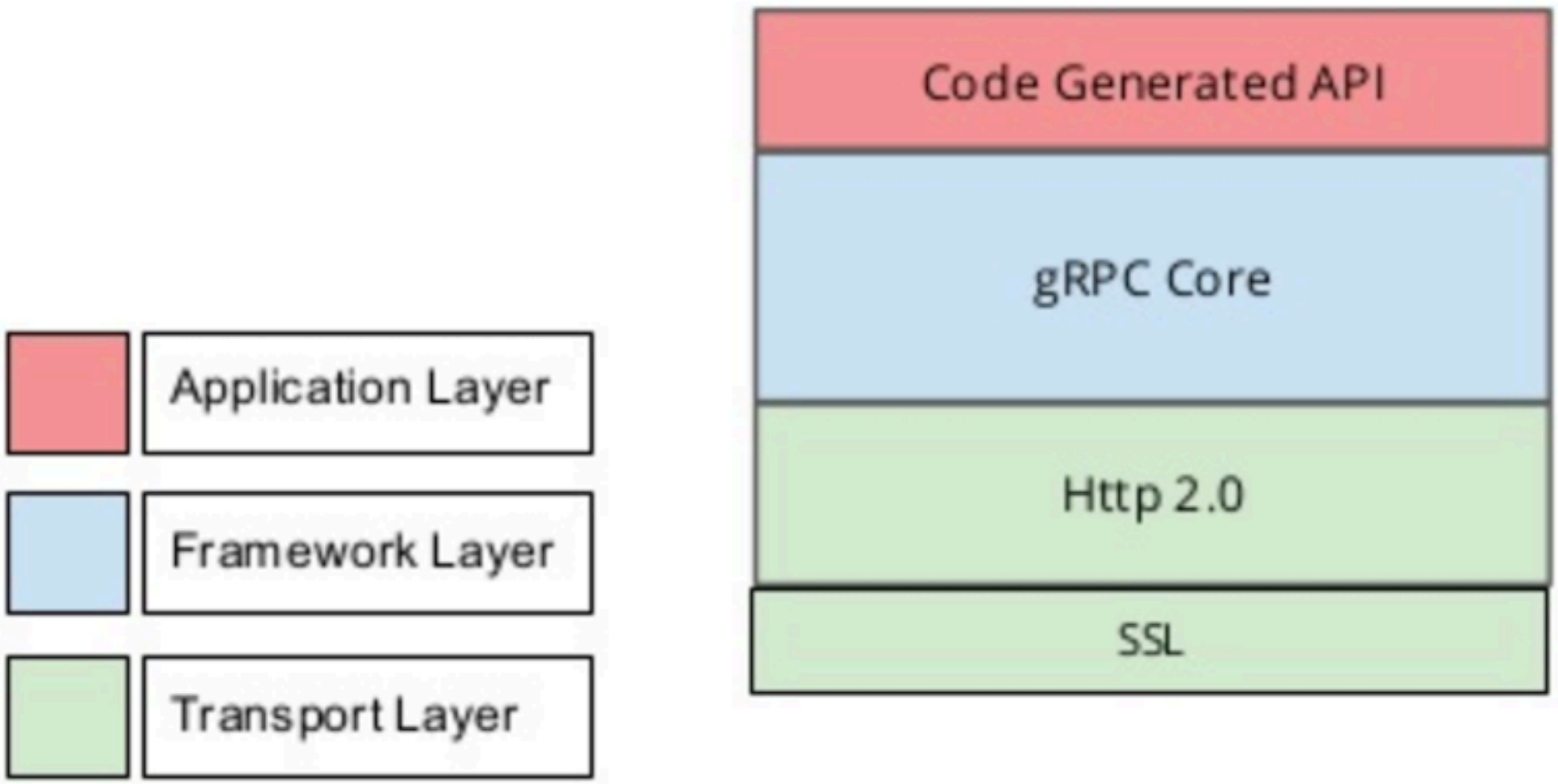
meitu

gRPC—特性

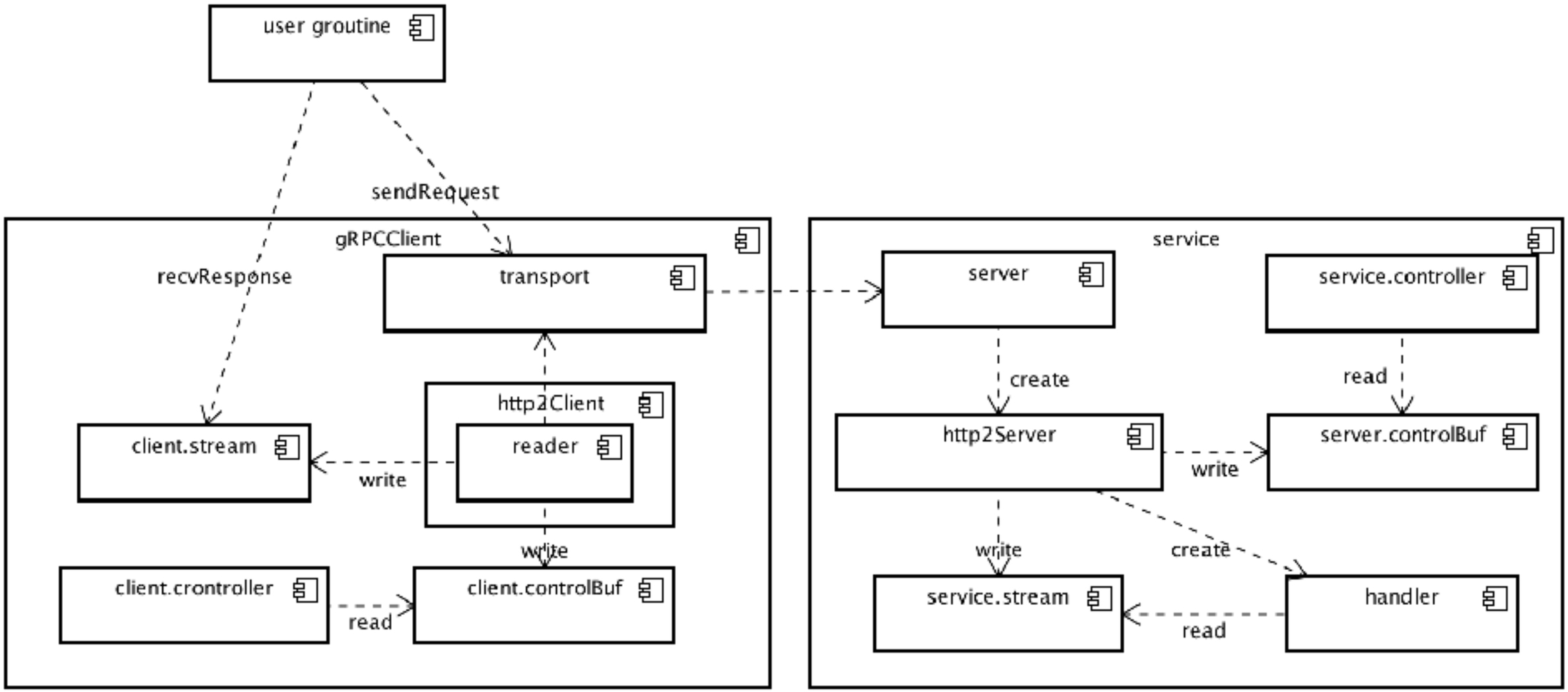
- 头部压缩
- 报文压缩
- Stream
- Protolbuf
- HTTP2
- TLS
- 响应码
- 流控
- 并发控制
- Middleware
