Skywalking跨进程链路信息传递协议(一)

易企秀工程师 发布于 2020-03-18

前言

在开发skywalking nginx探针过程中重点分析了“Skywalking跨进程链路信息传递协议”和“Skywalking链路数据采集协议”。skywalking nginx探针的实现目前已完成一个版本并在生产环境上线，有兴趣的朋友可以交流。
本文基于Skywalking v5.0.0-GA分析，版本虽然有些落后，但两个协议的核心没有多大变化。两个协议的官方英文称呼及文档参考如下链接：

- Skywalking-Cross-Process-Propagation-Headers-Protocol-v1
- Trace Data Protocol

为了便于理解，本文将`Skywalking-Cross-Process-Propagation-Headers-Protocol-v1`解读为`跨进程链路信息传递协议`，`Trace-Data-Protocol`解读为`Skywalking链路数据采集协议`。官方对两个协议称呼有中文翻译，可自行查阅。
“跨进程链路信息传递协议”本身理解不困难，但在实现时需要考虑很多因素，本系列文章将结合Java探针源码深入分析。本文重点讲解`链路ID`生成过程。

跨进程链路信息传递协议作用

在分布式系统中，用户发起一次数据请求后，后端系统会有多个服务服务节点处理数据请求。链路跟踪系统的重要作用之一就是要能将数据在各个服务节点处理的先后顺序，逻辑关系，经理时长，拓扑结构等信息客观真实的反应出来。系统中一般一个进程充当一个服务节点，`跨进程`的描述由此而来。
一个完整的分布式系统，各个服务节点往往是有不同的组件，不同的语言，不同的技术栈组成。如java服务，node服务，nginx组件等等。要解决链路数据在不同节点的生成、传递、识别、解析、编码等问题，就不得不有一套规范进行约定，以便不同的语言能处理链路数据。于是`跨进程链路信息传递协议`就诞生了。

Skywalking Java Agent源码结构简介

在讲trace segment id生成逻辑前先简要介绍一下Skywalking Java Agent的源码结构。
Skywalking v5.0.0-GA中，Java Agent对Skywalking链路信息的抽象与封装是位于源码的`apm-sniffer/apm-agent-core`模块中。本文对该模块的链路数据处理逻辑做如下分析。

- 链路数据模型抽象与封装

作用是封装链路数据协议约定的数据抽象，同时总体兼容了Opentracing规范。重点类有如下几个

-
  - `org.apache.skywalking.apm.agent.core.context.trace.TraceSegment`
  - `org.apache.skywalking.apm.agent.core.context.trace.TraceSegmentRef`
  - `org.apache.skywalking.apm.agent.core.context.trace.EntrySpan`
  - `org.apache.skywalking.apm.agent.core.context.trace.ExitSpan`
  - `org.apache.skywalking.apm.agent.core.context.ids.DistributedTraceIds`

- 跨进程链路数据抽象与封装

重点类是`org.apache.skywalking.apm.agent.core.context.ContextCarrier`。即是对本文所讲解的协议规范的封装。

- 链路数据采集模型抽象与封装

重点类是`org.apache.skywalking.apm.agent.core.context.TracingContext`。TracingContext保存在ThreadLocal中，包含了链路中的所有数据，以及对数据的管控方法，主要是对Span的管控。同时提供对ContextCarrier数据的处理，包括：

-
  - 将TraceSegment转换为ContextCarrier，即org.apache.skywalking.apm.agent.core.context.TracingContext#inject
  - 从ContextCarrier数据中抽取TraceSegment数据，即org.apache.skywalking.apm.agent.core.context.TracingContext#extract

  - 链路数据采集模块

重点是`org.apache.skywalking.apm.agent.core.context.ContextManager`类。ContextManager类是各种skywalking agent插件的枢纽。不同组件的skywalking插件，如mq，dubbo，tomcat，spring等skywalking agent插件均是通过调用ContextManager创建和管控TracingContext，TraceSegment，EntrySpan，ExitSpan，ContextCarrier等数据。可以说ContextManager管控着agent内的链路数据生命周期。

