


# Skywalking-06：OAL基础

 Switch 发布于 8 月 16 日

## OAL 基础知识

### 基本介绍

OAL(Observability Analysis Language) 是一门用来分析流式数据的语言。

因为 OAL 聚焦于度量 Service 、 Service Instance 和 Endpoint 的指标，所以它学习和使用起来非常简单。

OAL 基于 altlr 与 javassist 将 oal 脚本转化为动态生成的类文件。

自从 6.3 版本后， OAL 引擎内置在 OAP 服务器中，可以看做 oal-rt(OAL Runtime) 。 OAL 脚本位置 OAL 配置目录下( /config/oal )，使用者能够更改脚本并重启生效。注意： OAL 脚本仍然是一门编译语言， oal-rt 动态的生成 Java 代码。

如果你配置了环境变量 SW\_OAL\_ENGINE\_DEBUG=Y，能在工作目录下的 oal-rt 目录下找到生成的 Class 文件。

### 语法

```
// 声明一个指标
METRICS_NAME = from(SCOPE.(* | [FIELD][,FIELD ...])) // 从某一个SCOPE中获取数据
[.filter(FIELD OP [INT | STRING])] // 可以过滤掉部分数据
.FUNCTION([PARAM][, PARAM ...]) // 使用某个聚合函数将数据聚合

// 禁用一个指标
disable(METRICS_NAME);
```

### 语案例

oap-server/server-bootstrap/src/main/resources/oal/java-agent.oal

```
// 从ServiceInstanceJVMMemory的used获取数据，只需要 heapStatus 为 true的数据，并取long型的平均值
instance_jvm_memory_heap = from(ServiceInstanceJVMMemory.used).filter(heapStatus == true).longAvg();
```

org.apache.skywalking.oap.server.core.source.ServiceInstanceJVMMemory

```
@ScopeDeclaration(id = SERVICE_INSTANCE_JVM_MEMORY, name = "ServiceInstanceJVMMemory", catalog =
SERVICE_INSTANCE_CATALOG_NAME)
@ScopeDefaultColumn.VirtualColumnDefinition(fieldName = "entityId", columnName = "entity_id", isID = true, type =
String.class)
public class ServiceInstanceJVMMemory extends Source {
    @Override
    public int scope() {
        return DefaultScopeDefine.SERVICE_INSTANCE_JVM_MEMORY;
    }

    @Override
    public String getEntityId() {
        return String.valueOf(id);
    }

    @Getter @Setter
    private String id;
    @Getter @Setter @ScopeDefaultColumn.DefinedByField(columnName = "name", requireDynamicActive = true)
    private String name;
    @Getter @Setter @ScopeDefaultColumn.DefinedByField(columnName = "service_name", requireDynamicActive = true)
    private String serviceName;
    @Getter @Setter @ScopeDefaultColumn.DefinedByField(columnName = "service_id")
    private String serviceId;
    @Getter @Setter
    private boolean heapStatus;
    @Getter @Setter
```

可供参考的官方文档：[Observability Analysis Language](#)

# 从一个案例开始分析 OAL 原理

## 缺少的类加载信息监控

默认的 `APM/Instance` 页面，缺少关于 `JVM Class` 的信息（如下图所示），故这次将相关信息补齐。由这次案例来分析 `OAL` 的原理。

在 [Skywalking-04：扩展Metric监控信息](#) 中，讲到了如何在已有 `Source` 类的情况下，增加一些指标。

这次直接连 `Source` 类以及 `OAL` 词法语法关键字都自己定义。

可供参考的官方文档：[Source and Scope extension for new metrics](#)

## 确定增加的指标

通过[Java ManagementFactory解析](#)这篇文章，可以确定监控指标为“当前加载类的数量”、“已卸载类的数量”、“一共加载类的数量”三个指标

```
ClassLoadingMXBean classLoadingMXBean = ManagementFactory.getClassLoadingMXBean();
// 当前加载类的数量
int loadedClassCount = classLoadingMXBean.getLoadedClassCount();
// 已卸载类的数量
long unloadedClassCount = classLoadingMXBean.getUnloadedClassCount();
// 一共加载类的数量
long totalLoadedClassCount = classLoadingMXBean.getTotalLoadedClassCount();
```

