《31 讲带你搞懂 SkyWalking》

徐郡明 资深技术专家

— 拉勾教育出品 —

第28讲：深入 query-graphql 插件 SW Rocketbot 背后的英雄（上）

前言

SkyWalking OAP 目前只提供了query-graphql-plugin 这一款查询插件

从名字就可以看出它是使用 GraphQL 实现的查询 API

启动逻辑

1. 通过 GraphQL Java Tools 实现 GraphQL Schema 与 POJO 之间的映射

   创建相应的 GraphQLSchema 对象

2. 通过 GraphQL Java API 创建 GraphQL 对象，它将处理"/graphql"路径上的全部请求

3. 创建 GraphQLQueryHandler 实例并注册到 JettyServer

   GraphQLQueryHandler 会将收到的 Http 请求进行一次转换，并交给 GraphQL 对象进行处理

启动逻辑

```java
public class GraphQLQueryProvider extends ModuleProvider {
  private final GraphQLQueryConfig config = new GraphQLQueryConfig();
  private GraphQL graphQL;

  @Override public void prepare() throws ServiceNotProvidedException,
ModuleStartException {
      GraphQLSchema schema = SchemaParser.newParser()
        .file("query-protocol/common.graphqls")
        .resolvers(new Query(), new Mutation())
        ... ... //这里会添加所有 GraphQL Schema以及关联的 Resolver实现，后面会挑选几个展开详述
        .build() .makeExecutableSchema();
      // 创建 GraphQL 对象，GraphQL Java提供的API
      this.graphQL = GraphQL.newGraphQL(schema).build();
  }

  @Override public void start() throws ServiceNotProvidedException, ModuleStartException {
      //创建 GraphQLQueryHandler实例并注册到JettyServer中
      JettyHandlerRegister service =
```

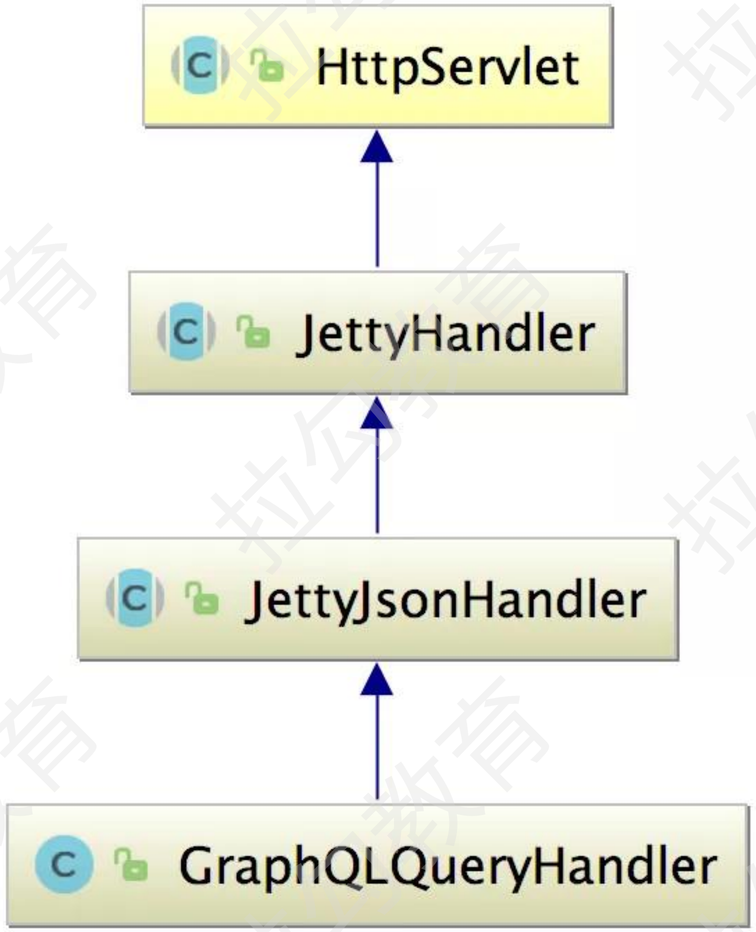
启动逻辑

```
ModuleStartException {
    GraphQLSchema schema = SchemaParser.newParser()
      .file("query-protocol/common.graphqls")
      .resolvers(new Query(), new Mutation())
      ... ... //这里会添加所有 GraphQL Schema以及关联的 Resolver实现，后面会挑选几个展开详述
      .build().makeExecutableSchema();
    // 创建 GraphQL 对象，GraphQL Java提供的API
    this.graphQL = GraphQL.newGraphQL(schema).build();
  }

  @Override public void start() throws ServiceNotProvidedException, ModuleStartException {
    // 创建 GraphQLQueryHandler实例并注册到JettyServer中
    JettyHandlerRegister service =
getManager().find(CoreModule.NAME).provider().getService(JettyHandlerRegister.class);
    // 这里的path在 application.yml中的query部分有相应配置项，默认是"/graphql"
    service.addHandler(new GraphQLQueryHandler(config.getPath(), graphQL));
  }
}
```

