《31 讲带你搞懂 SkyWalking》

徐郡明　前搜狗资深技术专家、源码剖析系列畅销书作者

第05讲：OpenTracing 简介
先有标准后有天

OpenTracing 简介，先有标准后有天

拉勾教育

Google Dapper 的论文发布

各个分布式链路追踪产品的 API 并不兼容，如果用户在各个产品之间进行切换，成本非常高

OpenTracing 通过提供平台无关、厂商无关的 API

帮助开发人员能够方便地添加（或更换）追踪系统

Trace 简介

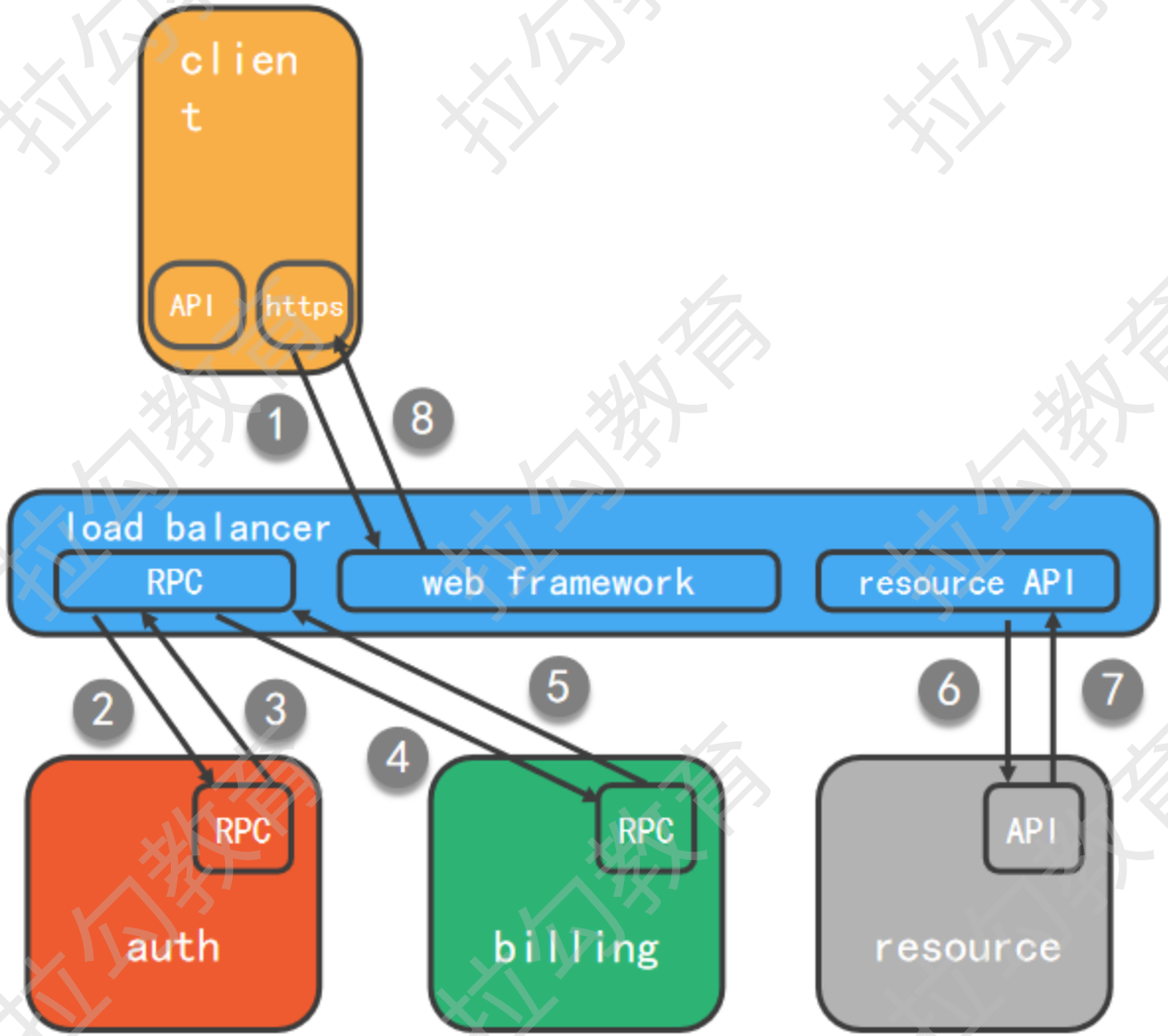
一个 Trace 代表一个事务、请求或是流程在分布式系统中的执行过程

OpenTracing 中的一条 Trace 被认为是一个由多个 Span 组成的有向无环图（ DAG 图）

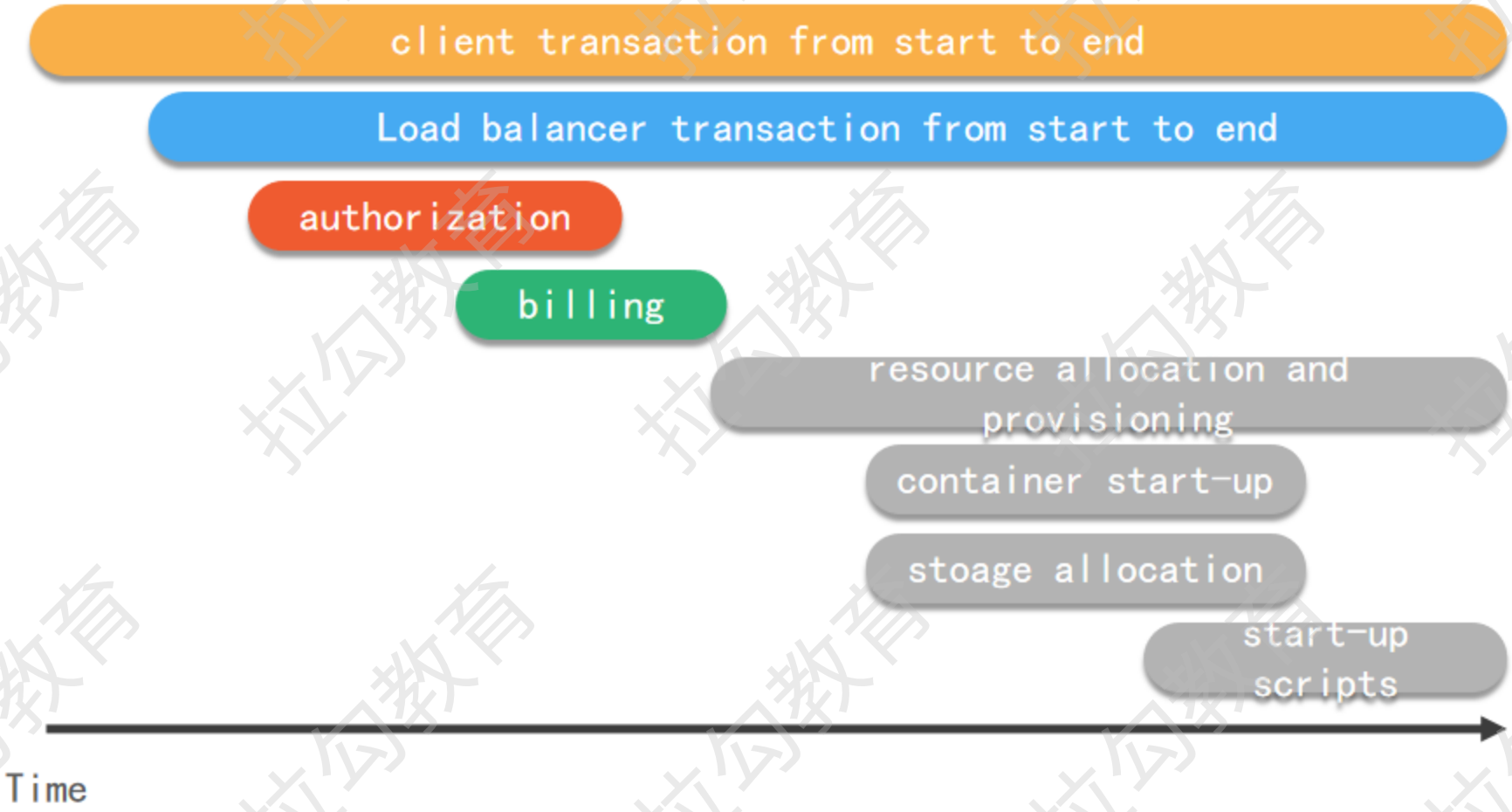
