分布式追踪 SkyWalking 源码分析四 Agent 收集 && 发送 trace数据

技术标签： 分布式链路追踪 (/tag/%E5%88%86%E5%B8%83%E5%BC%8F%E9%93%BE%E8%B7%AF%E8%BF%BD%E8%B8%AA/)

分布式链路追踪系统，链路的追踪大体流程如下：

1. Agent 收集 Trace 数据。
2. Agent 发送 Trace 数据给 Collector 。
3. Collector 接收 Trace 数据。
4. Collector 存储 Trace 数据到存储器，例如，数据库。

`org.skywalking.apm.agent.core.context.trace.TraceSegment` (https://github.com/YunaiV/skywalking/blob/2a75efbeddac2b9565816af0ab0873ec3d998424/apm-sniffer/apm-agent-core/src/main/java/org/skywalking/apm/agent/core/context/trace/TraceSegment.java) ，是**一次**分布式链路追踪(Distributed Trace) 的**一段**。

- **一条** TraceSegment ，用于记录所在**线程**(Thread)的链路。
- **一次**分布式链路追踪，可以包含**多条** TraceSegment ，因为存在**跨进程**(例如，RPC 、MQ 等等)，或者垮**线程**(例如，并发执行、异步回调等等)。

`traceSegmentId` 属性，TraceSegment 的编号，全局唯一

`spans` 属性，包含的 Span **数组**。这是 TraceSegment 的**主体**，总的来说，TraceSegment 是 Span 数组的封装。

我们先来看看**一个爸爸**的情况，常见于 RPC 调用。例如，【服务 A】调用【服务 B】时，【服务 B】新建一个 TraceSegment 对象：

- 将自己的 `refs` 指向【服务 A】的 TraceSegment 。
- 将自己的 `relatedGlobalTraces` 设置为 【服务 A】的 DistributedTraceId 对象。

2.1 ID

`org.skywalking.apm.agent.core.context.ids.ID` (https://github.com/YunaiV/skywalking/blob/cc27e35d69d922ba8fa38fbe4e8cc4704960f602/apm-sniffer/apm-agent-core/src/main/java/org/skywalking/apm/agent/core/context/ids/ID.java) ，编号。从类的定义上，这是一个**通用**的编号，由三段整数组成。

目前使用 GlobalIdGenerator 生成，作为**全局唯一编号**。属性如下：

- `part1` 属性，应用实例编号。
- `part2` 属性，线程编号。
- `part3` 属性，时间戳串，生成方式为 `${时间戳} * 10000 + 线程自增序列([0, 9999])` 。例如：15127007074950012 。具体生成方法的代码，在 GlobalIdGenerator 中详细解析。
- `encoding` 属性，编码后的字符串。格式为 `"${part1}.${part2}.${part3}"` 。例如， `"12.35.15127007074950000"` 。
  - 使用 `#encode()` (https://github.com/YunaiV/skywalking/blob/cc27e35d69d922ba8fa38fbe4e8cc4704960f602/apm-sniffer/apm-agent-core/src/main/java/org/skywalking/apm/agent/core/context/ids/ID.java#L83) 方法，编码编号。
- `isValid` 属性，编号是否合法。
  - 使用 `ID(encodingString)` (https://github.com/YunaiV/skywalking/blob/cc27e35d69d922ba8fa38fbe4e8cc4704960f602/apm-sniffer/apm-agent-core/src/main/java/org/skywalking/apm/agent/core/context/ids/ID.java#L56) 构造方法，解析字符串，生成 ID 。

org.skywalking.apm.agent.core.context.ids.NewDistributedTraceId (https://github.com/YunaiV/skywalking/blob/5fb841b3ae5b78f07d06c6186adf9a8c08295a07/apm-sniffer/apm-agent-core/src/main/java/org/skywalking/apm/agent/core/context/ids/NewDistributedTraceId.java)，**新建的**分布式链路追踪编号。当全局链路追踪开始，创建 TraceSegment 对象的过程中，会调用 `DistributedTraceId()` 构造方法 (https://github.com/YunaiV/skywalking/blob/5fb841b3ae5b78f07d06c6186adf9a8c08295a07/apm-sniffer/apm-agent-core/src/main/java/org/skywalking/apm/agent/core/context/ids/NewDistributedTraceId.java#L30)，创建 DistributedTraceId 对象。该构造方法内部会调用 `GlobalIdGenerator#generate()` 方法，创建 ID 对象。