GraphQLQueryHandler

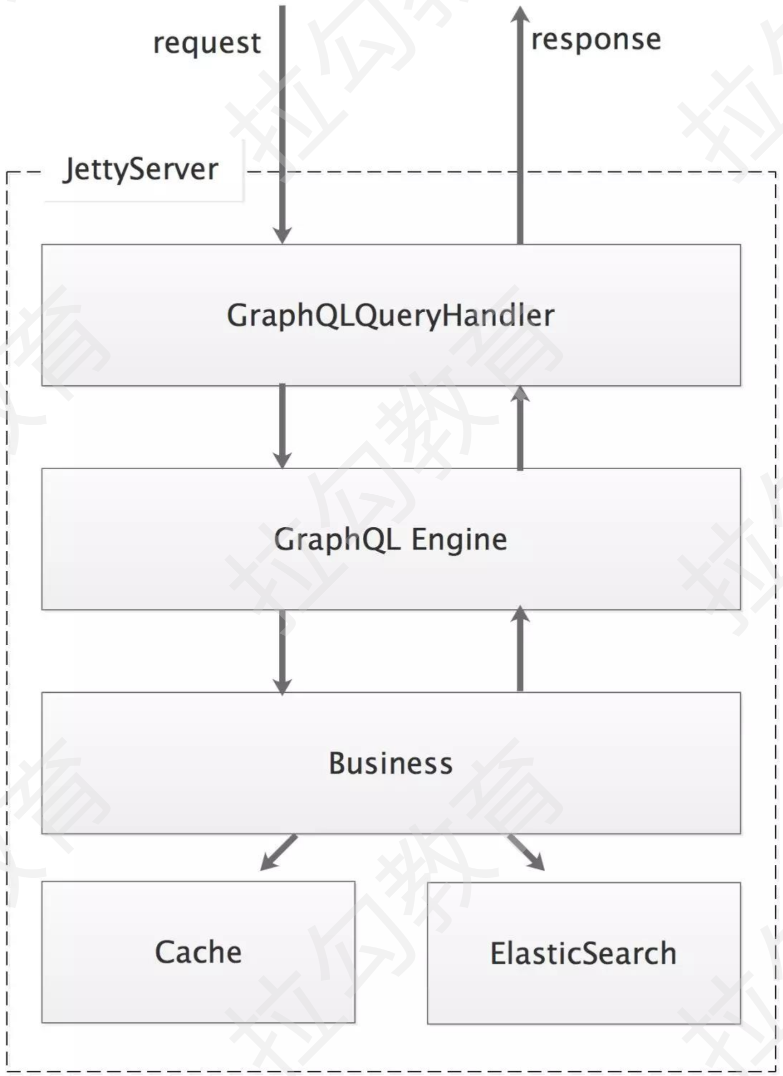
拉勾教育
— 互联网人实战大学 —



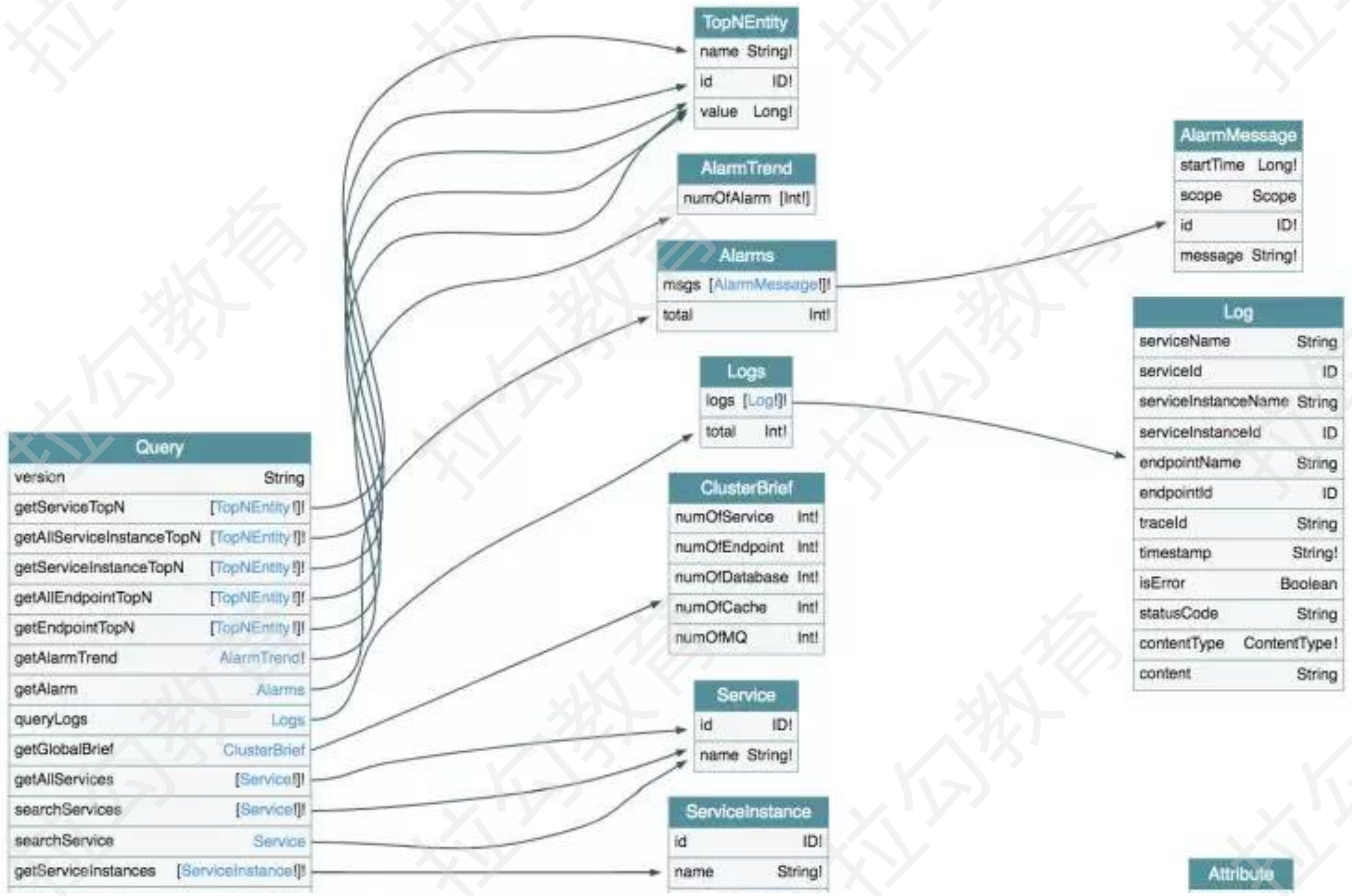
GraphQLQueryHandler

```
protected JsonElement doPost(HttpServletRequest req) throws IOException {
  BufferedReader reader = new BufferedReader(new InputStreamReader(req.getInputStream()));
  StringBuilder request = new StringBuilder();
  //省略读取reader的过程
  JsonObject requestJson = gson.fromJson(request.toString(), JsonObject.class);
  ExecutionInput executionInput = ExecutionInput.newExecutionInput()
    .query(requestJson.get(QUERY).getAsString())
    .variables(gson.fromJson(requestJson.get(VARIABLES), mapOfStringObjectType))
    .build();
  //在前文的示例中，Spring Boot 帮我们屏蔽了 execute()方法的调用，这里需要自己通过GraphQL
Java API进行调用
  ExecutionResult executionResult = graphQL.execute(executionInput);
  Object data = executionResult.getData(); // ???è????
  List<GraphQLError> errors = executionResult.getErrors(); // ??ä???
  JsonObject jsonObject = new JsonObject();
  //将正常查询结果记录到"data"字段，将异常信息记录到"error"(略)
  return jsonObject;
}
```