- 链路数据上传模块

重点是`org.apache.skywalking.apm.agent.core.remote`包中的`TraceSegmentServiceClient`类。
ContextManager采集节点内的链路数据片段（TraceSegment）后，通知TraceSegmentServiceClient将数据上报到Collector Server。
其中涉及到
`TracingContextListener`与skywalking封装的`内存MQ`组件，后面会详细分析。

以上每个部分都涉及复杂的处理逻辑，后面会分篇一一做详细讲解。

Skywalking链路ID生成流程及Java实现

官方文档Skywalking-Cross-Process-Propagation-Headers-Protocol-v1已做了详细的说明，本文结合Java实现做进一步的分析。

1. Trace Segment Id协议规范

> The trace segment id is the unique id for the part of the distributed trace. Each id is only used in a single thread. The id includes three parts(Long), e.g. "1.2343.234234234

1. The first one represents application instance id, which assigned by collector. (most likely just an integer value, would be helpful in protobuf)
2. The second one represents thread id. (In Java most likely just an integer value, would be helpful in protobuf)
3. The third one also has two parts

3.1. A timestamp, measured in milliseconds
3.2.A seq, in current thread, between 0(included) and 9999(included)

> If you are using other language, you can generate your own id, but make sure it is unique and combined by three longs.

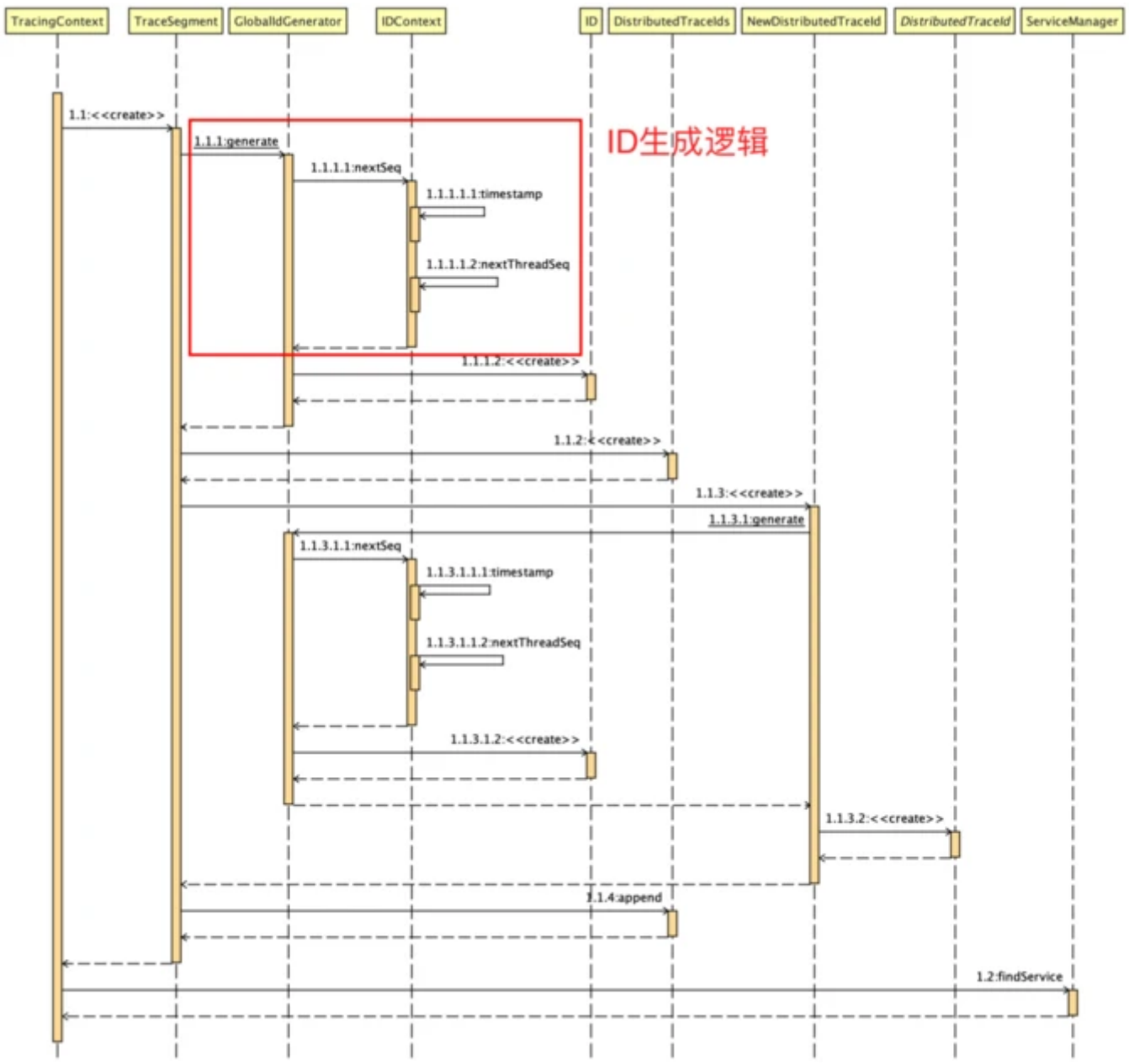
规范要点

- trace segment是分布式调用链的抽象，可以理解为分布式系统中的每个服务节点。
- 协议要求trace segment id由三部分组成，每部分是一个Long型的数值。三部分数据用点号连接，组成trace segment id字符串。
- 生成的trace segment id字符串，只需要保证ID由三个Long型构成，并全局唯一即可。并没有对三个部分的逻辑含义做约定，可以由开发者自行约定。
- java agent生成的ID的三部分规则为原文描述。即：第一部分代表应用的实例ID，第二部分为线程号，第三部分由时间戳和自增序列组成。

2. Trace Segment Id生成流程

首先skywalking java agent提供了`org.apache.skywalking.apm.agent.core.context.ids.ID`类用于对协议规范描述的数据结构，并提供了合法性校验方法isValid和序列化方法transform，重写了toString，equals，hashCode方法。
通过grpc上报到collector server的链路采集数据都提供了一个transform方法，用于序列化为protobuf数据。如