一个 Span 代表系统中具有开始时间和执行时长的逻辑单元

一条 Trace 中 Span 是首尾连接的

Trace 简介



Trace 简介



Span 简介

Span 代表系统中具有开始时间和执行时长的逻辑单元

Span 之间通过嵌套或者顺序排列建立逻辑因果关系

每个 Span 中可以包含以下的信息：

- 操作名称：例如访问的具体 RPC 服务，访问的 URL 地址等
- 起始时间
- 结束时间
- Span Tag：一组键值对构成的 Span 标签集合

  其中键必须为字符串类型，值可以是字符串、bo
- Span Log：一组 Span 的日志集合
- SpanContext：Trace 的全局上下文信息

Span 简介

- References：Span 之间的引用关系

在一个 Trace 中，一个 Span 可以和一个或者多个 Span 间存在因果关系

OpenTracing 定义了 ChildOf 和 FollowsFrom 两种 Span 之间的引用关系

这两种引用类型代表了子节点和父节点间的直接

Span 简介

- ChildOf 关系：一个 Span 可能是一个父级 Span 的孩子，即为 ChildOf 关系

一个 HTTP 请求中，被调用的服务端产生的 Span与发起调用的客户端产生的 Span，构成了 ChildOf 关系

一个 SQL Insert 操作的 Span，和 ORM 的 save 方法的 Span 构成 ChildOf 关系

ChildOf 关系中的父级 Span 要等待子 Span 的返回

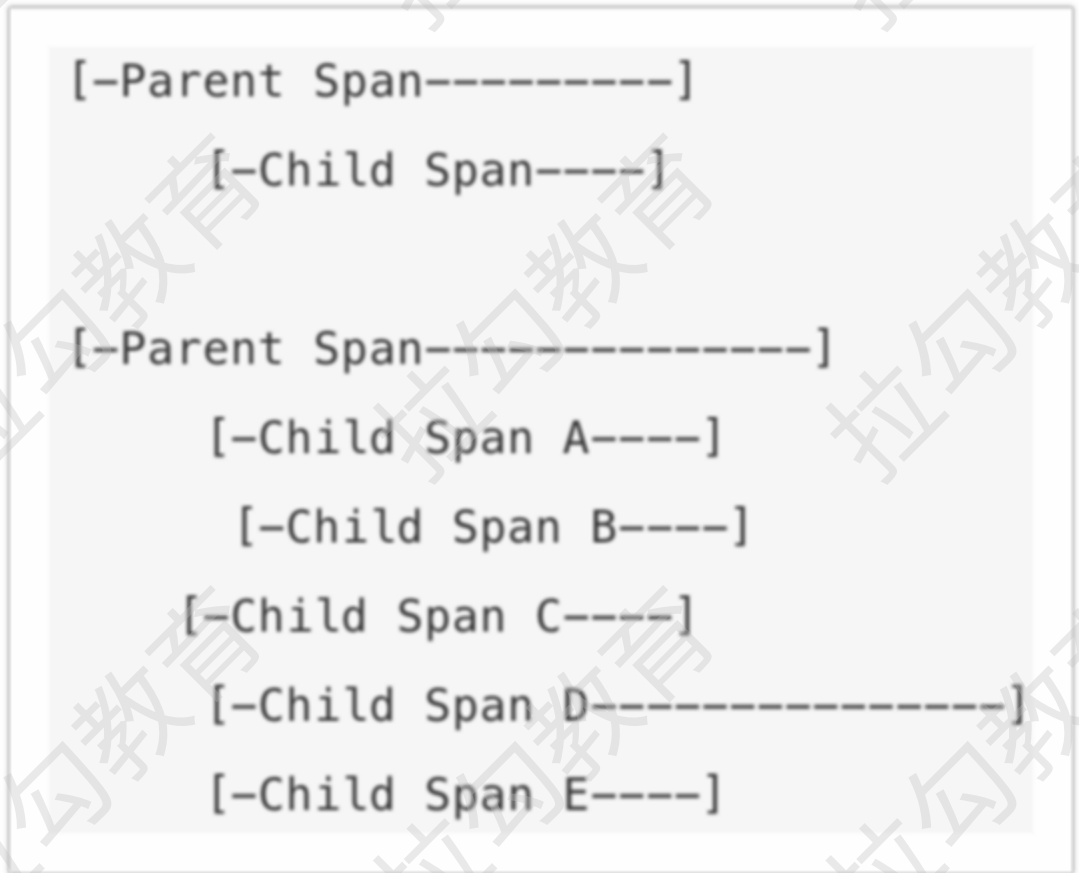
子 Span 的执行时间影响了其所在父级 Span 的执行时间，父级 Span 依赖子 Span 的执行结果

