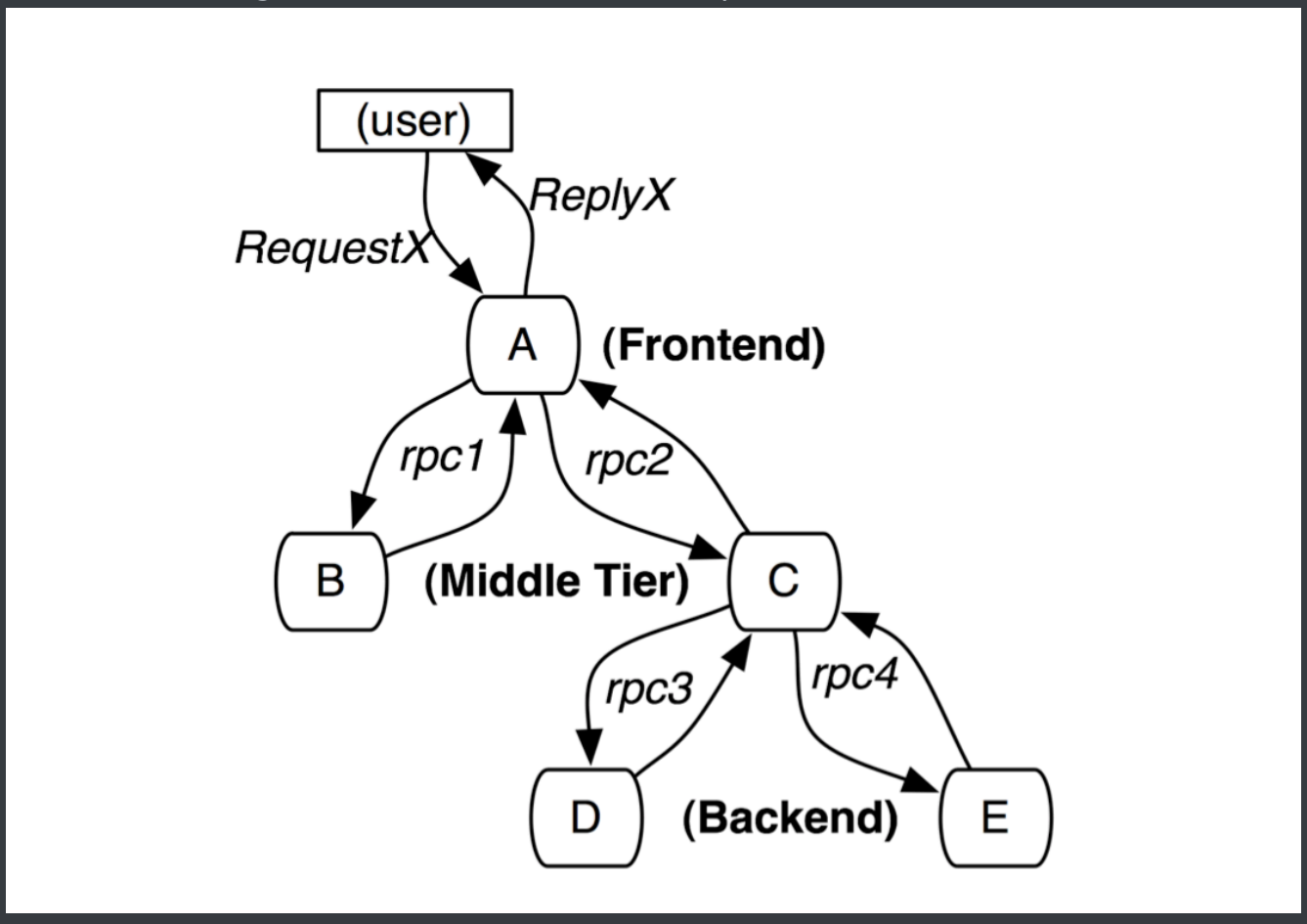
Tracing 介绍

- Tracing 概念
- Tracing 标准 - opentracing
- Tracing 实现
- Tracing 使用方法
- Tracing 集成在Gateway中

Tracing 概念

- 目标：追踪请求执行踪迹，特别是在分布式系统中
- 追踪请求，调用链
- 为什么需要tracing。一个请求案例，如果请求RequestX出现变慢，如何排查?
-



