【GO-Micro】jaeger 分布式链路追踪

作者：Allenxuxu
原文链接：https://ld246.com/article/1561380026174
来源网站：链滴

github完整代码地址

安装jaeger

jaeger提供一个all in one 的docker镜像，可以快速搭建实验环境

```
docker run -d --name jaeger
-e COLLECTOR_ZIPKIN_HTTP_PORT=9411
-p 5775:5775/udp
-p 6831:6831/udp
-p 6832:6832/udp
-p 5778:5778
-p 16686:16686
-p 14268:14268
-p 9411:9411
jaegertracing/all-in-one:1.6
```

OpenTracing

OpenTracing通过提供平台无关、厂商无关的API，使得开发人员能够方便的添加（或更换）追踪系的实现。 OpenTracing提供了用于运营支撑系统的和针对特定平台的辅助程序库。

jaeger兼容OpenTracing API，所以我们使用OpenTracing的程序库可以方便的替换追踪工具。

OpenTracing中文文档

jaeger使用

封住一下jaeger的初始化操作方便使用，详细用法可以查看 jaeger-client-go

```
// lib/tracer

// NewTracer 创建一个jaeger Tracer
func NewTracer(servicename string, addr string) (opentracing.Tracer, io.Closer, error) {
    cfg := jaegercfg.Configuration{
        ServiceName: servicename,
        Sampler: &jaegercfg.SamplerConfig{
            Type:  jaeger.SamplerTypeConst,
            Param: 1,
        },
        Reporter: &jaegercfg.ReporterConfig{
            LogSpans:            true,
            BufferFlushInterval: 1 * time.Second,
        },
    }

    sender, err := jaeger.NewUDPTransport(addr, 0)
    if err != nil {
        return nil, nil, err
    }

    reporter := jaeger.NewRemoteReporter(sender)
    // Initialize tracer with a logger and a metrics factory
```

```
    tracer, closer, err := cfg.NewTracer(
        jaegercfg.Reporter(reporter),
    )

    return tracer, closer, err
}

func main() {
t, io, err := tracer.NewTracer("tracer", "")
    if err != nil {
        log.Fatal(err)
    }
    defer io.Close()
    opentracing.SetGlobalTracer(t)
}
```

opentracing.SetGlobalTracer(t) 方法执行会将jaeger tracer注册到全局，接下来只需要使用opentraing 的标准API便可以了。

如果不想使用jaeger了，想替换成其他分布式追踪工具，只需要工具支持opentracing标准，并将mai函数的SetGlobalTracer操作替换即可，其他文件都不需要更改。

micro链路追踪插件

micro自带的opentracing插件

在micro自带的插件中已经有opentracing的插件了，包含server，client等，不过这个插件只能go-mcro构建的微服务（api，srv）中使用。因为micro网关有一个独立的插件系统，但是并没有提供opentacing相关的插件。

micro/go-plugins/wrapper/trace/opentracing/opentracing.go

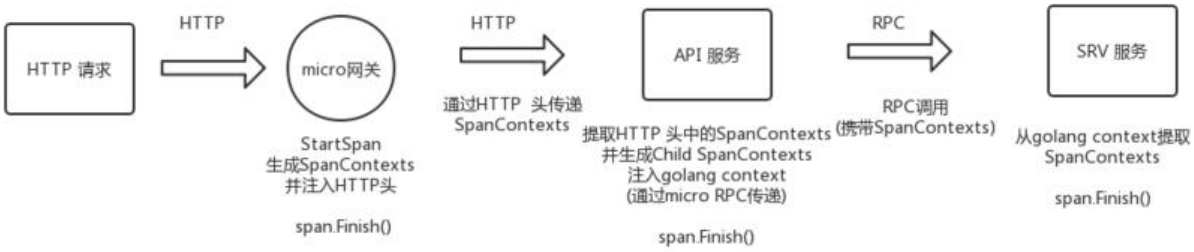
我们可以在构建服务的时候直接使用，只需要在服务初始化时增加一行函数就可以了。

```
service := micro.NewService(
        micro.Name(name),
        micro.Version("latest"),
        micro.WrapHandler(ocplugin.NewHandlerWrapper(opentracing.GlobalTracer())),
    )
```

srv/user/main.go 目录下的user 服务是一个完整的使用实例。

为micro网关增加opentracing插件

实现原理



外部HTTP请求首先经过API网关，网关生成第一个SpanContexts并且通过HTTP头传递到聚合层的AP服务，这边需要我们实现一个插件去做这件事，原理很简单，拦截每一次请求添加信息就可以了。