- 取消超时
- Lameducking
- Metadata

meitu

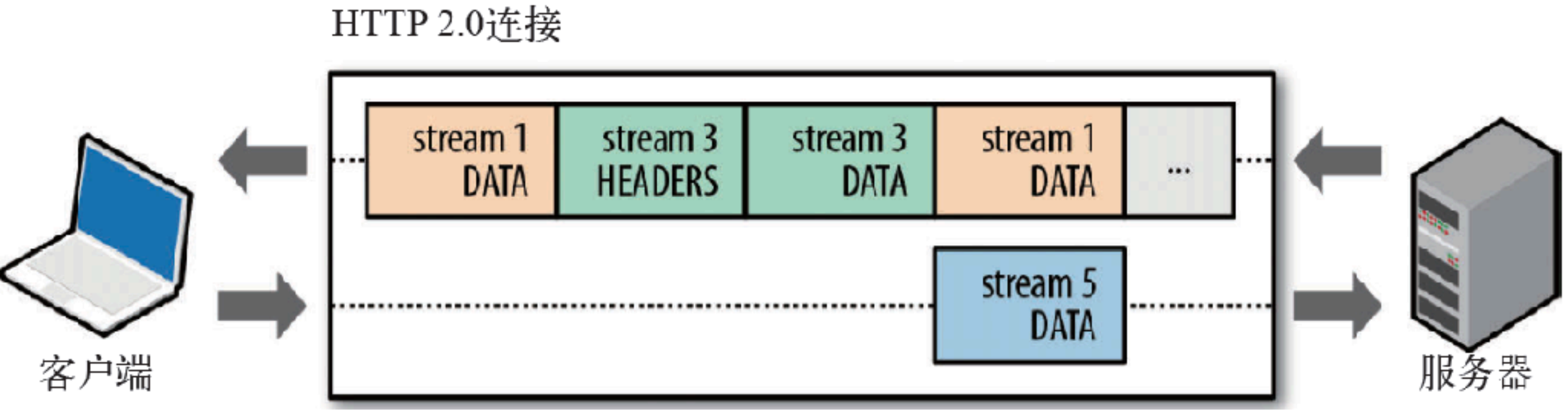
gRPC—架构


Application Layer
Framework Layer
Transport Layer
Code Generated API
gRPC Core
Http 2.0
SSL

