ALIBABA CLOUD

阿里云

链路追踪

产品简介

文档版本：20201222

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

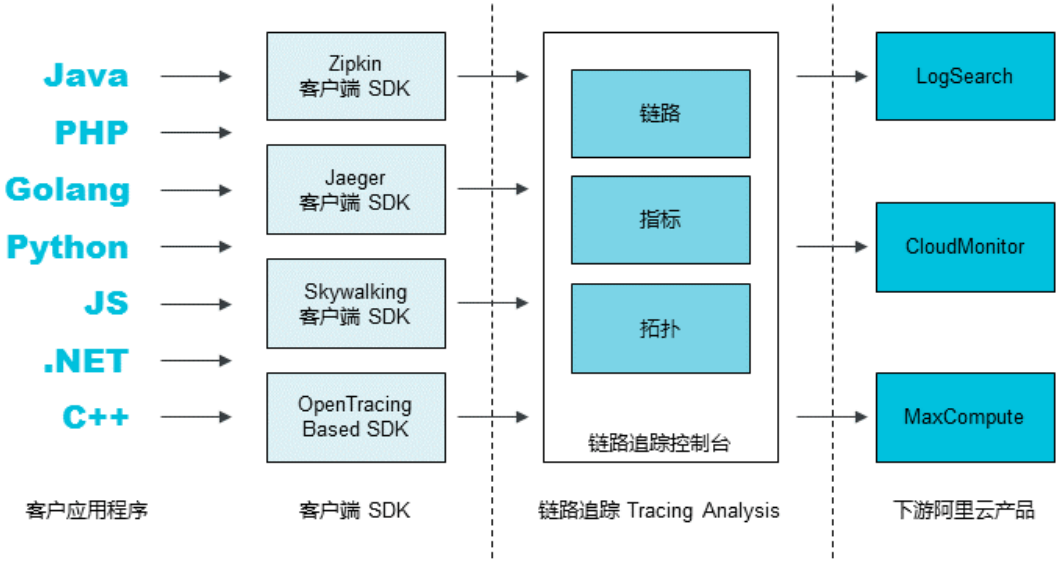
1.什么是链路追踪 Tracing Analysis

链路追踪 Tracing Analysis 为分布式应用的开发者提供了完整的调用链路还原、调用请求量统计、链路拓扑、应用依赖分析等工具，可以帮助开发者快速分析和诊断分布式应用架构下的性能瓶颈，提高微服务时代下的开发诊断效率。

产品架构

链路追踪的产品架构如图所示。

链路追踪产品架构



主要工作流程为：

- 客户侧的应用程序通过集成链路追踪的多语言客户端 SDK 上报服务调用数据。链路追踪支持多种开源社区的 SDK，且支持 OpenTracing 标准。
- 数据上报至链路追踪控制台后，链路追踪组件进行实时聚合计算和持久化，形成链路明细、性能总览、实时拓扑等监控数据。您可以据此进行问题排查与诊断。
- 调用链数据可对接下游阿里云产品，例如 LogSearch、CloudMonitor、MaxCompute 等，用于离线分析、报警等场景。

产品功能

链路追踪的主要功能有：

- 分布式调用链查询和诊断：追踪分布式架构中的所有微服务用户请求，并将它们汇总成分布式调用链。
- 应用性能实时汇总：通过追踪整个应用程序的用户请求，来实时汇总组成应用程序的单个服务和资源。
- 分布式拓扑动态发现：用户的所有分布式微服务应用和相关 PaaS 产品可以通过链路追踪收集到的分布式调用信息。
- 多语言开发程序接入：基于 OpenTracing 标准，全面兼容开源社区，例如 Jaeger 和 Zipkin。
- 丰富的下游对接场景：收集的链路可直接用于日志分析，且可对接到 MaxCompute 等下游分析平台。

2.基本概念

本文介绍在使用链路追踪之前需要了解的基本概念，包括分布式追踪系统的作用，什么是调用链，链路追踪所依赖的 OpenTracing 数据模型，以及在链路追踪产品里数据是如何上报的。

为什么需要分布式追踪系统？

为了应对各种复杂的业务，开发工程师开始采用敏捷开发、持续集成等开发方式。系统架构也从单机大型软件演化成微服务架构。微服务构建在不同的软件集上，这些软件模块可能是由不同团队开发的，可能使用不同的编程语言来实现，还可能发布在多台服务器上。因此，如果一个服务出现问题，可能导致几十个应用都出现服务异常。

