Go工程可观测性实践

周曙光

得物
Go开发

目 录

第一部分

可观测性概述

什么是可观测性？

广义的可观测性：可以根据系统的外部输出信息推断出系统内部状态的好坏。

软件系统的可观测性：一种度量能力，能帮你更好的理解系统当前所处的任何状态。如果无需发布新代码就可以理解任何新的或怪异的状态，那么系统就具备可观测性。

可观测性开源产品

Prometheus

Pinpoint

Zipkin

Jaeger

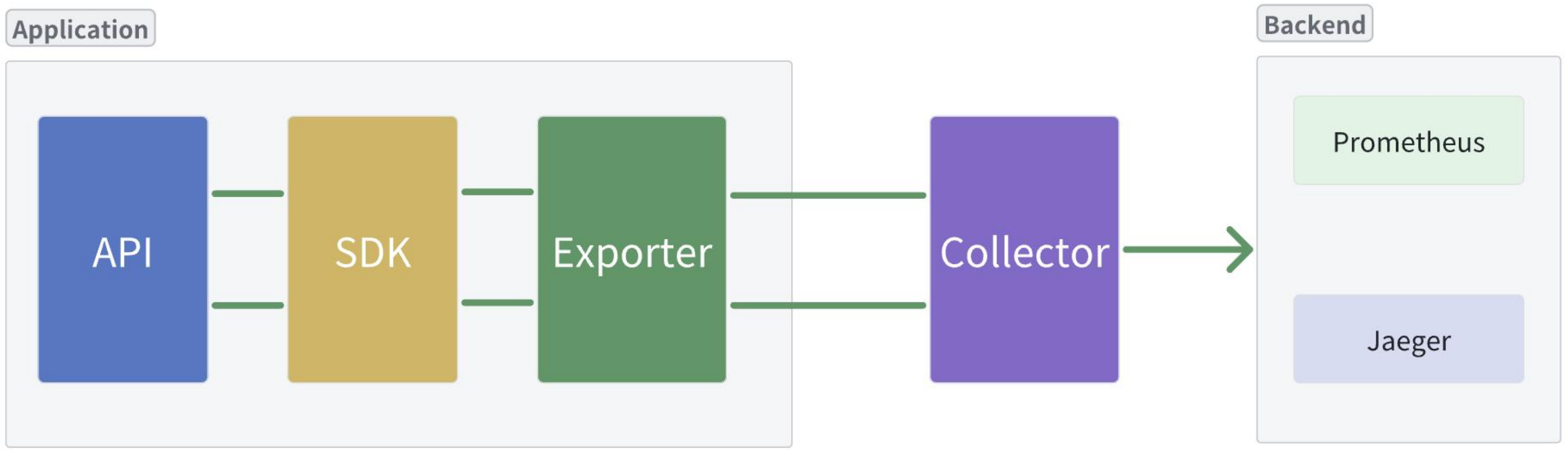
每种方案都有特定的、自定义的步骤来生产和转移符合后端存储的遥测(Telemetry)数据，这就带来了工具或者厂商的绑定性。

OpenTelemetry

为了解决“厂商锁定问题”，监控和可观测性社区过去创建了很多开源项目，比如OpenTracing和OpenCensus，这些标准允许用户实时收集遥测数据并传输到所选择的后端，最终在2019年，两个组织共同组建OTel项目，并由CNCF负责。