gRPC—golang架构图



gRPC—数据传输



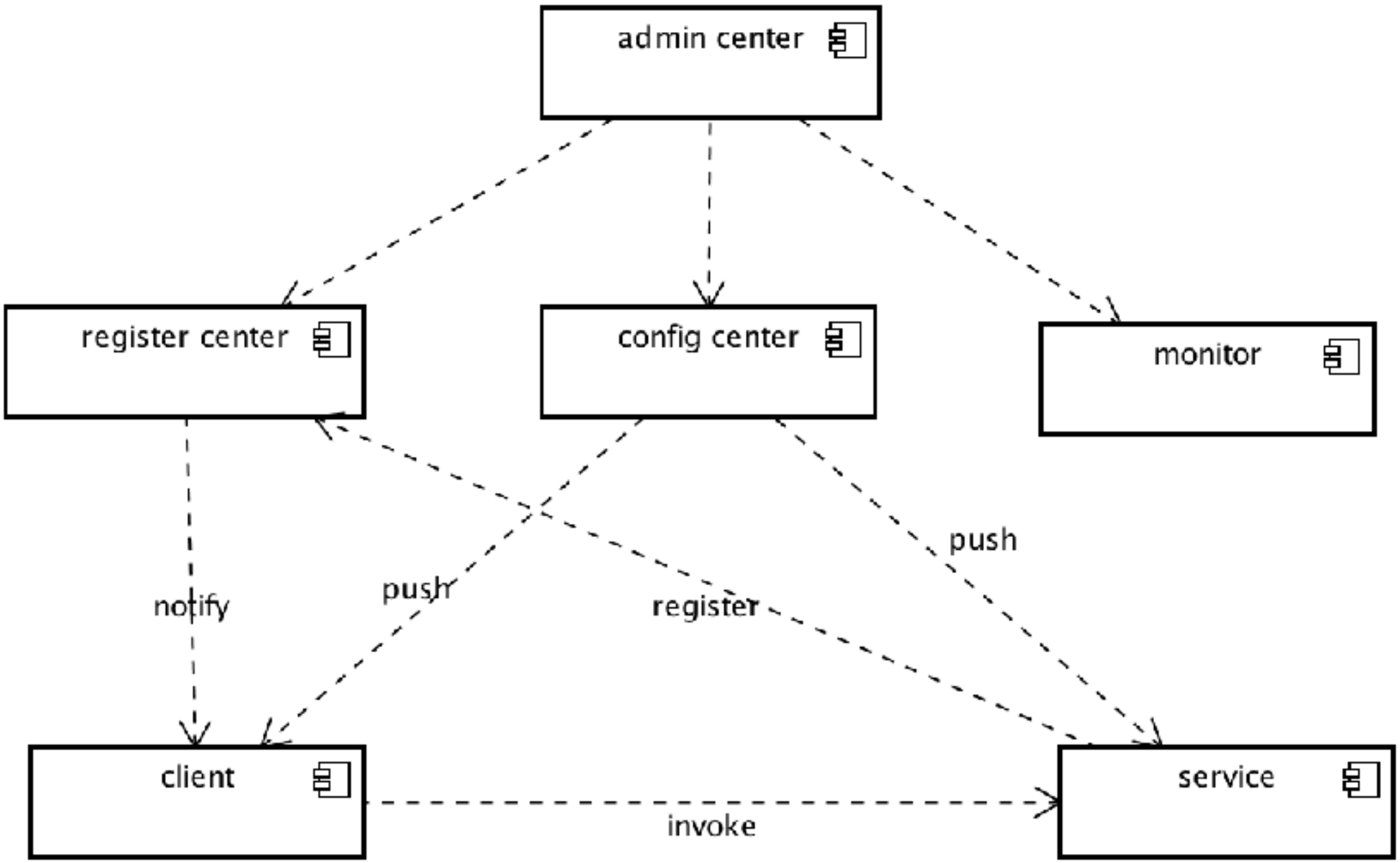
HTTP/2协议是一个二进制协议，二进制更易于frame(帧 数据包)的实现，HTTP/2有十个不同frame定义，其中两个最基础的对应于HTTP 1.1的是Data数据和HEADE头部，其后将描述。