Tracing 标准 - OpenTracing

- Trace: 请求的踪迹汇总
- span: 请求的上下文，主要包括traceID, SpanID，发生时间，结束时间。可以认为是一次函数调用
- Annotation: 请求执行过程中的发生的事件，用于具体的事件记录，例如记录函数开始时间
- Tag：与请求相关的标记内容，相当于日志记录。多用于debug的具体case搜索，例如记录每个请求的业务ID。
- reporter：代表传输组件，一般是单独实现，集成在span内

span 举例

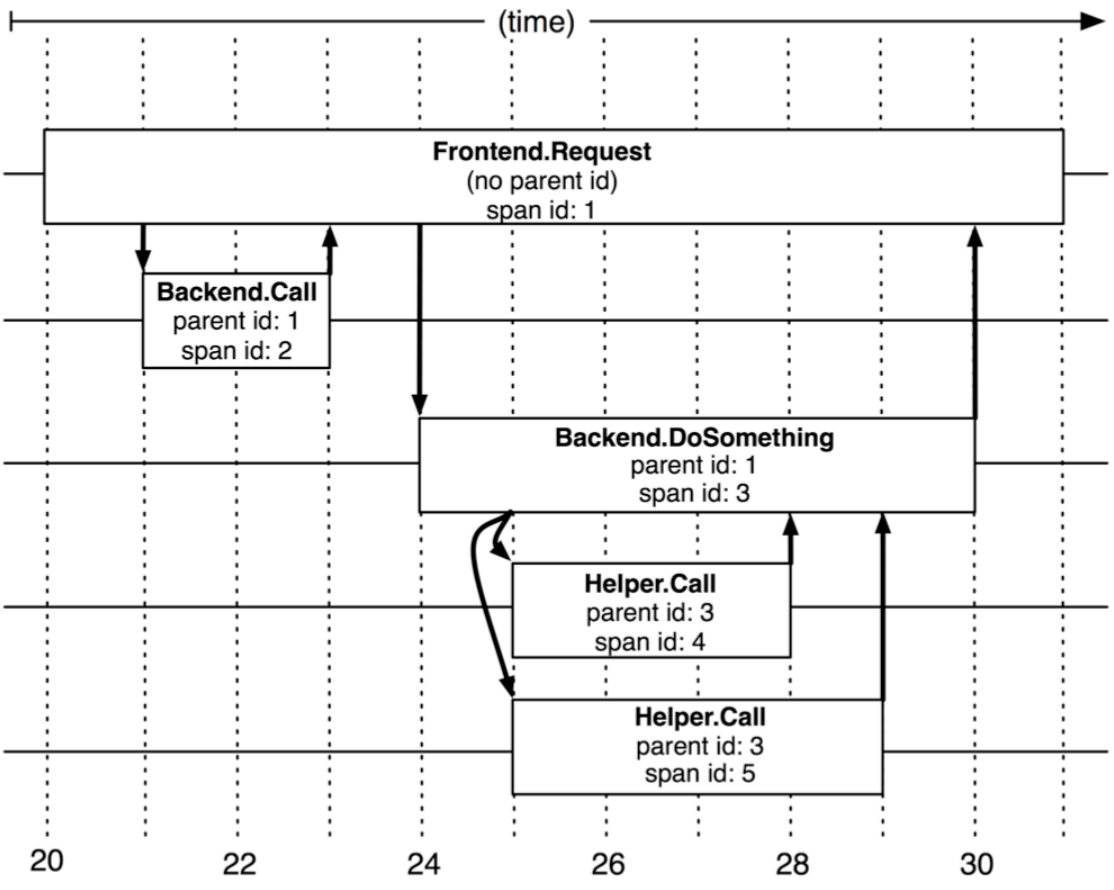
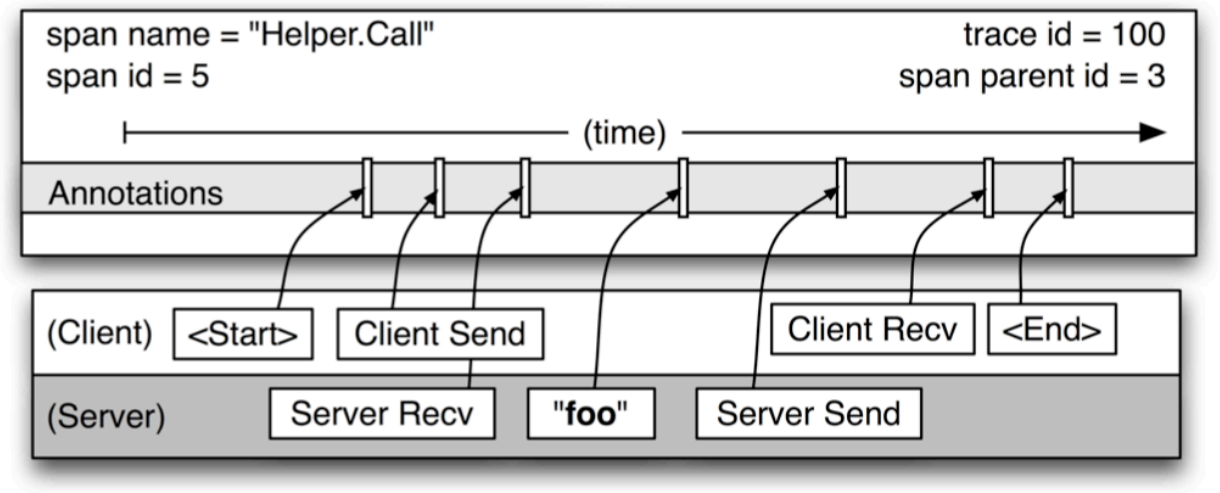
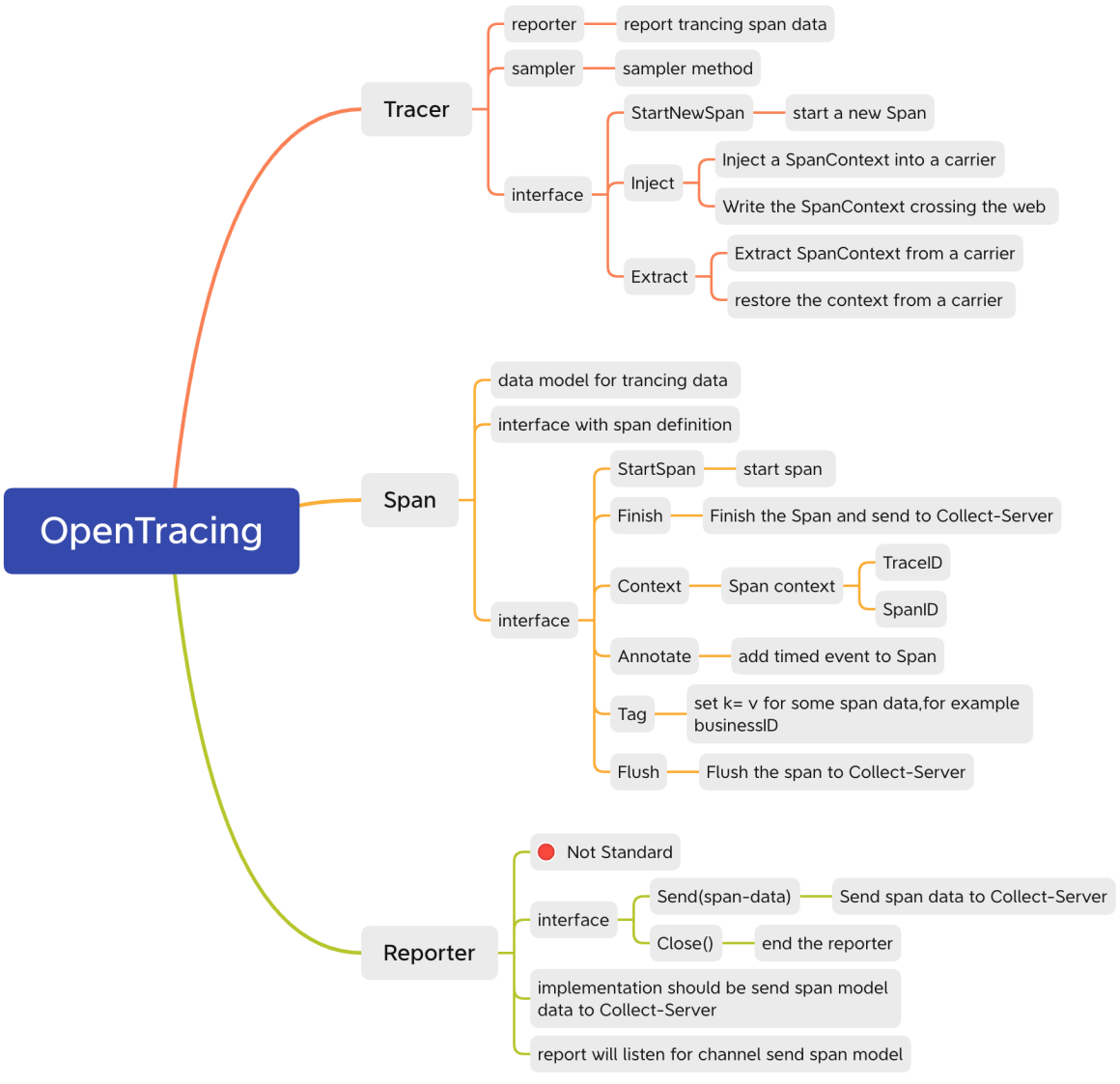
- span relationship
- 

Figure 2: The causal and temporal relationships between five spans in a Dapper trace tree.

