

# 16、skywalking的OAP-通过SegmentTrace构建Trace的过程

rock\_fish 关注

2021.06.10 17:35:47 字数 977 阅读 93

## SegmentTrace的核心处理流程

包含了从Kafka初始化, 接收数据、解析构建、存储; 核心的源码流程如下:

```
KafkaFetcher -> TraceSegmentHandler#handle -> SegmentParserServiceImpl#send ->
TraceAnalyzer#doAnalysis -> AnalysisListener#parsexxx -> AnalysisListener#build -
> SourceReceiver#receive -> dispatcherManager#forward -> SegmentDispatcher#dispatch
```

1. KafkaFetcherProvider 的 start 方法中注册了TraceSegmentHandler, 用于接收Trace的数据, 进行处理。

```
1 handlerRegister.register(new TraceSegmentHandler(getManager(), config));
```

2. TraceSegmentHandler#handle 处理Kafka数据

```
1 //将Kafka数据, 解析成SegmentObject
2 SegmentObject segment = SegmentObject.parseFrom(record.value().get());
3 //交给SegmentParserServiceImpl处理
4 segmentParserService.send(segment);
```

3. SegmentParserServiceImpl#send 中通过 TraceAnalyzer 解析 SegmentObject

```
1 public void send(SegmentObject segment) {
2     final TraceAnalyzer traceAnalyzer = new TraceAnalyzer(moduleManager, listenerManager,
3     traceAnalyzer.doAnalysis(segment);
4 }
```

其内部功能很多, 从是Segment这个数据流程来说就是: 构建多个监听器, 以监听器的模式来通过解析segmentObject各个属性, 通过构建 SourceBuilder 对象来承载上下游的链路相关信息, 并添加到entrySourceBuilders中; 在 build 环节, 进一步构建各维度的souce数据, 包括 Trace(链路),Metrics(调用统计如调用次数, pxx, 响应时长等) 信息都在这个环节创建。先大致看下其代码主体流程, 接下来会分析内部更多的细节逻辑:

```
1 public void doAnalysis(SegmentObject segmentObject) {
2     if (segmentObject.getSpansList().size() == 0) {
3         return;
4     }
5
6     createSpanListeners();//创建监听器
7
8     notifySegmentListener(segmentObject);//处理trace
9
10    segmentObject.getSpansList().forEach(spanObject -> {
11        if (spanObject.getSpanId() == 0) {
12            notifyFirstListener(spanObject, segmentObject);//根据第一个span的信息做一些处理
13        }
14
15        if (SpanType.Exit.equals(spanObject.getSpanType())) {
16            notifyExitListener(spanObject, segmentObject);
17        }
18    });
19 }
```



rock\_fish 总资产6

关注

- 21、Skywalking的埋点-Agent动态采样控制  
阅读 219
- 22、skywalking的Trace数据协议  
阅读 12
- skywalking的日常维护1:  
com.netflix.zuul.exception.ZuulExcept  
阅读 41

- 推荐阅读
- Gateway动态路由实现  
阅读 833
- Nacos注册中心之概要设计  
阅读 217
- 大数据开发: Flume日志采集框架简介  
阅读 170
- [快速入门]通过http方式监听besu上智能合约的事件  
阅读 233
- 【dubbo源码】4. 配置信息解析-XML  
阅读 89





```
22         log.error("span type value was unexpected, span type name: {}", spanObject.get  
23                                     .nam  
24     }  
25     });  
26  
27     notifyListenerToBuild();  
28 }
```

## 1. `SegmentAnalysisListener#parseSegment` 构建 `Segment` (Source) ,部分属性赋值

### 1.1 赋值 起止时间

### 1.2 赋值 是否error

### 1.3 赋值 是否采样, 这里是重点

## 2. `SegmentAnalysisListener#notifyFirstListener` 更多的属性赋值

## 3. 多个 `EntryAnalysisListener` 监听器处理Entry类型的span

### 3.1 `SegmentAnalysisListener#parseEntry` 赋值service和endpoint的Name和id

### 3.2 `NetworkAddressAliasMappingListener#parseEntry` 构造 `NetworkAddressAliasSetup` 完善ip\_port地址与别名之间的映射关系, 交给 `NetworkAddressAliasSetupDispatcher` 处理

### 3.3 `MultiScopesAnalysisListener#parseEntry` 遍历span列表

- 3.3.1 将每个span构建成 `SourceBuilder` , 设置上下游的Server、Instance、endpoint的name信息, 这里mq和网关特殊处理, 其上游保持ip端口, 因为mq、网关通常没有搭载agent, 没有相关的name信息。
- 3.3.2 `setPublicAttrs`: `SourceBuilder`中添加 tag信息, 重点是时间bucket, `setResponseCode`, `Status`, `type`(http, rpc, db)
- 3.3.3 `SourceBuilder`添加到 `entrySourceBuilders` ,
- 3.3.4 `parseLogicEndpoints`//处理span的tag是 `LOGIC_ENDPOINT = "x-le"` 类型的, 添加到 `logicEndpointBuilders`中 (用途待梳理)