GraphQLQueryHandler

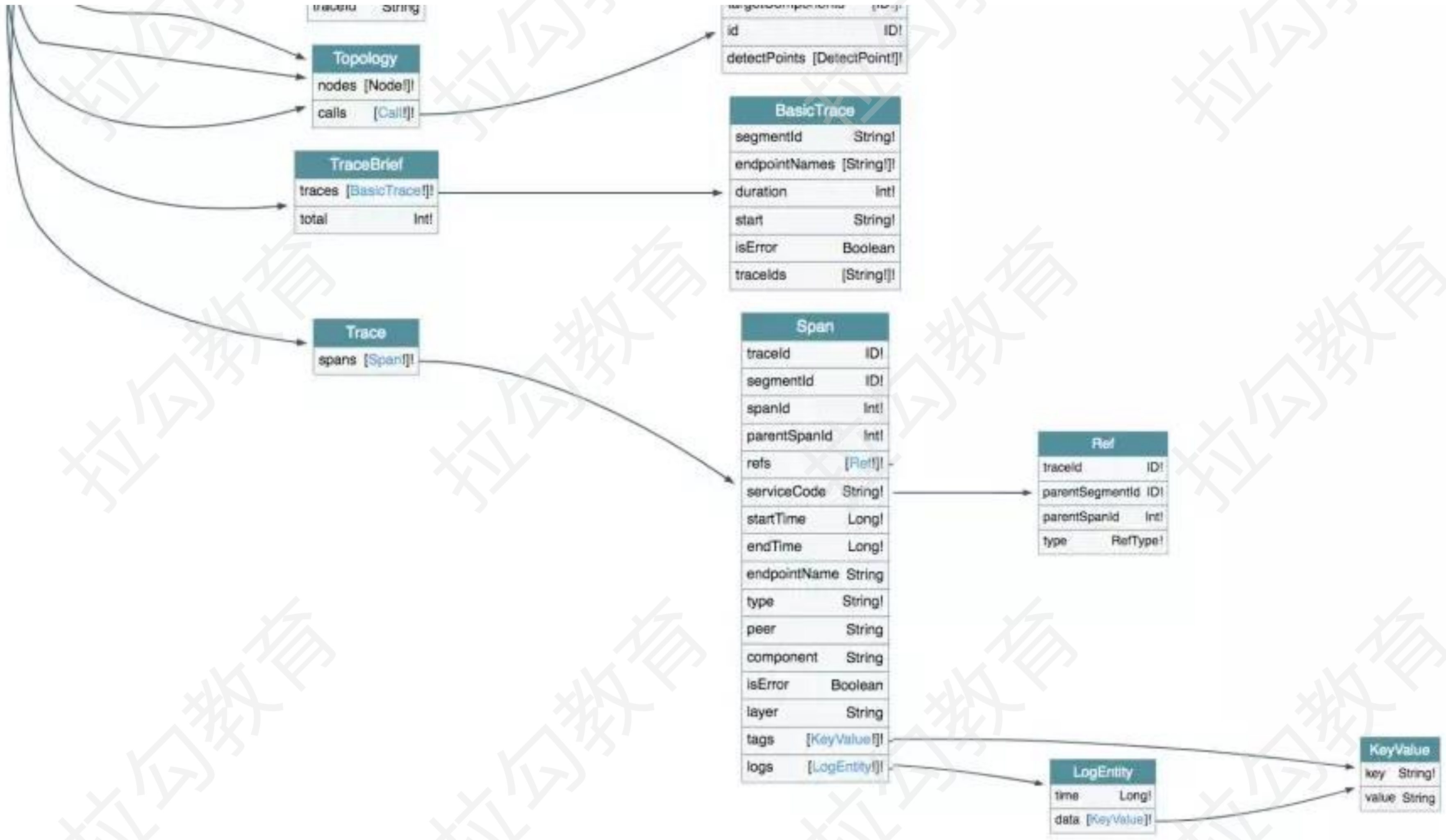


GraphQL Schema 鸟瞰

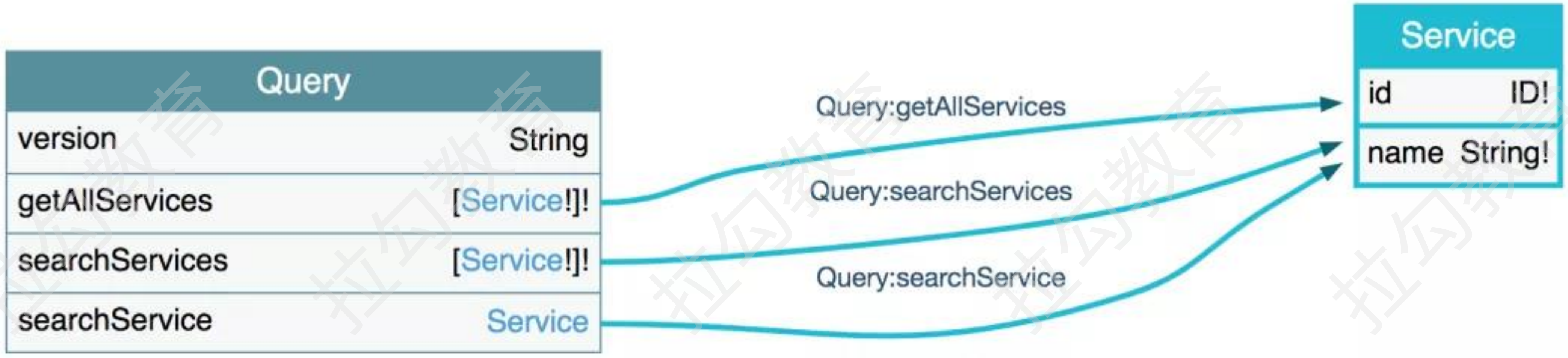


GraphQL Schema 鸟瞰

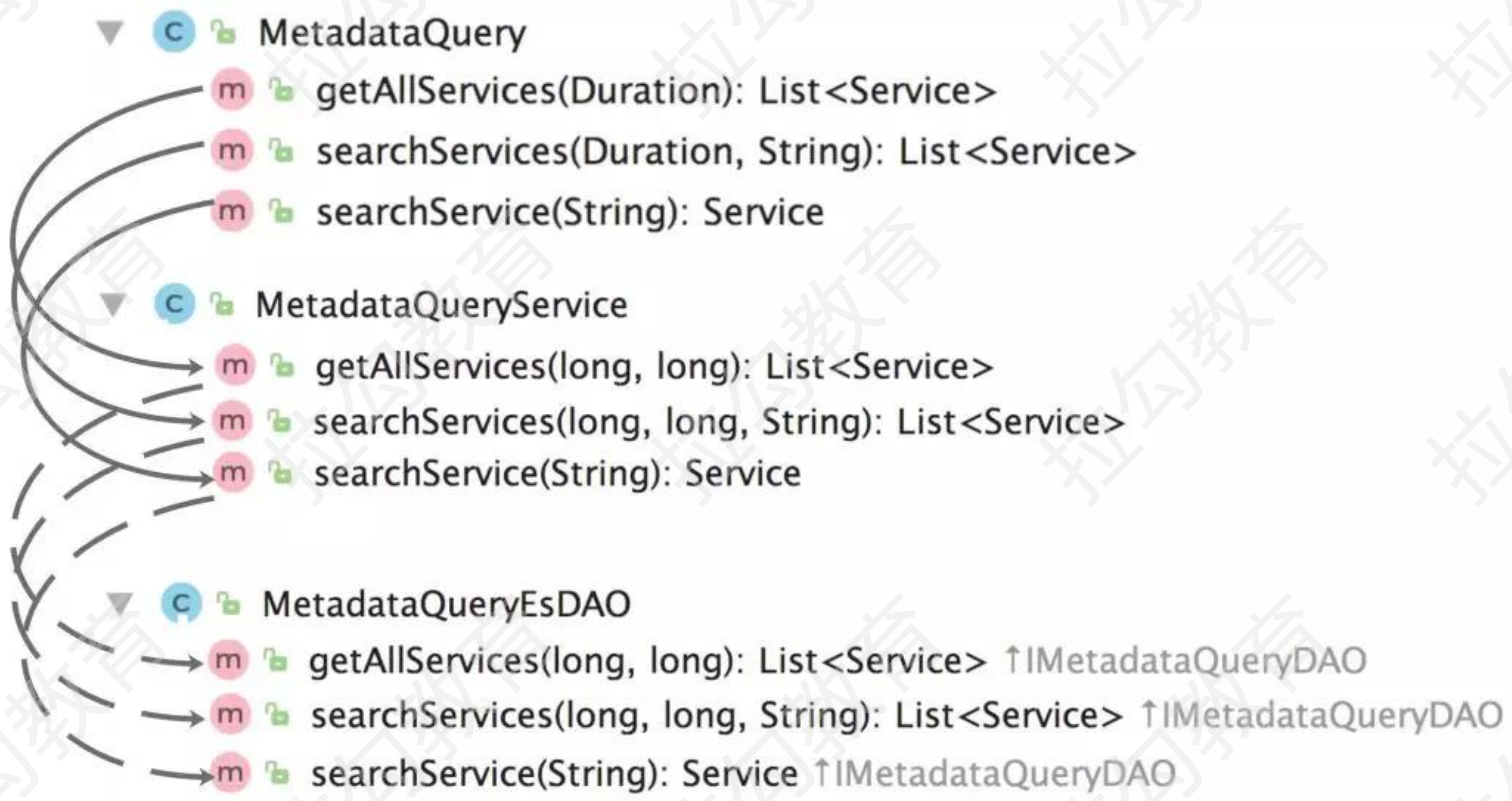
拉勾教育
— 互联网人实战大学 —



MetadataQuery



MetadataQuery



MetadataQuery
getAllServices(Duration): List<Service>
searchServices(Duration, String): List<Service>
searchService(String): Service