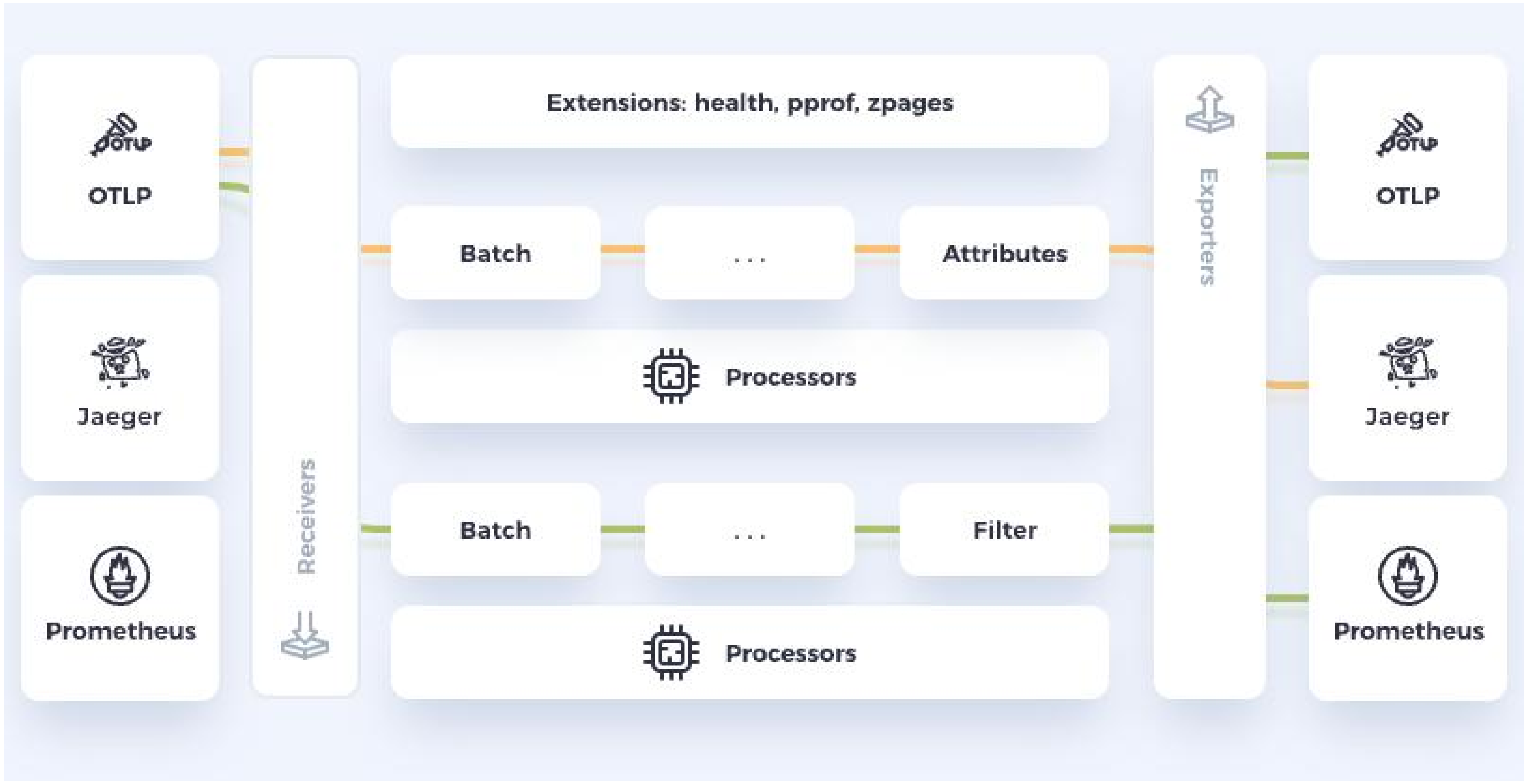
OTel目前已经成为可观测性方案开源标准，标准的好处就是有了很多选择。

OTel 组件



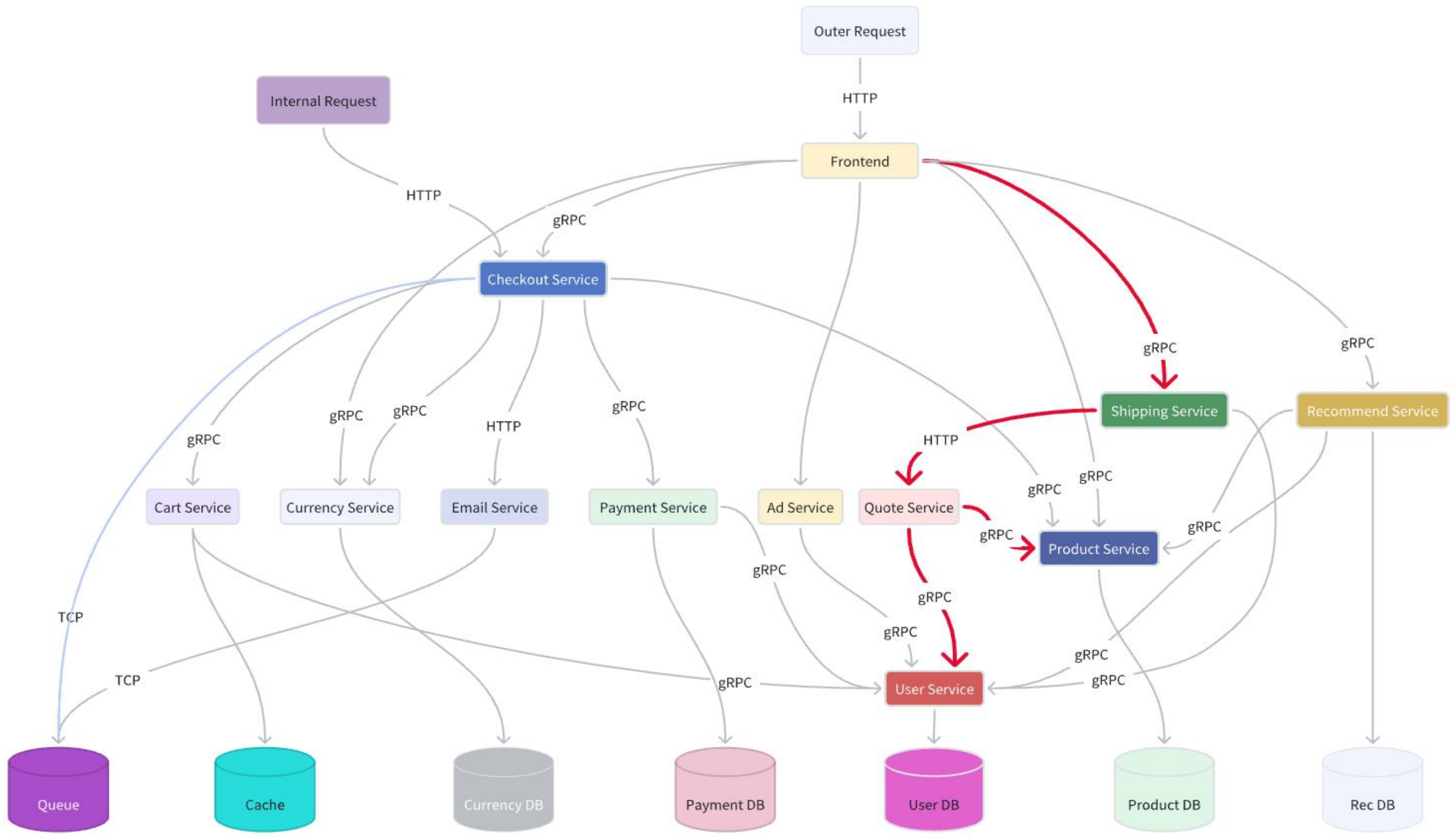
- API
- SDK
- Exporter
- Collector

OTel Collector



- Receiver
- Processor
- Exporter

微服务业务架构图

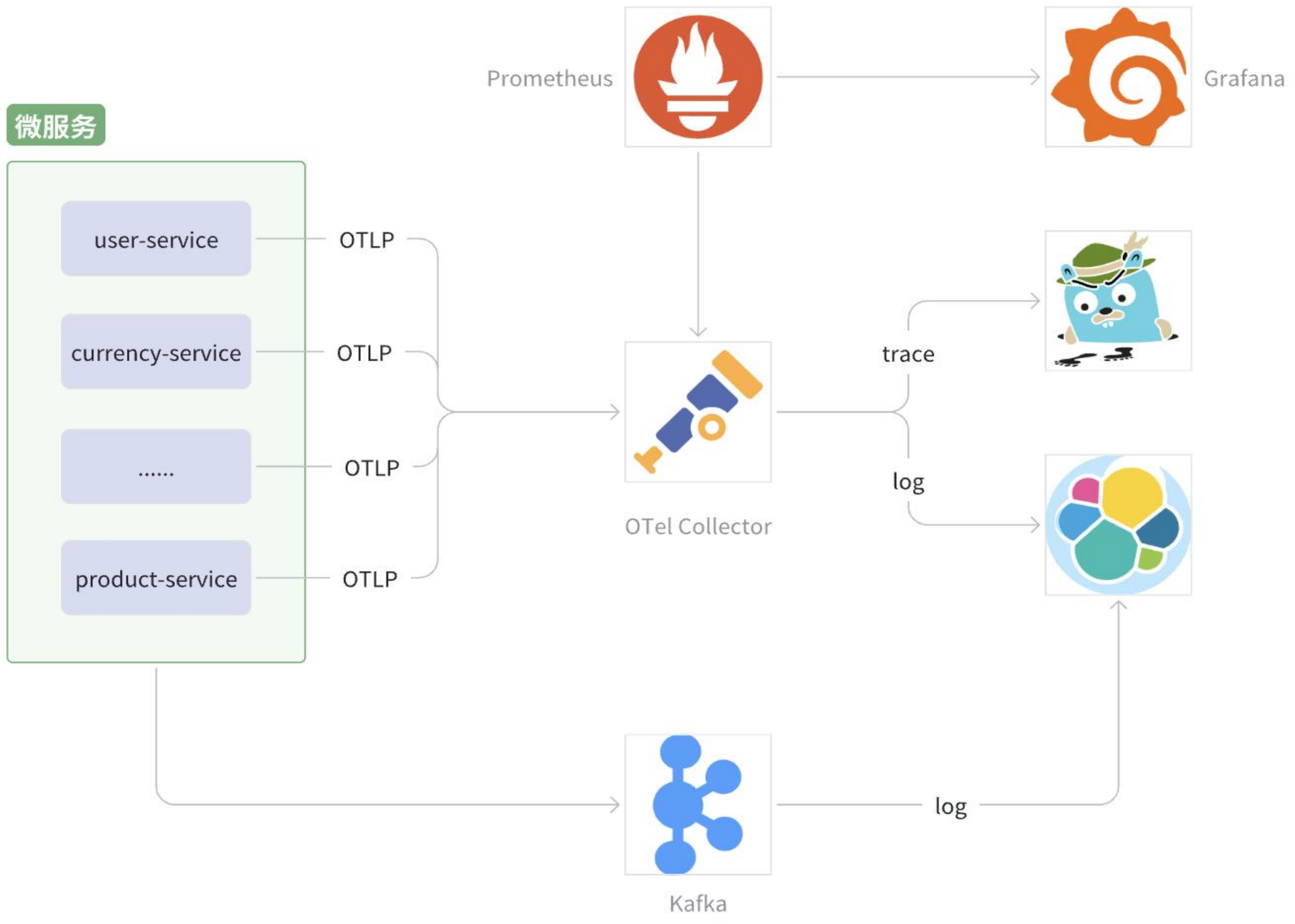


Outer Request
HTTP
Internal Request
Frontend
HTTP
gRPC
Checkout Service
gRPC
gRPC
Shipping Service
Recommend Service
gRPC
gRPC
gRPC
HTTP
gRPC
HTTP
Cart Service
Currency Service
Email Service
Payment Service
Ad Service
Quote Service
gRPC
gRPC
gRPC
Product Service
gRPC
gRPC
gRPC
gRPC
TCP
TCP
gRPC
gRPC
gRPC
User Service
Queue
Cache
Currency DB
Payment DB
User DB
Product DB
Rec DB

项目工程layout

```
./
├── Makefile
├── README.md
├── app
│   ├── ad
│   ├── backend
│   ├── cart
│   ├── checkout
│   ├── currency
│   ├── email
│   ├── frontend
│   ├── payment
│   ├── product
│   ├── quote
│   ├── recommend
│   ├── shipping
│   └── user
├── common
│   ├── metric
│   ├── middleware
│   └── trace
├── go.mod
└── go.sum
```

遥测数据处理架构


Prometheus
Grafana
微服务
user-service
currency-service
......
product-service
OTLP
OTLP
OTLP
OTLP
OTel Collector
trace
log
log
Kafka

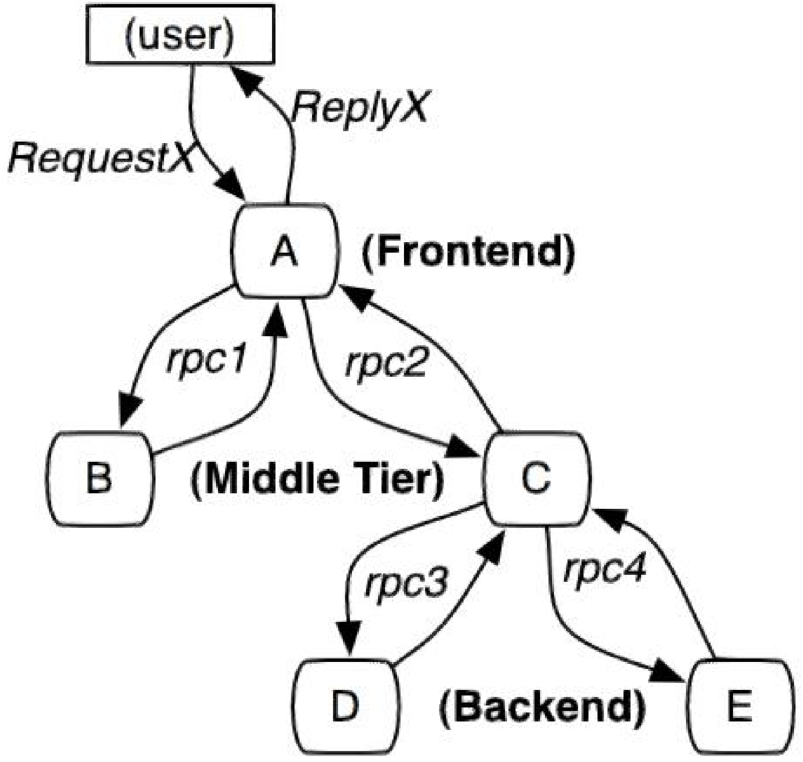
第二部分

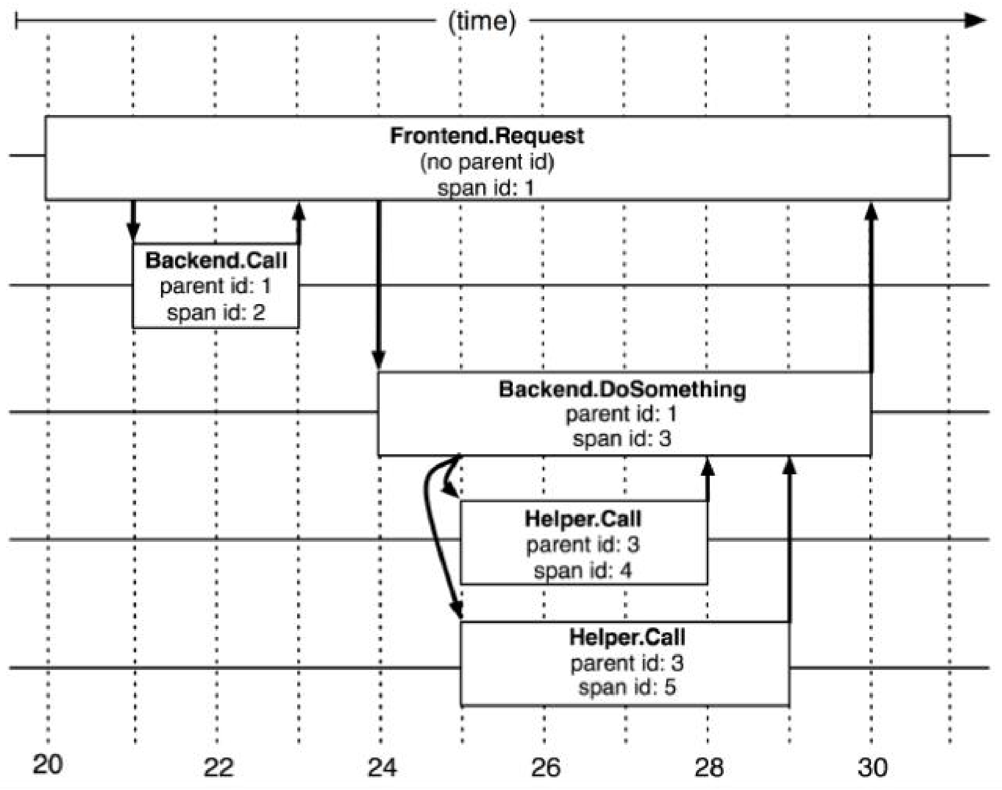
链路追踪

链路追踪设计目标

- 无所不在的部署
- 持续监控
- 低消耗
- 应用级透明
- 延展性

链路追踪 Dapper





每个请求都生成一个全局唯一的 traceid，端到端透传到上下游所有节点，每一层生成一个 spanid，通过traceid 将不同系统孤立的调用日志和异常信息串联一起，通过 spanid 和 parentid 表达节点的父子关系

链路追踪

在分布式系统中请求的路径经常很凌乱且无法预测，为了构建我们想要的任何路径的视图，无论多么复杂，每个组件都需要五段数据：

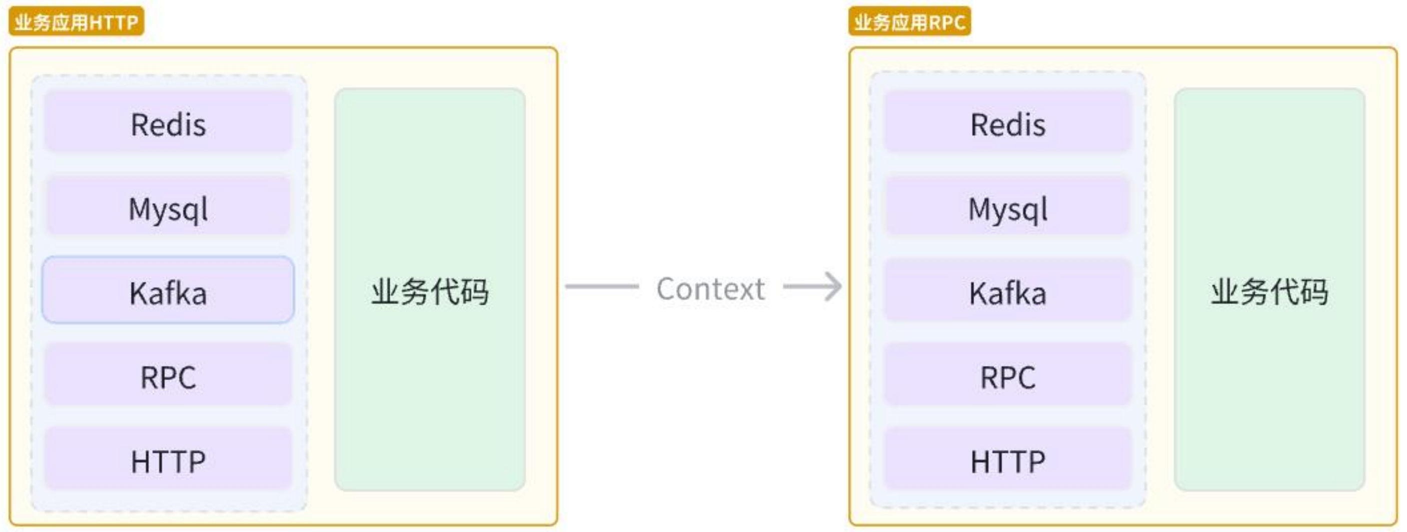
- TraceID：请求唯一标识符，由根span产生，贯穿请求的各个阶段。
- SpanID：span包含单一链路中一个工作单元收到的信息。
- ParentID：区别请求链路中的嵌套包含关系，根Span没有ParentID。
- 时间戳：每个Span必须展示开始时间。
- 执行时长：每个Span都必须记录工作开始到结束时花费的时长。