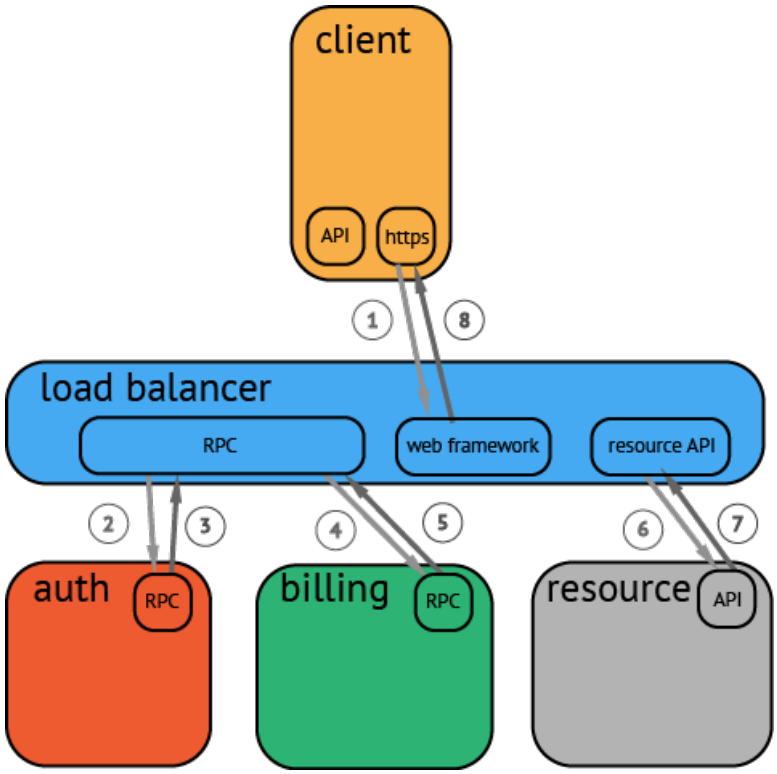
分布式追踪系统可以记录请求范围内的信息，例如一次远程方法调用的执行过程和耗时，是我们排查系统问题和系统性能的重要工具。

什么是调用链（Trace）？

在广义上，一个调用链代表一个事务或者流程在（分布式）系统中的执行过程。在 OpenTracing 标准中，调用链是多个 Span 组成的一个有向无环图（Directed Acyclic Graph，简称 DAG），每一个 Span 代表调用链中被命名并计时的连续性执行片段。

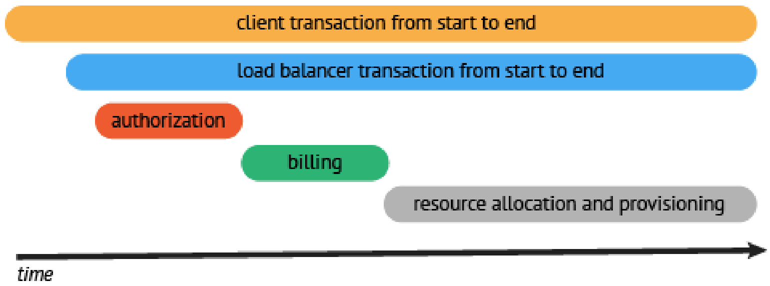
下图是一个分布式调用的例子：客户端发起请求，请求首先到达负载均衡器，接着经过认证服务、计费服务，然后请求资源，最后返回结果。

分布式调用示例



数据被采集存储后，分布式追踪系统一般会选择使用包含时间轴的时序图来呈现这个调用链。

包含时间轴的链路图



OpenTracing 数据模型

整体概念

OpenTracing 中的调用链（Trace）通过归属于此调用链的 Span 来隐性地定义。一条调用链可以视为一个由多个 Span 组成的有向无环图（DAG 图）。Span 之间的关系被命名为 References。例如下面的示例调用链就是由 8 个 Span 组成的。

```
单个 Trace 中 Span 间的因果关系
     [Span A]  ←←←(The root span)
         |
   +------+------+
  |        |
[Span B]    [Span C] ←←←(Span C 是 Span A 的子节点，ChildOf)
  |        |
[Span D]    +---+-------+
        |       |
      [Span E]   [Span F] >>> [Span G] >>> [Span H]
                     ↑
                     ↑
                     ↑
             (Span G 在 Span F 后被调用, FollowsFrom)
```

有些情况下，使用下面这种基于时间轴的时序图可以更好地展现调用链。

```
单个 Trace中 Span 间的时间关系
--|-------|-------|-------|-------|-------|-------|-------|-> time
[Span A · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · ·
· · · ]
 [Span B · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · ]
  [Span D · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · ]
 [Span C · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · ]
    [Span E · · · · · · · ]     [Span F · · ] [Span G · · ] [Span H · · ]
```

