拉勾教育
—互联网人实战大学—

《31 讲带你搞懂 SkyWalking》

徐郡明 资深技术专家

— 拉勾教育出品 —

加餐4：trace-receiver 插件番外篇 慢查询的处理

前言

在 mysql-8.x-plugin 插件中

会**拦截** preparedStatement.execute() 方法创建 Database 类型的 **ExitSpan**

并在 execute() 方法调用完成之后结束 ExitSpan

前言

multiScopesSpanListener.parseExit() 方法还会针对 Database 类型的 ExitSpan 进行特殊处理

该处理主要用于统计慢查询

这里的慢查询统计不仅是 DB 的慢查询，还包括其他常见的存储

例如：Redis、MongoDB 等等

前言


```
DatabaseSlowStatement statement = new DatabaseSlowStatement();
//记录发生此次慢查询的 traceId
statement.setTraceId(traceId);
//由 TraceSegment.id 以及 Span.id 构成的唯一标识
statement.setId(segmentCoreInfo.getSegmentId() + "-" + spanDecorator.getSpanId());
//记录存储对应的 ServiceId
statement.setDatabaseServiceId(sourceBuilder.getDestServiceId());
//此次慢查询的实际耗时
statement.setLatency(sourceBuilder.getLatency());
//秒级时间窗口
statement.setTimeBucket(TimeBucket.getSecondTimeBucket(segmentCoreInfo.getStartTime()));
for (KeyStringValuePair tag : spanDecorator.getAllTags()) { //遍历ExitSpan携带的 Tag 信息
```

前言

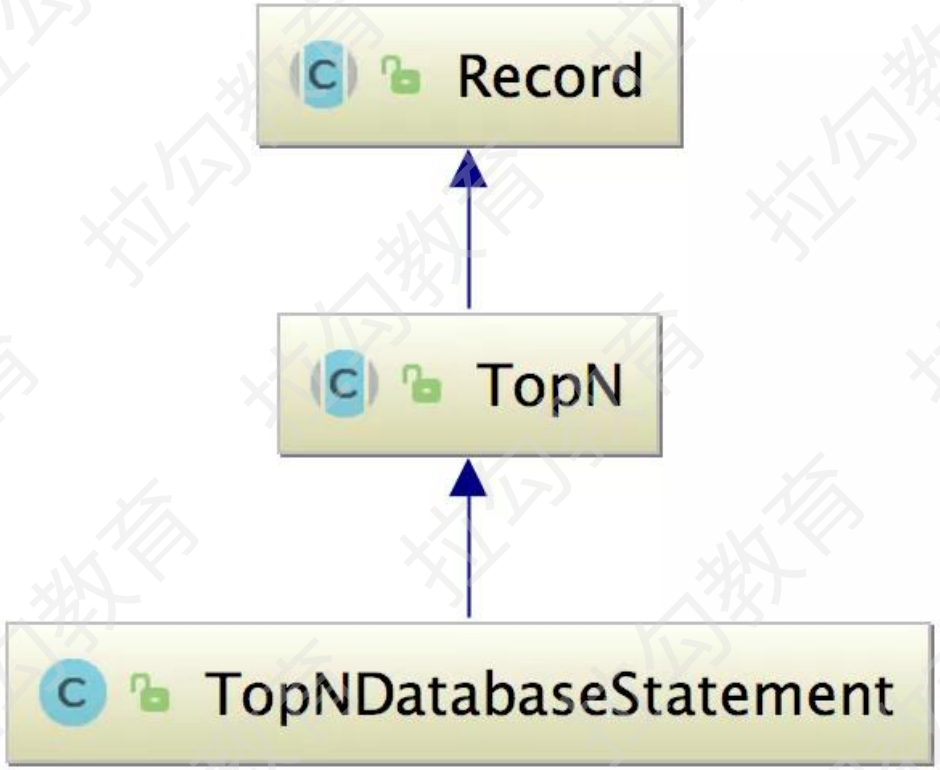

```
if (SpanTags.DB_STATEMENT.equals(tag.getKey())) {
    //具体执行的操作，例如，访问 DB 的话，就是 SQL 语句
    statement.setStatement(tag.getValue());
} else if (SpanTags.DB_TYPE.equals(tag.getKey())) {
    //在 application.yml 配置文件中配置了不同存储的慢查询阈值上限，
    //这里会根据 dbType（其值可以为 sql、Redis、MongoDB 等）查找其阈值
    String dbType = tag.getValue();
    DBLatencyThresholdsAndWatcher thresholds = config.getDbLatencyThresholdsAndWatcher();
    int threshold = thresholds.getThreshold(dbType);
    if (sourceBuilder.getLatency() > threshold) {
        isSlowDBAccess = true; //判断此次请求存储的操作是否为慢查询
    }
```