Go工程插桩(Instrument)

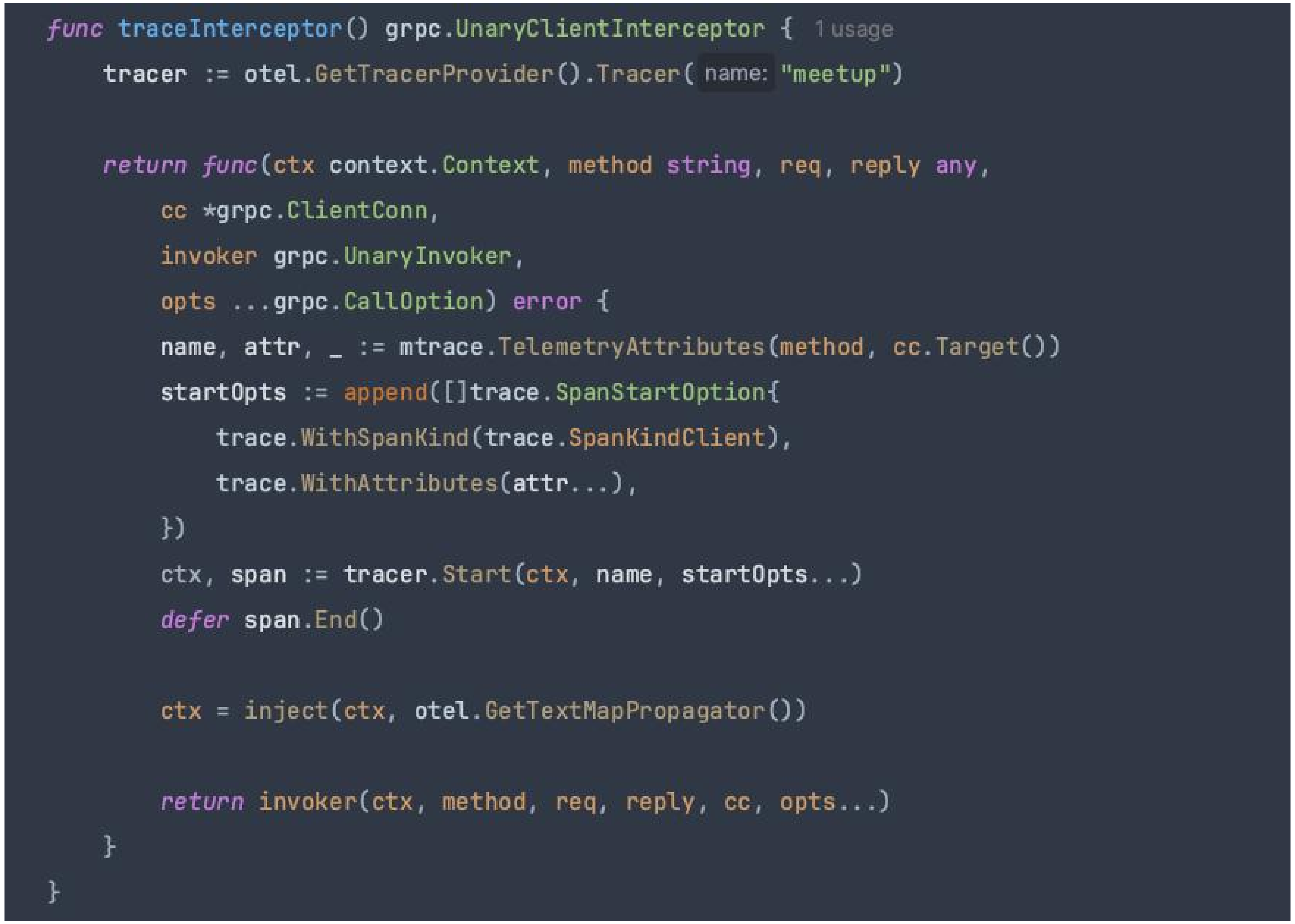
需要对业务开发者几乎零成本的接入链路追踪，几乎完全依赖于少量通用组件库的改造。

当一个请求在处理跟踪控制路径的过程中，需要把跟踪的上下文存储在ThreadLocal中，在Go中就是存储在Context中，一般约定每个方法第一个参数为Context（上下文）。

覆盖组件不限于：数据库、缓存、消息队列、RPC、HTTP等。



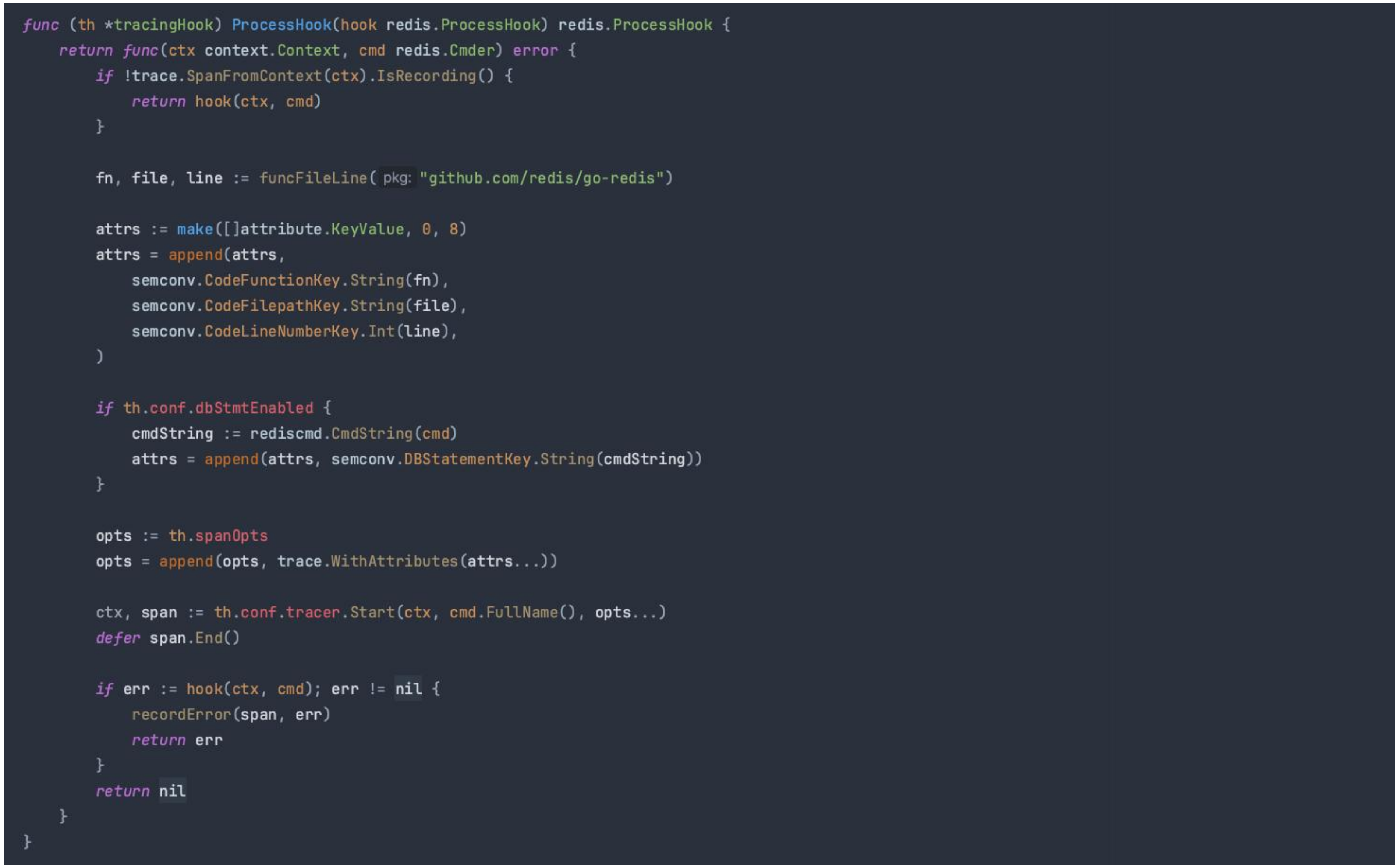
插桩(拦截器)



```
func traceInterceptor() grpc.UnaryClientInterceptor {  1 usage
    tracer := otel.GetTracerProvider().Tracer( name: "meetup")

    return func(ctx context.Context, method string, req, reply any,
        cc *grpc.ClientConn,
        invoker grpc.UnaryInvoker,
        opts ...grpc.CallOption) error {
        name, attr, _ := mtrace.TelemetryAttributes(method, cc.Target())
        startOpts := append([]trace.SpanStartOption{
            trace.WithSpanKind(trace.SpanKindClient),
            trace.WithAttributes(attr...),
        })
        ctx, span := tracer.Start(ctx, name, startOpts...)
        defer span.End()

        ctx = inject(ctx, otel.GetTextMapPropagator())

        return invoker(ctx, method, req, reply, cc, opts...)
    }
}
```

创建并命名上下文(HTTP请求或gRPC)中已存在的任意Span的子Span，开始计时器计时，计算Span的时长，在函数调用结束(defer)中完成用于传输给后端的Span。

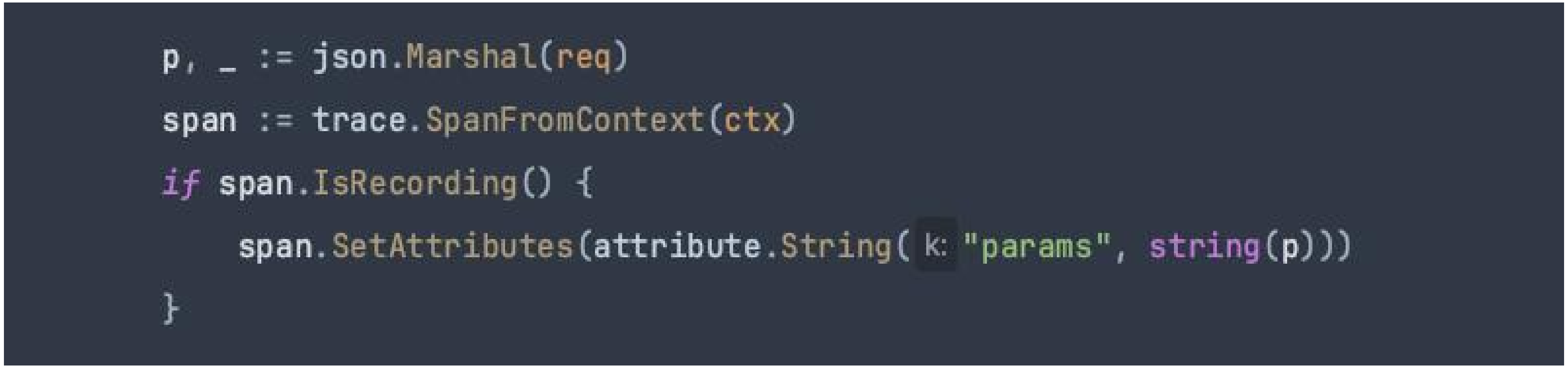
插桩(Hook)



```
func (th *tracingHook) ProcessHook(hook redis.ProcessHook) redis.ProcessHook {
    return func(ctx context.Context, cmd redis.Cmder) error {
        if !trace.SpanFromContext(ctx).IsRecording() {
            return hook(ctx, cmd)
        }

        fn, file, line := funcFileLine( pkg: "github.com/redis/go-redis")

        attrs := make([]attribute.KeyValue, 0, 8)
        attrs = append(attrs,
            semconv.CodeFunctionKey.String(fn),
            semconv.CodeFilepathKey.String(file),
            semconv.CodeLineNumberKey.Int(line),
        )

        if th.conf.dbStmtEnabled {
            cmdString := rediscmd.CmdString(cmd)
            attrs = append(attrs, semconv.DBStatementKey.String(cmdString))
        }

        opts := th.spanOpts
        opts = append(opts, trace.WithAttributes(attrs...))

        ctx, span := th.conf.tracer.Start(ctx, cmd.FullName(), opts...)
        defer span.End()

        if err := hook(ctx, cmd); err != nil {
            recordError(span, err)
            return err
        }
        return nil
    }
}
```

