

17、skywalking的OAP-通过SegmentTrace构建Trace相关的Metrics的过程

 rock_fish 关注

2021.06.21 18:42:21 字数 1,151 阅读 71

SegmentTrace

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 -> XXXDispatcher#dispatch
```

Kafka pull数据处理

TraceSegmentHandler#handle 中调用 SegmentParserServiceImpl#send(segment)

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

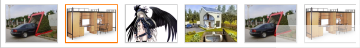
doAnalysis 解析segment


```
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        } else if (SpanType.Entry.equals(spanObject.getSpanType())) {
18            notifyEntryListener(spanObject, segmentObject);//这里有很重要的链路的metric信息
19        } else if (SpanType.Local.equals(spanObject.getSpanType())) {
20            notifyLocalListener(spanObject, segmentObject);
21        } else {
22            log.error("span type value was unexpected, span type name: {}", spanObject.get
23                .nam
24            );
25        }
26    });
27    notifyListenerToBuild();
28 }
```

notifyEntryListener 中调用 MultiScopesAnalysisListener#parseEntry ,把上下游的链路信息完善到 sourceBuilder里，并添加到entrySourceBuilders中，在 build 环节，进一步构建成各维度的 source数据，包括链路的trace,以及调用统计如调用次数，pxx，响应时长等metric信息都在这个环节创建。



学生公寓床



 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

- sentinel之dashboard改造
阅读 153
- Python 命令行（四）- 日志
阅读 90
- 4. ZooKeeper
阅读 302
- Spring动态添加定时任务
阅读 1,126
- JavaWeb开发之Filter过滤器
阅读 351



智能停车设备





```
7 sourceReceiver.receive(entrySourceBuilder.toServiceInstanceRelation());
8 EndpointRelation endpointRelation = entrySourceBuilder.toEndpointRelation();
9 sourceReceiver.receive(endpointRelation);
10 ...
```

从上边 `MultiScopesAnalysisListener#build` 代码片段中可以看到包含了 `Service`、`ServiceInstance`、`Endpoint`、`ServiceRelation`、`ServiceInstanceRelation`、`EndpointRelation` 这些类型的 `Source`；并将这些 `Source` 提交给 `sourceReceiver`，其底层封装的 `DispatcherManager` 会根据 `Source` 的类型选择相应的 `SourceDispatcher`，通过方法 `dispatch` 进一步处理。

具体的 `SourceDispatcher` 类是哪一个呢？

```
1 public class EndpointCallRelationDispatcher implements SourceDispatcher<EndpointRelation> {
2
3     @Override
4     public void dispatch(EndpointRelation source) {
5         switch (source.getDetectPoint()) {
6             case SERVER:
7                 serverSide(source);
8                 break;
9         }
10    }
```

从这个类 `EndpointCallRelationDispatcher#dispatch` 的参数可以看出，这个类负责 `EndpointRelation` 这种类型的 `Source`。查看 `dispatch` 的实现有以下这些：

```
1 BrowserAppTrafficSourceDispatcher.dispatch(SOURCE)
2 BrowserErrorLogRecordDispatcher.dispatch(BrowserErrorLog)
3 DatabaseStatementDispatcher.dispatch(DatabaseSlowStatement)
4 EndpointCallRelationDispatcher.dispatch(EndpointRelation)
5 EndpointMetaDispatcher.dispatch(EndpointMeta)
6 EndpointTrafficDispatcher.dispatch(Endpoint)
7 InstanceTrafficDispatcher.dispatch(ServiceInstance)
8 InstanceUpdateDispatcher.dispatch(ServiceInstanceUpdate)
9 LogRecordDispatcher.dispatch(Log)
10 NetworkAddressAliasSetupDispatcher.dispatch(NetworkAddressAliasSetup)
11 SegmentDispatcher.dispatch(Segment)
12 ServiceCallRelationDispatcher.dispatch(ServiceRelation)
13 ServiceInstanceCallRelationDispatcher.dispatch(ServiceInstanceRelation)
14 ServiceMetaDispatcher.dispatch(ServiceMeta)
15 ServiceTrafficDispatcher.dispatch(Service)
16 ZipkinSpanRecordDispatcher.dispatch(ZipkinSpan)
```

从以上清单中不难发现还缺少好多 `Source` 以及对应的 `Dispatcher` 类型；这些缺失的类，在 Skywalking 中是通过 OAL 机制在 OAP 启动时动态生成，OAL 脚本位于 `/config` 文件夹中，用户只需更改并重新启动服务器即可使其生效。但是，OAL 脚本还是编译语言，OAL 运行时会动态生成 Java 代码。

可以在系统环境中添加 `SW_OAL_ENGINE_DEBUG=Y` 打开开关，以查看生成了哪些类，在 `oal-rt` 目录下的 `dispatcher` 和 `metrics` 两个目录查看



image.png

这些生成的Metric的主要SCOPE为 All , Service , ServiceInstance , Endpoint , ServiceRelation , ServiceInstanceRelation , EndpointRelation 。此外, 还有一些辅助SCOPE。查看官网的[SCOPE定义](#), 可以找到所有现有的SCOPE和字段

Source 类的scope方法指定了SourceDispatcher的一个数字标识,