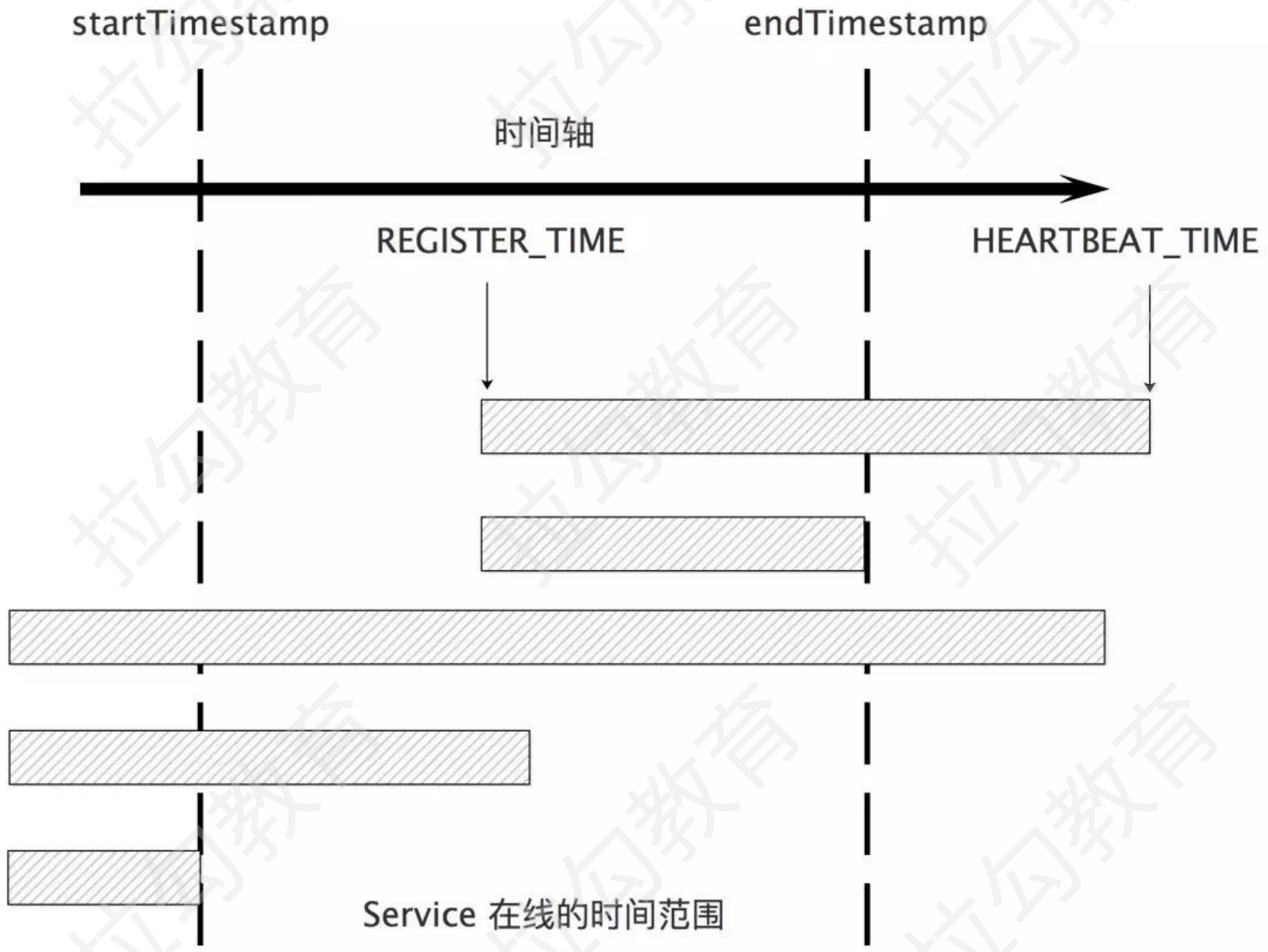
MetadataQueryService
getAllServices(long, long): List<Service>
searchServices(long, long, String): List<Service>
searchService(String): Service

MetadataQueryEsDAO
getAllServices(long, long): List<Service> ↑IMetadataQueryDAO
searchServices(long, long, String): List<Service> ↑IMetadataQueryDAO
searchService(String): Service ↑IMetadataQueryDAO