## 定义 agent 与 oap server 通讯类

在 `apm-protocol/apm-network/src/main/proto/language-agent/JVMMetric.proto` 协议文件中增加如下定义。

在 `apm-protocol/apm-network` 目录下执行 `mvn clean package -DskipTests=true` 会生成新的相关 Java 类, `org.apache.skywalking.apm.network.language.agent.v3.Class` 该类就是我们在代码中实际操作的类。

```
message Class {
  int64 loadedClassCount = 1;
  int64 unloadedClassCount = 3;
  int64 totalLoadedClassCount = 2;
}

message JVMMetric {
  int64 time = 1;
  CPU cpu = 2;
  repeated Memory memory = 3;
  repeated MemoryPool memoryPool = 4;
  repeated GC gc = 5;
  Thread thread = 6;
  // 在JVM指标中添加Class的定义
  Class clazz = 7;
}
```

## 收集 agent 的信息后，将信息发送至 oap server

收集 `Class` 相关的指标信息

```
package org.apache.skywalking.apm.agent.core.jvm.clazz;

import org.apache.skywalking.apm.network.language.agent.v3.Class;

import java.lang.management.ClassLoadingMXBean;
import java.lang.management.ManagementFactory;

public enum ClassProvider {
    /**
     * instance
     */
    INSTANCE;

    private final ClassLoadingMXBean classLoadingMXBean;

    ClassProvider() {
        this.classLoadingMXBean = ManagementFactory.getClassLoadingMXBean();
    }

    // 构建class的指标信息
    public Class getClassMetrics() {
        int loadedClassCount = classLoadingMXBean.getLoadedClassCount();
        long unloadedClassCount = classLoadingMXBean.getUnloadedClassCount();
        long totalLoadedClassCount = classLoadingMXBean.getTotalLoadedClassCount();
        return Class.newBuilder().setLoadedClassCount(loadedClassCount)
            .setUnloadedClassCount(unloadedClassCount)
    }
}
```

在 `org.apache.skywalking.apm.agent.core.jvm.JVMService#run` 方法中，将 `class` 相关指标设置到 `JVM` 指标类中

```
@Override
public void run() {
    long currentTimeMillis = System.currentTimeMillis();
    try {
        JVMMetric.Builder jvmBuilder = JVMMetric.newBuilder();
        jvmBuilder.setTime(currentTimeMillis);
        jvmBuilder.setCpu(CPUProvider.INSTANCE.getCpuMetric());
        jvmBuilder.addAllMemory(MemoryProvider.INSTANCE.getMemoryMetricList());
        jvmBuilder.addAllMemoryPool(MemoryPoolProvider.INSTANCE.getMemoryPoolMetricsList());
        jvmBuilder.addAllGc(GCProvider.INSTANCE.getGCList());
        jvmBuilder.setThread(ThreadProvider.INSTANCE.getThreadMetrics());
        // 设置class的指标
        jvmBuilder.setClazz(ClassProvider.INSTANCE.getClassMetrics());
        // 将JVM的指标放在阻塞队列中
        // org.apache.skywalking.apm.agent.core.jvm.JVMMetricsSender#run方法，会将相关信息发送至oap server
        sender.offer(jvmBuilder.build());
    } catch (Exception e) {
        LOGGER.error(e, "Collect JVM info fail.");
    }
}
```

