

分布式追踪中间件开发

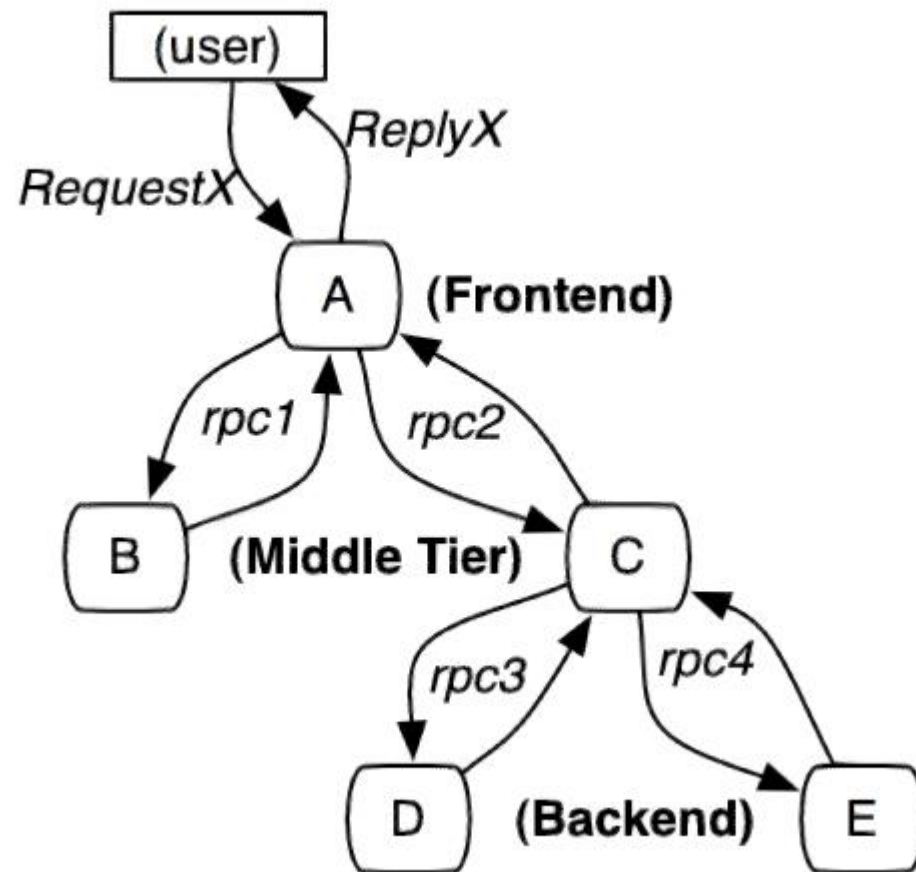
联系QQ: [2816010068](#), 加入会员群

目录

- 背景
- 思路
- Opentrace实战
- 分布式中间件开发

背景

- 调用树示例

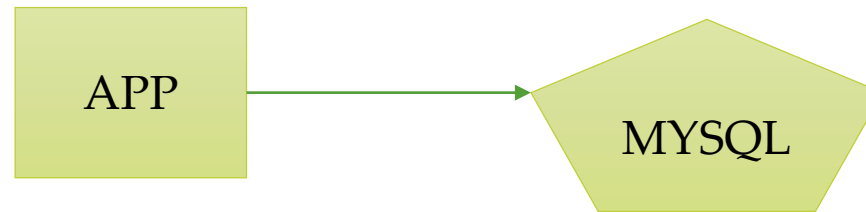


背景

- 面临的问题或痛点
 - 如果user服务某个请求特别慢, 如何去定位?
 - 如果user服务某个请求频繁报错了, 如何定位?
 - 1年以后, user服务的调用依赖图, 还有人能够快速的给出吗?

