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 —

拉勾教育

拉勾教育
互联网人实战大学

第25讲：trace-receiver 插件拆解 Trace 蕴含的宝贵信息（上）

L / A / G / O / U

TraceModuleProvider

拉勾教育
互联网人实战大学

目前 trace-receiver-plugin 插件同时支持处理 V1 和 V2 两个版本的 TraceSegment

本课时重点分析 V2 版本 TraceSegment

后面不进行特殊说明的情况下，都是指 V2 版本 TraceSegment

拉勾教育

TraceModuleProvider

拉勾教育
互联网人实战大学

在 trace-receiver-plugin 插件的 SPI 文件中指定的 ModuleProvider 实现是 TraceModuleProvider

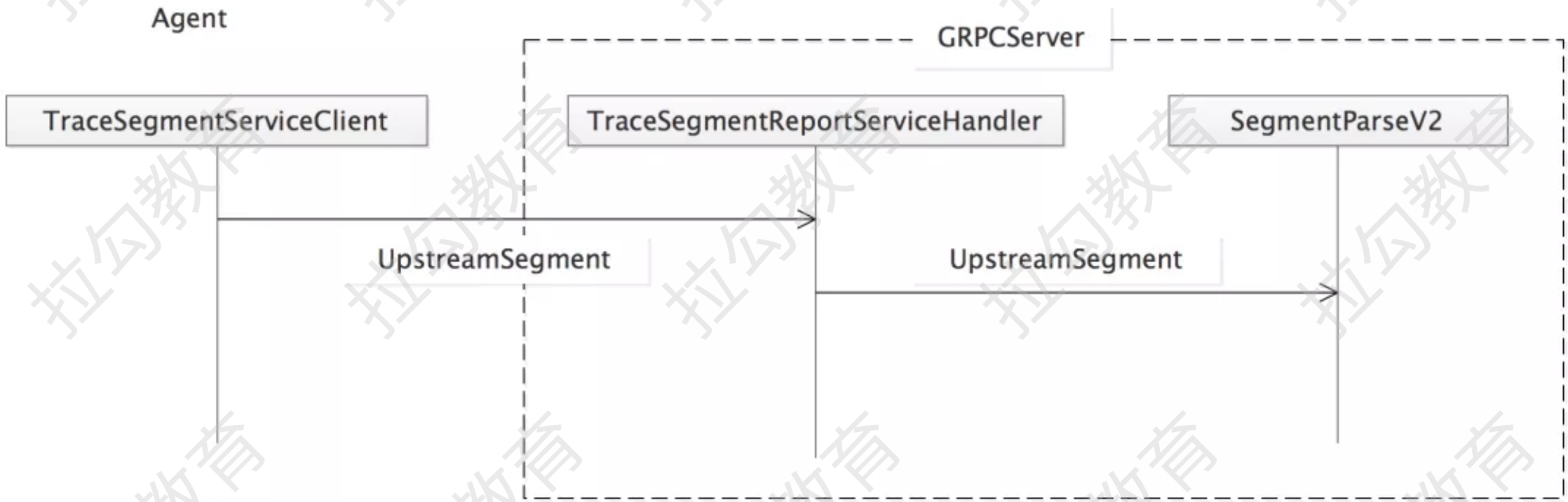
在 prepare() 方法中主要初始化 SegmentParseV2 解析器

SegmentParseV2 主要负责解析 TraceSegment 数据

拉勾教育

TraceModuleProvider

拉勾教育
互联网人实战大学



SegmentParseV2


```
//在解析过程中产生的 Segment 核心数据都会记录到 SegmentCoreInfo 中
private final SegmentCoreInfo segmentCoreInfo;

//在解析 TraceSegment 过程中会碰到不同类型的 Span
//会通过不同的 SpanListener 执行不同的操作
private final List<SpanListener> spanListeners;
```