frame是包含几个部分：类型Type, 长度Length, 标记Flags, 流标识Stream和frame payload有效载荷。

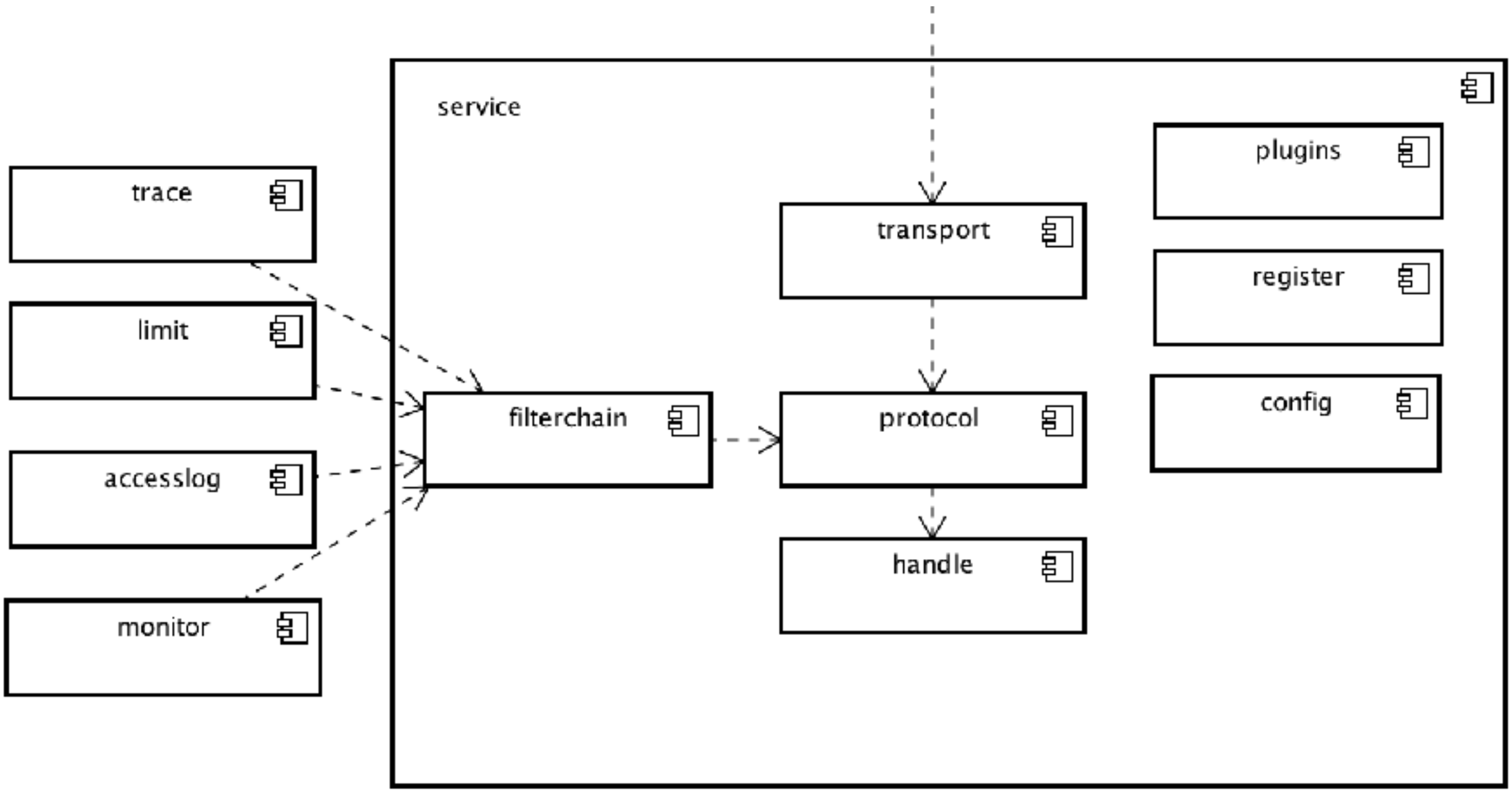
meitu服务化框架Tardis

- 整体设计
- API设计
- 服务发现
- 通讯
- 高可用