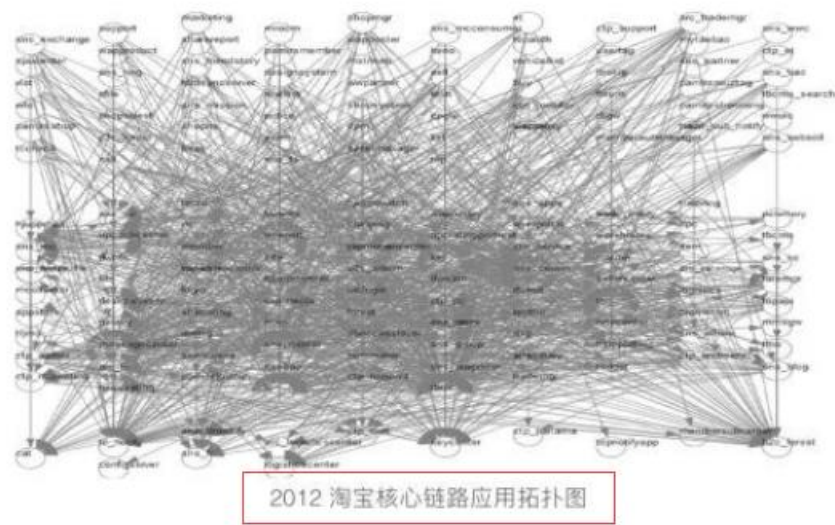
背景

- 单体应用
 - 绝大多数可以通过日志进行定位



微服务架构

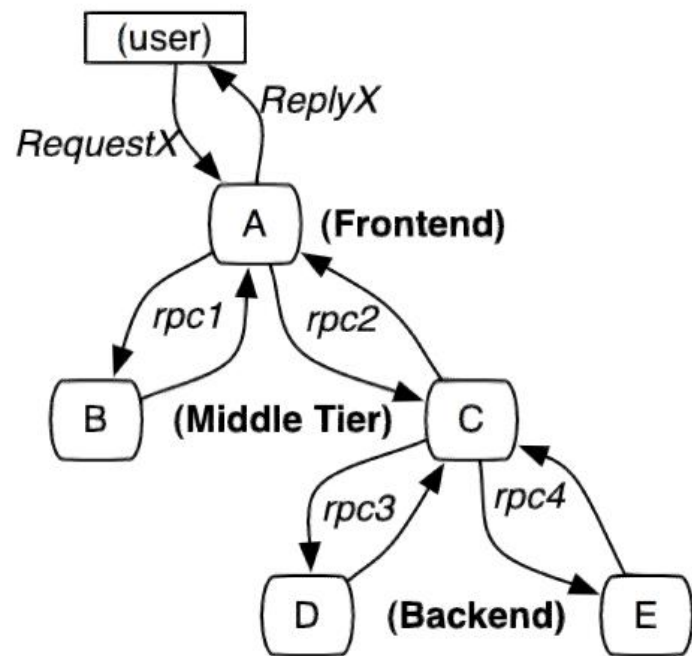
- 故障定位难
- 容量预估难
- 资源浪费多
- 链路梳理难



故障定位难
容量预估难
资源浪费多
链路梳理难

分布式追踪如何解决这个问题？

- trace_id概念
 - 为每个请求分配唯一的id, 通常叫做trace_id
 - 日志聚合

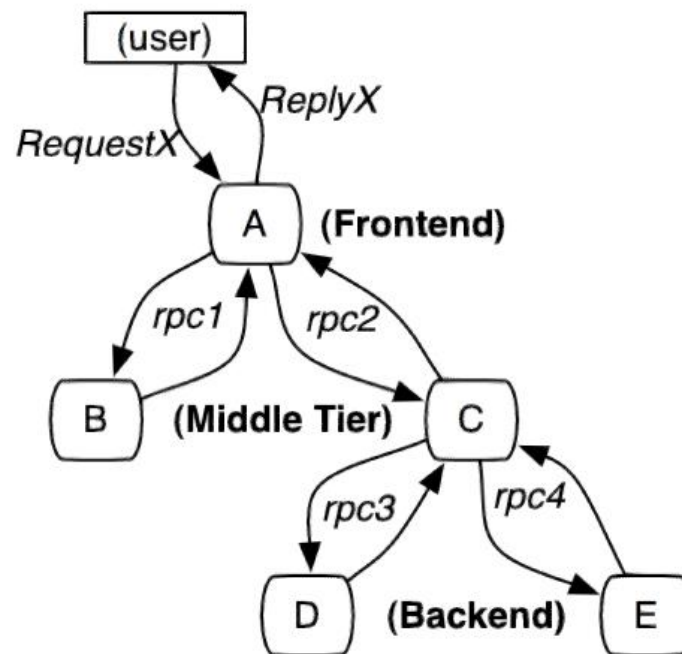


分布式追踪如何解决这个问题？

- Span概念
 - 如何知道每个子系统的详细的处理细节？
 - 通过span进行抽象

整个请求耗时3s