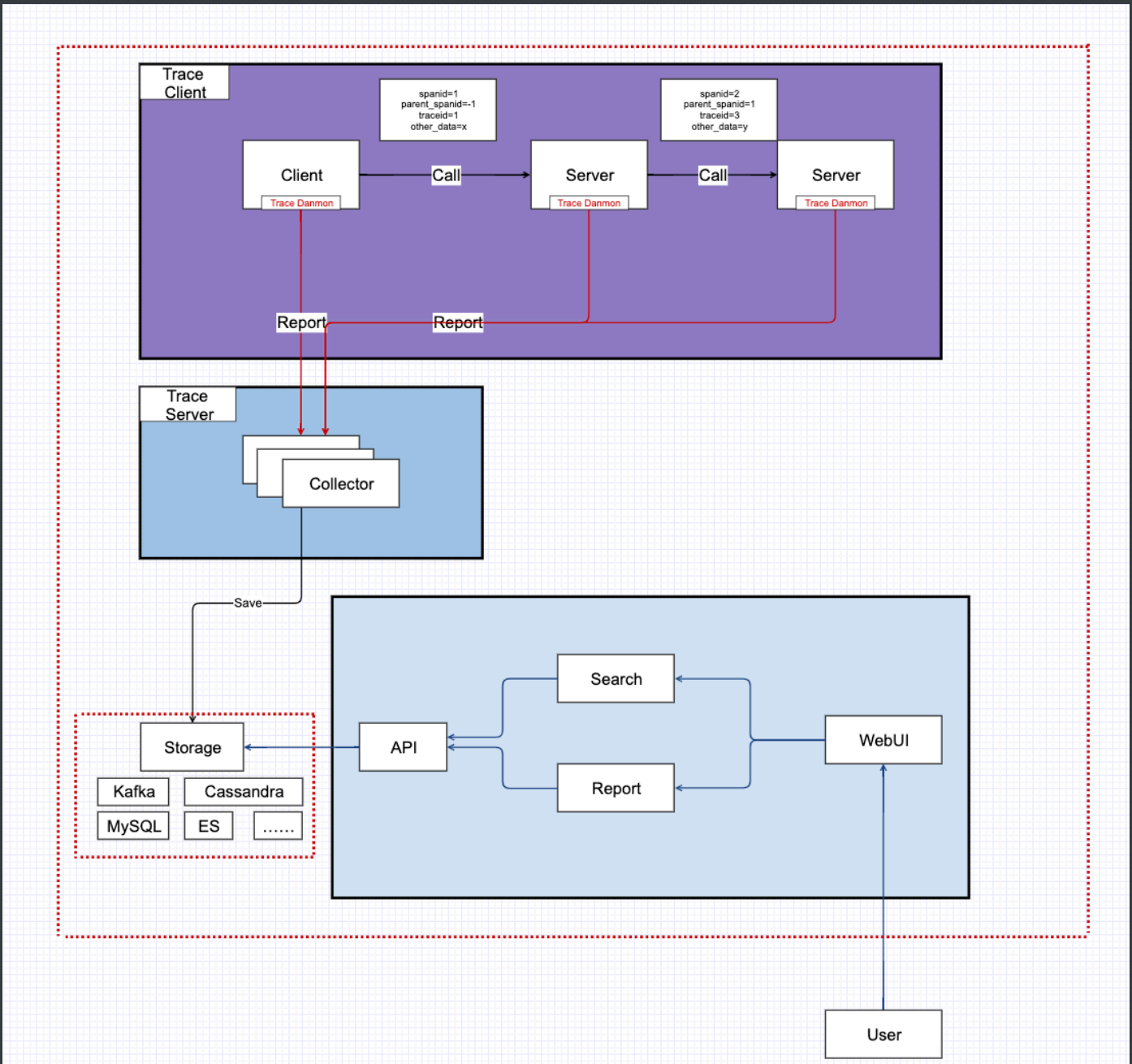
- one span example


span name = "Helper.Call"
span id = 5
trace id = 100
span parent id = 3
(time)
Annotations
(Client)
<Start>
Client Send
Client Recv
<End>
(Server)
Server Recv
"foo"
Server Send