## 创建 Source 类

```
public class DefaultScopeDefine {
    public static final int SERVICE_INSTANCE_JVM_CLASS = 11000;

    /** Catalog of scope, the metrics processor could use this to group all generated metrics by oal rt. */
    public static final String SERVICE_INSTANCE_CATALOG_NAME = "SERVICE_INSTANCE";
}
```

```
package org.apache.skywalking.oap.server.core.source;

import lombok.Getter;
import lombok.Setter;

import static org.apache.skywalking.oap.server.core.source.DefaultScopeDefine.SERVICE_INSTANCE_CATALOG_NAME;
import static org.apache.skywalking.oap.server.core.source.DefaultScopeDefine.SERVICE_INSTANCE_JVM_CLASS;

@ScopeDeclaration(id = SERVICE_INSTANCE_JVM_CLASS, name = "ServiceInstanceJVMClass", catalog =
SERVICE_INSTANCE_CATALOG_NAME)
@ScopeDefaultColumn.VirtualColumnDefinition(fieldName = "entityId", columnName = "entity_id", isID = true, type =
String.class)
public class ServiceInstanceJVMClass extends Source {
    @Override
    public int scope() {
        return SERVICE_INSTANCE_JVM_CLASS;
    }

    @Override
    public String getEntityId() {
        return String.valueOf(id);
    }

    @Getter @Setter
    private String id;
    @Getter @Setter @ScopeDefaultColumn.DefinedByField(columnName = "name", requireDynamicActive = true)
```

## 将从 agent 获取到的信息，发送至 SourceReceive

在 org.apache.skywalking.oap.server.analyzer.provider.jvm.JVMSourceDispatcher 进行如下修改

```
public void sendMetric(String service, String serviceInstance, JVMetric metrics) {
    long minuteTimeBucket = TimeBucket.getMinuteTimeBucket(metrics.getTime());

    final String serviceId = IDManager.ServiceID.buildId(service, NodeType.Normal);
    final String serviceInstanceId = IDManager.ServiceInstanceId.buildId(serviceId, serviceInstance);

    this.sendToCpuMetricProcess(
        service, serviceId, serviceInstance, serviceInstanceId, minuteTimeBucket, metrics.getCpu());
    this.sendToMemoryMetricProcess(
        service, serviceId, serviceInstance, serviceInstanceId, minuteTimeBucket, metrics.getMemoryList());
    this.sendToMemoryPoolMetricProcess(
        service, serviceId, serviceInstance, serviceInstanceId, minuteTimeBucket,
metrics.getMemoryPoolList());
    this.sendToGCMetricProcess(
        service, serviceId, serviceInstance, serviceInstanceId, minuteTimeBucket, metrics.getGcList());
    this.sendToThreadMetricProcess(
        service, serviceId, serviceInstance, serviceInstanceId, minuteTimeBucket, metrics.getThread());
    // class指标处理
    this.sendToClassMetricProcess(
        service, serviceId, serviceInstance, serviceInstanceId, minuteTimeBucket, metrics.getClazz());
}

private void sendToClassMetricProcess(String service,
    String serviceId,
    String serviceInstance,
    String serviceInstanceId,
```

## 在 OAL 词法定义和语法定义中加入 Source 相关信息

在 `oap-server/oal-grammar/src/main/antlr4/org/apache/skywalking/oal/rt/grammar/OALLexer.g4` 定义 `Class` 关键字

```
// Keywords

FROM: 'from';
FILTER: 'filter';
DISABLE: 'disable';
SRC_ALL: 'All';
SRC_SERVICE: 'Service';
SRC_SERVICE_INSTANCE: 'ServiceInstance';
SRC_ENDPOINT: 'Endpoint';
SRC_SERVICE_RELATION: 'ServiceRelation';
SRC_SERVICE_INSTANCE_RELATION: 'ServiceInstanceRelation';
SRC_ENDPOINT_RELATION: 'EndpointRelation';
SRC_SERVICE_INSTANCE_JVM_CPU: 'ServiceInstanceJVMCPU';
SRC_SERVICE_INSTANCE_JVM_MEMORY: 'ServiceInstanceJVMMemory';
SRC_SERVICE_INSTANCE_JVM_MEMORY_POOL: 'ServiceInstanceJVMMemoryPool';
SRC_SERVICE_INSTANCE_JVM_GC: 'ServiceInstanceJVMGC';
SRC_SERVICE_INSTANCE_JVM_THREAD: 'ServiceInstanceJVMThread';
SRC_SERVICE_INSTANCE_JVM_CLASS: 'ServiceInstanceJVMClass'; // 在OAL词法定义中添加Class的关键字
SRC_DATABASE_ACCESS: 'DatabaseAccess';
SRC_SERVICE_INSTANCE_CLR_CPU: 'ServiceInstanceCLRCPU';
SRC_SERVICE_INSTANCE_CLR_GC: 'ServiceInstanceCLRGC';
SRC_SERVICE_INSTANCE_CLR_THREAD: 'ServiceInstanceCLRThread';
SRC_ENVOY_INSTANCE_METRIC: 'EnvoyInstanceMetric';
```