- 监控

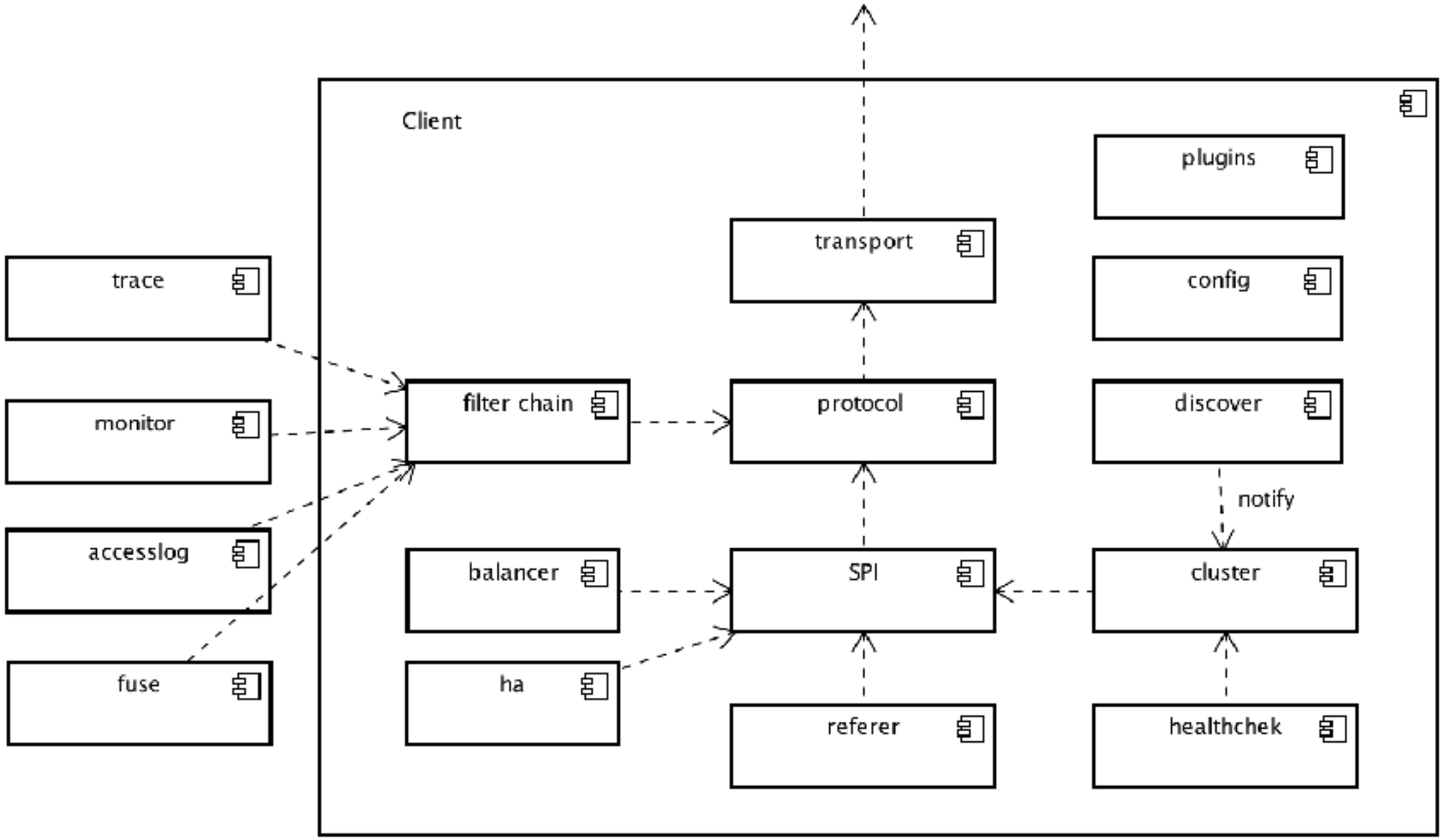
架构—整体设计


admin center
register center
config center
monitor
notify
push
register
push
client
service
invoke