拉勾教育

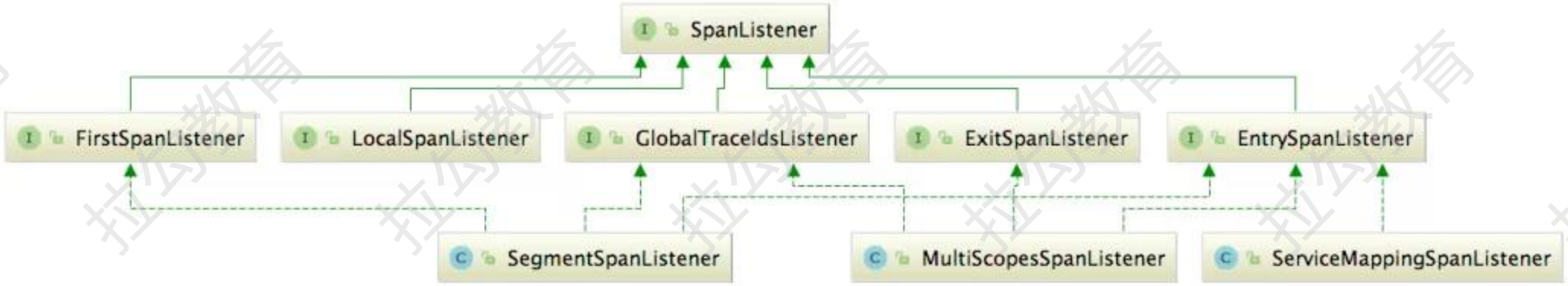
SegmentParseV2


```
// TraceSegment 编号，即 TraceSegment.traceSegmentId 。
private String segmentId;
private int serviceId; // Segment 所属的 Service 以及 ServiceInstance
private int serviceInstanceId;
private long startTime; // Segment 的开始时间和结束时间
private long endTime;
//如果 TraceSegment 范围内的任意一个 Span 被标记了 Error，则该字段会被设置为true
private boolean isError;
// TraceSegment 开始时间窗口(即第一个 Span 开始时间所处的分钟级时间窗口)
private long minuteTimeBucket;
//整个 TraceSegment 的字节数据
private byte[] dataBinary;
private boolean isV2; //是否为 V2 版本的 TraceSegment 数据
```

拉勾教育

SegmentParseV2

拉勾教育
互联网人实战大学



拉勾教育

SegmentParseV2

第二步会从 UpstreamSegment 中读取 TraceSegment 的数据，其中主要包括：

1. 与该 TraceSegment 关联的全部 TraceId

2. 反序列化 TraceSegment 关联的元数据以及 Span 数据，得到对应的 SegmentObject 对象

SegmentParseV2

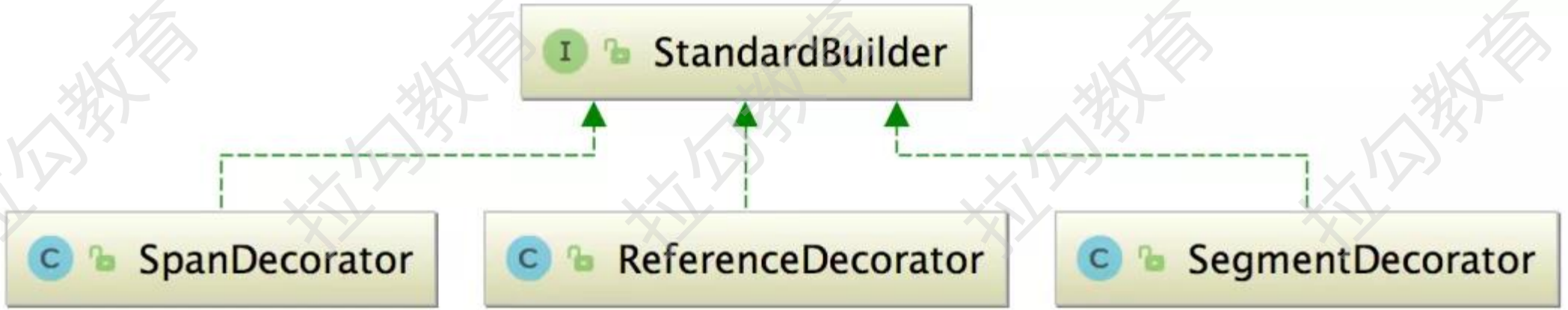
拉勾教育
互联网人实战大学

```
UpstreamSegment upstreamSegment = bufferData.getMessageType();
//获取该 TraceSegment 关联的全部 TraceId
List<UniqueId> traceIds = upstreamSegment.getGlobalTraceIdsList();
//反序列化 UpstreamSegment.segment，得到 SegmentObject 对象
SegmentObject segmentObject = parseBinarySegment(upstreamSegment);
//将 SegmentObject 封装成 SegmentDecorator
SegmentDecorator segmentDecorator = new SegmentDecorator(segmentObject);
```