MetadataQuery

```java
public List<Service> searchServices(long startTimestamp, long endTimestamp,
    String keyword) throws IOException {
    SearchSourceBuilder sourceBuilder = SearchSourceBuilder.searchSource();
    BoolQueryBuilder boolQueryBuilder = QueryBuilders.boolQuery();
    //查询的时间范围
    boolQueryBuilder.must().add(timeRangeQueryBuild(startTimestamp, endTimestamp));
    //不查询 NetWorkAddress在service_inventory索引中的数据
    boolQueryBuilder.must().add(QueryBuilders.termQuery(ServiceInventory.IS_ADDRESS,
BooleanUtils.FALSE));
    if (!Strings.isNullOrEmpty(keyword)) {
      // serviceName匹配用户指定的关键字(keyword)
      String matchCName = MatchCNameBuilder.INSTANCE.build(ServiceInventory.NAME);
      boolQueryBuilder.must().add(QueryBuilders.matchQuery(matchCName, keyword));
    }
    sourceBuilder.query(boolQueryBuilder);
    sourceBuilder.size(queryMaxSize); //查询返回Document的个数上限，默认上限5000个
    //通过 RestHighLevelClient 执行SearchRequest查询
    SearchResponse response = getClient().search(ServiceInventory.INDEX_NAME, sourceBuilder);
    //从 SearchResponse响应中获取相应的 Service信息并返回
    return buildServices(response);
}
```

MetadataQuery



MetadataQuery

拉勾教育
— 互 联 网 人 实 战 大 学 —

- **查询 ServiceInstance**

  getServiceInstances() 方法可以按照时间范围和 serviceId 查询关联的 ServiceInstance 集合

- **查询 Endpoint**

  - searchEndpoint() 方法会根据 serviceId 以及关键字查询相应的 Endpoint 集合
  - getEndpointInfo() 方法会根据指定的 endpointId 查询 Endpoint 信息

MetadataQuery

- **查询 Databases**

  getAllDatabases() 方法其实也是查询 Service，只不过指定了 node_type 字段的取值为 Database 而已

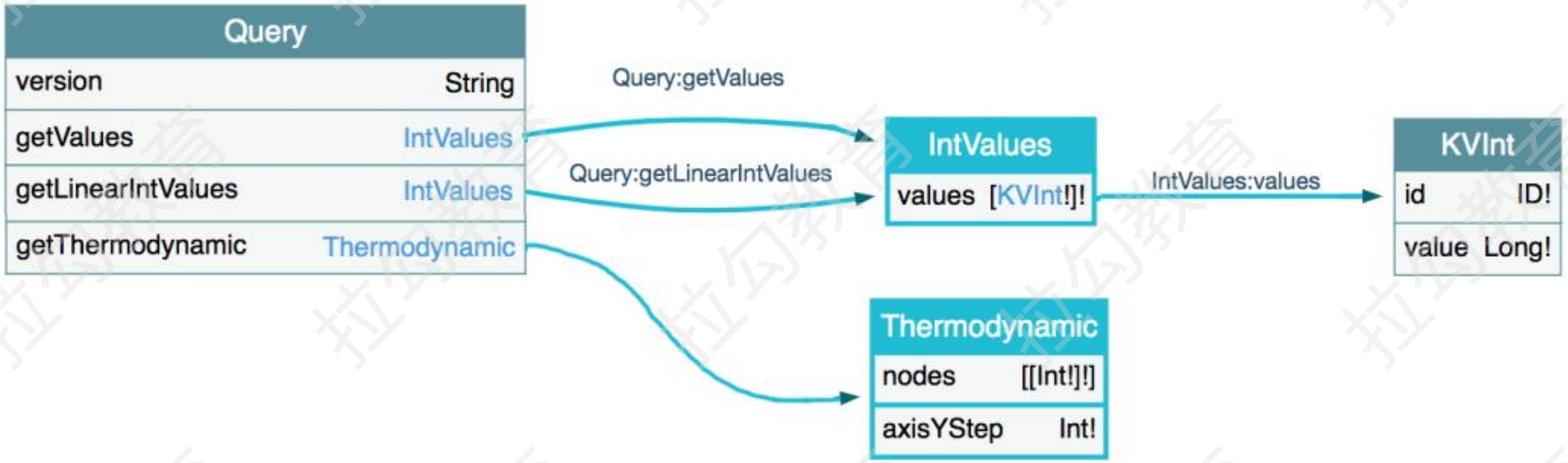
- **查询 ClusterBrief**

  getGlobalBrief() 方法会按照时间范围查询整个 OAP 集群所能感知到的各类组件的个数

  然后封装成 ClusterBrief 对象返回

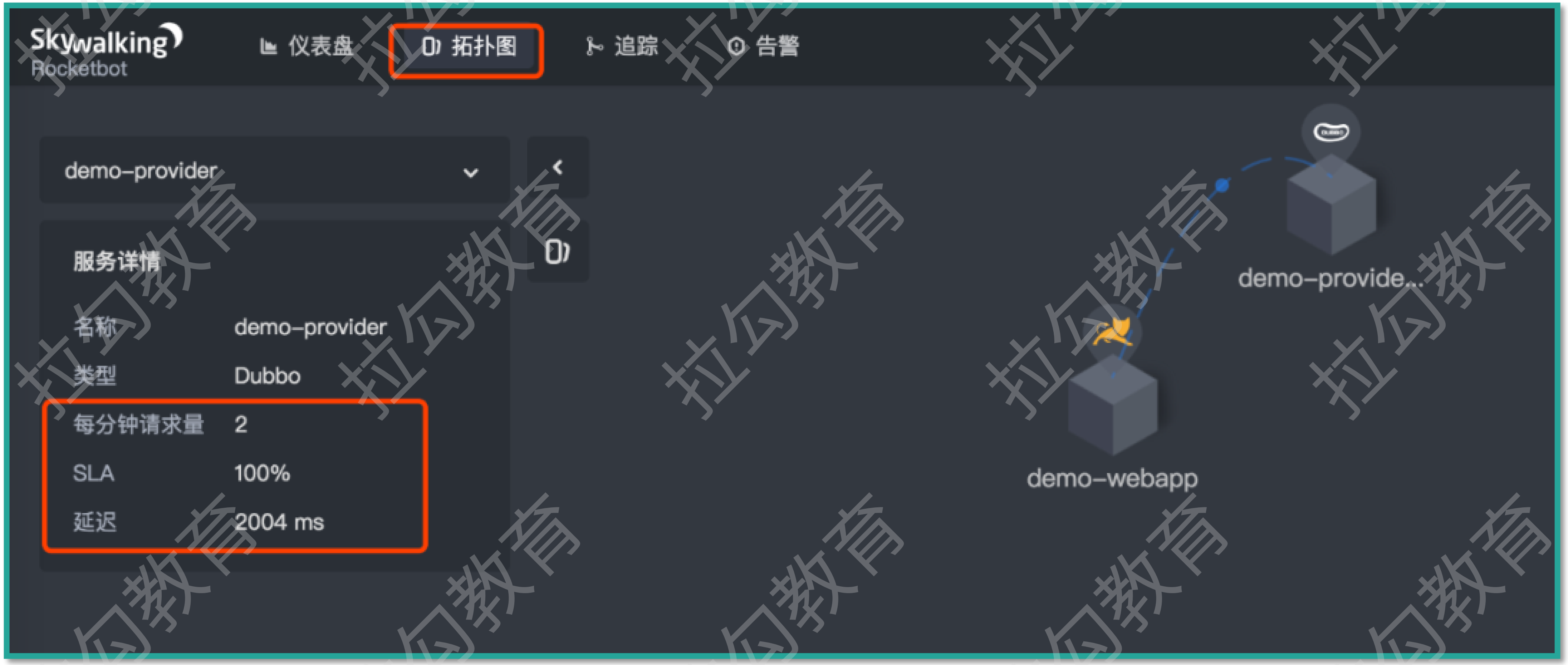
  在 ClusterBrief 中包括 Service 数量、 Endpoint 数量、Database 数量、Cache 数量以及 MQ 数量