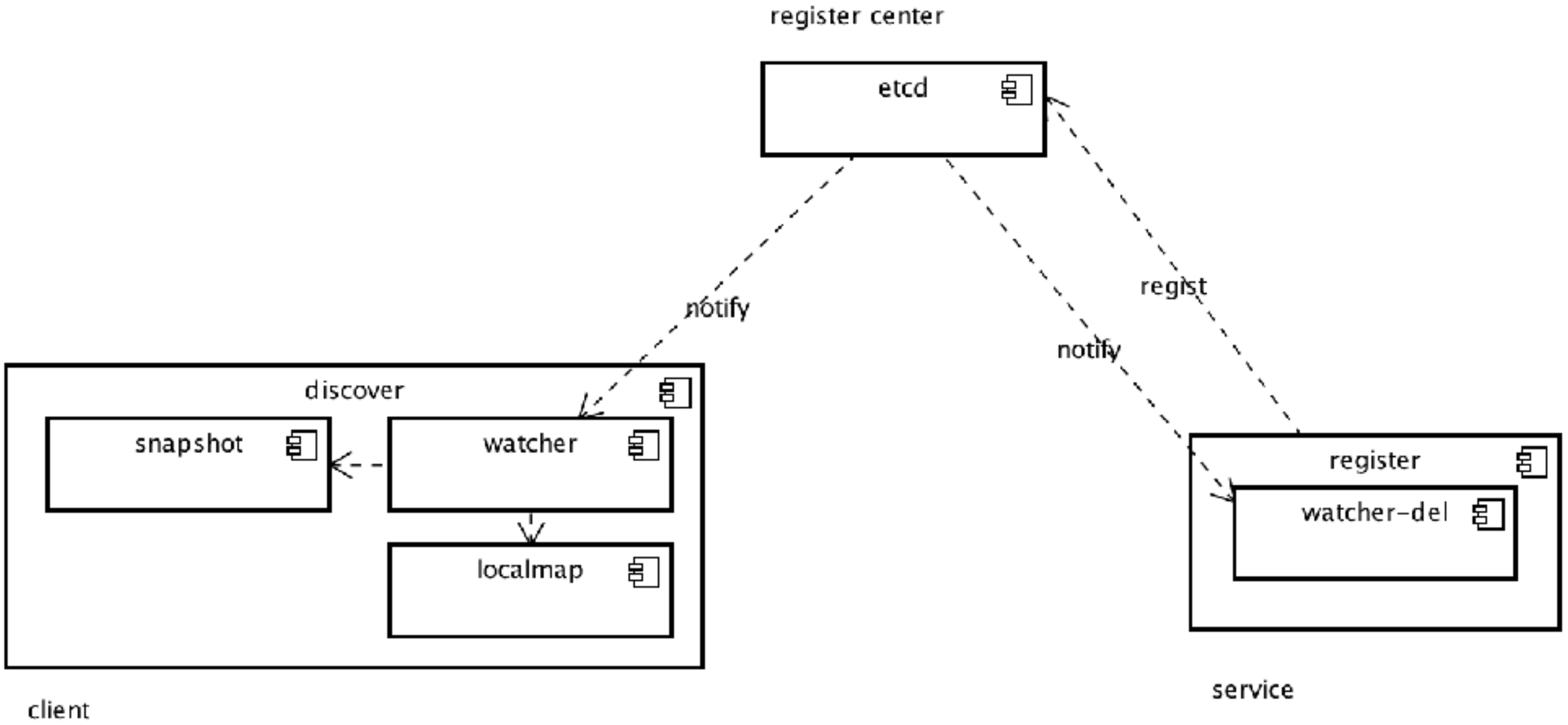
架构—Service


service
trace
limit
accesslog
monitor
filterchain
transport
protocol
handle
plugins
register
config

架构—Client


Client
plugins
transport
trace
config
filter chain
protocol
discover
monitor
notify
accesslog
balancer
SPI
cluster
fuse
ha
referer
healthchek

服务发现


register center
etcd
notify
notify
regist
discover
snapshot
watcher
localmap
register
watcher-del
client
service

服务发现—原理

- Service

  注册节点信息 status，report ，value

  监听对应路径的delete事件，防止误操作

  register / deregister 接口

- Client

  支持snashopt

  支持断线重新监听

  通过Revision 防止数据丢失

  增量和全量的接口

API的设计

- 非嵌入式

  减少对接层本

  减少学习层本

  保留gRPC 的特性

数据传输 —gRPC协议

Request

```
HEADERS (flags = END_HEADERS)
:method = POST
:scheme = http
:path = /google.pubsub.v2.PublisherService/CreateTopic
:authority = pubsub.googleapis.com
grpc-timeout = 1S
content-type = application/grpc+proto
grpc-encoding = gzip
authorization = Bearer y235.wef315yfh138vh31hv93hv8h3v

DATA (flags = END_STREAM)
<Delimited Message>
```

Response

```
HEADERS (flags = END_HEADERS)
:status = 200
grpc-encoding = gzip

DATA
<Delimited Message>

HEADERS (flags = END_STREAM, END_HEADERS)
grpc-status = 0 # OK
trace-proto-bin = jher831yy13JHy3hc
```

meitu

数据传输—gRPC

No.	Time	Source	Destination	Protocol	Length	Info
33	3.923512	127.0.0.1	127.0.0.1	TCP	68	59342 → 50051 [SYN] Seq=0…
34	3.923589	127.0.0.1	127.0.0.1	TCP	68	50051 → 59342 [SYN, ACK] …
35	3.923600	127.0.0.1	127.0.0.1	TCP	56	59342 → 50051 [ACK] Seq=1…
36	3.923608	127.0.0.1	127.0.0.1	TCP	56	[TCP Window Update] 50051…
37	3.923694	127.0.0.1	127.0.0.1	HTTP2	80	Magic
38	3.923713	127.0.0.1	127.0.0.1	TCP	56	50051 → 59342 [ACK] Seq=1…
39	3.923717	127.0.0.1	127.0.0.1	HTTP2	65	SETTINGS
40	3.923726	127.0.0.1	127.0.0.1	HTTP2	71	SETTINGS
41	3.923728	127.0.0.1	127.0.0.1	HTTP2	69	WINDOW_UPDATE
42	3.923729	127.0.0.1	127.0.0.1	TCP	56	50051 → 59342 [ACK] Seq=1…
43	3.923732	127.0.0.1	127.0.0.1	TCP	56	59342 → 50051 [ACK] Seq=4…
44	3.923736	127.0.0.1	127.0.0.1	TCP	56	50051 → 59342 [ACK] Seq=1…
45	3.923738	127.0.0.1	127.0.0.1	HTTP2	69	WINDOW_UPDATE
46	3.923751	127.0.0.1	127.0.0.1	TCP	56	59342 → 50051 [ACK] Seq=4…
47	3.923839	127.0.0.1	127.0.0.1	HTTP2	65	SETTINGS
48	3.923848	127.0.0.1	127.0.0.1	HTTP2	237	HEADERS
49	3.923860	127.0.0.1	127.0.0.1	TCP	56	59342 → 50051 [ACK] Seq=2…
50	3.923869	127.0.0.1	127.0.0.1	TCP	56	50051 → 59342 [ACK] Seq=3…
51	3.923890	127.0.0.1	127.0.0.1	HTTP2	65	SETTINGS
52	3.923903	127.0.0.1	127.0.0.1	TCP	56	50051 → 59342 [ACK] Seq=3…
53	3.923920	127.0.0.1	127.0.0.1	HTTP2	77	DATA
54	3.923931	127.0.0.1	127.0.0.1	TCP	56	50051 → 59342 [ACK] Seq=3…
55	3.924197	127.0.0.1	127.0.0.1	HTTP2	79	HEADERS