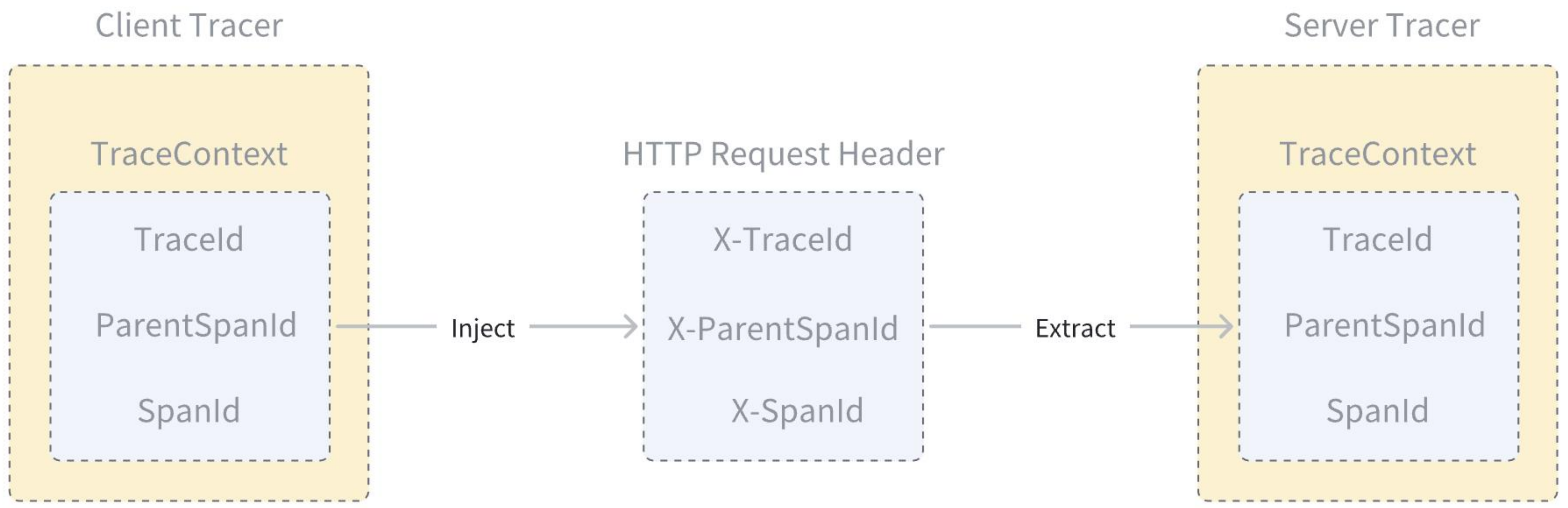
Span的开始和结束

对于Go来说，在进程或者goroutine上下文中出现活跃的Span的任何地方，都可以调用 sp := trace.SpanFromContext(ctx) 从活跃的上下文对象中获取当前活跃的span。获取后就可以给其添加属性。



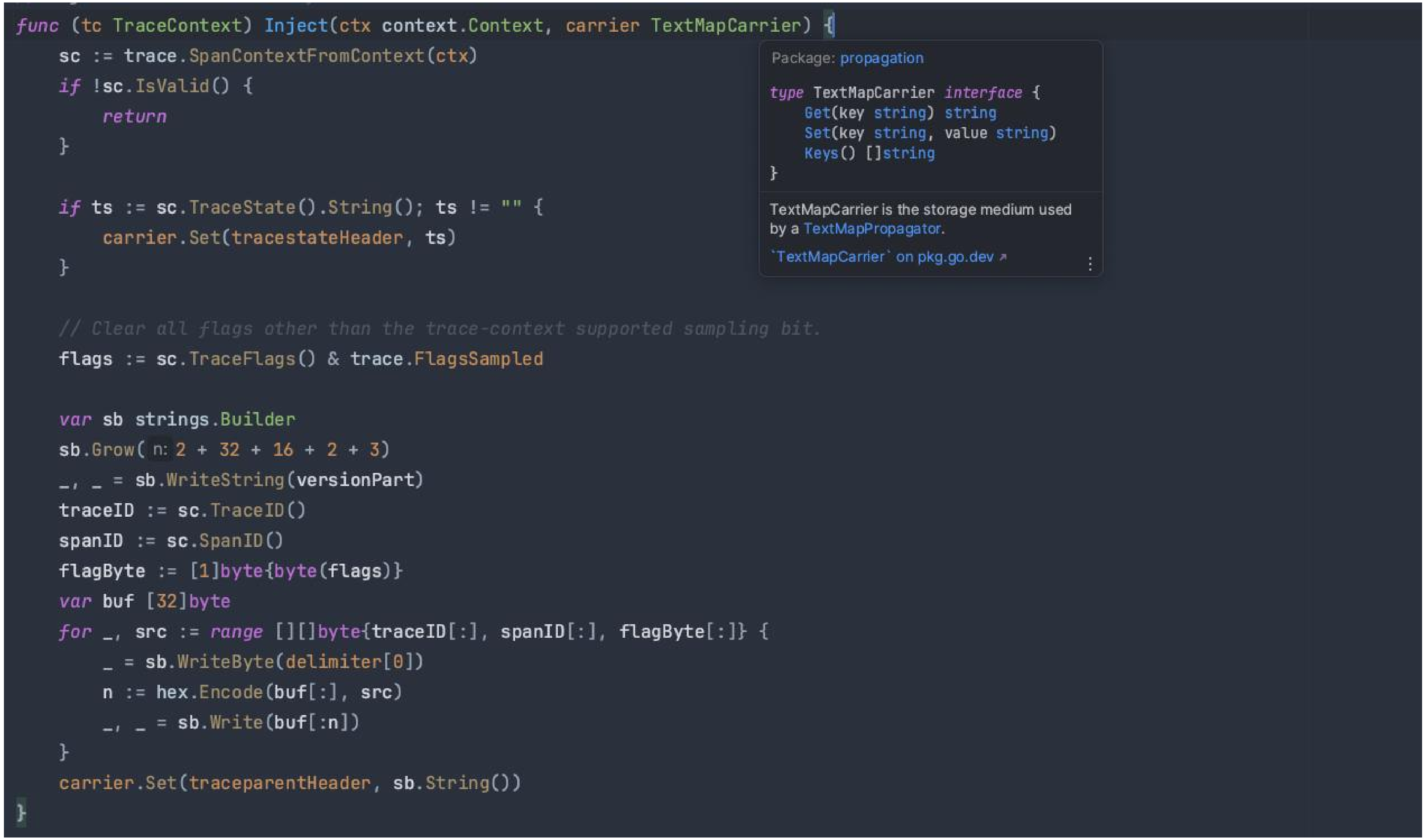
```
p, _ := json.Marshal(req)
span := trace.SpanFromContext(ctx)
if span.IsRecording() {
    span.SetAttributes(attribute.String( k: "params", string(p)))
}
```

链路传递



Client请求Server通信，同时Client把自己的链路信息传递给Server。链路信息通过一个叫TraceContext的对象封装起来，通过Http Header来存取这个对象，最后达到传播的效果，TraceContext就是一个Context上下文对象。

Inject



```go
func (tc TraceContext) Inject(ctx context.Context, carrier TextMapCarrier) {
	sc := trace.SpanContextFromContext(ctx)
	if !sc.IsValid() {
		return
	}

	if ts := sc.TraceState().String(); ts != "" {
		carrier.Set(tracestateHeader, ts)
	}

	// Clear all flags other than the trace-context supported sampling bit.
	flags := sc.TraceFlags() & trace.FlagsSampled

	var sb strings.Builder
	sb.Grow( n: 2 + 32 + 16 + 2 + 3)
	_, _ = sb.WriteString(versionPart)
	traceID := sc.TraceID()
	spanID := sc.SpanID()
	flagByte := [1]byte{byte(flags)}
	var buf [32]byte
	for _, src := range [][]byte{traceID[:], spanID[:], flagByte[:]} {
		_ = sb.WriteByte(delimiter[0])
		n := hex.Encode(buf[:], src)
		_, _ = sb.Write(buf[:n])
	}
	carrier.Set(traceparentHeader, sb.String())
}
```

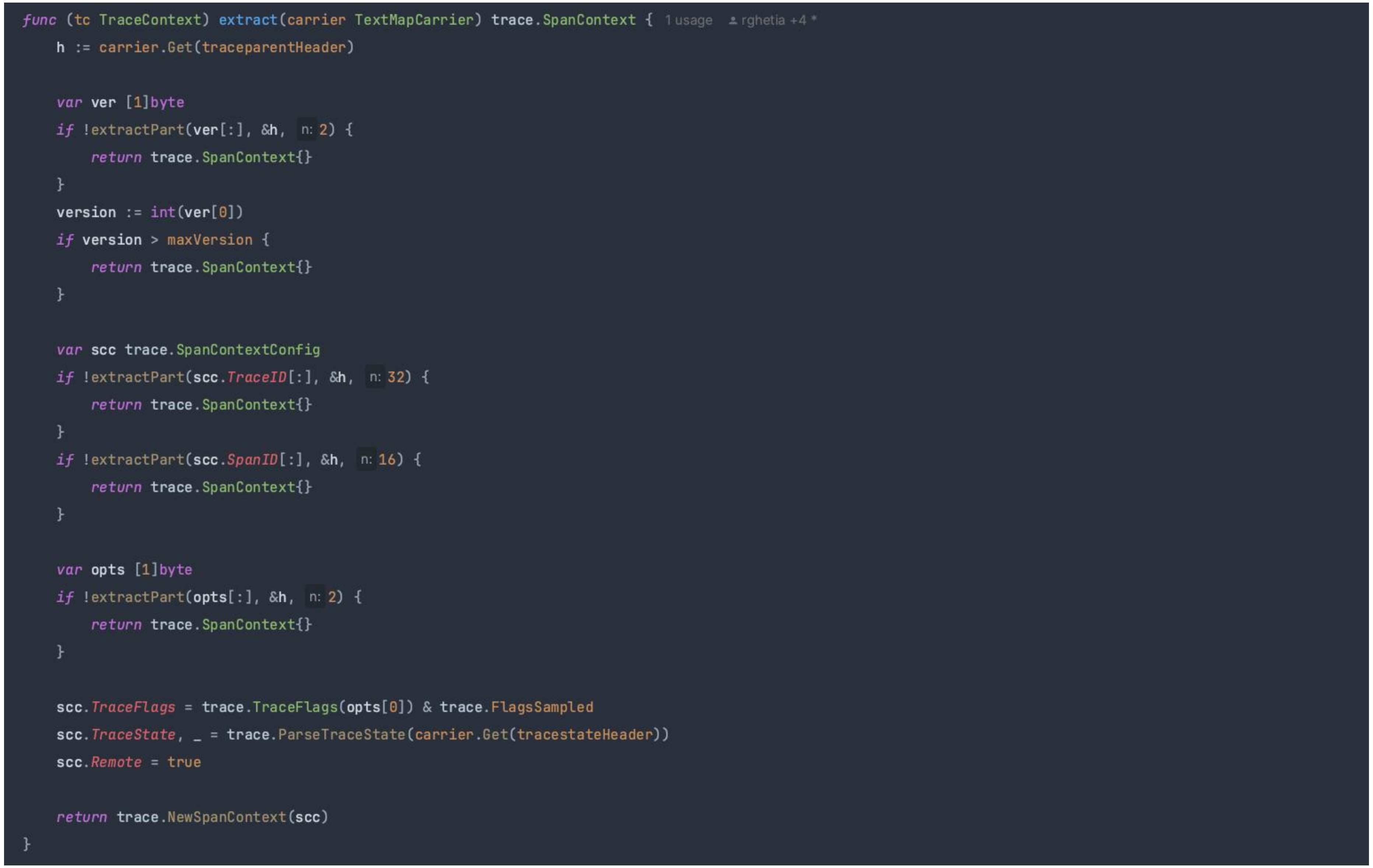
Package: propagation

```go
type TextMapCarrier interface {
	Get(key string) string
	Set(key string, value string)
	Keys() []string
}
```

TextMapCarrier is the storage medium used by a TextMapPropagator.