拉勾教育

StandardBuilder

拉勾教育
互联网人实战大学



拉勾教育

StandardBuilder

拉勾教育
互联网人实战大学

```
//底层封装的 SpanObjectV2 对象
private SpanObjectV2 spanObjectV2;
// SpanObjectV2 关联的 Builder
private SpanObjectV2.Builder spanBuilderV2;
// Builder 是否已经创建
private boolean isOrigin = true;
```

拉勾教育

StandardBuilder

SpanDecorator 暴露出来的主要是 SpanObjectV2 相应字段的 getter/setter 方法：

1. 其 getter 方法底层调用 SpanObjectV2 或是 Builder 相应的 getter 方法读取相应字段数据
2. 其 setter 方法底层只能通过 Builder 的 setter 方法修改相应字段的数据

   （通过 isOrigin 字段确定 SpanBuilderV2 是否已经初始化，若未初始化则先进行初始化）

StandardBuilder

拉勾教育
互联网人实战大学

```
public void setComponentId(int value) {
  if (isOrigin) { //先检查 isOrigin，确定 spanBuilderV2 字段是否已经初始化
    toBuilder(); //初始化 spanBuilderV2 字段，即创建 SpanObjectV2 关联的 Builder 对象
  }
  //通过 Builder 完成更新（这里省略 V1 版本的相关代码）
  spanBuilderV2.setComponentId(value);
}
```

preBuild

预构建首先会通过 notifyGlobalsListener() 方法将该 TraceSegment 关联的全部 TraceId 交给 GlobalTraceIdsListener 进行解析

GlobalTraceIdsListener 接口只定义了一个 parseGlobalTraceId() 方法

SegmentSpanListener 、MultiScopesSpanListener 都是该接口的实现类

preBuild


```
long sampleValue = lastLong % 10000;
if (sampleValue < sampleRate) {
    return true;
}
return false;
```