meitu

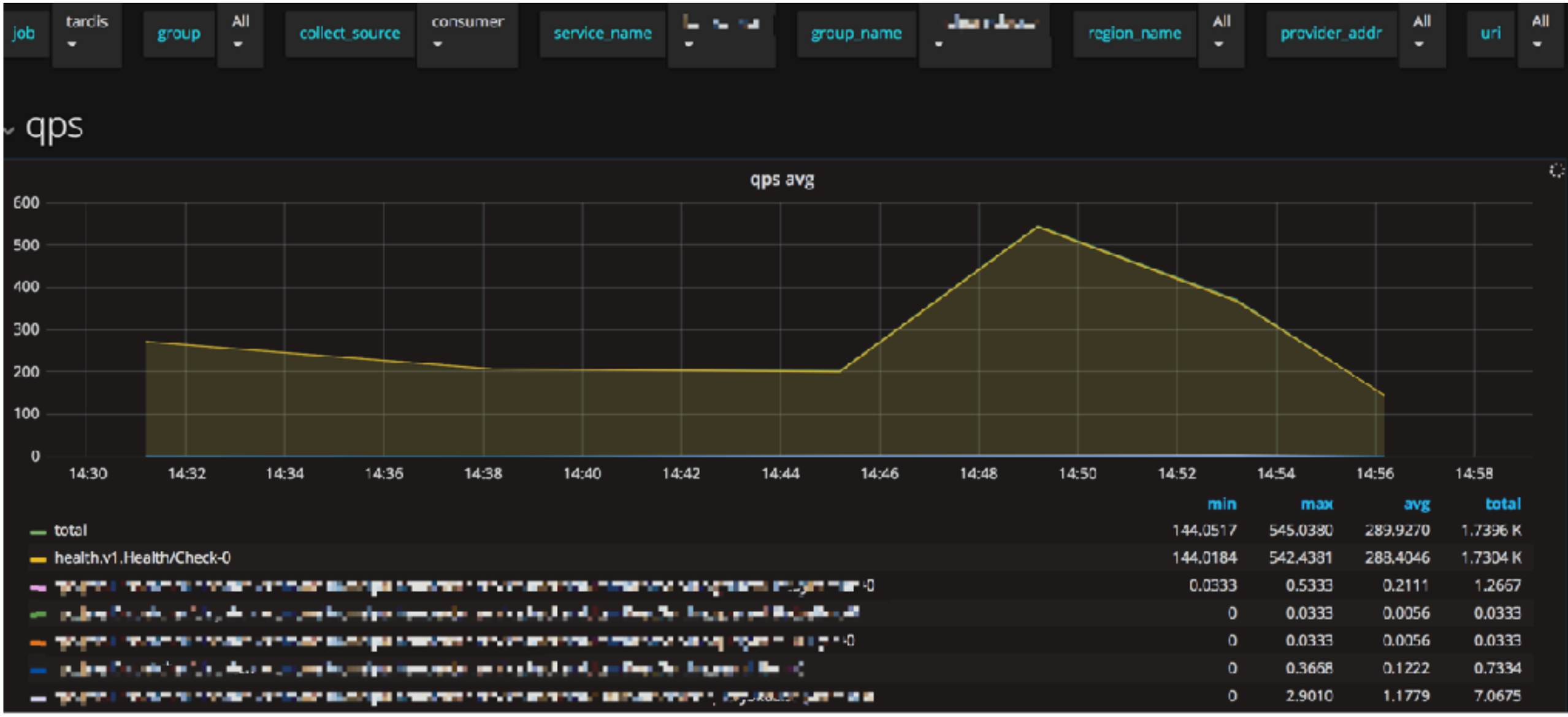
可用性—策略

- 流控、熔断、过载保护
- 局部容错、跨集群的容错
- 柔性降级
- 可扩展的自定义策略

服务管控平台

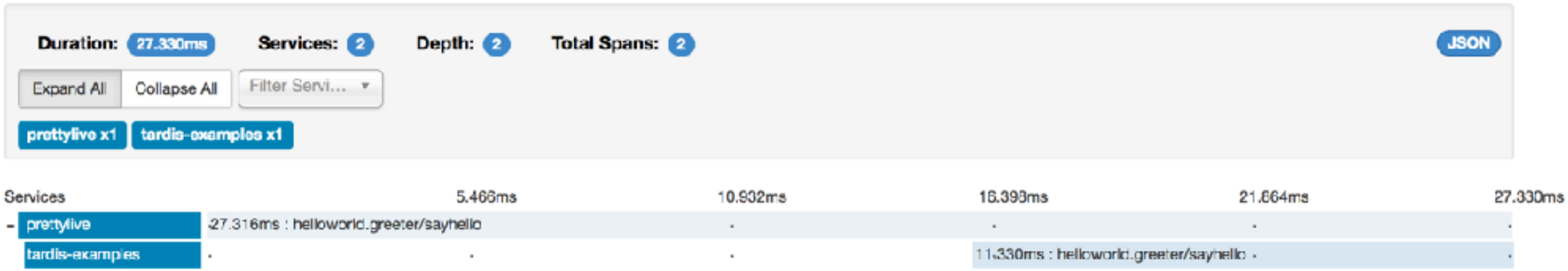
- 分布式trace
- 服务依赖关系
- 监控、 metric、 报警
- 动态配置
- 实时流量拓扑
- SLA

可用性—监控


job
tarcis
group
All
collect_source
consumer
service_name
group_name
region_name
All
provider_addr
All
uri
All
qps
qps avg
600
500
400
300
200
100
0
14:30
14:32
14:34
14:36
14:38
14:40
14:42
14:44
14:46
14:48
14:50
14:52
14:54
14:56
14:58
min
max
avg
total
total
144.0517
545.0380
289.9270
1.7396 K
health.v1.Health/Check-0
144.0184
542.4381
288.4046
1.7304 K
0.0333
0.5333
0.2111
1.2667
0
0.0333
0.0056
0.0333
0
0.0333
0.0056
0.0333
0
0.3668
0.1222
0.7334
0
2.9010
1.1779
7.0675

meitu

可用性—Trace


Duration: 27.330ms
Services: 2
Depth: 2
Total Spans: 2
JSON
Expand All
Collapse All
Filter Servi...
prettylive x1
tardis-examples x1
Services
5.466ms
10.932ms
16.398ms
21.864ms
27.330ms
prettylive
27.316ms : helloworld.greeter/sayhello
tardis-examples
11.330ms : helloworld.greeter/sayhello

关于gRPC的一些设计

- 采用HTTP2，好处在于,因为添加了头信息,可以方便在**框架层面**对调用做拦截和控制(比如说限流,调用链分析,安全认证等)，而且协议相对简单

- 通过IDL生成各种语言SDK

- 头部压缩

- stream的方式减少报文

- 通过setting fream 交换两端信息

meitu

gRPC支持通讯模式

	Stream	Unary
优点	1. 共享Stream，减少报文	1.失败重试，不用重传 2.符合编程习惯