在 `oap-server/oal-grammar/src/main/antlr4/org/apache/skywalking/oal/rt/grammar/OALParser.g4` 添加 `Class` 关键字

```
source
: SRC_ALL | SRC_SERVICE | SRC_DATABASE_ACCESS | SRC_SERVICE_INSTANCE | SRC_ENDPOINT |
SRC_SERVICE_RELATION | SRC_SERVICE_INSTANCE_RELATION | SRC_ENDPOINT_RELATION |
SRC_SERVICE_INSTANCE_JVM_CPU | SRC_SERVICE_INSTANCE_JVM_MEMORY | SRC_SERVICE_INSTANCE_JVM_MEMORY_POOL |
SRC_SERVICE_INSTANCE_JVM_GC | SRC_SERVICE_INSTANCE_JVM_THREAD | SRC_SERVICE_INSTANCE_JVM_CLASS | // 在OAL语法定
义中添加词法定义中定义的关键字
SRC_SERVICE_INSTANCE_CLR_CPU | SRC_SERVICE_INSTANCE_CLR_GC | SRC_SERVICE_INSTANCE_CLR_THREAD |
SRC_ENVOY_INSTANCE_METRIC |
SRC_BROWSER_APP_PERF | SRC_BROWSER_APP_PAGE_PERF | SRC_BROWSER_APP_SINGLE_VERSION_PERF |
SRC_BROWSER_APP_TRAFFIC | SRC_BROWSER_APP_PAGE_TRAFFIC | SRC_BROWSER_APP_SINGLE_VERSION_TRAFFIC
;
```

在 `oap-server/oal-grammar` 目录下执行 `mvn clean package -DskipTests=true` 会生成新的相关 `Java` 类

## 定义 OAL 指标

在 `oap-server/server-bootstrap/src/main/resources/oal/java-agent.oal` 中添加基于 `OAL` 语法的 `Class` 相关指标定义

```
// 当前加载类的数量
instance_jvm_class_loaded_class_count = from(ServiceInstanceJVMClass.loadedClassCount).longAvg();
// 已卸载类的数量
instance_jvm_class_unloaded_class_count = from(ServiceInstanceJVMClass.unloadedClassCount).longAvg();
// 一共加载类的数量
instance_jvm_class_total_loaded_class_count = from(ServiceInstanceJVMClass.totalLoadedClassCount).longAvg();
```

## 配置 UI 面板

将如下界面配置导入 `APM` 面板中

```
{
  "name": "Instance",
  "children": [{
    "width": "3",
    "title": "Service Instance Load",
    "height": "250",
    "entityType": "ServiceInstance",
    "independentSelector": false,
    "metricType": "REGULAR_VALUE",
    "metricName": "service_instance_cpm",
    "queryMetricType": "readMetricsValues",
    "chartType": "ChartLine",
    "unit": "CPM - calls per minute"
  },
  {
    "width": 3,
    "title": "Service Instance Throughput",
    "height": "250",
    "entityType": "ServiceInstance",
    "independentSelector": false,
    "metricType": "REGULAR_VALUE",
    "metricName": "service_instance_throughput_received,service_instance_throughput_sent",
    "queryMetricType": "readMetricsValues",
    "chartType": "ChartLine",
    "unit": "Bytes"
  }
],
```