拉勾教育

preBuild


```
segment.setTraceId(traceIdBuilder.toString()); // 记录traceId
```

拉勾教育

preBuild

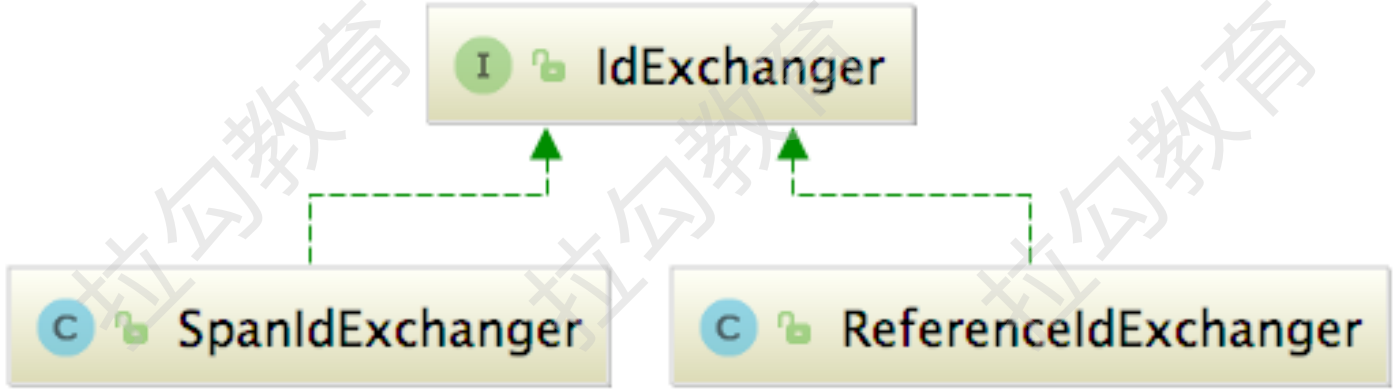
拉勾教育
互联网人实战大学

```
segment.setTraceId(traceIdBuilder.toString()); // 记录traceId
```

预构建接下来的步骤叫 "exchange"

该步骤主要实现字符串到 id 的转换

preBuild



拉勾教育

preBuild


```
private final ServiceInventoryCache serviceInventoryCacheDAO;
private final IServiceInventoryRegister serviceInventoryRegister;
private final IEndpointInventoryRegister endpointInventoryRegister;
private final INetworkAddressInventoryRegister
networkAddressInventoryRegister;
private final IComponentLibraryCatalogService
componentLibraryCatalogService;
```

拉勾教育

preBuild


```
//这里的 standardBuilder 就是前面介绍的 SpanDecorator，用于读写SpanObjectV2 中的数据
int peerId = standardBuilder.getPeerId();
//检测该 SpanObjectV2 的 peer 是否需要进行转换
if (peerId == 0 && !Strings.isNullOrEmpty(standardBuilder.getPeer())) {
  //通过 NetworkAddressInventoryRegister 获取 peer 对应的 id peerId=
  peerId = networkAddressInventoryRegister.getOrCreate(standardBuilder.getPeer(),
      buildServiceProperties(standardBuilder));

  if (peerId == 0) { //该 peer 字符串没有对应的id
    exchanged = false;
  } else { //记录 peerId，并清空 peer 字段
    standardBuilder.toBuilder();
    standardBuilder.setPeerId(peerId);
    standardBuilder.setPeer(Const.EMPTY_STRING);
  }
}
```

preBuild


```
for (int i = 0; i < segmentDecorator.getSpansCount(); i++) {
    SpanDecorator spanDecorator = segmentDecorator.getSpans(i);
    //针对 TraceSegment 中第一个 Span 的处理
    if (spanDecorator.getSpanId() == 0) {
        notifyFirstListener(spanDecorator);
    }
    //根据 SpanType 处理各个 Span
    if (SpanType.Exit.equals(spanDecorator.getSpanType())) {
        notifyExitListener(spanDecorator);
    } else if (SpanType.Entry.equals(spanDecorator.getSpanType())) {
        notifyEntryListener(spanDecorator);
    } else if (SpanType.Local.equals(spanDecorator.getSpanType())) {
        notifyLocalListener(spanDecorator);
    }
}
```

拉勾教育

notifyFirstListener

notifyFirstListener() 方法会调用所有 FirstSpanListener 的 parseFirst() 方法处理 TraceSegment 中的第一个 Span，这里只有 SegmentSpanListener 实现了该方法，具体实现分为三步：