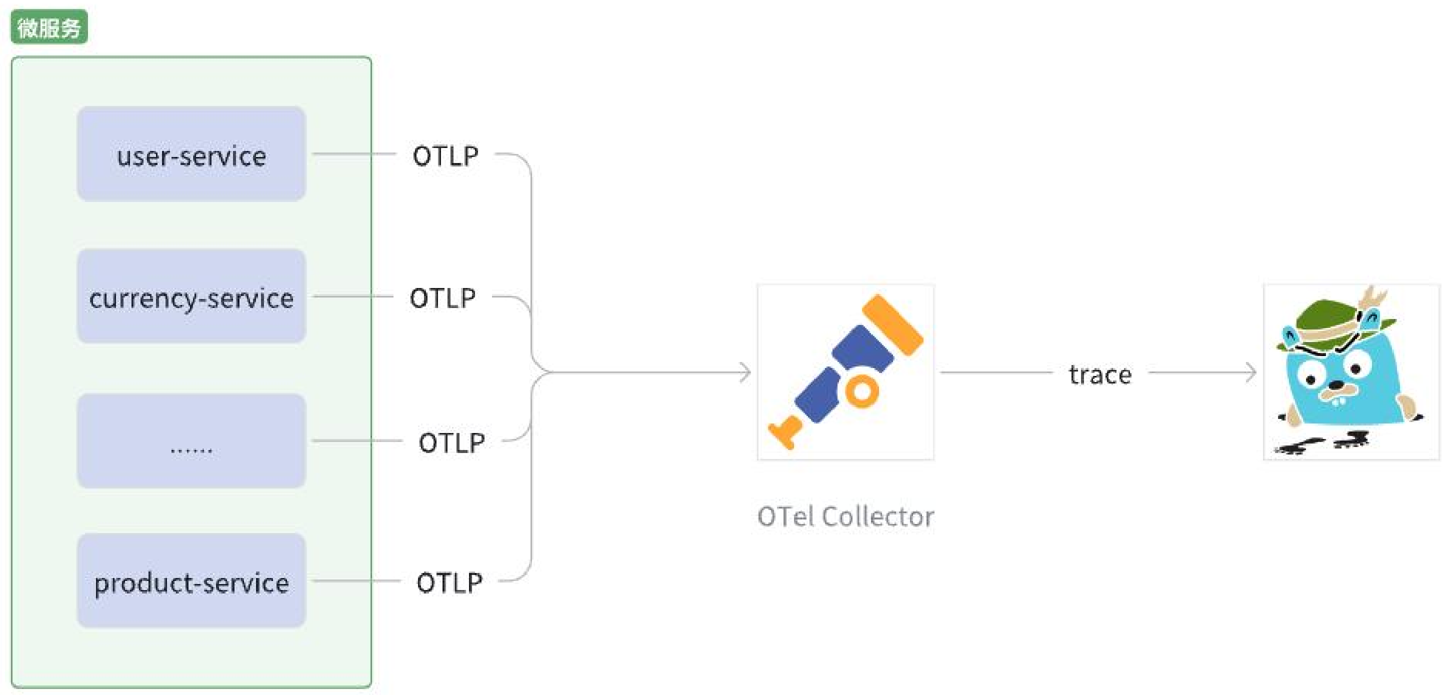
`TextMapCarrier` on pkg.go.dev ↗

Extract



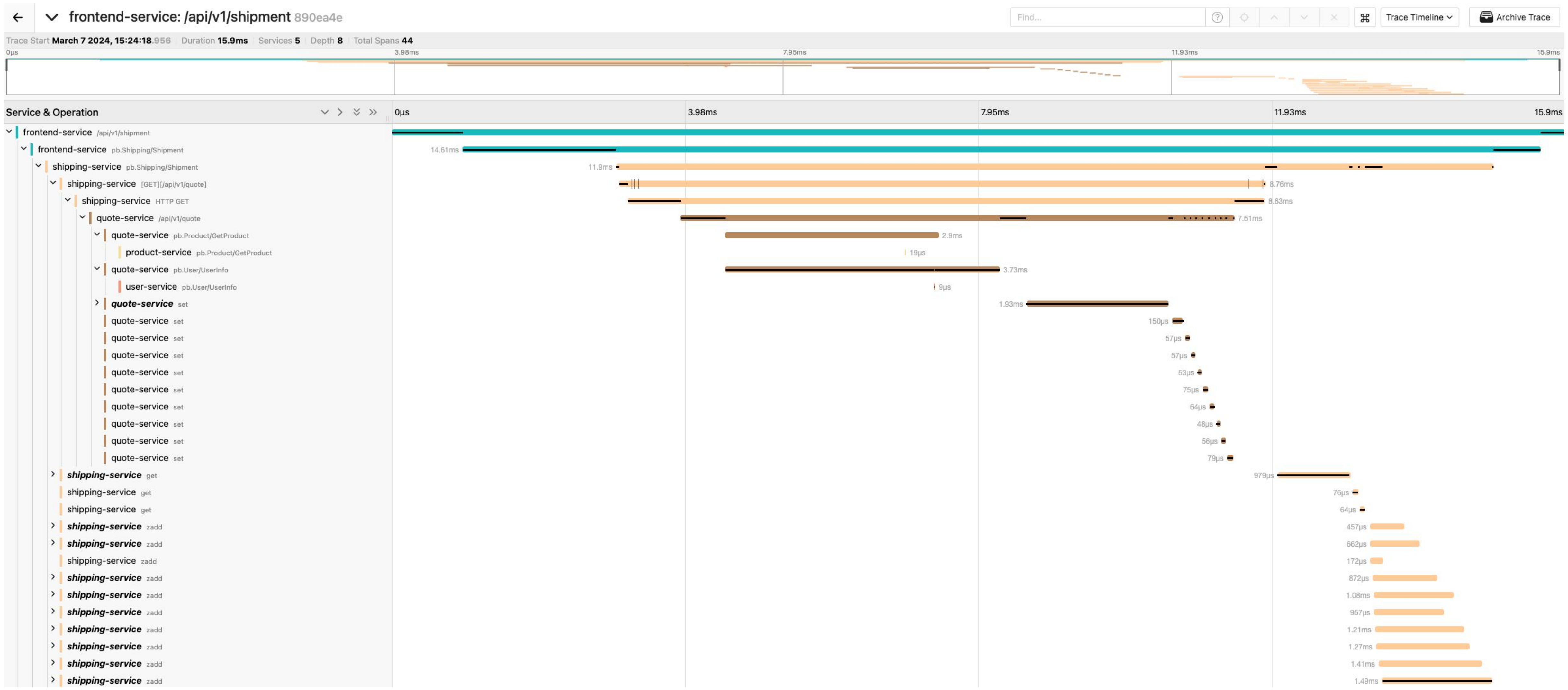
```
func (tc TraceContext) extract(carrier TextMapCarrier) trace.SpanContext {  1 usage  rghetia +4 *
    h := carrier.Get(traceparentHeader)

    var ver [1]byte
    if !extractPart(ver[:], &h,  n: 2) {
        return trace.SpanContext{}
    }
    version := int(ver[0])
    if version > maxVersion {
        return trace.SpanContext{}
    }

    var scc trace.SpanContextConfig
    if !extractPart(scc.TraceID[:], &h,  n: 32) {
        return trace.SpanContext{}
    }
    if !extractPart(scc.SpanID[:], &h,  n: 16) {
        return trace.SpanContext{}
    }

    var opts [1]byte
    if !extractPart(opts[:], &h,  n: 2) {
        return trace.SpanContext{}
    }

    scc.TraceFlags = trace.TraceFlags(opts[0]) & trace.FlagsSampled
    scc.TraceState, _ = trace.ParseTraceState(carrier.Get(tracestateHeader))
    scc.Remote = true

    return trace.NewSpanContext(scc)
}
```

数据流转



使用Collector的好处在于一些计算操作可以再Collector中统一处理，一些逻辑如压缩、过滤、配置变更等可以集中到Collector中实现，服务只需要实现很薄的一层埋点、采样逻辑即可，这也能使得链路追踪对业务服务本身的影响降到最低。

链路追踪分析



```
frontend-service: /api/v1/shipment 890ea4e
Trace Start March 7 2024, 15:24:18.956   Duration 15.9ms   Services 5   Depth 8   Total Spans 44

Service & Operation                               0µs    3.98ms    7.95ms    11.93ms    15.9ms
frontend-service  /api/v1/shipment
  frontend-service  pb.Shipping/Shipment                       14.61ms
    shipping-service  pb.Shipping/Shipment                     11.9ms
      shipping-service  [GET][/api/v1/quote]                   8.76ms
        shipping-service  HTTP GET                             8.63ms
          quote-service  /api/v1/quote                         7.51ms
            quote-service  pb.Product/GetProduct               2.9ms
              product-service  pb.Product/GetProduct           19µs
            quote-service  pb.User/UserInfo                    3.73ms
              user-service  pb.User/UserInfo                   9µs
            quote-service  set                                 1.93ms
            quote-service  set                                 150µs
            quote-service  set                                 57µs
            quote-service  set                                 57µs
            quote-service  set                                 53µs
            quote-service  set                                 75µs
            quote-service  set                                 64µs
            quote-service  set                                 48µs
            quote-service  set                                 56µs
            quote-service  set                                 79µs
      shipping-service  get                                    979µs
      shipping-service  get                                    76µs
      shipping-service  get                                    64µs
      shipping-service  zadd                                   457µs
      shipping-service  zadd                                   662µs
      shipping-service  zadd                                   172µs
      shipping-service  zadd                                   872µs
      shipping-service  zadd                                   1.08ms
      shipping-service  zadd                                   957µs
      shipping-service  zadd                                   1.21ms
      shipping-service  zadd                                   1.27ms
      shipping-service  zadd                                   1.41ms
      shipping-service  zadd                                   1.49ms
```