`#setOperationId(operationId)` 方法，设置操作编号。考虑到操作名是字符串，Agent 发送给 Collector 占用流量较大。因此，Agent 会将操作注册到 Collector ，生成操作编号。

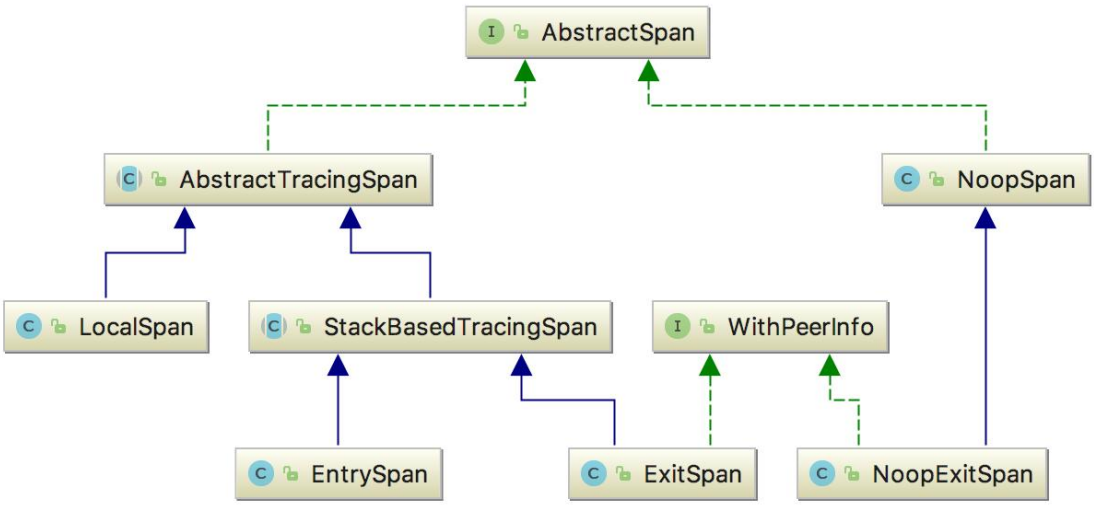
2.2.1 Tag

2.2.1.1 AbstractTag

`org.skywalking.apm.agent.core.context.tag.AbstractTag<T>` (https://github.com/YunaiV/skywalking/blob/e0c449745dfabe847b2e918d5352381f191a4469/apm-sniffer/apm-agent-core/src/main/java/org/skywalking/apm/agent/core/context/tag/AbstractTag.java)，标签**抽象类**。注意，这个类的用途是将标签属性设置到 Span 上，或者说，它是设置 Span 的标签的**工具类**。代码如下：

- `key` 属性，标签的键。
- `#set(AbstractSpan span, T tagValue)` **抽象**方法，设置 Span 的标签键 `key` 的值为 `tagValue`

关于span的类继承图



Span 只有三种实现类：

- EntrySpan： 入口 Span
- LocalSpan： 本地 Span
- ExitSpan： 出口 Span

2.2.2.2.1 EntrySpan

`org.skywalking.apm.agent.core.context.trace.EntrySpan`

(https://github.com/YunaiV/skywalking/blob/d36f6a47a208720f4caac9d9a8b7263bd36f2187/apm-sniffer/apm-agent-core/src/main/java/org/skywalking/apm/agent/core/context/trace/EntrySpan.java)，实现 StackBasedTracingSpan 抽象类，**入口** Span，用于服务提供者(Service Provider)，例如 Tomcat 。

那么为什么 EntrySpan 继承 StackBasedTracingSpan？

例如，我们常用的 SprintBoot 场景下，Agent 会在 SkyWalking 插件在 Tomcat 定义的方法切面，创建 EntrySpan 对象，也会在 SkyWalking 插件在 SpringMVC 定义的方法切面，创建 EntrySpan 对象。那岂不是出现**两个** EntrySpan，一个 TraceSegment 出现了两个入口 Span？

答案是当然不会！Agent 只会在第一个方法切面，生成 EntrySpan 对象，第二个方法切面，栈深度 + 1。这也是上面我们看到的 `#finish(TraceSegment)` 方法，只在栈深度为零时，出栈成功。通过这样的方式，保持一个 TraceSegment 有且仅有一个 EntrySpan 对象。

对新进入的方法切面，就把栈深度+1

而对于StackBasedTracingSpan的finish方法，把栈深度减少

如下是一个 EntrySpan 在 SkyWalking 展示的例子：

2.2.2.2.2 ExitSpan

`org.skywalking.apm.agent.core.context.trace.ExitSpan`

(https://github.com/YunaiV/skywalking/blob/958830d8db481b5b8a70498a09bc18eb7c721737/apm-sniffer/apm-agent-core/src/main/java/org/skywalking/apm/agent/core/context/trace/ExitSpan.java)，继承 StackBasedTracingSpan 抽象类，**出口** Span，用于服务消费者(Service Consumer)，例如 HttpClient 、MongoDBClient 。

那么为什么 ExitSpan 继承 StackBasedTracingSpan ?