1. 检测当前 TraceSegment 是否被成功采样
2. 将 segmentCoreInfo 中记录的 TraceSegment 数据拷贝到 segment 字段中
3. 将 endpointNameId 记录到 firstEndpointId 字段，endpointNameId 在 Spring MVC 里面对应的是 URL 在 Dubbo 里面对应的是 RPC 请求 path 与方法名称的拼接

notifyFirstListener

拉勾教育
互联网人实战大学

```
public void parseFirst(SpanDecorator spanDecorator, SegmentCoreInfo segmentCoreInfo){
    if (sampleStatus.equals(SAMPLE_STATUS.IGNORE)) {
        return; // 检测是否采样成功
    }
    long timeBucket = TimeBucket.getSecondTimeBucket(segmentCoreInfo.getStartTime());
    // 将 segmentCoreInfo 中记录的数据全部拷贝到 Segment 中
    segment.setSegmentId(segmentCoreInfo.getSegmentId());
    segment.setServiceId(segmentCoreInfo.getServiceId());
    segment.setServiceInstanceId(segmentCoreInfo.getServiceInstanceId());
    segment.setLatency((int)(segmentCoreInfo.getEndTime() - segmentCoreInfo.getStartTime()));
    segment.setStartTime(segmentCoreInfo.getStartTime());
    segment.setEndTime(segmentCoreInfo.getEndTime());
    segment.setIsError(BooleanUtils.booleanToValue(segmentCoreInfo.isError()));
    segment.setTimeBucket(timeBucket);
    segment.setDataBinary(segmentCoreInfo.getDataBinary());
    segment.setVersion(segmentCoreInfo.isV2() ? 2 : 1);
    // 记录 endpointNameId
    firstEndpointId = spanDecorator.getOperationNameId();
}
```

L / A / G / O / U

notifyEntryListener

拉勾教育
互联网人实战大学

在 notifyEntryListener() 方法中

会调用所有 EntrySpanListener 实现的 parseEntry() 方法对于 Entry 类型的 Span 进行处理

```
entryEndpointId = spanDecorator.getOperationNameId();
```

拉勾教育

notifyLocalListener

拉勾教育
互联网人实战大学

上述的三个 SpanListener 实现类中，**全部都没有实现 LocalSpanListener 接口**

所以在 trace-receiver-plugin 插件中并不会处理 Local 类型的 Span

notifyListenerToBuild

如果预构建（preBuild）中的 exchange 过程已经将全部字符串转换成了相应的 id

则会通过 notifyListenerToBuild() 方法调用所有 SpanListener 实现的 build() 方法

重点来看 SegmentSpanListener 的实现：

1. 首先会检测 TraceSegment 是否已被采样，它只会处理被采样的 TraceSegment
2. 设置 Segment 的 endpointName 字段
3. 将 Segment 交给 SourceReceiver 继续处理