查看micro自带的opentracing插件，可以发现是通过golang的context传递，micro的RPC已经封装了通过context在跨进程服务间传递SpanContexts机制，所以我们需要在API服务层实现一个插件，HTTP头中取出SpanContexts并按照micro自带的方式注入golang context。

```
// micro opentracing插件中wHandlerWrappe
// NewHandlerWrapper accepts an opentracing Tracer and returns a Handler Wrapper
func NewHandlerWrapper(ot opentracing.Tracer) server.HandlerWrapper {
    return func(h server.HandlerFunc) server.HandlerFunc {
        return func(ctx context.Context, req server.Request, rsp interface{}) error {
            name := fmt.Sprintf("%s.%s", req.Service(), req.Endpoint())
            ctx, span, err := traceIntoContext(ctx, ot, name)
            if err != nil {
                return err
            }
            defer span.Finish()
            return h(ctx, req, rsp)
        }
    }
}
```

micro API网关插件

lib/wrapper/tracer/opentracing/stdhttp/stdhttp.go

和实现JWT鉴权插件一样，实现一个HTTP中间件通过mciro的插件机制全局注册就可以实现拦截每次求并处理。

```
// TracerWrapper tracer wrapper
func TracerWrapper(h http.Handler) http.Handler {
    return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        spanCtx, _ := opentracing.GlobalTracer().Extract(opentracing.HTTPHeaders, opentracing.
TTPHeadersCarrier(r.Header))
        sp := opentracing.GlobalTracer().StartSpan(r.URL.Path, opentracing.ChildOf(spanCtx))
        defer sp.Finish()

        if err := opentracing.GlobalTracer().Inject(
            sp.Context(),
            opentracing.HTTPHeaders,
            opentracing.HTTPHeadersCarrier(r.Header)); err != nil {
            log.Println(err)
```

```
        }

        sct := &status_code.StatusCodeTracker{ResponseWriter: w, Status: http.StatusOK}
        h.ServeHTTP(sct.WrappedResponseWriter(), r)

        ext.HTTPMethod.Set(sp, r.Method)
        ext.HTTPUrl.Set(sp, r.URL.EscapedPath())
        ext.HTTPStatusCode.Set(sp, uint16(sct.Status))
        if sct.Status >= http.StatusInternalServerError {
            ext.Error.Set(sp, true)
        }
    })
}
```

Tracer相关的概念可以查看这个文档

1. opentracing.GlobalTracer().Extract 方法提取HTTP头中的spanContexts
2. opentracing.ChildOf 方法基于提取出来的spanContexts生成新的child spanContexts
3. opentracing.GlobalTracer().StartSpan 方法生成一个新的span
4. github.com/opentracing/opentracing-go/ext 通过ext可以为追踪添加一些tag来展示更多信息比如URL，请求类型(GET，POST...), 返回码
5. sp.Finish() 结束这一个span

API服务（使用gin）插件

lib/wrapper/tracer/opentracing/gin2micro/gin2micro.go

```
// TracerWrapper tracer 中间件
func TracerWrapper(c *gin.Context) {
    md := make(map[string]string)
    spanCtx, _ := opentracing.GlobalTracer().Extract(opentracing.HTTPHeaders, opentracing.HT
PHeadersCarrier(c.Request.Header))
    sp := opentracing.GlobalTracer().StartSpan(c.Request.URL.Path, opentracing.ChildOf(spanCt
))
    defer sp.Finish()

    if err := opentracing.GlobalTracer().Inject(sp.Context(),
        opentracing.TextMap,
        opentracing.TextMapCarrier(md)); err != nil {
        log.Log(err)
    }

    ctx := context.TODO()
    ctx = opentracing.ContextWithSpan(ctx, sp)
    ctx = metadata.NewContext(ctx, md)
    c.Set(contextTracerKey, ctx)

    c.Next()

    statusCode := c.Writer.Status()
    ext.HTTPStatusCode.Set(sp, uint16(statusCode))
    ext.HTTPMethod.Set(sp, c.Request.Method)
```

```
    ext.HTTPUrl.Set(sp, c.Request.URL.EscapedPath())
    if statusCode >= http.StatusInternalServerError {
        ext.Error.Set(sp, true)
    }
}

// ContextWithSpan 返回context
func ContextWithSpan(c *gin.Context) (ctx context.Context, ok bool) {
    v, exist := c.Get(contextTracerKey)
    if exist == false {
        ok = false
        return
    }

    ctx, ok = v.(context.Context)
    return
}
```