## 结果校验

可以看到导入的界面中，已经有 `Class` 相关指标了

# 代码贡献

- [Add some new thread metric and class metric to JVMMetric #7230](#)
- [add some new thread metric and class metric to JVMMetric #52](#)
- [Remove Terminated State and New State in JVMMetric \(#7230\) #53](#)
- [Add some new thread metric and class metric to JVMMetric \(#7230\) #7243](#)

# 参考文档

- [Observability Analysis Language](#)
- [Source and Scope extension for new metrics](#)
- [Java ManagementFactory解析](#)

分享并记录所学所见



阅读 56 • 发布于 8 月 16 日



赞



收藏



分享

本作品系原创，采用《署名-非商业性使用-禁止演绎 4.0 国际》许可协议



Switch

9 声望

0 粉丝

关注作者

0 条评论

得票数

最新



撰写评论 ...



提交评论

你知道吗？

不要站着调试程序，那会使得你的耐心减半，你需要的是全神贯注。

注册登录

继续阅读



### 聊聊nacos Service的processClientBeat

[nacos-1.1.3/naming/src/main/java/com/alibaba/nacos/naming/core/Service.java](#)

[codecraft](#) · 阅读 651

### Skywalking-05：在Skywalking RocketBot上添加监控图表

标题为： JVM Thread State Count (Java Service)指标为： read all values in the duration instance jvm\_thread\_new\_thread\_count,in...

[Switch](#) · 阅读 167

### 聊聊skywalking的jvm-receiver-plugin

[skywalking-6.6.0/oap-server/server-receiver-plugin/skywalking-jvm-receiver-plugin/src/main/java/org/apache/skywalking/oap/se...](#)

[codecraft](#) · 阅读 1.2k

### 聊聊skywalking的JVMService

[skywalking-6.6.0/apm-sniffer/apm-agent-core/src/main/java/org/apache/skywalking/apm/agent/core/boot/BootService.java](#)

[codecraft](#) · 阅读 617

### 聊聊NacosDiscoveryClient

[spring-cloud-alibaba-0.9.0.RELEASE/spring-cloud-alibaba-nacos-discovery/src/main/java/org/springframework/cloud/alibaba/nac...](#)

[codecraft](#) · 阅读 1.4k

### 聊聊skywalking的ServiceResetCommand

[skywalking-6.6.0/apm-protocol/apm-network/src/main/java/org/apache/skywalking/apm/network/trace/component/command/S...](#)

[codecraft](#) · 阅读 541

### 聊聊skywalking的ServiceAndEndpointRegisterClient

[skywalking-6.6.0/apm-sniffer/apm-agent-core/src/main/java/org/apache/skywalking/apm/agent/core/remote/ServiceAndEndpoi...](#)

[codecraft](#) · 阅读 866

### Skywalking-08：OAL原理——如何动态生成Class类

启动 OAP 配置中，配置下环境变量 SW\_OAL\_ENGINE\_DEBUG=Y，这样能在工作目录下的 oal-rt 目录下找到生成的 Class 文件。

[Switch](#) · 阅读 66

#### 产品

[热门问答](#)

[热门专栏](#)

[热门课程](#)

[最新活动](#)

[技术圈](#)

[酷工作](#)

#### 课程

[Java 开发课程](#)

[PHP 开发课程](#)

[Python 开发课程](#)

[前端开发课程](#)

[移动开发课程](#)

#### 资源

[每周精选](#)

[用户排行榜](#)

[勋章](#)

[帮助中心](#)

[声望与权限](#)

[社区服务中心](#)

[建议反馈](#)

#### 合作

[关于我们](#)

[广告投放](#)

[职位发布](#)

[讲师招募](#)

[联系我们](#)

[合作伙伴](#)

#### 关注

[产品技术日志](#)

[社区运营日志](#)

[市场运营日志](#)

[团队日志](#)

[社区访谈](#)

#### 条款

[服务协议](#)

[隐私政策](#)

[下载 App](#)