`org.apache.skywalking.apm.agent.core.context.trace.TraceSegment#transform`，
`org.apache.skywalking.apm.agent.core.context.trace.TraceSegmentRef#transform`等。

Trace Segment Id的创建发生在实例化`TracingContext`过程中。`TracingContext.new()`的序列图参考如下：



上图红框为生成`Trace Segment Id`的流程，通过调用`org.apache.skywalking.apm.agent.core.context.ids.GlobalIdGenerator#generate`方法生成满足协议规范的ID字符串。

3. Global(Distributed) Trace Id与Trace Segment Id

3.1 区别

skywalking中分别有两类全局唯一的id，即Global(Distributed) Trace Id与Trace Segment Id。Global(Distributed) Trace Id是指分布系统下的某条链路的唯一ID。Trace Segment Id是指链路所经过的分布系统服务节点事生成的链路片段ID。两者是一对多的关系。

3.2 生成流程

上述序列图表明，在实例化TracingContext时，会实例化TraceSegment，而在实例化TraceSegment时，同时生成了Global(Distributed) Trace Id与Trace Segment Id，源码如下：

```
    public TraceSegment() {
        this.traceSegmentId = GlobalIdGenerator.generate();
        this.spans = new LinkedList<AbstractTracingSpan>();
        this.relatedGlobalTraces = new DistributedTraceIds();
        this.relatedGlobalTraces.append(new NewDistributedTraceId());
    }
```

- 生成trace segment id

其中，第1行代码`this.traceSegmentId = GlobalIdGenerator.generate();`，即是生成Trace Segment Id，调用了一次GlobalIdGenerator.generate()方法。
在第4行代码的`new NewDistributedTraceId()`创建了Global(Distributed) Trace Id。源码如下，可见同样是调用`GlobalIdGenerator.generate()`方法。

```
public class NewDistributedTraceId extends DistributedTraceId {
    public NewDistributedTraceId() {
        super(GlobalIdGenerator.generate());
    }
}
```

- 生成Global(Distributed) Trace Id

全局链路id的的逻辑较复杂，java agent中为这个id封装了`DistributedTraceIds`类，通过该类的append方法添加`NewDistributedTraceId` 对象，表示一个Global(Distributed) Trace Id数据。NewDistributedTraceId是DistributedTraceId的子类，而DistributedTraceId类就是对ID类的包装，提供了一些工具方法。
`DistributedTraceIds`代表了相关链路id的集合，大多数情况下仅包含一条链路id。