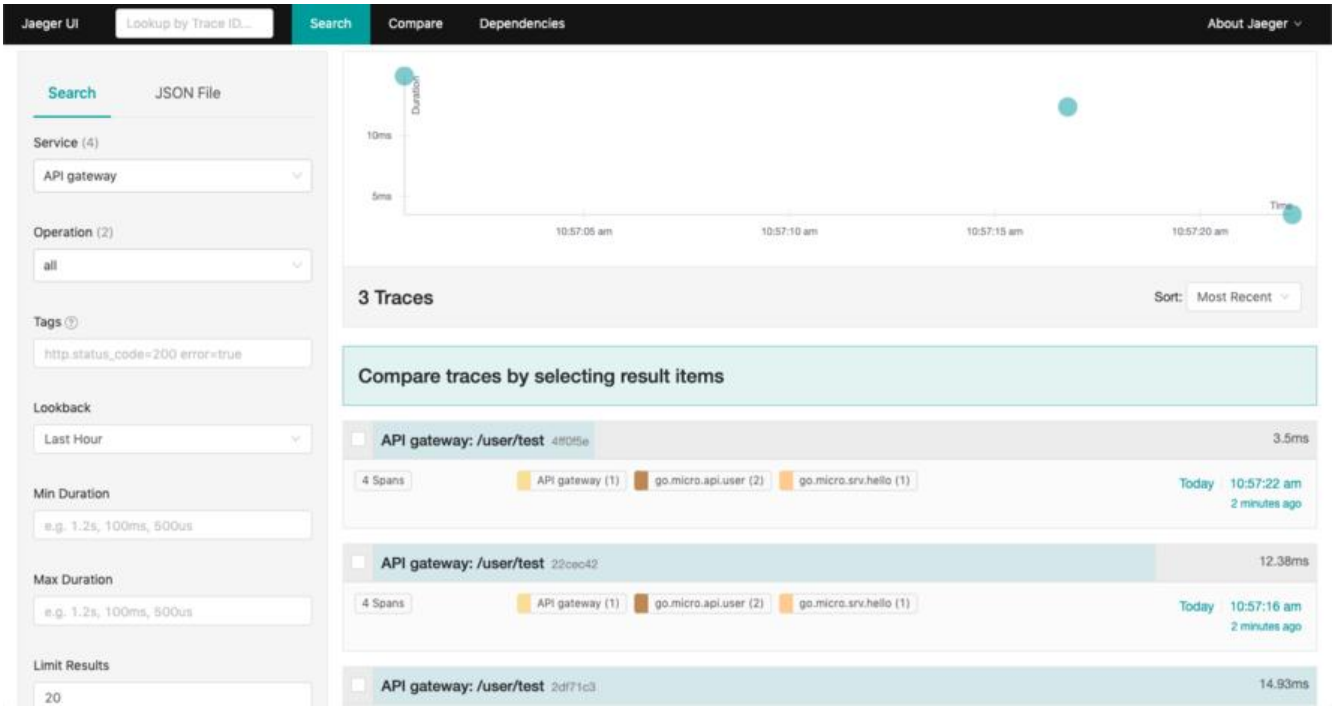
基本操作流程和给micro编写的插件相同，但是有两点不同。其一，因为我使用gin开发API服务，所基于gin的API。其二，因为micro内部提供通过golang context传递spanContexts的机制，所以将边会将child spanContexts注入到gin 的context，在API服务通过micro提供RPC接口(生成的XX.micr.go文件中调用函数第一个参数都是context)调用其他服务时传入提取的context，如下：

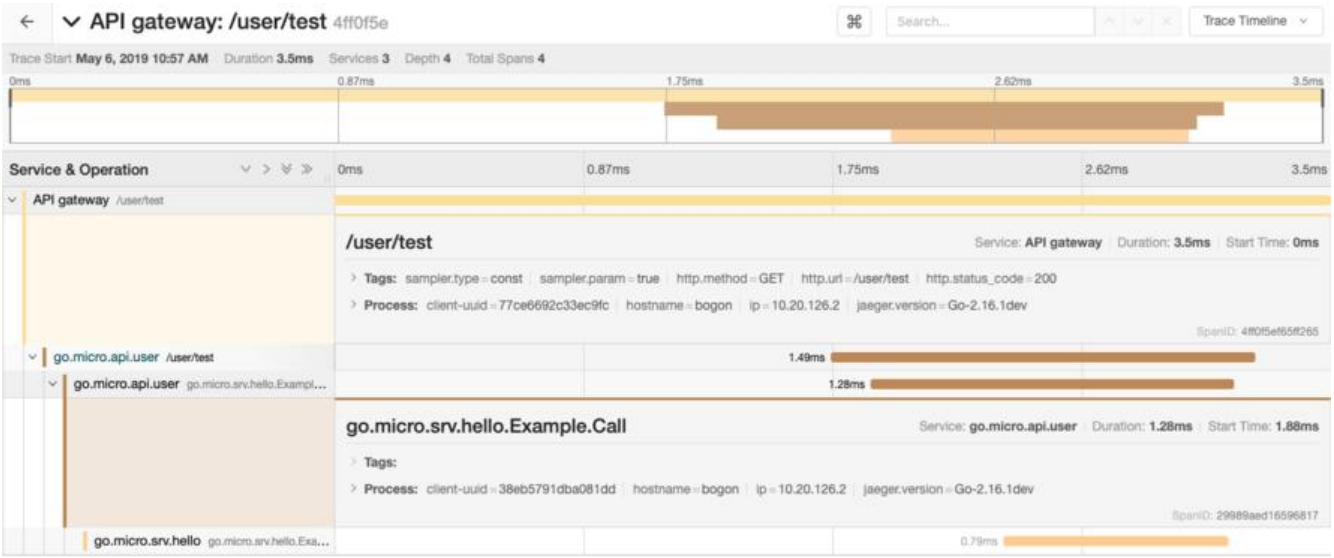
```
...
ctx, ok := gin2micro.ContextWithSpan(c)
    if ok == false {
        log.Log("get context err")
    }

    res, err := s.helloC.Call(ctx, &helloS.Request{Name: "xuxu"})
...
```

完整的实现细节可以查看，github仓库中 lib/wrapper/tracer/opentracing， 这里。

完整体验





原文链接：【GO-Micro】jaeger 分布式链路追踪