前言


```
    //这里会根据 dbType（其值可以为 sql、Redis、MongoDB 等）查找其阈值
    String dbType = tag.getValue();
    DBLatencyThresholdsAndWatcher thresholds = config.getDbLatencyThresholdsAndWatcher();
    int threshold = thresholds.getThreshold(dbType);
    if (sourceBuilder.getLatency() > threshold) {
      isSlowDBAccess = true; //判断此次请求存储的操作是否为慢查询
    }
  }

  if (isSlowDBAccess) { //将慢查询记录到 slowDatabaseAccesses 集合中
    slowDatabaseAccesses.add(statement);
  }
```

前言

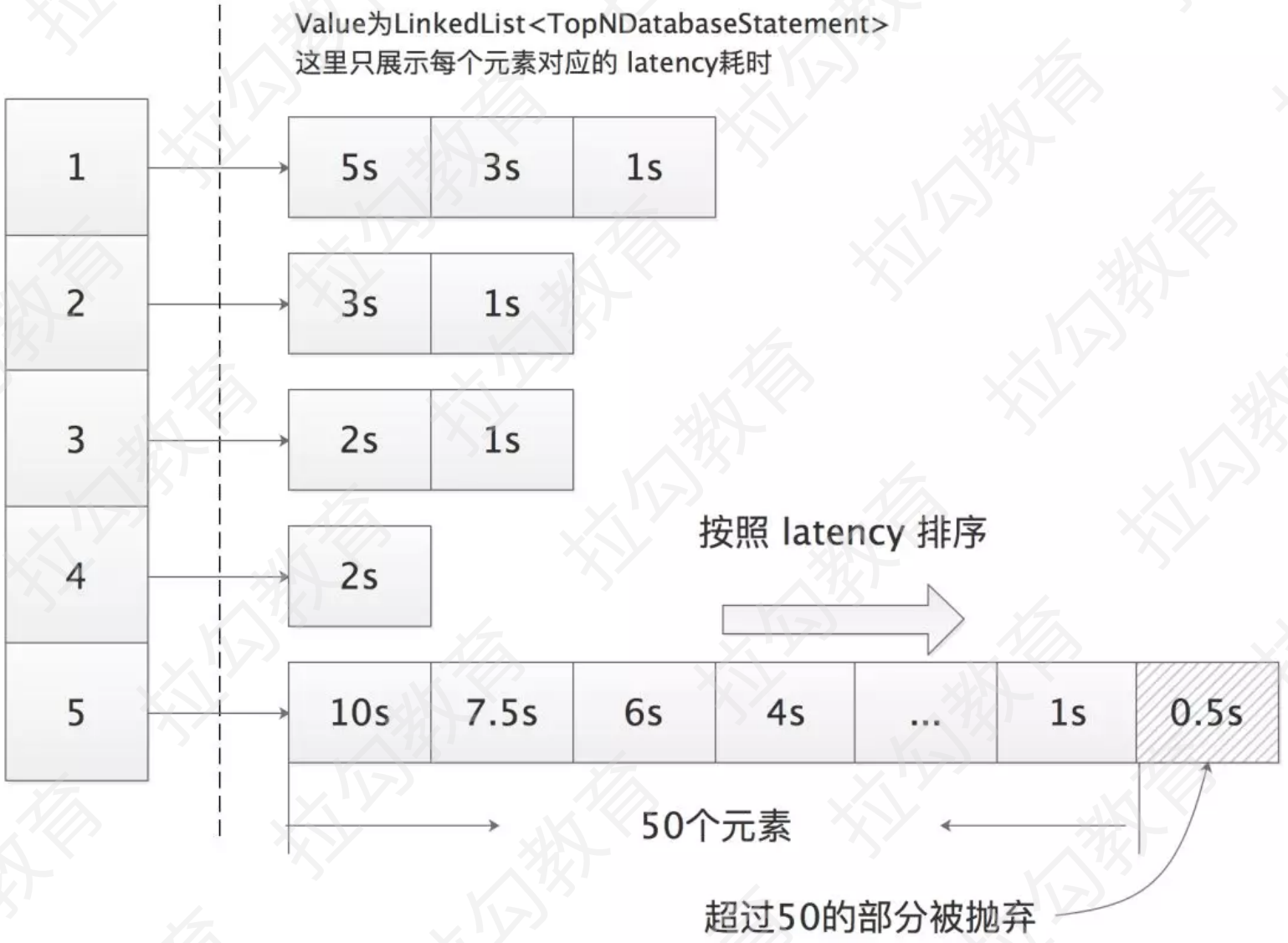


前言


```
@Getter @Setter @Column(columnName = "statement", content = true) private String statement;
@Getter @Setter @Column(columnName = "latency") private long latency;
@Getter @Setter @Column(columnName = "trace_id") private String traceId;
@Getter @Setter @Column(columnName = "service_id") private int serviceId;
```

TopNWorker

TopNWorker



TopNWorker


```
void onWork(TopN data) {
    limitedSizeDataCache.writing();
    try {
        limitedSizeDataCache.add(data);
    } finally {
        limitedSizeDataCache.finishWriting();
    }
}
```

PersistenceTimer

在 PersistenceWorker 的三个实现类中

MetricsPersistentWorker 和 RecordPersistentWorker 启动的 Consumer 直接使用了继承自

PersistenceWorker 的 onWork() 方法

该实现只会在 DataCache 缓存的数据到达一定阈值时，才会触发 ElasticSearch 的写入

如果缓存量长时间达不到阈值，就会导致监控数据和 Trace 数据写入延迟

PersistenceTimer


```