RecordStreamProcessor

拉勾教育
互联网人实战大学

SourceReceiver 底层封装的 DispatcherManager

会根据 Segment 选择相应的 SourceDispatcher 实现 —— **SegmentDispatcher 进行分发**

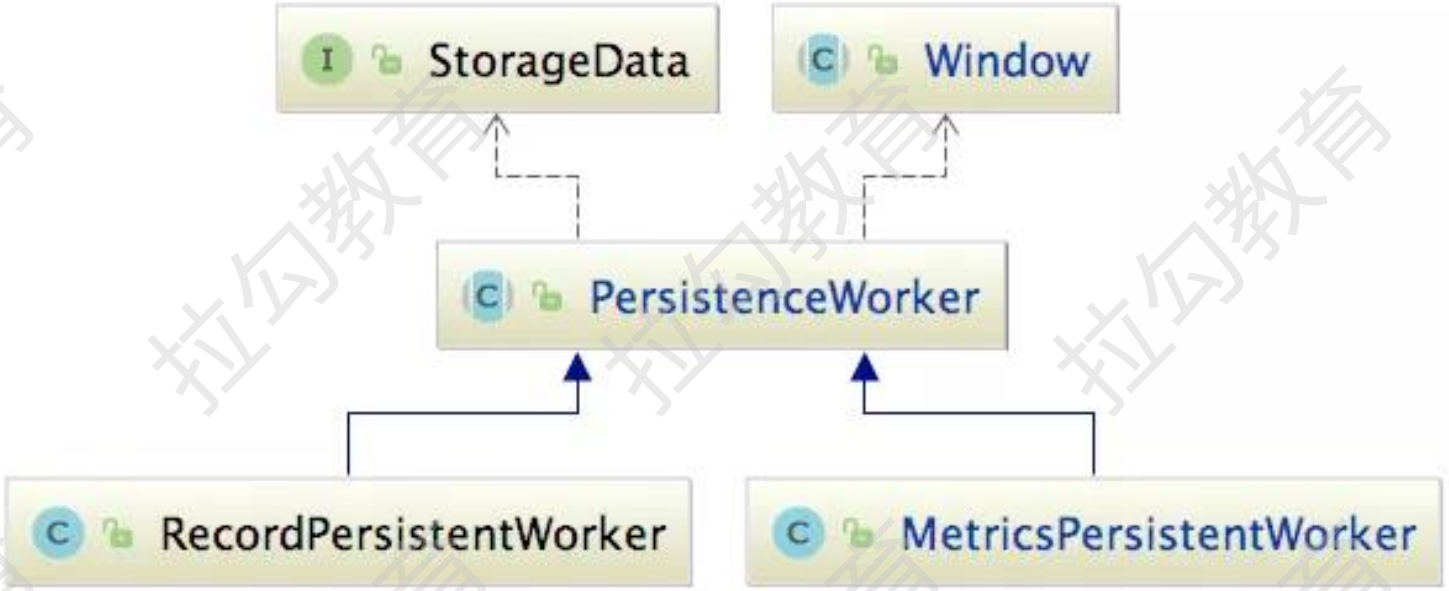
SegmentDispatcher.dispatch() 方法中会将 Segment 中的数据拷贝到 SegmentRecord 对象中

RecordStreamProcessor

拉勾教育
互联网人实战大学

```
// @Stream 注解的 name 属性指定了 index 的名称(index 前缀)，processor 指定了处理该类型数据的 StreamProcessor 实现
@Stream(name = "segment", processor = RecordStreamProcessor.class...)
public class SegmentRecord extends Record {
    // @Column 注解中指定了该字段在 index 中对应的 field 名称
    @Setter @Getter @Column(columnName = "segment_id") private String segmentId;
    @Setter @Getter @Column(columnName = "trace_id") private String traceId;
    @Setter @Getter @Column(columnName = "service_id") private int serviceId;
    @Setter @Getter @Column(columnName = "service_instance_id") private int serviceInstanceId;
    @Setter @Getter @Column(columnName = "endpoint_name, matchQuery = true) private String endpointName;
    @Setter @Getter @Column(columnName = "endpoint_id") private int endpointId;
    @Setter @Getter @Column(columnName = "start_time") private long startTime;
    @Setter @Getter @Column(columnName = "end_time") private long endTime;
    @Setter @Getter @Column(columnName = "latency") private int latency;
    @Setter @Getter @Column(columnName = "is_error") private int isError;
    @Setter @Getter @Column(columnName = "data_binary") private byte[] dataBinary;
    @Setter @Getter @Column(columnName = "version") private int version;
}
```