MetricQuery



查询单个聚合值

拉勾教育
— 互 联 网 人 实 战 大 学 —



demo–provider

服务详情

名称	demo–provider
类型	Dubbo
每分钟请求量	2
SLA	100%
延迟	2004 ms

查询单个聚合值

拉勾教育
互联网人实战大学

指定 time_bucket 字段的时间范围，即 time_bucket 字段值必须在 startTB 和 endTB 之间

```
RangeQueryBuilder rangeQueryBuilder =
QueryBuilders.rangeQuery(Metrics.TIME_BUCKET).gte(startTB).lte(endTB);
BoolQueryBuilder boolQuery = QueryBuilders.boolQuery();
boolQuery.must().add(rangeQueryBuilder);
```

查询单个聚合值

拉勾教育

精确匹配 Document 中的 entity_id 字段值，示例中 entity_id 字段分别为 2 和 3

```
where.getKeyValues().forEach(keyValues -> {
  if (keyValues.getValues().size() > 1) {
    boolQuery.must().add(QueryBuilders.termsQuery(keyValues.getKey(),
keyValues.getValues()));
  } else {
    boolQuery.must().add(QueryBuilders.termQuery(keyValues.getKey(),
keyValues.getValues().get(0)));
  }
});
```

查询单个聚合值

拉勾教育
— 互 联 网 人 实 战 大 学 —

按照 entity_id 分组聚合查询到的 SLA 值（即 Document 中的 percentage 字段）
具体聚合方式是计算平均值

```
TermsAggregationBuilder entityIdAggregation =

AggregationBuilders.terms(Metrics.ENTITY_ID).field(Metrics.ENTITY_ID).size(1000);

parentAggBuilder.subAggregation(AggregationBuilders.avg(valueCName).field(valueCName));
```

查询单个聚合值

拉勾教育
— 互联网人实战大学 —

将上述构造的查询条件和聚合函数构造成 SearchRequest 请求发送给 ElasticSearch 集群完成查询

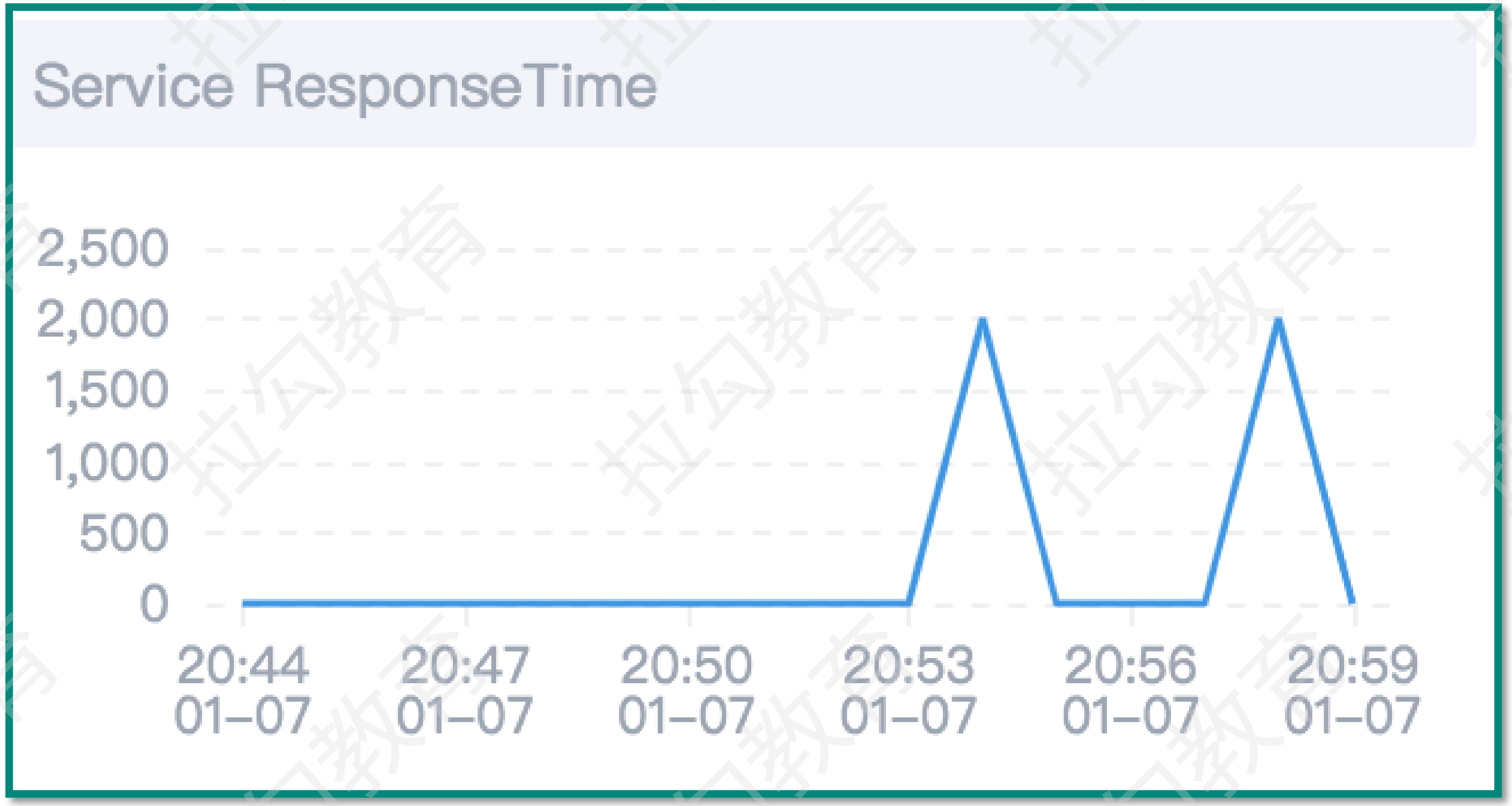
```
SearchSourceBuilder sourceBuilder = SearchSourceBuilder.searchSource();
sourceBuilder.query(boolQuery);
sourceBuilder.size(0);
sourceBuilder.aggregation(entityIdAggregation);
SearchResponse response = getClient().search(indexName, sourceBuilder);
```