例如，我们可能在使用的 Dubbox 场景下，【Dubbox 服务 A】使用 HTTP 调用【Dubbox 服务 B】时，实际过程是，【Dubbox 服务 A】=》【HttpClient】=》【Dubbox 服务 B】。Agent 会在【Dubbox 服务 A】创建 ExitSpan 对象，也会在 【HttpClient】创建 ExitSpan 对象。那岂不是**一次出口**，出现**两个** ExitSpan ?

答案是当然不会！Agent 只会在【Dubbox 服务 A】，生成 EntrySpan 对象，第二个方法切面，栈深度 **+ 1**。这也是上面我们看到的 `#finish(TraceSegment)` 方法，只在栈深度为零时，出栈成功。通过这样的方式，保持**一次出口**有且仅有一个 ExitSpan 对象。

当然，一个 TraceSegment 会有多个 ExitSpan 对象，例如【服务 A】远程调用【服务 B】，然后【服务 A】再次远程调用【服务 B】，或者然后【服务 A】远程调用【服务 C】。

2.3 TraceSegmentRef

`org.skywalking.apm.agent.core.context.trace.TraceSegmentRef` (https://github.com/YunaiV/skywalking/blob/49dc81a8bcaad1879b3a3be9917944b0b8b5a7a4/apm-sniffer/apm-agent-core/src/main/java/org/skywalking/apm/agent/core/context/trace/TraceSegmentRef.java)，TraceSegment 指向，通过 `traceSegmentId` 和 `spanId` 属性，指向父级 TraceSegment 的指定 Span 。

3. Context

在 「2. Trace」 (http://www.iocoder.cn/SkyWalking/agent-collect-trace/#) 中，我们看了 Trace 的数据结构，本小节，我们一起来看看 Context 是怎么收集 Trace 数据的。

3.1 ContextManager

`org.skywalking.apm.agent.core.context.ContextManager` ，实现了 BootService 、TracingContextListener 、IgnoreTracerContextListener 接口，链路追踪上下文管理器。

`CONTEXT` (https://github.com/YunaiV/skywalking/blob/ad259ad680df86296036910ede262765ffb44e5e/apm-sniffer/apm-agent-core/src/main/java/org/skywalking/apm/agent/core/context/ContextManager.java#L52) **静态**属性，线程变量，存储 AbstractTracerContext 对象。为什么是**线程变量**呢?

一个 TraceSegment 对象，关联到**一个**线程，负责收集该线程的链路追踪数据，因此使用线程变量。

而**一个** AbstractTracerContext 会关联**一个** TraceSegment 对象，ContextManager 负责获取、创建、销毁 AbstractTracerContext 对象。

`#getOrCreate(operationName, forceSampling)` (https://github.com/YunaiV/skywalking/blob/ad259ad680df86296036910ede262765ffb44e5e/apm-sniffer/apm-agent-core/src/main/java/org/skywalking/apm/agent/core/context/ContextManager.java#L61) **静态**方法，获取 AbstractTracerContext 对象。若不存在，进行**创建**。

- **要**需要收集 Trace 数据的情况下，创建 TracingContext (https://github.com/YunaiV/skywalking/blob/ad259ad680df86296036910ede262765ffb44e5e/apm-sniffer/apm-agent-core/src/main/java/org/skywalking/apm/agent/core/context/TracingContext.java) 对象。
- **不**需要收集 Trace 数据的情况下，创建 IgnoredTracerContext (https://github.com/YunaiV/skywalking/blob/ad259ad680df86296036910ede262765ffb44e5e/apm-sniffer/apm-agent-core/src/main/java/org/skywalking/apm/agent/core/context/IgnoredTracerContext.java) 对象。