user request span



分布式追踪如何解决这个问题？

- Span概念
 - 如何知道每个子系统的详细的处理细节？
 - 通过span进行抽象，Span之间有父子的关系

整个请求耗时3s

user request span 3s

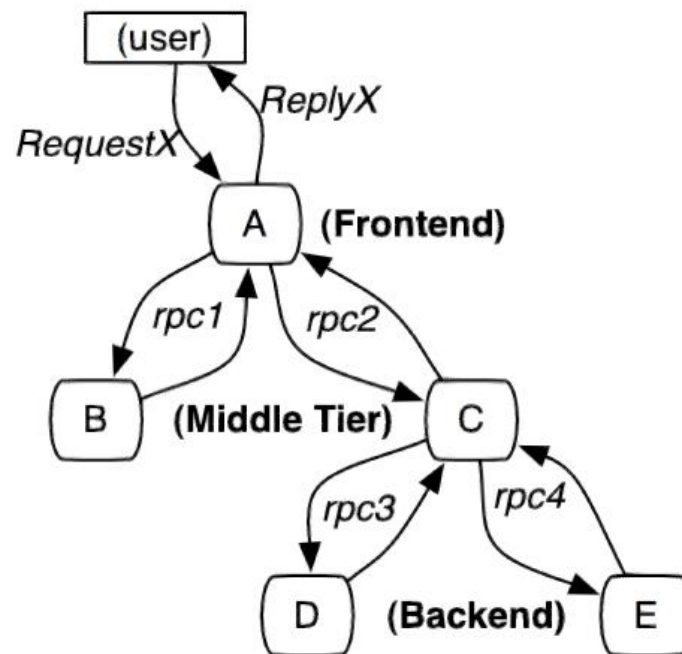
A request span 2.8s


B request span 0.8s


C request span 2s

C 0.5s

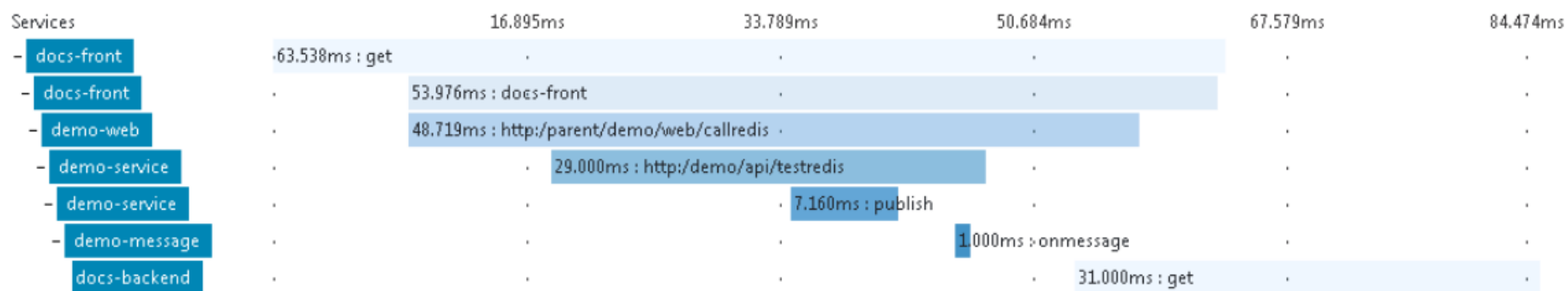
D 1.5s



Duration: 84.474ms Services: 5 Depth: 7 Total Spans: 7 [JSON](#) 