查询单个聚合值

解析 SearchResponse 得到查询结果，即示例中每个 Service 的 SLA 平均值

```
IntValues intValues = new IntValues();
Terms idTerms = response.getAggregations().get(Metrics.ENTITY_ID);
for (Terms.Bucket idBucket : idTerms.getBuckets()) {
    KVInt kvInt = new KVInt();
    // key为 entity_id，即示例中的serviceId
    kvInt.setId(idBucket.getKeyAsString());
    // value为该 entity_id对应的 SLA平均值
    kvInt.setValue(idBucket.getAggregations().get(valueCName).getValue());
    intValues.getValues().add(kvInt); //记录上述查询解析结果
}
return intValues;
```

查询时序


Service ResponseTime
2,500
2,000
1,500
1,000
500
0
20:44 01–07
20:47 01–07
20:50 01–07
20:53 01–07
20:56 01–07
20:59 01–07

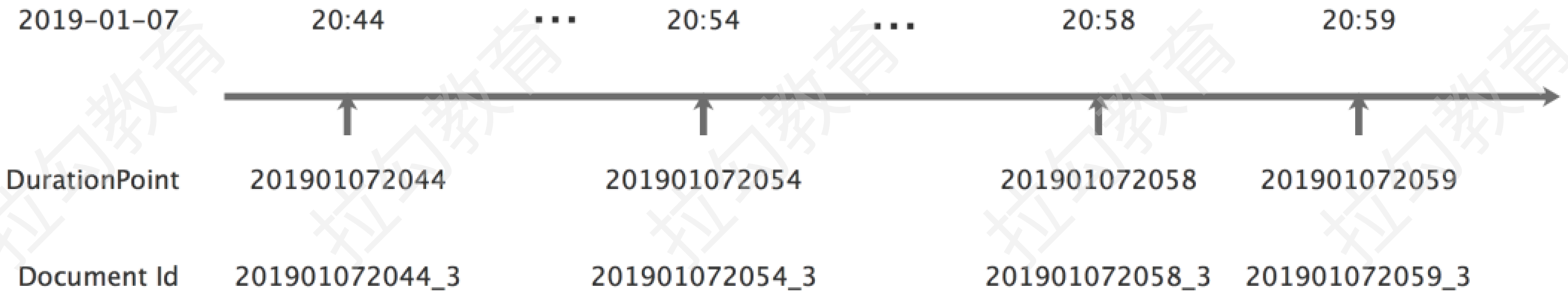
查询时序

拉勾教育

— 互 联 网 人 实 战 大 学 —

```
//按照 DownSampling单位以及查询时间范围，确定有多少个Document需要查询
List<DurationPoint> durationPoints =
DurationUtils.INSTANCE.getDurationPoints(downsampling, startTB, endTB);
List<String> ids = new ArrayList<>();
//构造每个 DurationPoint对应的 Document Id
durationPoints.forEach(durationPoint -> ids.add(durationPoint.getPoint() +
Const.ID_SPLIT + id));
```

查询时序


2019-01-07
20:44
...
20:54
...
20:58
20:59
DurationPoint
201901072044
201901072054
201901072058
201901072059
Document Id
201901072044_3
201901072054_3
201901072058_3
201901072059_3