链路

Tracer 接口用于创建 Span（startSpan 函数）、解析上下文（Extract 函数）和透传上下文（Inject 函数）。它具有以下能力：

- 创建一个新 Span 或者设置 Span 属性

```
/** 创建和开始一个span，返回一个span，span中包括操作名称和设置的选项。
** 例如：
**  创建一个无parentSpan的Span：
**  sp := tracer.StartSpan("GetFeed")
**  创建一个有parentSpan的Span
**  sp := tracer.StartSpan("GetFeed",opentracing.ChildOf(parentSpan.Context()))
**/
StartSpan(operationName string, opts ...StartSpanOption) Span
```

  每个 Span 包含以下对象：

  - Operation name：操作名称 （也可以称作 Span name）。
  - Start timestamp：起始时间。
  - Finish timestamp：结束时间。
  - Span tag：一组键值对构成的 Span 标签集合。键值对中，键必须为 String，值可以是字符串、布尔或者数字类型。
  - Span log：一组 Span 的日志集合。每次 Log 操作包含一个键值对和一个时间戳。键值对中，键必须为 String，值可以是任意类型。
  - SpanContext：pan 上下文对象。每个 SpanContext 包含以下状态：
    - 要实现任何一个 OpenTracing，都需要依赖一个独特的 Span 去跨进程边界传输当前调用链的状态（例如：Trace 和 Span 的 ID）。
    - Baggage Items 是 Trace 的随行数据，是一个键值对集合，存在于 Trace 中，也需要跨进程边界传输。
  - References（Span 间关系）：相关的零个或者多个 Span（Span 间通过 SpanContext 建立这种关系）。

- 透传数据

  透传数据分为两步：

  i. 从请求中解析出 SpanContext。

```
// Inject() takes the `sm` SpanContext instance and injects it for
// propagation within `carrier`. The actual type of `carrier` depends on
// the value of `format`.
/** 根据format参数从请求（Carrier）中解析出SpanContext（包括traceId、spanId、baggage）。
** 例如：
**  carrier := opentracing.HTTPHeadersCarrier(httpReq.Header)
**  clientContext, err := tracer.Extract(opentracing.HTTPHeaders, carrier)
**/
Extract(format interface{}, carrier interface{}) (SpanContext, error)
```

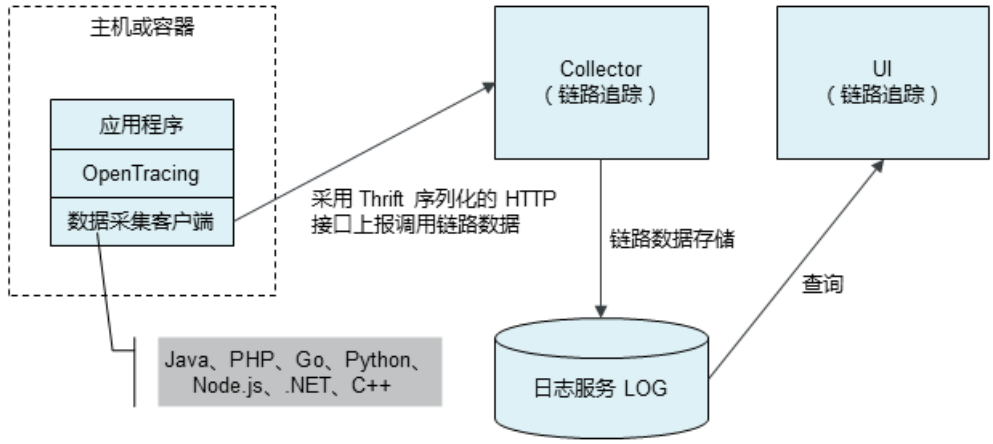
ii. 将 SpanContext 注入到请求中。

```
/**
** 将SpanContext中的traceId，spanId，Baggage等根据format参数注入到请求中（Carrier），
** e.g
** carrier := opentracing.HTTPHeadersCarrier(httpReq.Header)
** err := tracer.Inject(span.Context(), opentracing.HTTPHeaders, carrier)
**/
Inject(sm SpanContext, format interface{}, carrier interface{}) error
```

数据是如何上报的？

不通过 Agent 而直接上报数据的原理如下图所示。

直接上报数据



通过 Agent 上报数据的原理如下图所示。

通过 Agent 上报数据