逻辑中很多并行的任务对应的 Span 也是并行的

一个父级 Span 可以合并所有子 Span 的执行结果并等待所有并行子 Span 结束

Span 简介

两种 ChildOf 关系 的 Span



Span 简介

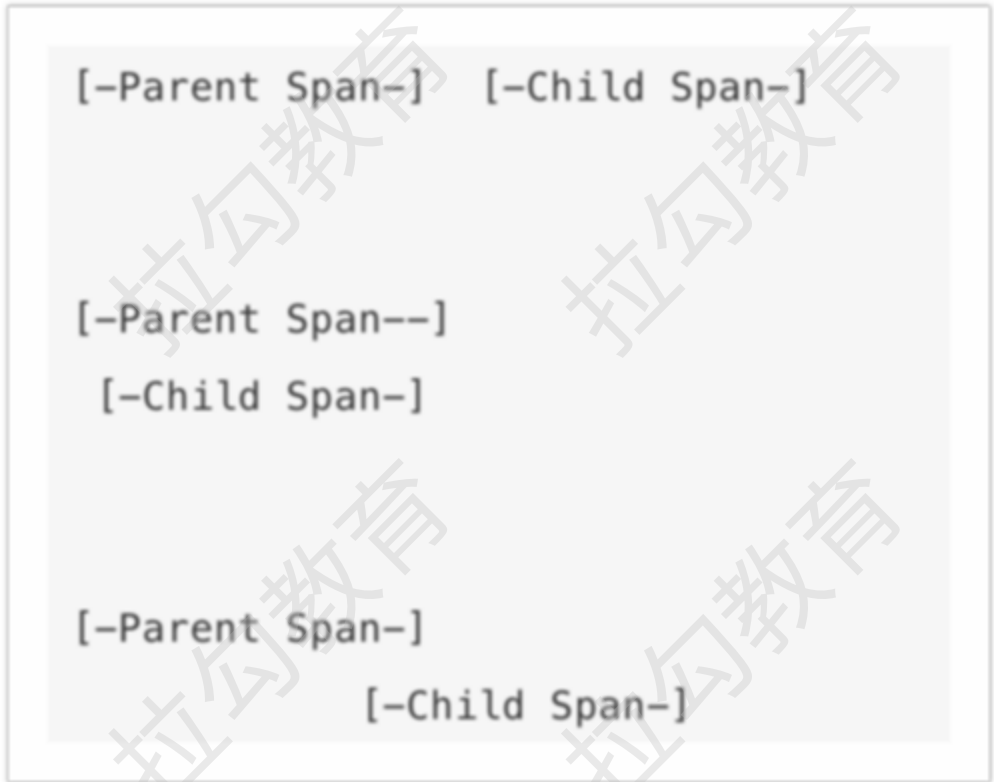
- FollowsFrom 关系：

在分布式系统中，一些上游系统（父节点）不以任何方式依赖下游系统（子节点）的执行结果

例如，上游系统通过消息队列向下游系统发送消息

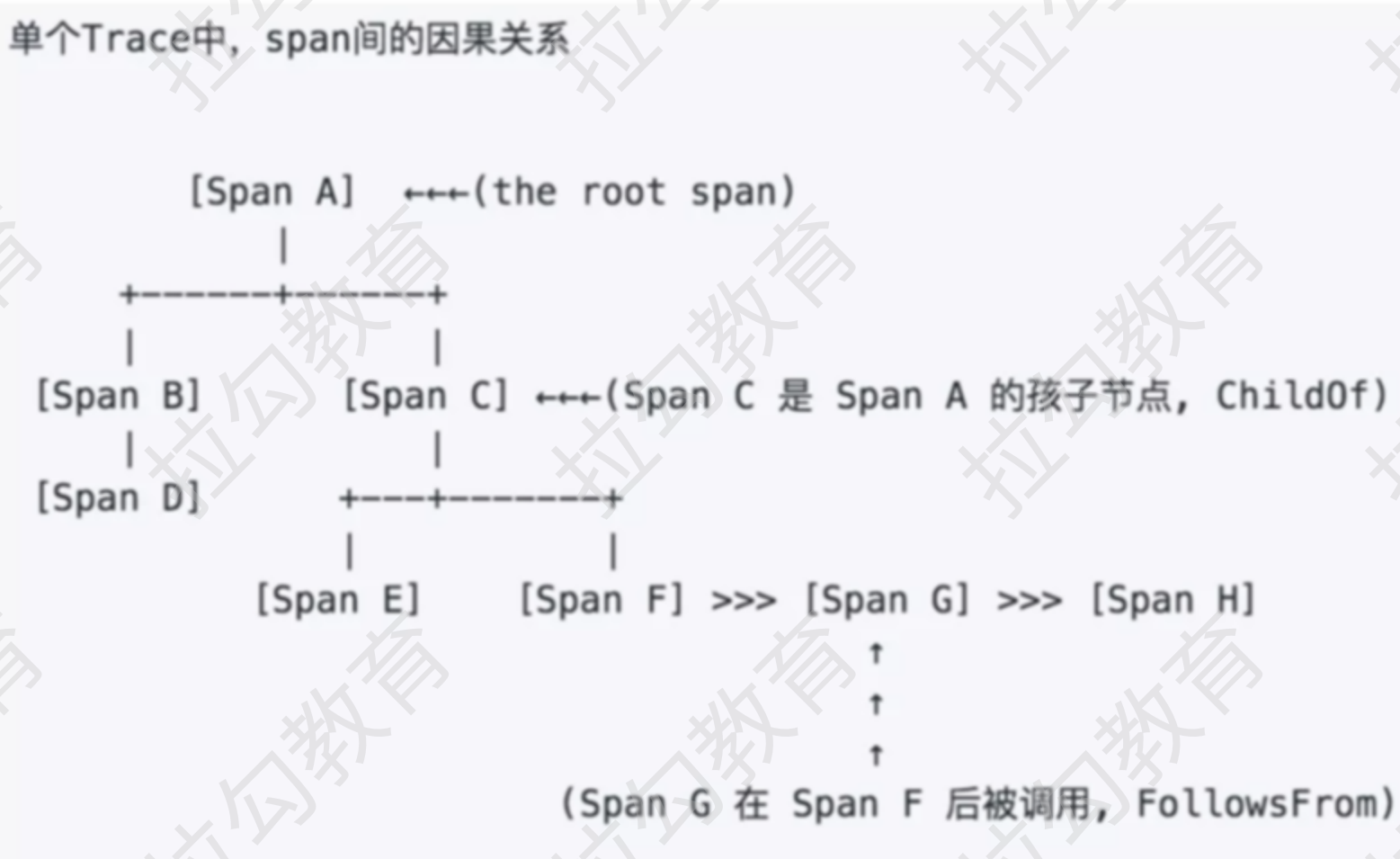
这种情况下，下游系统对应的子 Span 和

上游系统对应的父级 Span 之间是 FollowsFrom 关系



Span 简介

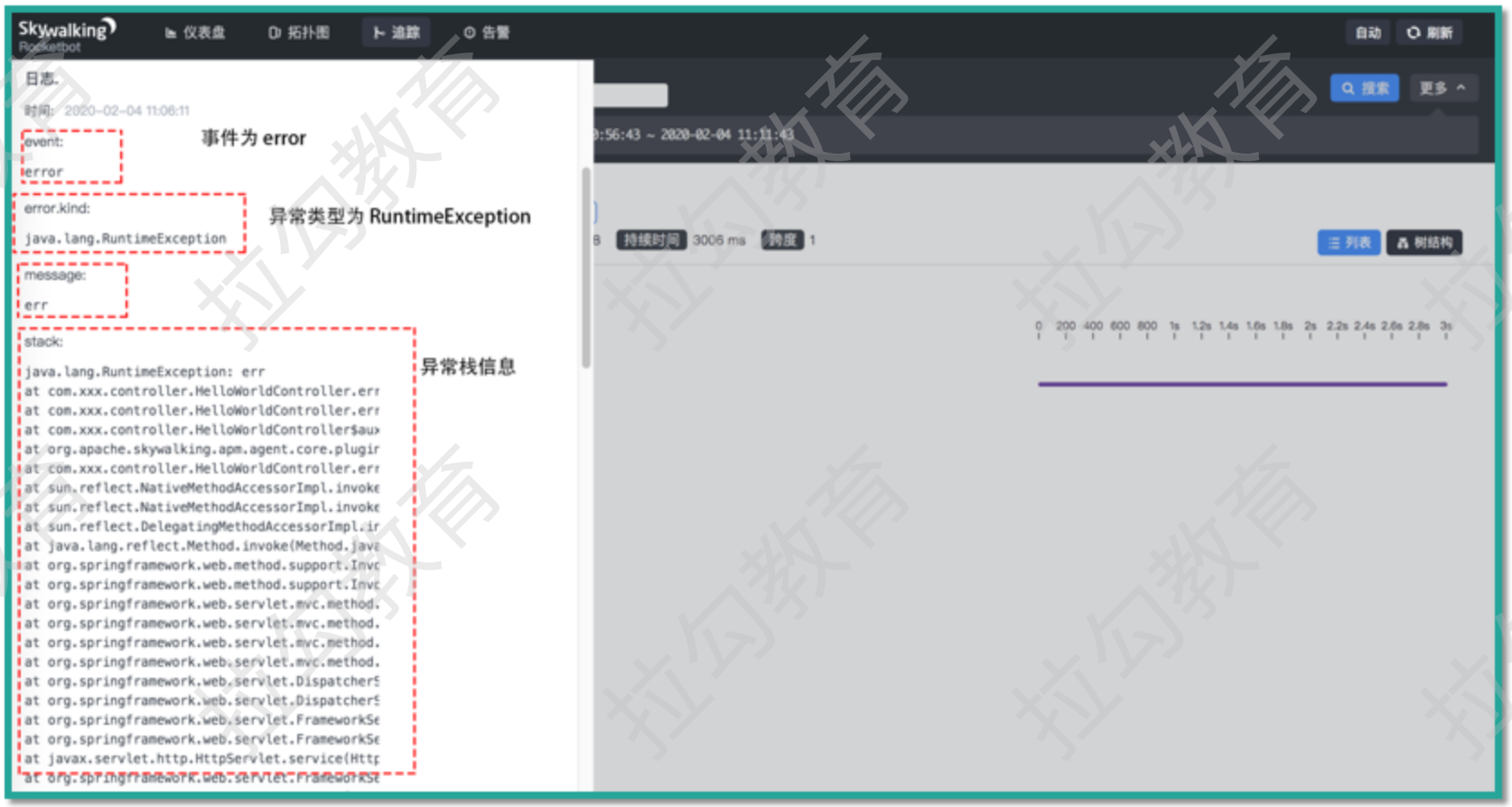
单个Trace中，span间的因果关系



Logs 简介

每个 Span 可以进行多次 Logs 操作，每一次 Logs 操作，都需要带一个时间戳，以及一个可选的附加信息

前文搭建的环境中，请求 http://localhost:8000/err 得到的 Trace 中就会通过 Logs 记录异常堆栈信息