3.3 关于GlobalIdGenerator

两个id均是调用同样的方法生成，即`org.apache.skywalking.apm.agent.core.context.ids.GlobalIdGenerator#generate`。GlobalIdGenerator是生产全局ID字符串的逻辑封装类。下面看看`GlobalIdGenerator#generate`的源码。

```
    public static ID generate() {
        if (RemoteDownstreamConfig.Agent.APPLICATION_INSTANCE_ID == DictionaryUtil.nullValue()) {
            throw new IllegalStateException();
        }
        IDContext context = THREAD_ID_SEQUENCE.get();

        return new ID(
            RemoteDownstreamConfig.Agent.APPLICATION_INSTANCE_ID,
            Thread.currentThread().getId(),
            context.nextSeq()
        );
    }
```

上述源码最终返回的即是根据协议规范封装的ID数据结构，第一个参数为当前应用的实例ID，第二个参数为当前线程号。第三个参数是通过IDContext.nextSeq()方法获取。而IDContext是从ThreadContext中获取的，所以IDContext是线程独有的。IDContext的源码比较好理解，IDContext.nextSeq()返回当前时间戳与线程内的自增序号组合的字符串。

4. 总结

综上，在新链路开始，生成TraceSegment实例时，第一次调用`GlobalIdGenerator.generate()`作为`Trace Segment Id`，第二次调用`GlobalIdGenerator.generate()`作为`Global(Distributed) Trace Id`。

5. 预告

`org.apache.skywalking.apm.agent.core.context.trace.TraceSegment`是skywalking java agent链路逻辑的核心类，后面会单独讲解。

作者：易企秀高级工程师-Frank

devops java skywalking

阅读 2.7k • 发布于 2020-03-18

易企秀工程师

88 声望 **10** 粉丝

关注作者

0 条评论

得票数 最新

撰写评论 ...

提交评论

继续阅读

使用 docker 部署 spring boot 并接入 skywalking

最近在研究skywalking，打算使用k8s部署 skywalking 并将 pod 中的应用接入 skywalking 进行服务链路追踪。这篇文章先不介绍 s...

惜鸟 • 阅读 3.4k • 17 赞

k8s 部署 skywalking 并将 pod 应用接入链路追踪

前面写了两篇文章介绍使用 docker 部署 spring boot 和 tomcat 项目，并将其接入skywalking，这篇文章主要介绍使用 k8s 部署 sk...

惜鸟 • 阅读 4.1k • 13 赞

skywalking实现分布式系统链路追踪

随着微服务的越来越流行，我们服务之间的调用关系就显得越来越复杂，我们急需一个APM工具来分析系统中存在的各种性能指标...

huan1993 • 阅读 3.9k • 2 赞

Kubernetes + Spring Cloud 集成链路追踪 SkyWalking

一、概述1、什么是 SkyWalking ？ 分布式系统的应用程序性能监视工具，专为微服务、云原生架构和基于容器（Docker、 K8s、 M...

ZeeBJ • 阅读 181 • 2 赞

基于docker部署的项目如何和skywalking agent进行整合

skywalking简介skywalking是一款开源的应用性能监控系统，包括指标监控，分布式追踪，分布式系统性能诊断skywalking官方中...

linyb极客之路 • 阅读 1.6k • 1 赞

史上最全SpringCloud整合skywalking

SkyWalking 是什么？ FROM [链接]分布式系统的应用程序性能监视工具，专为微服务、云原生架构和基于容器（Docker、 K8s、 Me...

liumang • 阅读 663

计算机网络基础(四)---网络层-ARP协议与RARP协议

还是看在上一篇文章中提到的这张图，计算机A将数据跨设备传输给C。A发出目的地为C的IP数据报，查询路由表发现下一跳为E，...

书旅 • 阅读 1.6k

聊聊skywalking的TraceSegmentServiceClient

skywalking-6.6.0/apm-sniffer/apm-agent-core/src/main/java/org/apache/skywalking/apm/agent/core/context/TracingContextList...

codecraft • 阅读 563