第二部分

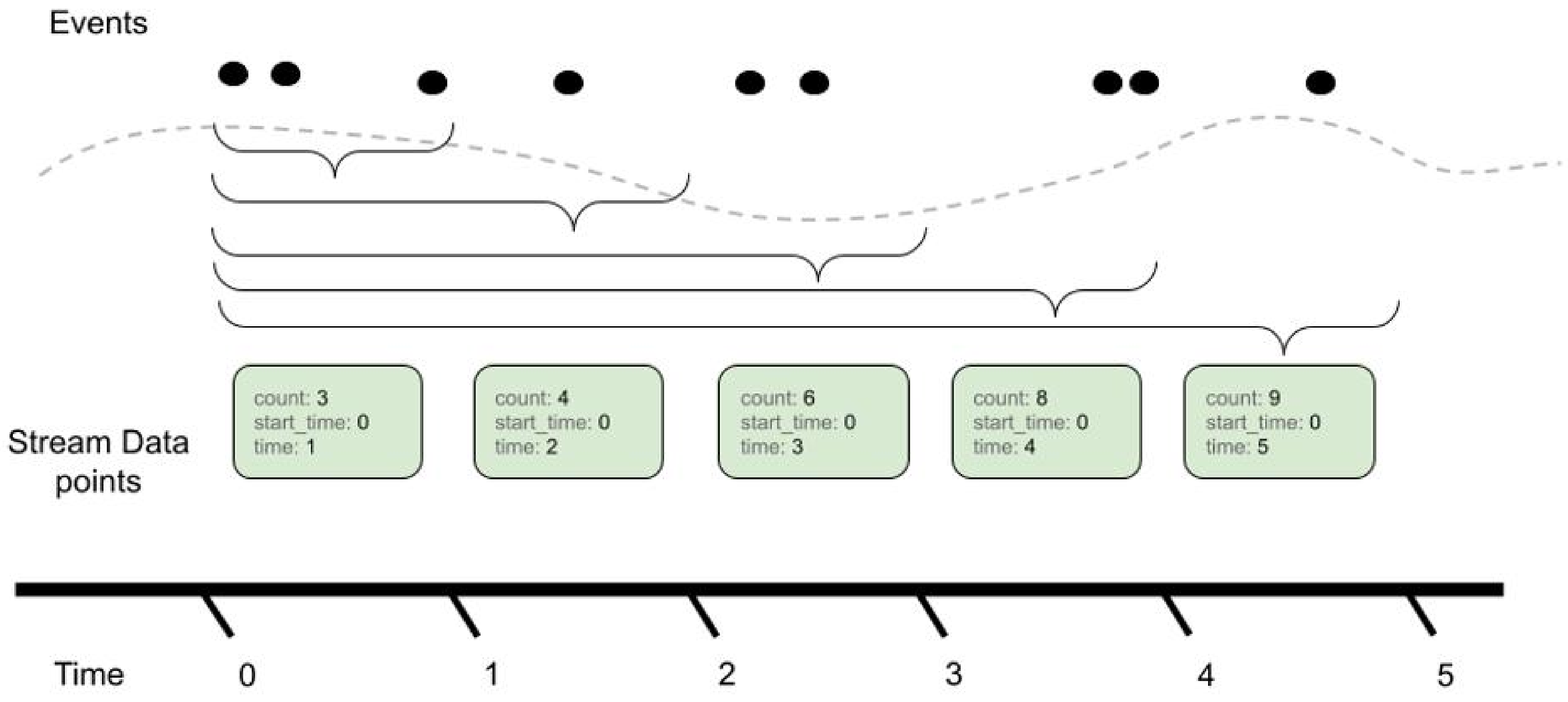
指标

指标数据类型

- Counter
- Gauge
- Histogram

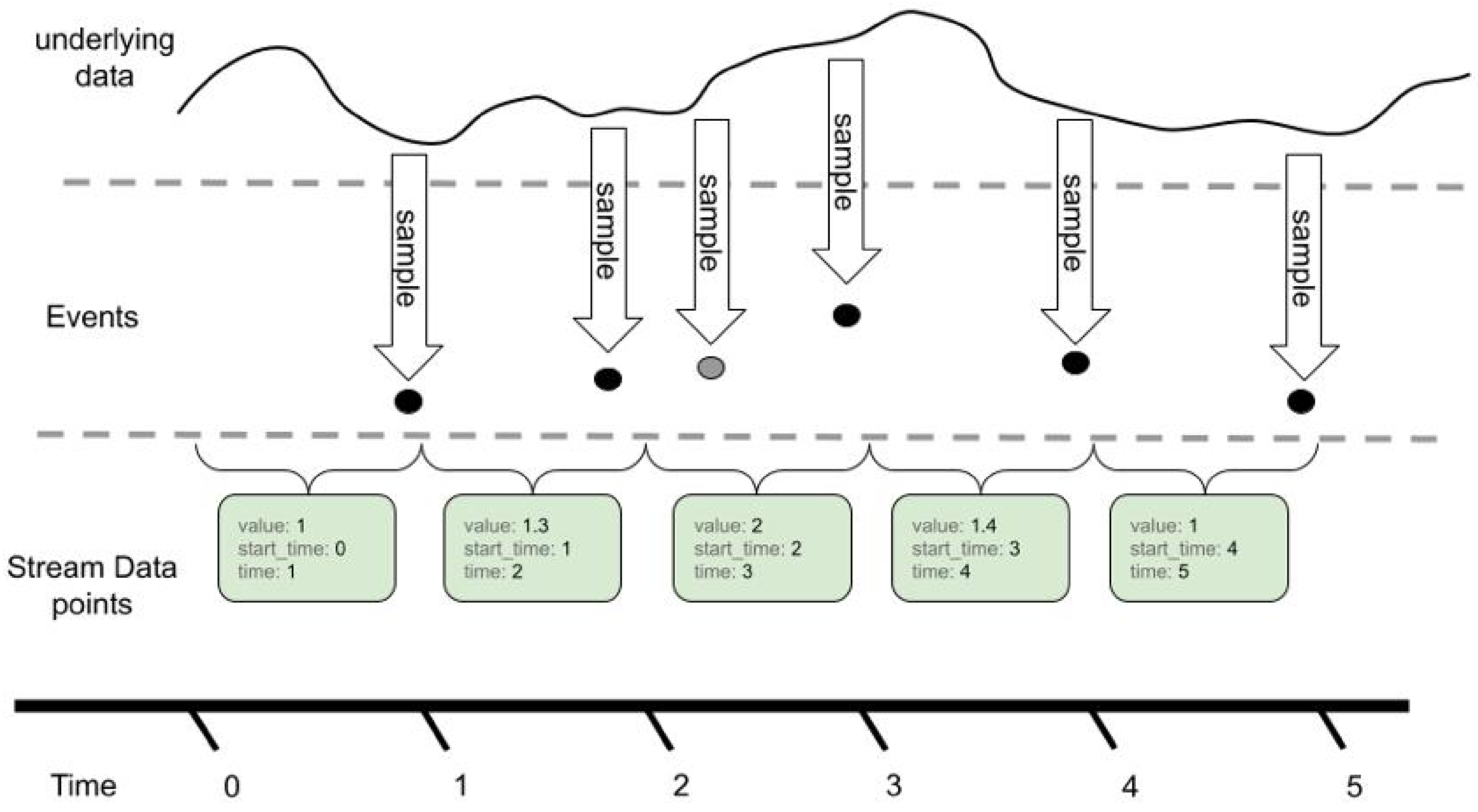
Counter

Counter 类型代表一种样本数据单调递增的指标，即只增不减，除非监控系统发生了重置。例如，你可以使用 counter 类型的指标来表示服务的请求数、已完成的任务数、错误发生的次数等。



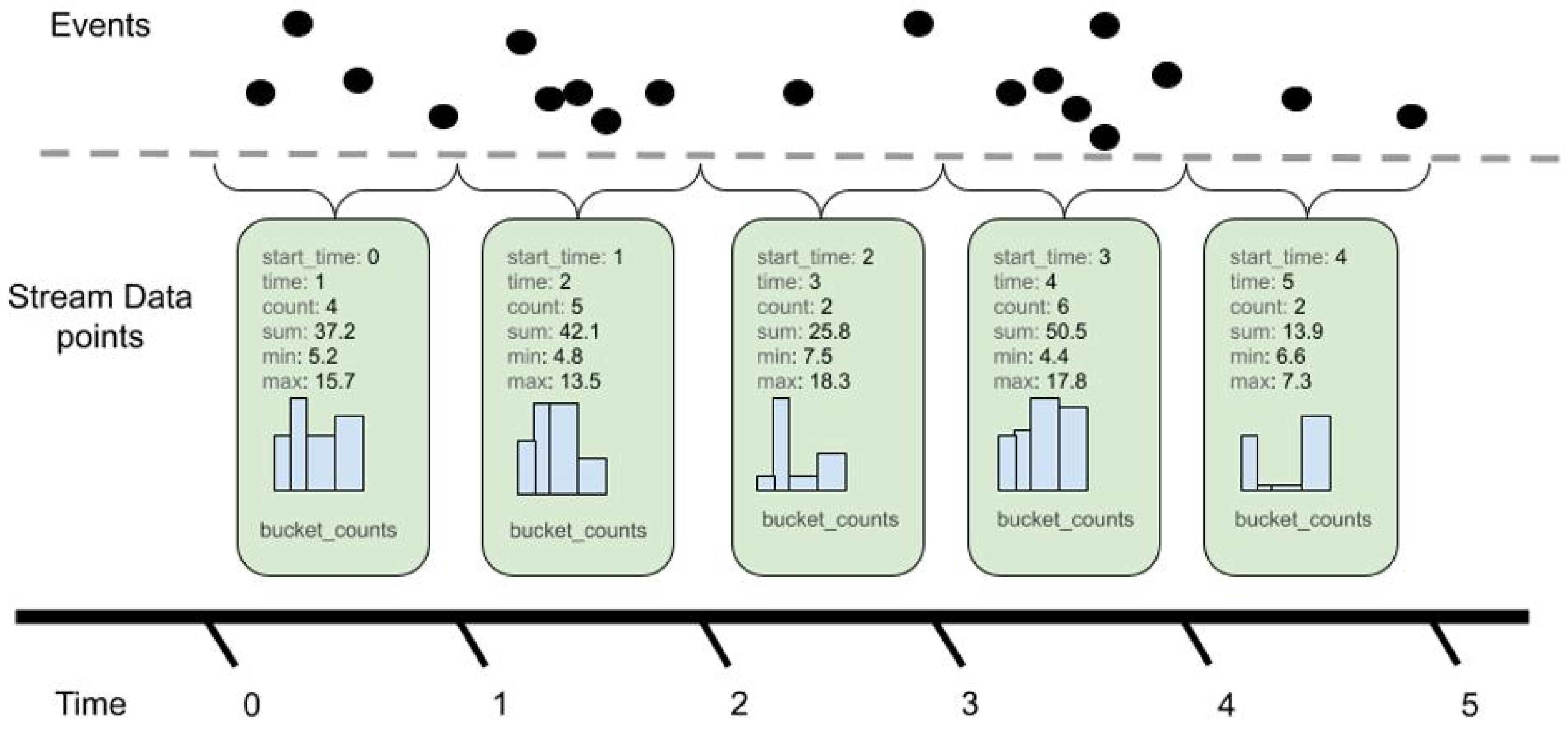
Gauge

Gauge 类型代表一种样本数据可以任意变化的指标，即可增可减。Gauge 通常用于像温度或者内存使用率这种指标数据，也可以表示能随时增加或减少的“总数”，例如：当前并发请求的数量。