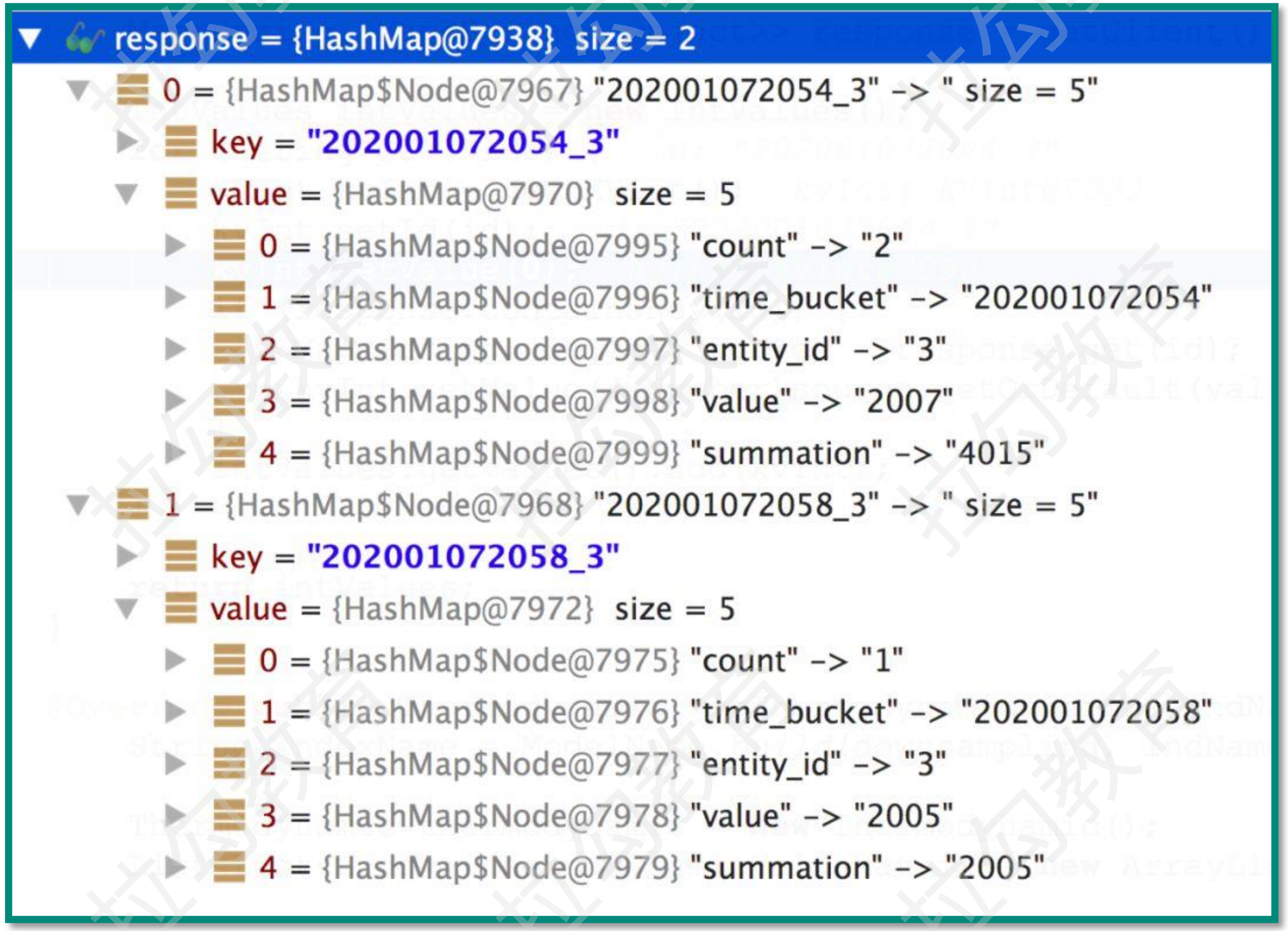
查询时序

拉勾教育
— 互 联 网 人 实 战 大 学 —

```
SearchRequest searchRequest = new SearchRequest(indexName);
searchRequest.types(TYPE);
//指定查询的 Document Id
searchRequest.source().query(QueryBuilders.idsQuery().addIds(ids)).size(ids.length);
SearchResponse response = client.search(searchRequest);
//将返回的 SearchResponse转换成 Map后返回，第一层 Key是DocumentId，第二层 Key是 Field名称，
第二层 Value是字段对应的 Value值
Map<String, Map<String, Object>> result = new HashMap<>();
SearchHit[] hits = response.getHits().getHits();
for (SearchHit hit : hits) {
  result.put(hit.getId(), hit.getSourceAsMap());
}
return result;
```

查询时序

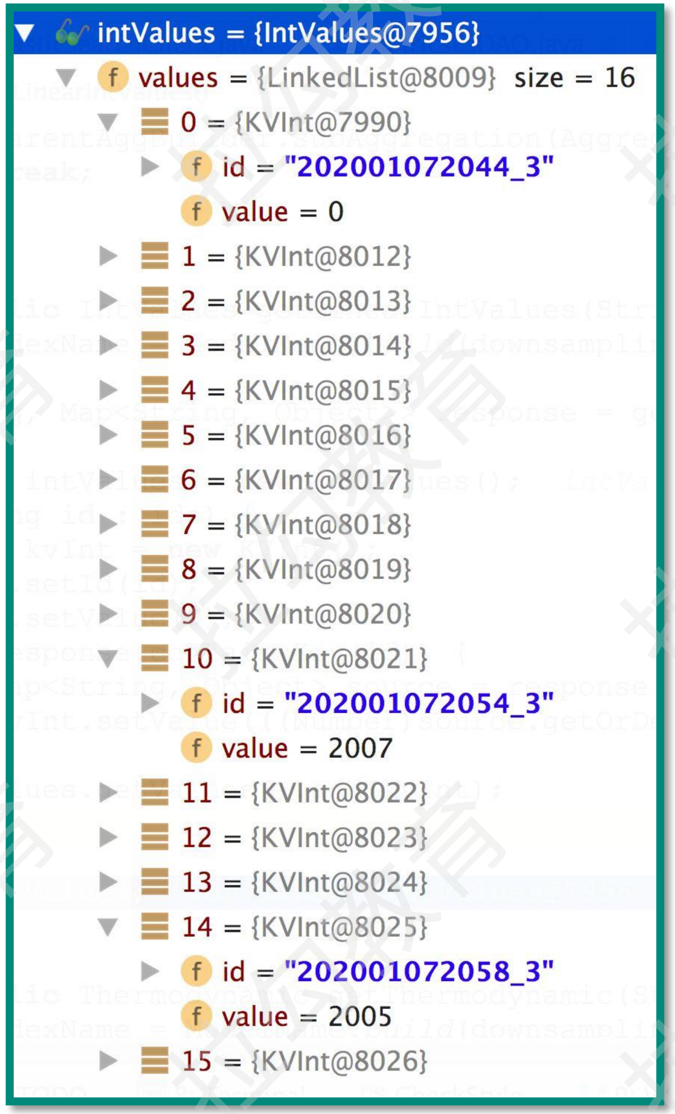
拉勾教育
— 互 联 网 人 实 战 大 学 —



```
▼ response = {HashMap@7938}  size = 2
    ▼ 0 = {HashMap$Node@7967} "202001072054_3" -> " size = 5"
        ▶ key = "202001072054_3"
        ▼ value = {HashMap@7970}  size = 5
            ▶ 0 = {HashMap$Node@7995} "count" -> "2"
            ▶ 1 = {HashMap$Node@7996} "time_bucket" -> "202001072054"
            ▶ 2 = {HashMap$Node@7997} "entity_id" -> "3"
            ▶ 3 = {HashMap$Node@7998} "value" -> "2007"
            ▶ 4 = {HashMap$Node@7999} "summation" -> "4015"
    ▼ 1 = {HashMap$Node@7968} "202001072058_3" -> " size = 5"
        ▶ key = "202001072058_3"
        ▼ value = {HashMap@7972}  size = 5
            ▶ 0 = {HashMap$Node@7975} "count" -> "1"
            ▶ 1 = {HashMap$Node@7976} "time_bucket" -> "202001072058"
            ▶ 2 = {HashMap$Node@7977} "entity_id" -> "3"
            ▶ 3 = {HashMap$Node@7978} "value" -> "2005"
            ▶ 4 = {HashMap$Node@7979} "summation" -> "2005"
```

查询时序

拉勾教育
— 互联网人实战大学 —



```
▼ intValues = {IntValues@7956}
  ▼ f values = {LinkedList@8009}  size = 16
    ▼ 0 = {KVInt@7990}
      ▶ f id = "202001072044_3"
        f value = 0
    ▶ 1 = {KVInt@8012}
    ▶ 2 = {KVInt@8013}
    ▶ 3 = {KVInt@8014}
    ▶ 4 = {KVInt@8015}
    ▶ 5 = {KVInt@8016}
    ▶ 6 = {KVInt@8017}
    ▶ 7 = {KVInt@8018}
    ▶ 8 = {KVInt@8019}
    ▶ 9 = {KVInt@8020}
    ▼ 10 = {KVInt@8021}
      ▶ f id = "202001072054_3"
        f value = 2007
    ▶ 11 = {KVInt@8022}
    ▶ 12 = {KVInt@8023}
    ▶ 13 = {KVInt@8024}
    ▼ 14 = {KVInt@8025}
      ▶ f id = "202001072058_3"
        f value = 2005
    ▶ 15 = {KVInt@8026}
```

Next：第29讲《深入 query-graphql 插件SW Rocketbot 背后的英雄（下）》