在下面的 `#createEntrySpan(...)` 、 `#createLocalSpan(...)` 、 `#createExitSpan(...)` 等等方法中，都会调用 AbstractTracerContext 提供的方法。这些方法的代码，我们放在 「3.2 AbstractTracerContext」 (http://www.iocoder.cn/SkyWalking/agent-collect-trace/#) 一起解析，保证流程的整体性。

另外，ContextManager 封装了**所有** AbstractTracerContext 提供的方法，从而实现，外部调用者，例如 SkyWalking 的插件，只调用 ContextManager 的方法，而不调用 AbstractTracerContext 的方法。

创建出traceContext

++

核心类实现TracingContext

创建EntrySpan

```
/**
 * Create an entry span
 *
 * @param operationName most likely a service name
 * @return span instance. Ref to {@link EntrySpan}
 */
@Override
public AbstractSpan createEntrySpan(final String operationName) {
    if (isLimitMechanismWorking()) {
        NoopSpan span = new NoopSpan();
        return push(span);
    }
    AbstractSpan entrySpan;
    final AbstractSpan parentSpan = peek();
    final int parentSpanId = parentSpan == null ? -1 : parentSpan.getSpanId();
    if (parentSpan != null && parentSpan.isEntry()) {
        entrySpan = (AbstractTracingSpan)DictionaryManager.findEndpointSection()
            .findOnly(segment.getServiceId(), operationName)
            .doInCondition((FoundAndObtain) (operationId) -> {
                    return parentSpan.setOperationId(operationId);
            }, () -> { return parentSpan.setOperationName(operationName); });
        return entrySpan.start();
    } else {
        entrySpan = (AbstractTracingSpan)DictionaryManager.findEndpointSection()
            .findOnly(segment.getServiceId(), operationName)
            .doInCondition((FoundAndObtain) (operationId) -> {
                    return new EntrySpan(spanIdGenerator++, parentSpanId, operationId);
            }, () -> {
                    return new EntrySpan(spanIdGenerator++, parentSpanId, operationName);
            });
        entrySpan.start();
        return push(entrySpan);
    }
}
```

父span存在，就直接start；父span不存在，就新建一个EntrySpan

创建exitSpan，原理类似

++

结束span

调用pop弹栈，然后调用finish，结束本线程的traceSegment

3.2.3.3 传输

`org.skywalking.apm.agent.core.context.CarrierItem`

(https://github.com/YunaiV/skywalking/blob/dd6d9bff2d160f3aa60bc0be5152c49ecc9d94a4/apm-sniffer/apm-agent-core/src/main/java/org/skywalking/apm/agent/core/context/CarrierItem.java)，传输载体**项**。代码如：

- `headKey` 属性，Header 键。
- `headValue` 属性，Header 值。
- `next` 属性，下一个项。

CarrierItem 有两个子类：

- CarrierItemHead (https://github.com/YunaiV/skywalking/blob/dd6d9bff2d160f3aa60bc0be5152c49ecc9d94a4/apm-sniffer/apm-agent-core/src/main/java/org/skywalking/apm/agent/core/context/CarrierItemHead.java)： Carrier 项的头(Head)，即首个元素。
- SW3CarrierItem (https://github.com/YunaiV/skywalking/blob/dd6d9bff2d160f3aa60bc0be5152c49ecc9d94a4/apm-sniffer/apm-agent-core/src/main/java/org/skywalking/apm/agent/core/context/SW3CarrierItem.java)： `header = sw3` ，用于传输 ContextCarrier 。

++

Agent发送trace数据

Agent 收集到 Trace 数据后，不是写入外部消息队列(例如，Kafka)或者日志文件，而是 Agent 写入**内存消息队列**，**后台线程【异步】**发送给 Collector 。

核心类为TraceSegmentServiceClient，负责将 TraceSegment **异步**发送到 Collector

核心方法consume

1.判断状态是Connected

2.开启一个观察器 upstreamSegmentStreamObserver

```
        try {
            for (TraceSegment segment : data) {
                UpstreamSegment upstreamSegment = segment.transform();
                upstreamSegmentStreamObserver.onNext(upstreamSegment);
            }
        } catch (Throwable t) {
            logger.error(t, format: "Transform and send UpstreamSegment to collector fail.");
        }

        upstreamSegmentStreamObserver.onCompleted();

        status.wait4Finish();
        segmentUplinkedCounter += data.size();
    } else {
        segmentAbandonedCounter += data.size();
    }
```

3.循环data，然后转换并且把这个upstreamSegment，发送到collector

智能推荐

(/article/7709987839/)

分布式链路追踪 SkyWalking 源码分析 —— Collector 初始化 (/article/7709987839/)

点击上方“芋道源码”，选择“设为星标” 做积极的人，而不是积极废人！ 源码精品专栏 中文详细注释的开源项目 消息中间件 RocketMQ 源码解析 数据库中间件 Sharding-JDBC 和 MyCAT 源码解析 作业调度中间件 Elastic-Job 源码解析 分布式事务中间件 TCC-Transaction 源码解析 Eureka ...