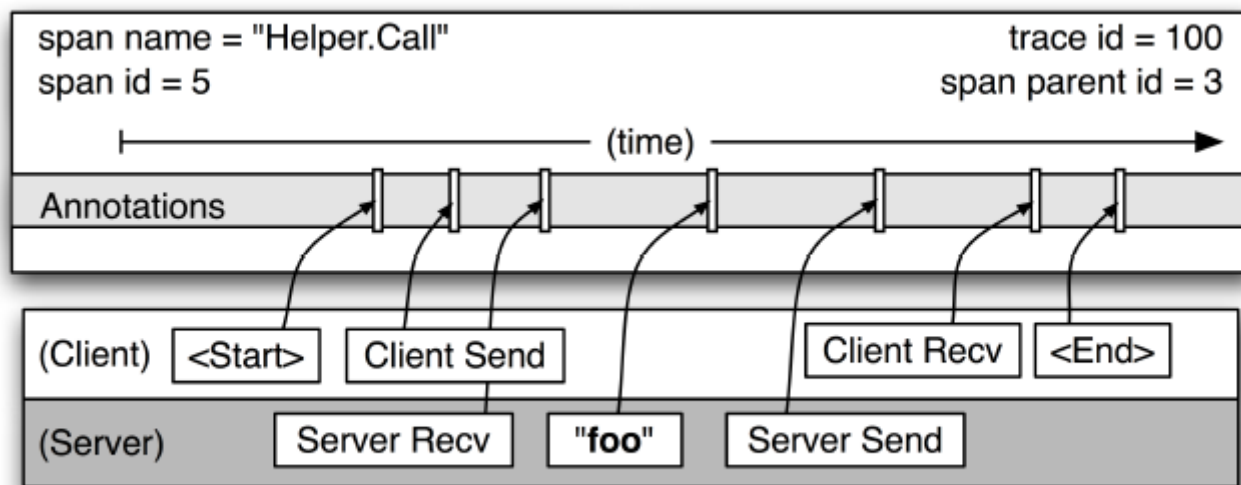
[Expand All](#) [Collapse All](#) 

[demo-message x2](#) [demo-service x2](#) [demo-web x2](#) [docs-backend x1](#) [docs-front x3](#)



分布式追踪如何解决这个问题？

- 单个span详细解剖



分布式追踪如何解决这个问题？

- Span Context概念
 - 传播问题
 - 进程内传播
 - 进程之间传播
 - http协议
 - 通过http头部进行透明传播
 - Tcp协议
 - Thrift
 - 需要改造thrift进行支持

开源实现

- Google
 - Dapper论文
 - <https://bigbully.github.io/Dapper-translation/>
- Uber
 - [Jaeger已经开源](#)
 - <https://www.jaegertracing.io/>
- Twitter
 - Zipkin
 - <https://github.com/openzipkin/zipkin>
- Opentracing, 标准化组织
 - Opentracing
 - <https://github.com/opentracing/opentracing-go>