```
1 | public abstract class Source {  
2 |     public abstract int scope();
```

最终这些Source会在SourceDispatcher的dispatch中, 转换成StorageData, 并交由MetricsStreamProcessor#in 进入L1、L2的聚合处理, 报警处理, 导出处理。

L1聚合

创建Worker, 并构建worker链路:

1. 启动扫描Stream注解的时候,在StreamAnnotationListener#notify中, 通过MetricsStreamProcessor#create方法为每种Metrics生成一个MetricsAggregateWorker (当前实例内L1聚合), 创建并注册一个这种Metric类型的远程Worker服务MetricsPersistentWorker (给其他实例的数据做L2聚合和报警、存储)
2. 创建MetricsRemoteWorker并指定为MetricsAggregateWorker (L1聚合) 的nextWorker, 当完成L1聚合后将通过MetricsRemoteWorker当前的数据传递给远程的Worker服务MetricsPersistentWorker用于L2处理
3. 数据在worker链路的流传的逻辑为: MetricsAggregateWorker(本实例做L1) -> MetricsRemoteWorker(本实例传递给远程MetricsPersistentWorker) -> MetricsPersistentWorker(远程实例, 完成L2处理) -> min级数据存储/更新->执行Hour聚合处理->执行day聚合处理->提交给AlarmWorker->提交给ExportWorker

image.png

4. MetricsAggregateWorker的一些实现细节： 接收到Metrics数据后，放入dataCarrier(10000*2)中，然后有一个线程去消费处理Metric，将metric丢入MergableBufferedData中执行初次的聚合，MergableBufferedData中是一个map，遇到id相同的则执行聚合

```
1 public void accept(final METRICS data) {
2     final String id = data.id();
3     final METRICS existed = buffer.get(id);
4     if (existed == null) {
5         buffer.put(id, data);
6     } else {
7         final boolean isAbandoned = !existed.combine(data);
8         if (isAbandoned) {
9             buffer.remove(id);
10        }
11    }
12 }
```

MetricsPersistentWorker完成L2聚合

MetricsPersistentWorker内部使用了读写buffer缓冲，且buffer是可聚合的
即处理数据的时候，是：

- 1. 丢入写buffer，这个写buffer在接收数据的时候具有聚合的作用
- 2. 定时任务读buffer，这时候交换buffer的读写标识，把之前已写入数据的buffer变成读buffer，将数据读出来，进行下一步的处理。

MetricsRemoteWorker对应的远程服务是MetricsPersistentWorker，其内部有这三个很重要的worker，从其名字基本就可知道这些worker完成什么任务。

```
1 this.nextAlarmWorker = Optional.ofNullable(nextAlarmWorker);
2 this.nextExportWorker = Optional.ofNullable(nextExportWorker);
3 this.transWorker = Optional.ofNullable(transWorker);
```

认是3秒)

```
1 public void buildBatchRequests(List<PrepareRequest> prepareRequests) {
2     //取出一批
3     final List<INPUT> dataList = getCache().read();
4     //预处理
5     prepareBatch(dataList, prepareRequests);
6 }
```

prepareBatch中是最核心的逻辑：

1. 在prepareBatch中遍历Metrics
2. 每个metric记录都要交给transWorker做处理
3. 当已处理的数据满2000条的时候写ES
4. 当当前批次全部处理完的时候写ES
5. 写ES的时候，如果记录已存在，则先聚合老数据再更新
6. 写ES完成后，尝试将数据交给nextAlarmWorker和nextExportWorker。

 0人点赞 >



 监控



更多精彩内容，就在简书APP



"小礼物走一走，来简书关注我"

赞赏支持

还没有人赞赏，支持一下

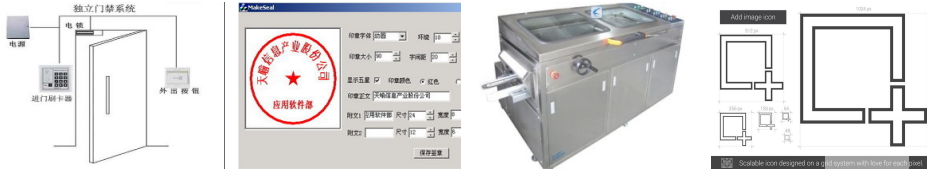


rock_fish

总资产6 共写了9.4W字 获得82个赞 共27个粉丝

关注

安印电子印章



推荐阅读

更多精彩内容 >

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

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：
K...

 rock_fish


阅读 94 评论 0 赞 0

写下你的评论...

评论0

赞

架...

 smooth00 阅读 469 评论 0 赞 2



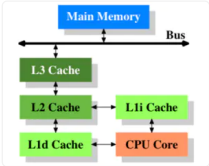
新款雷克萨斯ES上市售价10.98万元起



如何构建一个交易系统（八）


最近事情比较多，所有搁了些日子才写， 此篇主要讲技术， 不喜绕开。 先声明此篇文章和具体的语言无关， 语言之间的比...

 莲安宇秀 阅读 3,628 评论 0 赞 214




2018-07-18

董多娇第226天坚持分享， 焦点相信， 每个人在每一刻都会为自己做出一个决定与选择， 是他们当时认为最合适自己的， 所以任...

 良知良能良知良能 阅读 1,271 评论 1 赞 1

初识jQuery之jQuery设计思想（一）

一、jQuery简介 JQ是JS的一个优秀的库， 大型开发必备。在此，我想说的是， JQ里面很多函数使用和JS类似， 所...

 Welkin_qing 阅读 1,408 评论 0 赞 1