缺点	1.head of line block 2.异常处理，重传机制	1.幂等 2.报文比较多

gRPC遇到的问题

- gRPC兼容问题：1.2少量api不兼容1.0
- java 1.0版本：基于netty4.1.3 有内存泄漏问题，netty 提高到4.1.6 fix了
- 单连接的限制
- 在一些语言层面支持不完善
- HTTP2握手失败后迅速断开重连，导致CPU异常

meitu

gRPC的实践经验

- 通过context 做策略的扩展
- 使用非block的方式，启动断线重连
- 使用go-grpc-middleware 实现拦截链
- metadata使用时要注意 Join，income，outcome
- 基于标准错误码的监控

meitu

gRPC的实践经验

无缝升级

- 通过额外的库facebookgo/grace
- 结合配置中心和优雅停止

http协议兼容

- https://github.com/grpc-ecosystem/grpc-gateway

meitu

gRPC的实践经验

- grpc.UnknownServiceHandle

  实现异常处理

  实现gRPC网关

- IDL扩展

- Metadata

  实现鉴权

  信息传递（trace，统计）

  减少报文传输

meitu

扩展POJO

- 方案1 底层转化为gRPC对象

  生成gRPC对象难度复杂，主要是范型

- 方案2 基于gRPC 的协议上走java 的序列化

  比较简单，但是只能在java间使用

dispatch 的定义

```
 syntax = "proto3";

  package rpc.java;

  option java_multiple_files = true;

  option java_package = "com.meitu.tardis.grpc.proto.rpc.java";

  option java_outer_classname = "DispatcherProto";
service Dispatcher {
    rpc dispatch (Request) returns (Response);
  }

//因为请求入口统一处理的一些需求,interfaceName、methodName放到了meta信息里而没放在request body里
//paramTypes,serialize也放到header里
//放header的一个好处是,grpc基于http2,对header有做优化节省传输数据体积
message Request {
    //参数值
    repeated bytes paramValues=1;
}

message Response {
    //返回值
    bytes value=1;
}
```

meitu

Q&A