private void extractDataAndSave(IBatchDAO batchDAO) {
    //三个 PersistenceWorker 实现构成的列表
    List<PersistenceWorker> persistenceWorkers = new ArrayList<>();
    persistenceWorkers.addAll(MetricsStreamProcessor.getInstance().getPersistentWorkers());
    persistenceWorkers.addAll(RecordStreamProcessor.getInstance().getPersistentWorkers());
    persistenceWorkers.addAll(TopNStreamProcessor.getInstance().getPersistentWorkers());
    persistenceWorkers.forEach(worker -> {
        //逐个 PersistenceWorker 实现的 flushAndSwitch()方法，
        //其中主要是对 DataCache 队列的切换
        if (worker.flushAndSwitch()) {
            //调用 PersistenceWorker.buildBatchCollection()为 DataCache中每个元素创建相应的
IndexRequest 以及 UpdateRequest 请求
            List<?> batchCollection = worker.buildBatchCollection();
            batchAllCollection.addAll(batchCollection);
        }
    });
    //执行三个 PersistenceWorker 生成的全部 ElasticSearch 请求
    batchDAO.batchPersistence(batchAllCollection);
}
```

PersistenceTimer


```
public boolean flushAndSwitch() {
    long now = System.currentTimeMillis();
    if (now - lastReportTimestamp <= reportCycle) {
        return false; //默认 10min 执行一次
    }
    lastReportTimestamp = now; //重置 lastReportTimestamp
    return super.flushAndSwitch(); //调用 PersistenceWorker 实现
}
```

Next：第27讲《实战入门 GraphQL，如何将 REST API 换成 GraphQL》

拉勾教育

—互联网人实战大学—

关注拉勾「教育公众号」
获取更多课程信息