## 4. `MultiScopesAnalysisListener#parseExit` 监听器处理Exit类型的span

### 4.1 将span构建成 `SourceBuilder` , 设置上下游的Server、Instance、Endpoint的name信息, 尝试把下游的ip\_port信息修改成别名。

### 4.2 `setPublicAttrs`: `SourceBuilder`中添加 tag信息, 重点是时间bucket, `setResponseCode`, `Status`, `type`(http, rpc, db)

### 4.3 `SourceBuilder` 添加到 `exitSourceBuilders` ,

### 4.4 如果是db类型, 构造`slowStatementBuilder`, 判断时长设置慢查询标识, 存入 `dbSlowStatementBuilders`中。这里是全局的阈值 是个改造点。

## 5. `MultiScopesAnalysisListener#parseLocal` 监听器处理 `Local` 类型的span,通过 `parseLogicEndpoints` 方法处理span的tag是 `LOGIC_ENDPOINT = "x-le"` 类型的, 添加到 `logicEndpointBuilders`中 (用途待梳理)

## 6. 这里是重点, 以上构建的 `SourceBuilder` 就在这一步使用, 执行各个 `AnalysisListener#build`

### 6.1 `SegmentAnalysisListener#build` ,设置endpoint的 id 和name, 然后将 `Segment` 交给 `SourceReceiver#receive` 处理, 而 `SourceReceiver#receive` 就是调用 `dispatcherManager#forward` ,最终



## SegmentDispatcher 处理 Segment

简单来说，这里的逻辑就是把Source转换成StorageData，交给 `RecordStreamProcessor` 里的一组 `AbstractWorker`，用于完成以记录存储为主的相关工作。

```
1 public class SegmentDispatcher implements SourceDispatcher<Segment> {
2
3     @Override
4     public void dispatch(Segment source) {
5         //Segment(Source) 转换成 SegmentRecord(StorageData)
6         SegmentRecord segment = new SegmentRecord();
7         segment.setSegmentId(source.getSegmentId());
8         ...
9         segment.setTags(Tag.Util.toStringList(source.getTags()));
10        //交给worker链路处理StorageData
11        RecordStreamProcessor.getInstance().in(segment);
12    }
13 }
```

具体看下 `RecordStreamProcessor#in` 中 `RecordPersistentWorker` 是如何把记录存储到ES中的。

`RecordPersistentWorker#in` 中的逻辑很清晰：

```
1 //1. 把SegmentRecord，通过RecordEsDAO的方法转换成ES的 InsertRequest
2 InsertRequest insertRequest = recordDAO.prepareBatchInsert(model, record);
3 //2. 异步的方式写es
4 batchDAO.asynchronous(insertRequest);
```

1. `BatchProcessEsDAO#prepareBatchInsert` 方法中，通过 `storageBuilder` 把 `SegmentRecord` 转换成 map，再讲map构建成 `XContentBuilder`，进而构建成ES里的 `InsertRequest` 实例

```
1 @Override
2 public InsertRequest prepareBatchInsert(Model model, Record record) throws IOException {
3     XContentBuilder builder = map2builder(
4         IndexController.INSTANCE.appendMetricTableColumn(model, storageBuilder.entity2StorageBuilder(record)),
5         String modelName = TimeSeriesUtils.writeIndexName(model, record.getTimeBucket());
6         String id = IndexController.INSTANCE.generateDocId(model, record.id());
7         return getClient().prepareInsert(modelName, id, builder);
8     }
```

2. `BatchProcessEsDAO#asynchronous`

首先初始化 `bulkProcessor`，将 `InsertRequest` 提交给 `bulkProcessor`，`bulkProcessor` 是一个异步批插入的操作，细节可另行百度。

至此Trace写到ES后，这个流程就算结束了。



0人点赞 >



📄 监控



更多精彩内容，就在简书APP



"小红书" 小红书关注我

写下你的评论...

评论0

赞



rock\_fish

总资产6 共写了9.4W字 获得82个赞 共27个粉丝

关注

损坏的硬盘或u盘数据恢复，98%可恢复（一辈子收藏）



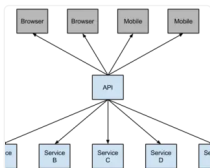
推荐阅读

Spring Cloud

Spring Cloud为开发人员提供了快速构建分布式系统中一些常见模式的工具（例如配置管理，服务发现，断路器，智...


 卡卡罗2017 阅读 120,800 评论 16 赞 134

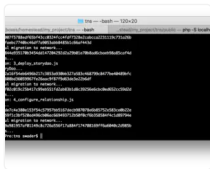
更多精彩内容>



以太坊构建DApps系列教程(七):为DAO合约构建Web3 UI

在本系列关于使用以太坊构建DApps教程的第6部分中，我们通过添加投票，黑名单，股息分配和撤销来完成DAO合约，同...

 编程狂魔 阅读 219 评论 0 赞 0




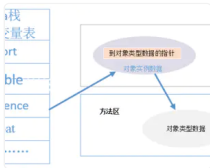
损坏的硬盘或u盘数据恢复，98%可恢复（一辈子收藏）



guava cache原理解析

缓存在日常开发中举足轻重，如果你的应用对某类数据有着较高的读取频次，并且改动较小时那就非常适合利用缓存来提高性能。...

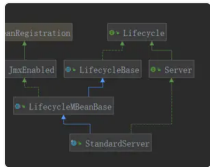
 tracy\_668 阅读 3,404 评论 0 赞 9



Tomcat之启动过程源码分析


LifecycleBase生命周期抽象类 在分析启动流程之前，有必要对于实现了LifecycleBase这个类的所...

 loveFXX 阅读 632 评论 0 赞 0



Kafka-2.配置-Broker Configs

Kafka在 property file format 使用键值对作为配置。这些值无论来自文件还是以编程的方式，都...

 悠扬前奏 阅读 554 评论 0 赞 0