OpenTracing
Tracer
reporter
report trancing span data
sampler
sampler method
interface
StartNewSpan
start a new Span
Inject
Inject a SpanContext into a carrier
Write the SpanContext crossing the web
Extract
Extract SpanContext from a carrier
restore the context from a carrier
Span
data model for trancing data
interface with span definition
interface
StartSpan
start span
Finish
Finish the Span and send to Collect-Server
Context
Span context
TraceID
SpanID
Annotate
add timed event to Span
Tag
set k= v for some span data,for example businessID
Flush
Flush the span to Collect-Server
Reporter
Not Standard
interface
Send(span-data)
Send span data to Collect-Server
Close()
end the reporter
implementation should be send span model data to Collect-Server
report will listen for channel send span model

Tracing系统的基本架构

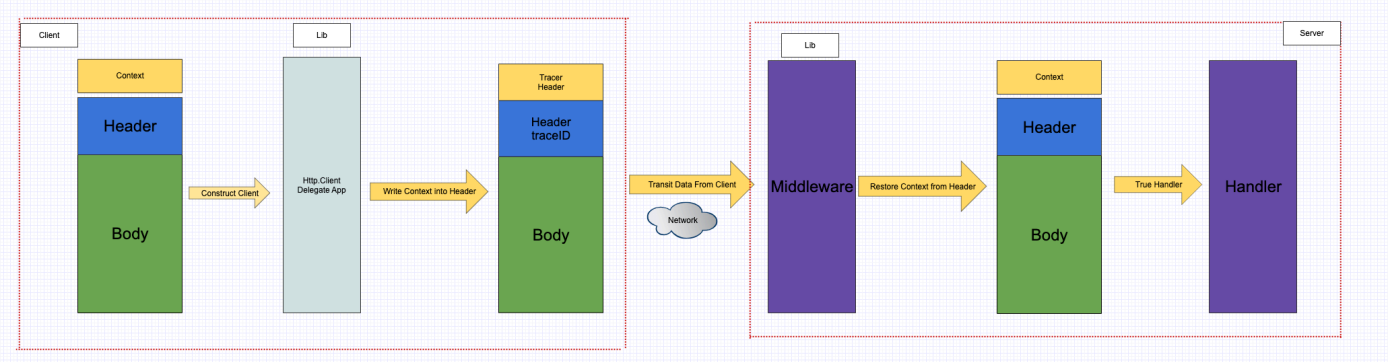
- Tracing结构图
- Tracing Client: 上报trace
- Tracing Server : 收集trace数据
- Storage：存储trace数据
- User API : 用户界面
- Tracing具体实现以lib包的形式提供给应用程序