(/article/67662821256/)

服务器采集数据源码,Skywalking数据采集与收集源码分析 (/article/67662821256/)

skywalking的架构图如下： Skywalking的agent负责采集数据，发送到collector，collector聚合，并且存储这些数据，且提供了一个简洁使用的UI端，可共我们查看监控的指标。 下面我们来开始分析skywalking的源码。 下载源码并构建 因为skywalking为了实现高性能通信，采用的是grpc的方式来实现服务器与客户端的数据传输的，所以导入之后我们需要稍微做一些...

三种分布 （边缘 & 联合 & 条件） (/article/987910832 92/)

(/article/98791083292/)

自己的理解： 联合分布：观察变量大于等于2时出现的概念，即多种变量同时出现时的概率（上图例子是变量为离散的情况，也有连续的情况）。 边际分布/边缘分布：（和"边缘"两个字本身没太大关系，因为是求和，在表格中往往将这种值放在margin（表头）的位置，所以叫margin distribution），可以大致理解为，多种变量出现时，某一变量出现的概率。（关于求和的说法，可以参看...

SkyWalking 分布式追踪系统 (/article/21471614214/)

(/article/21471614214/)

随着微服务架构的流行，一些微服务架构下的问题也会越来越突出，比如一个请求会涉及多个服务，而服务本身可能也会依赖其他服务，整个请求路径就构成了一个网状的调用链，而在整个调用链中一旦某个节点发生异常，整个调用链的稳定性就会受到影响，所以会深深的感受到 "银弹" 这个词是不存在的，每种架构都有其优缺点 。 面对以上情况， 我们就需要一些可以帮助理解系统行为、用于分析性能问题的工具...

skywalking插入agent数据过长问题解决 (/article/66041119704/)

(/article/66041119704/)

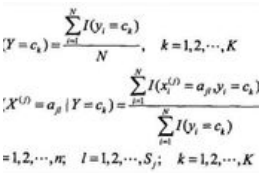
今天在paas平台上的skywalking server出现一个问题，就是agent发送过来的心跳数据超长了，导致skywalking server的表字段不够用，出现插入数据错误，如下图： 经过检查，发现agent部署在paas上，获取当前docker的IPV4地址，会拿到很多，目前是1百多个，估计是获取方式在容器上出现了问题导致。 目前的解决办法是将agent里的获取IPV...

猜你喜欢

支付宝资深架构师的分布式追踪 & APM 系统 SkyWalking 源码分析— DataCarrier 异步处理库 (/article/97291055687/)

(/article/97291055687/)

1. 概述 本文主要分享 SkyWalking DataCarrier 异步处理库。 基于生产者消费者的模式，大体结构如下图： 实际项目中，没有 Producer 这个类。所以本文提到的 Producer ，更多的是一种角色。 下面我们来看看整体的项目结构，如下图所示： 2. buffer org.skywalking.apm.commons.datacarrier.buffer&nb...



统计学习方法读书笔记（四） (/article/26251197687/)

(/article/26251197687/)

朴素贝叶斯法的学习与分类 基本方法： 假设输入空间是n维向量的集合，输出空间x是定义在输入空间X上的随机变量，y是定义在输出空间Y的随机变量，P（X,Y）是X,Y的联合分布。朴素贝叶斯就是通过训练数据集学习联合分布。实质就是学习先验概率和条件概率。先验概率如下：. 条件概率如下： 。通过这种方式就学习到了联合概率分布。 朴素贝叶斯法对条件概率做了独立性的假设，具体独立性假设如下： 对于给定的输入x...

zabbix添加触发器 (/article/2994818442/)

(/article/2994818442/)

写触发器（一般和邮件一起用）报警邮件点我 某个分区容量少于10G提示你 下面的问题事件生成模式选多重（如果选单个，触发器Top 100里就只存在几分钟） 低于70% 本身是有的，是20% 复制一个，改70%就行 &nbs...