Span context其他用途

- 测试标签

分布式trace实战

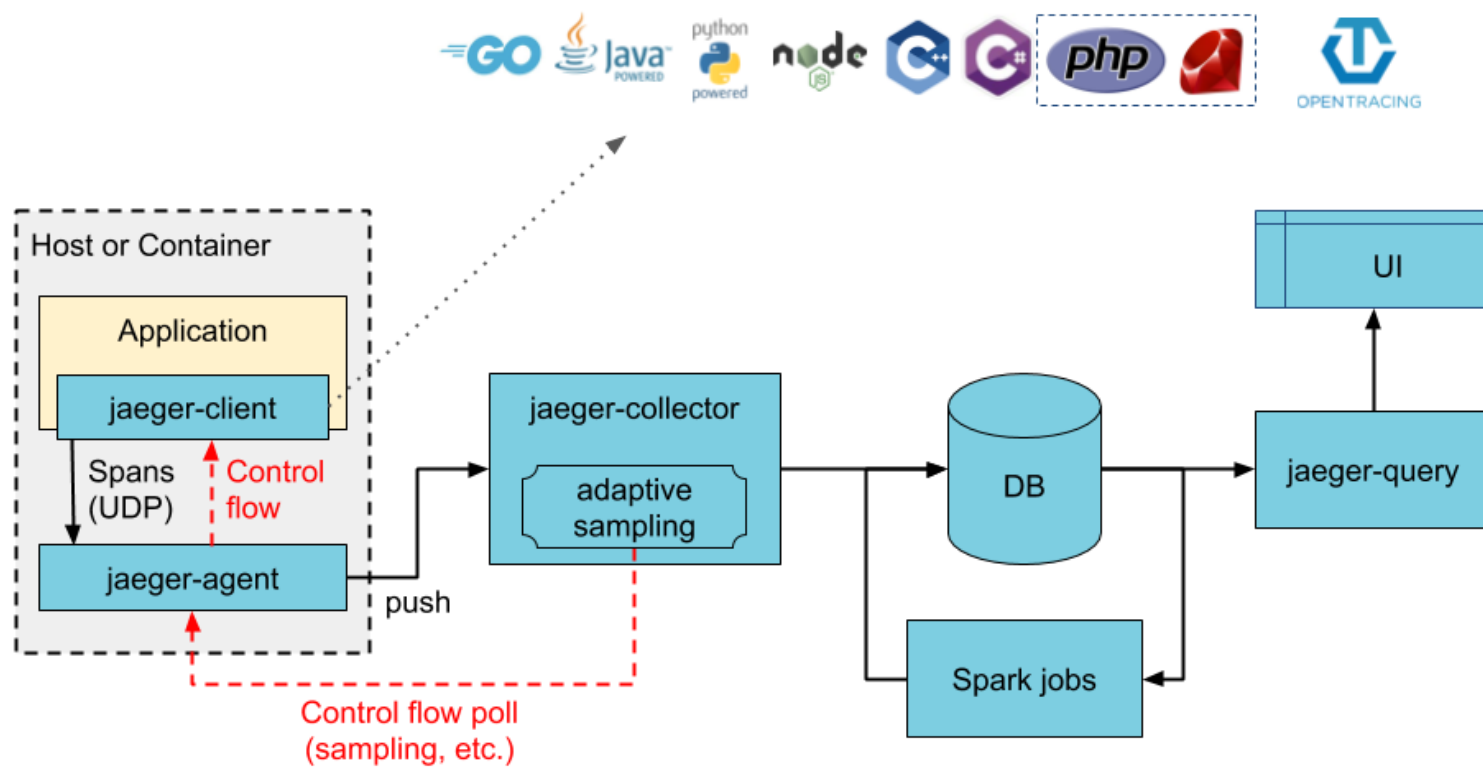
技术选型

- 使用opentracing提供的通用接口
- 底层jagger作为分布式系统

环境准备

- Jagger部署
 - 生产环境:
 - <https://juejin.im/post/5cc6c23ae51d456e660d4542>
 - 开发环境: 使用docker镜像
 - Docker安装: <https://qizhanming.com/blog/2019/01/25/how-to-install-docker-ce-on-centos-7>
 - 测试环境: <http://60.205.218.189:9411/api/v1/spans>

Jeaggar架构



实战一

- Hello world
 - 代码: `koala/example/trace_example/hello`
 - 演示jeagger的基本使用

实战二

- 追踪函数
 - 追踪指定函数的执行情况
 - span和context进行结合，并在进程内进行传播
 - 代码： `koala/example/trace_example/function`

实战三

- 追踪网络调用
 - span在网络请求中进行传递

```
ext.SpanKindRPCClient.Set(span)
ext.HTTPUrl.Set(span, url)
ext.HTTPMethod.Set(span, "GET")
span.Tracer().Inject(
    span.Context(),
    opentracing.HTTPHeaders,
    opentracing.HTTPHeadersCarrier(req.Header),
)
```

```
http.HandleFunc("/format", func(w http.ResponseWriter, r *http.Request) {
    spanCtx, _ := tracer.Extract(opentracing.HTTPHeaders, opentracing.HTTPHeadersCarrier(r.Header))
    span := tracer.StartSpan("format", ext.RPCServerOption(spanCtx))
    defer span.Finish()
    // 从http头部提取出传输的span

    helloTo := r.FormValue("helloTo")
    helloStr := fmt.Sprintf("Hello, %s!", helloTo)
    span.LogFields(
        otlog.String("event", "string-format"),
        otlog.String("value", helloStr)
    )
})
```

koala分布式中间件开发

- 配置项开发
- 中间件开发