tracing 实现

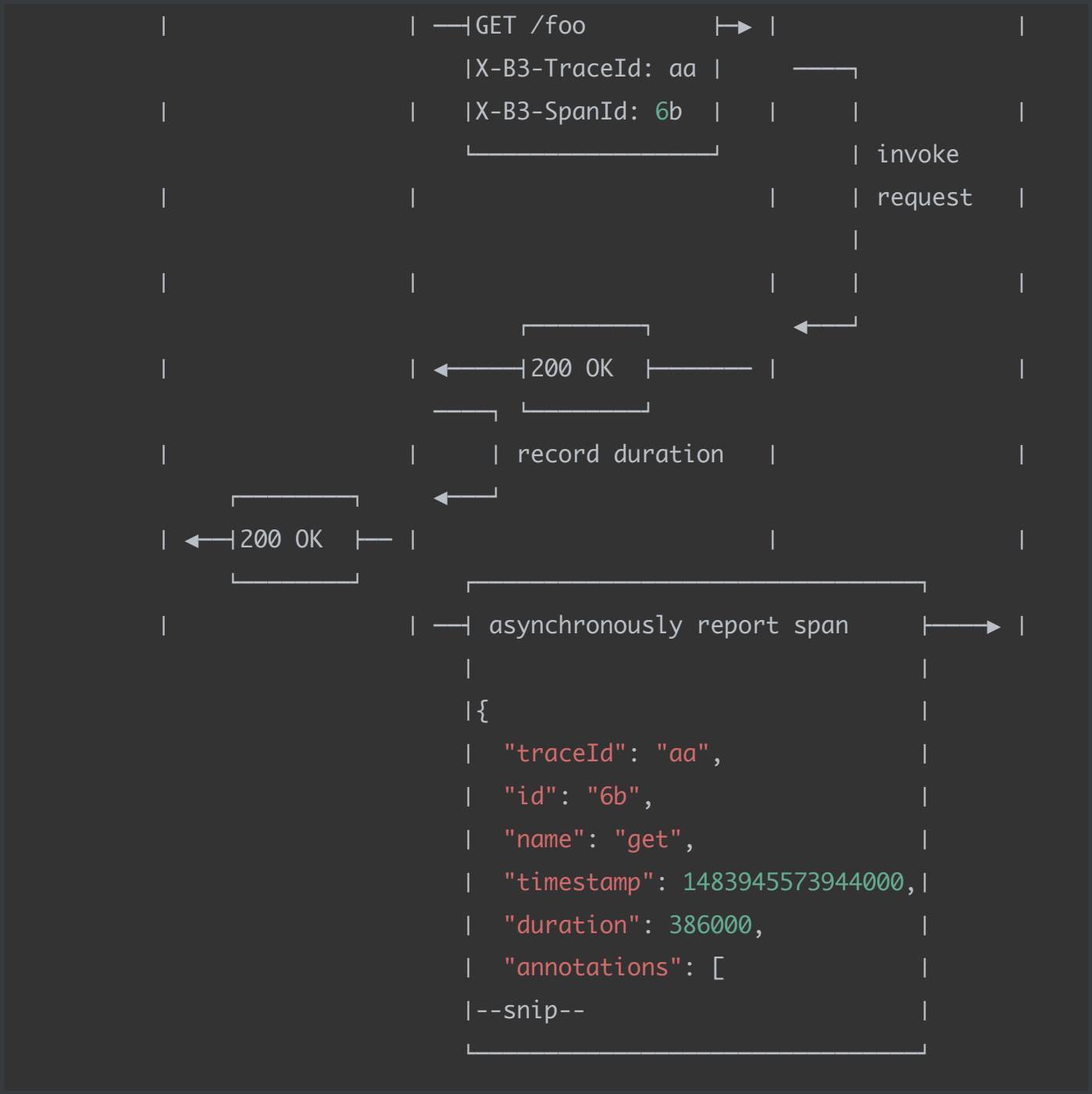
http实现

- http代理流程



- 使用http对Header的代理示例

```
┌──────────────┐ ┌──────────────────────────┐  ┌──────────────┐
 ┌──────────────────┐
| User Code    | | Trace Instrumentation |  | Http Client |  | Zipkin
Collector |
└──────────────┘ └──────────────────────────┘  └──────────────┘
 └──────────────────┘
      |                |                          |                |
          ┌─────────┐
      | ──┤GET /foo ├─▶ | ────┐                    |                |
          └─────────┘          | record tags
      |                | ◀───┘                    |                |
                           ────┐
      |                |      | add trace headers |                |
                           ◀───┘
      |                | ────┐                    |                |
                                | record timestamp
      |                | ◀───┘                    |                |
                             ┌──────────────────┐
```



- 具体header注入的标记

```
// Default B3 Header keys
const (
  TraceID      = "x-b3-traceid"
  SpanID       = "x-b3-spanid"
  ParentSpanID = "x-b3-parentspanid"
  Sampled      = "x-b3-sampled"
  Flags        = "x-b3-flags"
  Context      = "b3"
)
```

- 其他内容，性能监控，采样频率等

grpc实现

- Metadata with intercepter

Tracing 使用

zipkin go的使用

```
// create reporter to localhost zipkin-server
reporter := zipkinHTTP.NewReporter("http://localhost:9411/api/v2/spans")
defer reporter.Close()
//create tracer
tracer, err = openzipkin.NewTracer(reporter,
openzipkin.WithLocalEndpoint(localEndpoint))
```

客户端

- http.Client的代理

```
  localEndpoint, err := openzipkin.NewEndpoint("local-client",
"192.168.1.61:8082")

  reporter :=
zipkinHTTP.NewReporter("http://localhost:9411/api/v2/spans")
  defer reporter.Close()

  //exporter := zipkin.NewExporter(reporter, localEndpoint)
  tracer, err = openzipkin.NewTracer(reporter,
openzipkin.WithLocalEndpoint(localEndpoint))

  // create  zipkin traced http client
  client, err := zipkinmw.NewClient(tracer, zipkinmw.ClientTrace(true),
zipkinmw.ClientTags(map[string]string{"type:": "from-raw-http-client"}))

  // initiate a call to list
  url := "http://localhost:8080/list"
  req, err := http.NewRequest("GET", url, nil)

  res, err := client.DoWithAppSpan(req, "raw-http-client")

  defer res.Body.Close() // close will send the span data
  body, err := ioutil.ReadAll(res.Body)
  log.Printf("%s", body)
```