Employee (/article/4879479708/)

(/article/4879479708/)

public class Employee { private int id; private String name; private double salary; private double byPercent; public Employee(int id,String name,double salary,double byPercent){ this.id = id; this.nam...

(/article/6620369160/)

iOS App设置中添加版本号 (/article/6620369160/)

1.创建 Settings.bundle 2.配置Root.plist文件 3.完成 效果图 项目名=测试新 版本号=1.0 缺点是版本号不能自动关联info.plist版本号 每次改版本号要改两个地方 ...

相关文章

分布式链路追踪 SkyWalking 源码分析 —— Agent 收集 Trace 数据 (/article/9446987572/)

分布式追踪 SkyWalking 源码分析六 Collector 接收和发送 trace 数据 (/article/74331466420/)

分布式追踪 SkyWalking 源码分析一 Agent初始化 (/article/8789781211/)

分布式追踪 SkyWalking 源码分析七 agent和byteBuddy 原理 (/article/4349861236/)

分布式链路追踪 SkyWalking 源码分析 —— Agent DictionaryManager 字典管理 (/article/8919987593/)

分布式链路追踪 SkyWalking 源码分析 —— Agent Remote 远程通信服务 (/article/3897987629/)

链路追踪 SkyWalking 源码分析 —— Agent 初始化 (/article/4315987861/)

分布式追踪 & APM 系统 SkyWalking 源码分析 —— Collector Streaming Computing 流式处理（二） (/article/9652750056/)

分布式追踪 & APM 系统 SkyWalking 源码分析 —— Collector Server Component 服务器组件 (/article/2592811670/)

分布式链路追踪 SkyWalking 源码分析 —— 应用于应用实例的注册 (/article/7242987602/)

热门文章

ubuntu制作usb启动盘 (/article/6748558624/)

发现最新的区块链应用-8月16日 (/article/4182609091/)

梯度下降法的三种形式批量梯度下降法、随机梯度下降以及小批量梯度下降法 (/article/613520162/)

java自定义登录_JavaWeb-SpringSecurity自定义登陆页面 (/article/66532474772/)

navicat找mysql的代码_本文为大家分享了使用navicat将csv文件导入mysql的具体代码，供大家参考，具体内容如下1.打开navicat，连接到数据库并找到自己想要导入数据的表... (/article/84662403510/)

Mac 命令行启动并连接Redis (/article/71061053078/)

vs2017解决scanf函数报错的问题 (/article/9791483757/)

MTOP2015双11整体网络拓扑 (/article/914251554/)

Python-S9—Day86-ORM项目实战之会议室预定相关 (/article/7451520829/)

项目进度计划的基本方法 (/article/91931024893/)

推荐文章

Oracle与MySQL的区别2 (/article/5665621117/)

Android学习笔记三之Android基础 (/article/73881575925/)

三维全景地图是怎么实现的？三维全景图制作教程 (/article/37142101076/)

个推php ios端教程,个推 -- iOS SDK 1.2.0 集成步骤 (/article/66532565698/)

2018云栖大会深圳峰会-企业级互联网架构专场看点提前大放送！ (/article/9634391345/)

看完泪奔：程序猿苦逼的一生 | 每日趣闻 (/article/52201655171/)

学校运动会主题的微信公众号图文排版有哪些技巧？ (/article/96501211491/)

询问HTG：白噪声屏幕保护程序，有效的文件命名以及从密码泄露中恢复 (/article/65712017722/)

java swing mysql 物资管理系统 (/article/61612721422/)

孝庄秘史第四集 (/article/4833928161/)

相关标签

分布式链路追踪 (/tag/%E5%88%86%E5%B8%83%E5%BC%8F%E9%93%BE%E8%B7%AF%E8%BF%BD%E8%B8%AA/)

服务器采集数据源码 (/tag/%E6%9C%8D%E5%8A%A1%E5%99%A8%E9%87%87%E9%9B%86%E6%95%B0%E6%8D%AE%E6%BA%90%E7%A0%81/)

v (/tag/v/)

Linux Shell编程理论+实战学习合集 (/tag/Linux+Shell%E7%BC%96%E7%A8%8B%E7%90%86%E8%AE%BA%2B%E5%AE%9E%E6%88%98%E5%AD%A6%E4%B9%A0%E5%90%88%E9%9B%86/)

SkyWalking (/tag/SkyWalking/)

docker (/tag/docker/)

运维 (/tag/%E8%BF%90%E7%BB%B4/)

zabbix (/tag/zabbix/)

centos (/tag/centos/)