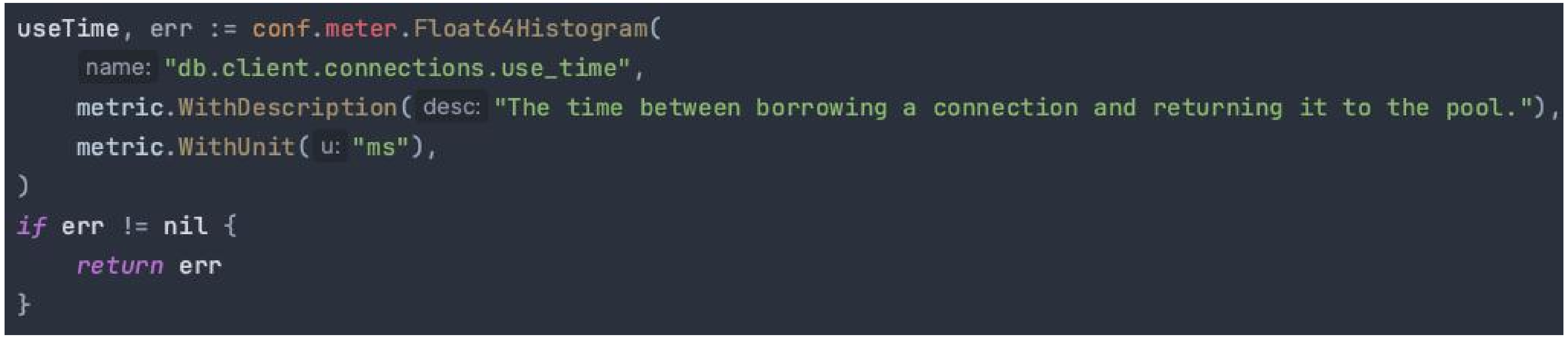


Histogram

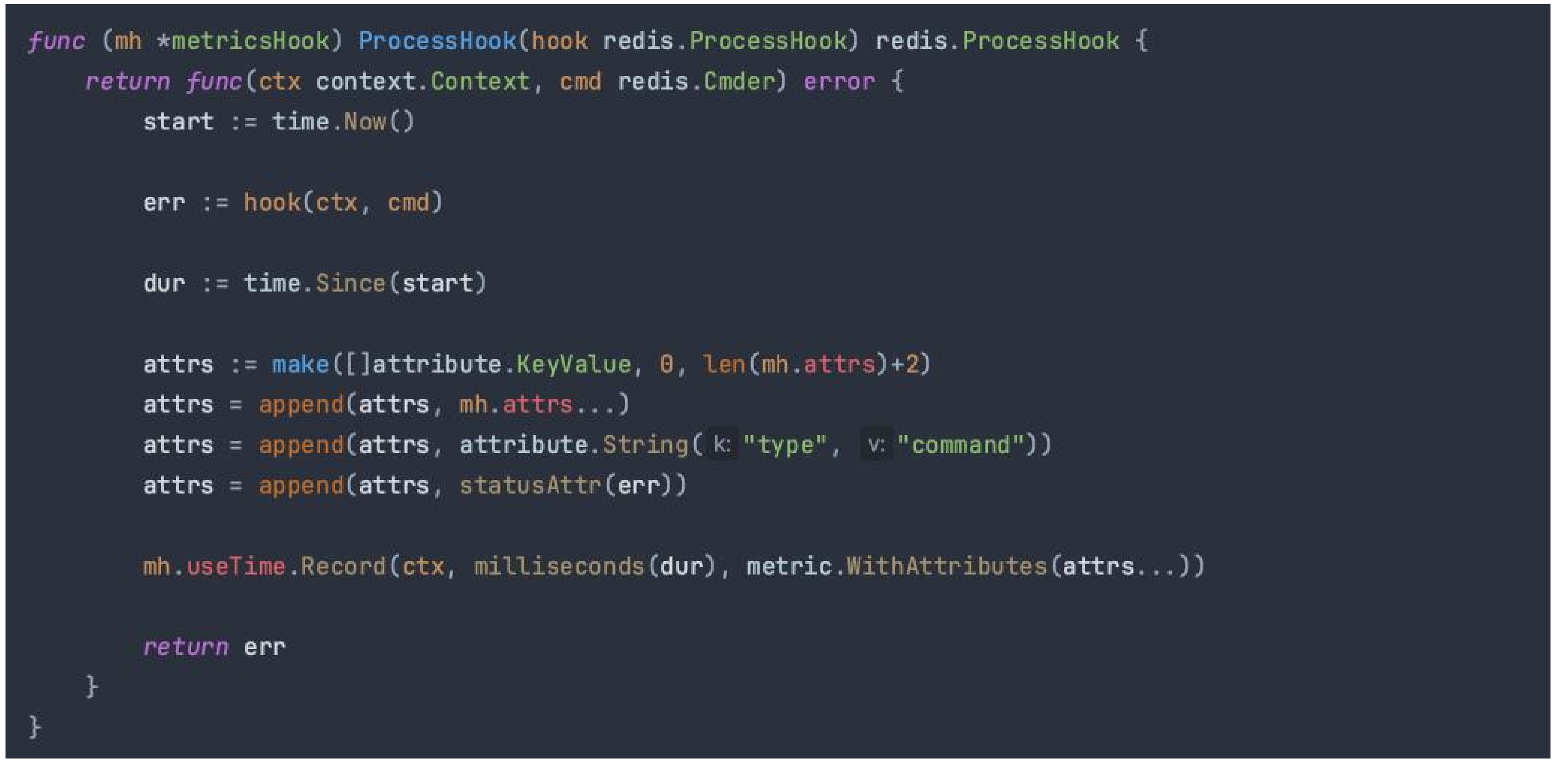
Histogram 在一段时间范围内对数据进行采样，并将其计入可配置的存储桶（bucket）中，后续可通过指定区间筛选样本，也可以统计样本总数。



插桩(Hook)

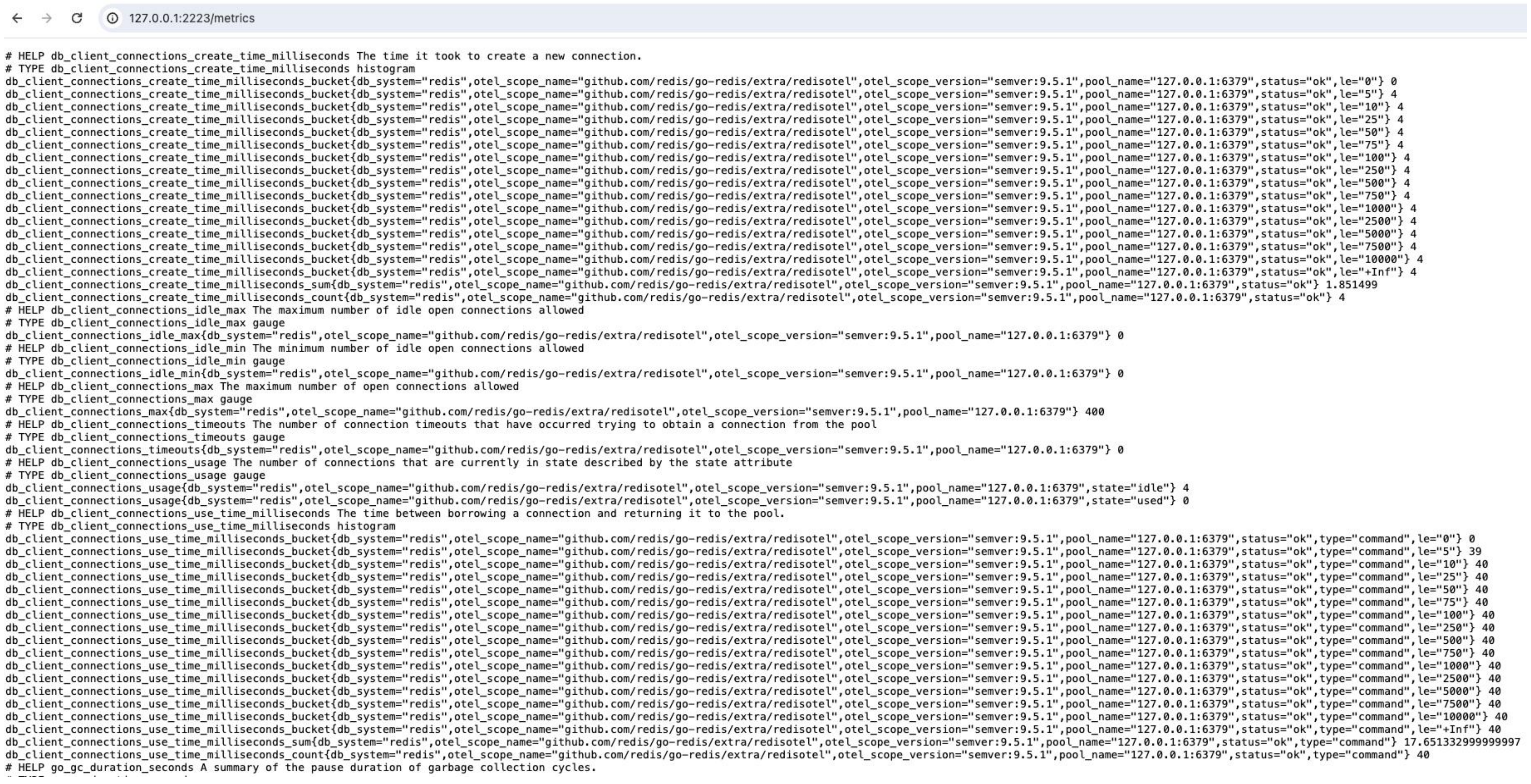


```
useTime, err := conf.meter.Float64Histogram(
    name: "db.client.connections.use_time",
    metric.WithDescription( desc: "The time between borrowing a connection and returning it to the pool."),
    metric.WithUnit( u: "ms"),
)
if err != nil {
    return err
}
```



```
func (mh *metricsHook) ProcessHook(hook redis.ProcessHook) redis.ProcessHook {
    return func(ctx context.Context, cmd redis.Cmder) error {
        start := time.Now()

        err := hook(ctx, cmd)

        dur := time.Since(start)

        attrs := make([]attribute.KeyValue, 0, len(mh.attrs)+2)
        attrs = append(attrs, mh.attrs...)
        attrs = append(attrs, attribute.String( k: "type",  v: "command"))
        attrs = append(attrs, statusAttr(err))

        mh.useTime.Record(ctx, milliseconds(dur), metric.WithAttributes(attrs...))

        return err
    }
}
```