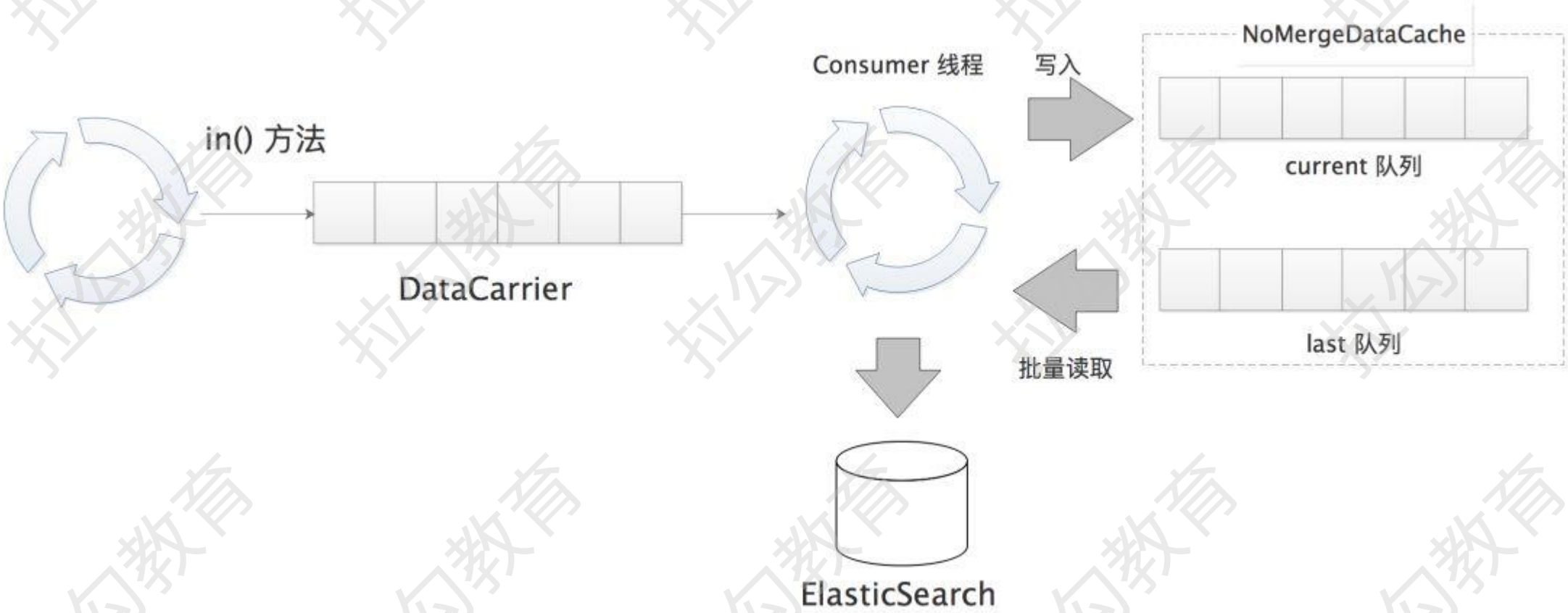
RecordStreamProcessor



拉勾教育

RecordStreamProcessor

拉勾教育
互联网人实战大学



拉勾教育

RecordStreamProcessor

RecordPersistentWorker 有两个地方与 MetricsPersistentWorker 有些区别：

1. RecordPersistentWorker 中使用的 DataCache（以及 Window）实现是 NoMergeDataCache

   它与 MergeDataCache 的唯一区别就是没有提供判断数据是否存在的 containKey() 方法

   这样就只提供了缓存数据的功能，调用方无法合并重复数据

2. 当 NoMergeDataCache 中缓存的数据到达阈值之后，RecordPersistentWorker 会通过 RecordDAO 生成批量的 IndexRequest 请求，Trace 数据没有合并的情况

   所以 RecordDAO 以及 IRecordDAO 接口没有定义 prepareBatchUpdate() 方法

RecordStreamProcessor

拉勾教育
互联网人实战大学

```
public IndexRequest prepareBatchInsert(Model model, Record record) throws IOException {
    XContentBuilder builder = map2builder(storageBuilder.data2Map(record));
    //生成的是最终 Index 名称，这里的 Index 由前缀字符串(即"segment")+TimeBucket 两部分构成
    String modelName = TimeSeriesUtils.timeSeries(model, record.getTimeBucket());
    //创建 IndexRequest 请求
    return getClient().prepareInsert(modelName, record.id(), builder);
}
```

Next：第26讲《trace-receiver 插件拆解，Trace 蕴含的宝贵信息（下）》

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

L / A / G / O / U