服务端

- raw http 的使用

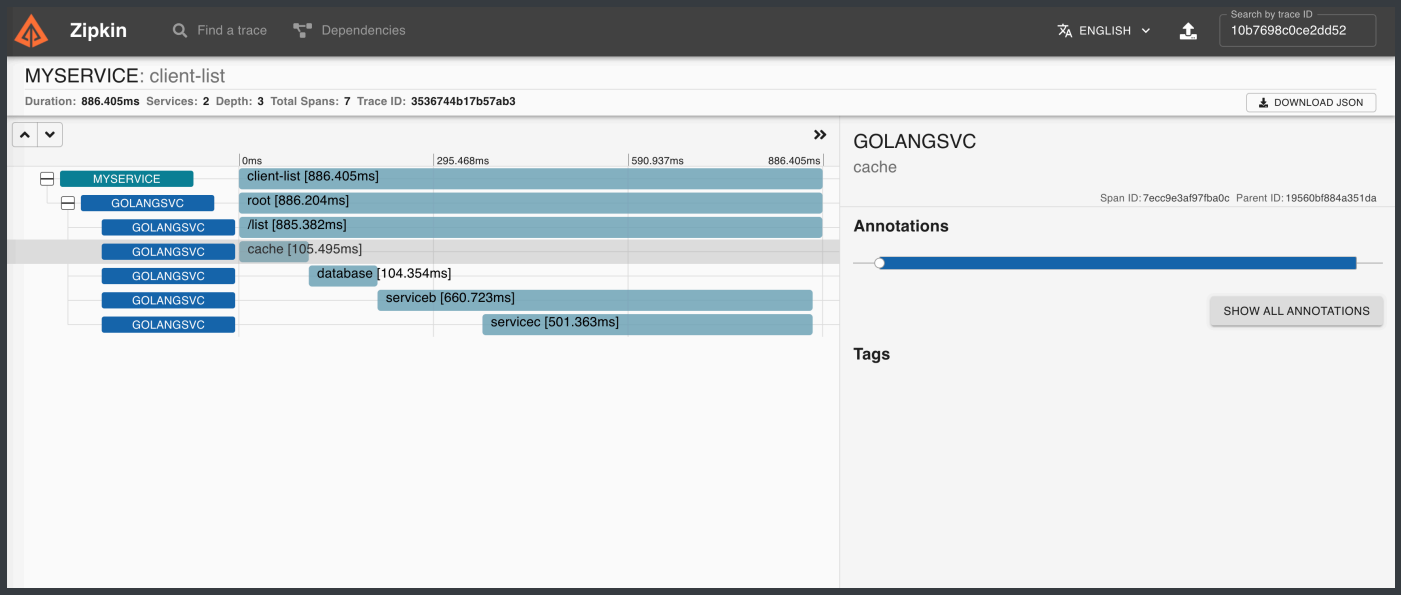
```
```

```
func main() {

  localEndpoint, err := openzipkin.NewEndpoint("golangsvc",
"192.168.1.61:8082")

  reporter :=
zipkinHTTP.NewReporter("http://localhost:9411/api/v2/spans")
  defer reporter.Close()

  //exporter := zipkin.NewExporter(reporter, localEndpoint)
  tracer, err = openzipkin.NewTracer(reporter,
openzipkin.WithLocalEndpoint(localEndpoint))

  mux := http.NewServeMux()
  mux.HandleFunc("/list", list)
  spanName := "root"
  // use the middleware around the true http.Handler
  middler := zipkinmw.NewServerMiddleware(
    tracer,
    zipkinmw.SpanName(spanName),
    zipkinmw.TagResponseSize(true),
  )

  h := middler(mux)
  port := ":8080"
  log.Printf("Server listening! %s ...", port)
  log.Fatal(http.ListenAndServe(port, h))

}

func list(w http.ResponseWriter, r *http.Request) {
  log.Printf("Serving request: %s", r.URL.Path)
  span, _ := tracer.StartSpanFromContext(r.Context(), r.URL.Path)
  defer span.Finish()
  database(r)
  serviceb(r)
```

```
  res := strings.Repeat("o", rand.Intn(100)+1)
  time.Sleep(time.Duration(rand.Intn(100)+1) * time.Millisecond)
  w.Write([]byte("Hello, w" + res + "rld!"))
}

func database(r *http.Request) {
  span, _ := tracer.StartSpanFromContext(r.Context(), "database")
  defer span.Finish()
  cache(r)
  x := rand.Intn(4) + 100
  time.Sleep(time.Duration(x) * time.Millisecond)
  span.Tag("sleep-time", fmt.Sprintf("database-cost:%d", x))
}

func cache(r *http.Request) {
  span, _ := tracer.StartSpanFromContext(r.Context(), "cache")
  defer span.Finish()
  x := rand.Intn(4) + 100
  time.Sleep(time.Duration(x) * time.Millisecond)
  span.Annotate(time.Now(), fmt.Sprintf("cost:%d", x))
}

func serviceb(r *http.Request) {
  span, pc := tracer.StartSpanFromContext(r.Context(), "serviceb")
  defer span.Finish()
  time.Sleep(time.Duration(rand.Intn(100)+100) * time.Millisecond)
  servicec(pc) // servicec is childof serviceb
  span.Annotate(time.Now(), "endtime")
}

//func servicec(r *http.Request) {
func servicec(c context.Context) {
  span, _ := tracer.StartSpanFromContext(c, "servicec")
  defer span.Finish()
  time.Sleep(time.Duration(rand.Intn(700)+100) * time.Millisecond)
  span.Tag("servicec", "C") // set tags for search servicec
```

```
}
```