Tags 简介

每个 Span 可以有多个键值对形式的 Tags

Tags 是没有时间戳的，只是为 Span 添加一些简单解释和补充信息

SpanContext 和 Baggage

SpanContext 表示进程边界，在跨进调用时需要将一些全局信息

例如，TraceId、当前 SpanId 等信息封装到 Baggage 中传递到另一个进程（下游系统）中

Baggage 是存储在 SpanContext 中的一个键值对集合

它会在一条 Trace 中全局传输，该 Trace 中的所有 Span 都可以获取到其中的信息

由于 Baggage 需要跨进程全局传输，会涉及相关数据的序列化和反序列化操作

如果在 Baggage 中存放过多的数据，就会导致序列化和反序列化操作耗时变长

使整个系统的 RPC 的延迟增加、吞吐量下降

SpanContext 和 Baggage

拉勾教育

Baggage 与 Span Tags 都是键值对集合

两者最大区别： Span Tags 中的信息不会跨进程传输，而 Baggage 需要全局传输

- OpenTracing 要求实现提供 Inject 和 Extract 两种操作
- SpanContext 通过 Inject 操作向 Baggage 中添加键值对数据

通过 Extract 从 Baggage 中获取键值

核心接口语义

OpenTracing 希望各个实现平台能够根据上述的核心概念来建模实现

OpenTracing 提供了核心接口的描述，帮助开发人员更好的实现 OpenTracing 规范

核心接口语义

Span 接口

Span接口必须实现以下的功能：

- 获取关联的 SpanContext：通过 Span 获取关联的 SpanContext 对象
- 关闭（Finish）Span：完成已经开始的 Span
- 添加 Span Tag：为 Span 添加 Tag 键值对
- 添加 Log：为 Span 增加一个 Log 事件
- 添加 Baggage Item：向 Baggage 中添加一组键值对
- 获取 Baggage Item：根据 Key 获取 Baggage 中的元素

核心接口语义

SpanContext 接口

SpanContext 接口必须实现以下功能：

用户可以通过 Span 实例或者 Tracer 的 Extract 能力获取 SpanContext 接口实例

- 遍历 Baggage 中全部的 KV

核心接口语义

Tracer 接口

Tracer 接口必须实现以下功能：

- 创建 Span：创建新的 Span
- 注入 SpanContext：主要是将跨进程调用携带的 Baggage 数据记录到当前 SpanContext 中
- 提取 SpanContext ：主要是将当前 SpanContext 中的全局信息提取出来

  封装成 Baggage 用于后续的跨进程调用

总结

- 介绍 Trace、Span、Logs、Tags、SpanContext 等 OpenTracing 规范的核心概念
- 介绍 OpenTracing 规范定义的核心接口的功能
- SkyWalking 作为 OpenTracing 的实现之一
- 了解 OpenTracing 的核心概念可以更好的帮助理解 SkyWalking 的实现

Next：第06讲《 SPI 技术》

拉勾教育

—互联网人实战大学—

关注拉勾「教育公众号」
获取更多课程信息

L / A / G / O / U