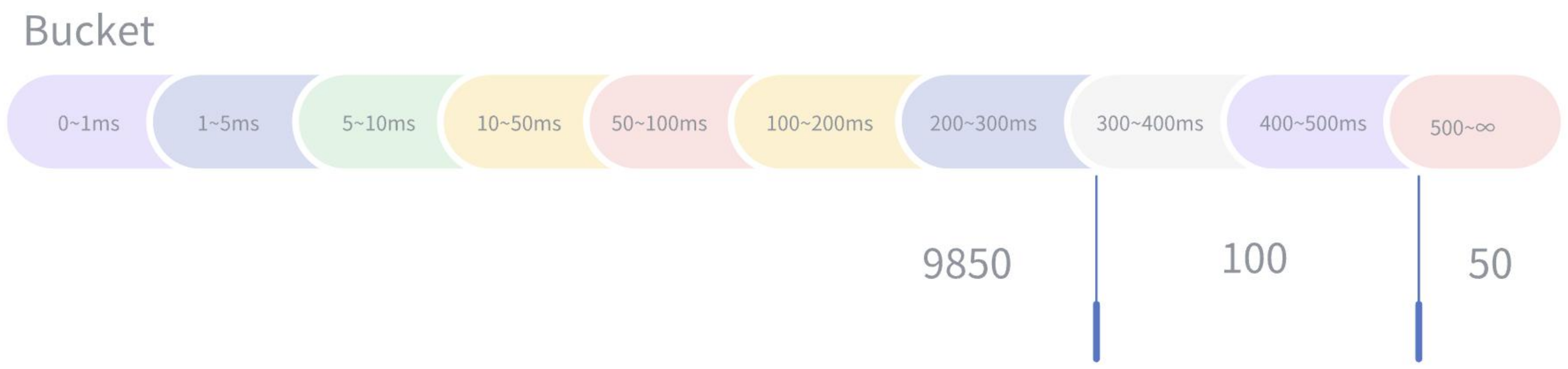
指标数据



127.0.0.1:2223/metrics

```
# HELP db_client_connections_create_time_milliseconds The time it took to create a new connection.
# TYPE db_client_connections_create_time_milliseconds histogram
db_client_connections_create_time_milliseconds_bucket{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",le="0"} 0
db_client_connections_create_time_milliseconds_bucket{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",le="5"} 4
db_client_connections_create_time_milliseconds_bucket{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",le="10"} 4
db_client_connections_create_time_milliseconds_bucket{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",le="25"} 4
db_client_connections_create_time_milliseconds_bucket{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",le="50"} 4
db_client_connections_create_time_milliseconds_bucket{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",le="75"} 4
db_client_connections_create_time_milliseconds_bucket{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",le="100"} 4
db_client_connections_create_time_milliseconds_bucket{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",le="250"} 4
db_client_connections_create_time_milliseconds_bucket{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",le="500"} 4
db_client_connections_create_time_milliseconds_bucket{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",le="750"} 4
db_client_connections_create_time_milliseconds_bucket{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",le="1000"} 4
db_client_connections_create_time_milliseconds_bucket{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",le="2500"} 4
db_client_connections_create_time_milliseconds_bucket{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",le="5000"} 4
db_client_connections_create_time_milliseconds_bucket{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",le="7500"} 4
db_client_connections_create_time_milliseconds_bucket{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",le="10000"} 4
db_client_connections_create_time_milliseconds_bucket{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",le="+Inf"} 4
db_client_connections_create_time_milliseconds_sum{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok"} 1.851499
db_client_connections_create_time_milliseconds_count{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok"} 4
# HELP db_client_connections_idle_max The maximum number of idle open connections allowed
# TYPE db_client_connections_idle_max gauge
db_client_connections_idle_max{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379"} 0
# HELP db_client_connections_idle_min The minimum number of idle open connections allowed
# TYPE db_client_connections_idle_min gauge
db_client_connections_idle_min{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379"} 0
# HELP db_client_connections_max The maximum number of open connections allowed
# TYPE db_client_connections_max gauge
db_client_connections_max{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379"} 400
# HELP db_client_connections_timeouts The number of connection timeouts that have occurred trying to obtain a connection from the pool
# TYPE db_client_connections_timeouts gauge
db_client_connections_timeouts{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379"} 0
# HELP db_client_connections_usage The number of connections that are currently in state described by the state attribute
# TYPE db_client_connections_usage gauge
db_client_connections_usage{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",state="idle"} 4
db_client_connections_usage{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",state="used"} 0
# HELP db_client_connections_use_time_milliseconds The time between borrowing a connection and returning it to the pool.
# TYPE db_client_connections_use_time_milliseconds histogram
db_client_connections_use_time_milliseconds_bucket{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",type="command",le="0"} 0
db_client_connections_use_time_milliseconds_bucket{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",type="command",le="5"} 39
db_client_connections_use_time_milliseconds_bucket{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",type="command",le="10"} 40
db_client_connections_use_time_milliseconds_bucket{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",type="command",le="25"} 40
db_client_connections_use_time_milliseconds_bucket{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",type="command",le="50"} 40
db_client_connections_use_time_milliseconds_bucket{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",type="command",le="75"} 40
db_client_connections_use_time_milliseconds_bucket{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",type="command",le="100"} 40
db_client_connections_use_time_milliseconds_bucket{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",type="command",le="250"} 40
db_client_connections_use_time_milliseconds_bucket{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",type="command",le="500"} 40
db_client_connections_use_time_milliseconds_bucket{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",type="command",le="750"} 40
db_client_connections_use_time_milliseconds_bucket{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",type="command",le="1000"} 40
db_client_connections_use_time_milliseconds_bucket{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",type="command",le="2500"} 40
db_client_connections_use_time_milliseconds_bucket{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",type="command",le="5000"} 40
db_client_connections_use_time_milliseconds_bucket{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",type="command",le="7500"} 40
db_client_connections_use_time_milliseconds_bucket{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",type="command",le="10000"} 40
db_client_connections_use_time_milliseconds_bucket{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",type="command",le="+Inf"} 40
db_client_connections_use_time_milliseconds_sum{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",type="command"} 17.651332999999997
db_client_connections_use_time_milliseconds_count{db_system="redis",otel_scope_name="github.com/redis/go-redis/extra/redisotel",otel_scope_version="semver:9.5.1",pool_name="127.0.0.1:6379",status="ok",type="command"} 40
# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection cycles.
```

指标计算



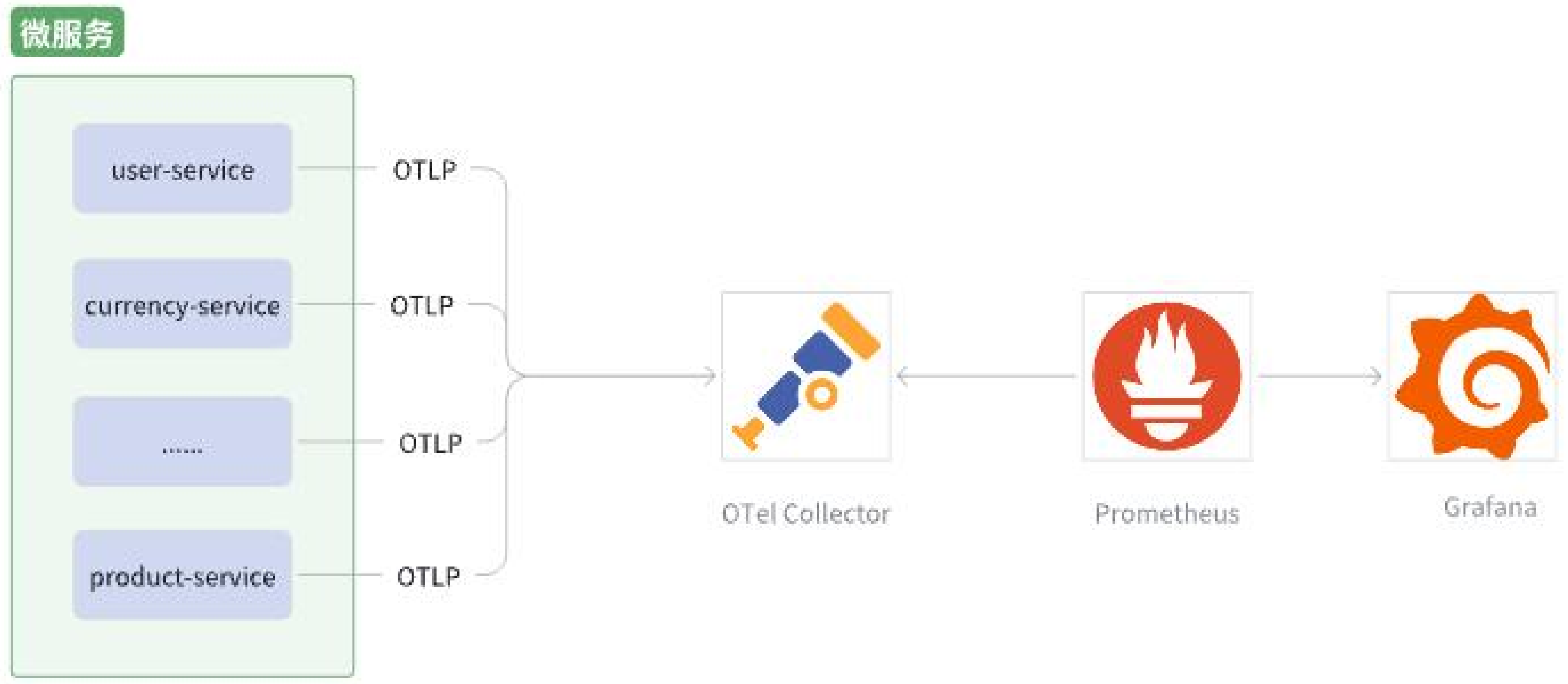
P99 = 10000 * 0.99 = 9900

9900 - 9850 = 第50个请求

(500 - 300) * (50/100) + 300 = 400ms

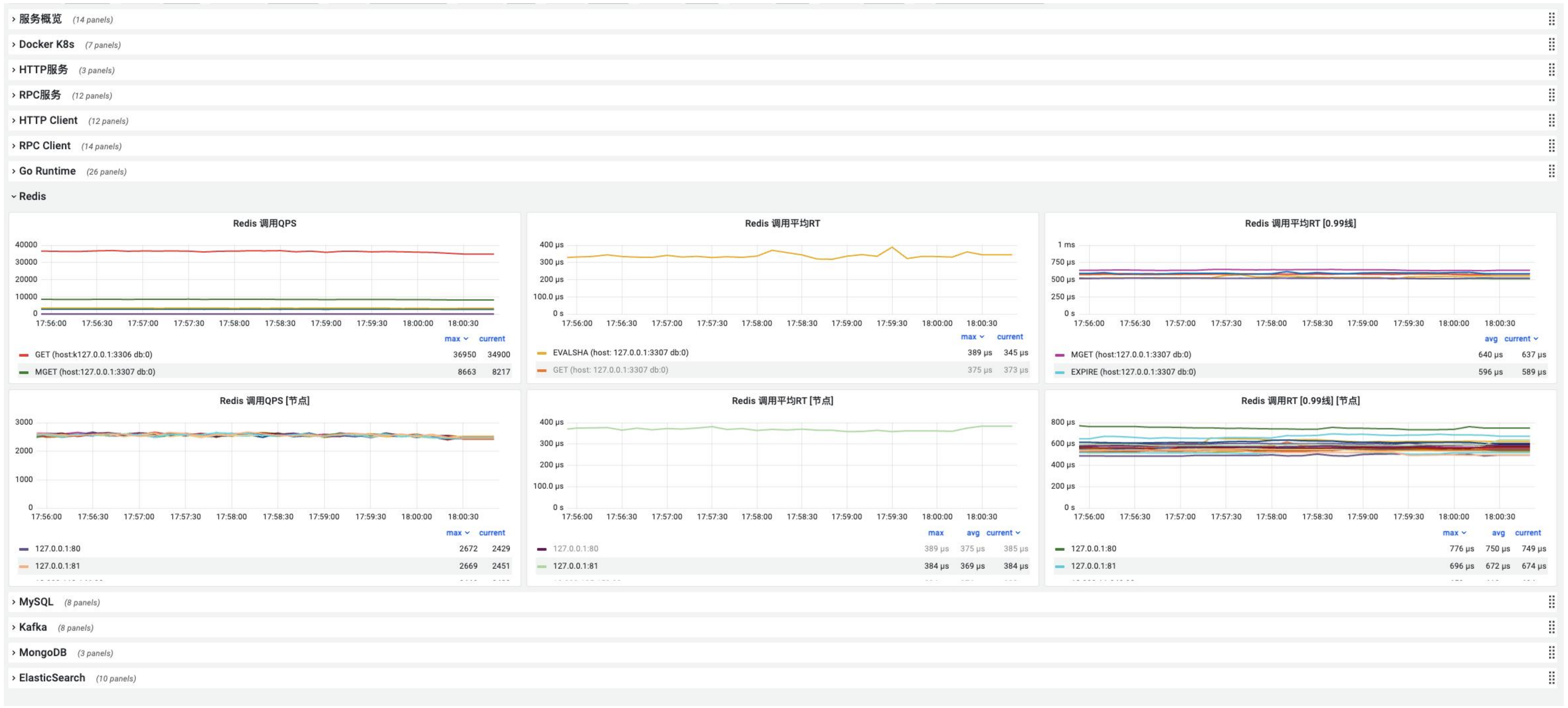
P99 = 400ms

数据流转



使用Collector的好处在于一些计算操作可以再Collector中统一处理，一些逻辑如压缩、过滤、配置变更等可以集中到Collector中实现，服务只需要实现很薄的一层埋点、采样逻辑即可，这也能使得链路追踪对业务服务本身的影响降到最低。

数据大盘



› 服务概览 *(14 panels)*

› Docker K8s *(7 panels)*

› HTTP服务 *(3 panels)*

› RPC服务 *(12 panels)*

› HTTP Client *(12 panels)*

› RPC Client *(14 panels)*

› Go Runtime *(26 panels)*

⌄ Redis

Redis 调用QPS

	max	current
🟥 GET (host:k127.0.0.1:3306 db:0)	36950	34900
🟩 MGET (host:127.0.0.1:3307 db:0)	8663	8217

Redis 调用平均RT

	max	current
🟨 EVALSHA (host: 127.0.0.1:3307 db:0)	389 µs	345 µs
🟧 GET (host: 127.0.0.1:3307 db:0)	375 µs	373 µs

Redis 调用平均RT [0.99线]

	avg	current
🟪 MGET (host:127.0.0.1:3307 db:0)	640 µs	637 µs
🟦 EXPIRE (host:127.0.0.1:3307 db:0)	596 µs	589 µs

Redis 调用QPS [节点]

	max	current
🟦 127.0.0.1:80	2672	2429
🟧 127.0.0.1:81	2669	2451

Redis 调用平均RT [节点]

	max	avg	current
🟪 127.0.0.1:80	389 µs	375 µs	385 µs
🟩 127.0.0.1:81	384 µs	369 µs	384 µs

Redis 调用RT [0.99线] [节点]

	max	avg	current
🟩 127.0.0.1:80	776 µs	750 µs	749 µs
🟦 127.0.0.1:81	696 µs	672 µs	674 µs

› MySQL *(8 panels)*

› Kafka *(8 panels)*

› MongoDB *(3 panels)*

› ElasticSearch *(10 panels)*

谢谢