- 服务端追踪展示效果

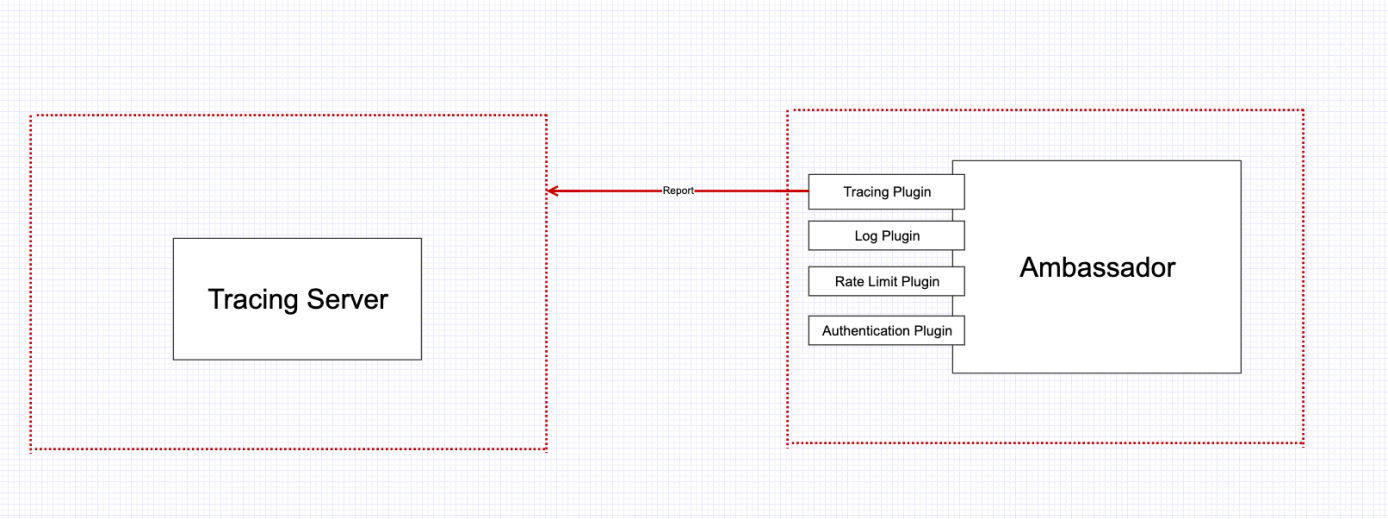


- zipkin gin 的使用

```
//use gin.Engine as a http.Handler with middleware
// init tracer omit
middler := zipkinmw.NewServerMiddleware(
    tracer,
    zipkinmw.SpanName(spanName),
    zipkinmw.TagResponseSize(true),
  )
  rgin := gin.Default()
  h := middler(rgin)
  port := ":8080"
  log.Fatal(http.ListenAndServe(port, h))
```

Tracing在Gateway中集成

- 以插件的形式 提供给Gateway
- 在gateway里提供tracing服务的地址，Gateway以tracing server client形式存在



参考

- Dapper paper from google
- Zipkin
- Ambassador

附录

- zipkin server的搭建 java8以上

```
curl -sSL https://zipkin.io/quickstart.sh | bash -s
java -jar zipkin.jar
// open default url localhost:9411/zipkin
```

- http server端代码 参见这里
- http client 端代码 参见这里