


Skywalking-08：OAL原理——如何动态生成Class类

 **Switch** 发布于 8 月 26 日

OAL 如何动态生成 Class 类

代码入口

在 `org.apache.skywalking.oal.rt.OALRuntime#start` 方法

```
public void start(ClassLoader currentClassLoader) throws ModuleStartException, OALCompileException {
    if (!IS_RT_TEMP_FOLDER_INIT_COMPLETED) {
        prepareRTTempFolder();
        IS_RT_TEMP_FOLDER_INIT_COMPLETED = true;
    }

    this.currentClassLoader = currentClassLoader;
    Reader read;

    try {
        read = ResourceUtils.read(oalDefine.getConfigFile());
    } catch (FileNotFoundException e) {
        throw new ModuleStartException("Can't locate " + oalDefine.getConfigFile(), e);
    }

    OALScripts oalScripts;
    try {
        ScriptParser scriptParser = ScriptParser.createFromFile(read, oalDefine.getSourcePackage());
        // 解析oal脚本，生成OALScripts对象
        oalScripts = scriptParser.parse();
    } catch (IOException e) {
        throw new ModuleStartException("OAL script parse analysis failure.", e);
    }
    // OALScripts对象动态生成需要的类
    this.generateClassAtRuntime(oalScripts);
}
```

时序图

[OALRuntime-generate-class-at-runtime.sdt](#) 该文件可以在 IDEA 的 Sequence Diagram 插件中打开

案例

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。

通过如下目录结构，可以看出有三种 `Class`：

- `dispatcher`：调度器，将指标对象发送 `MetricsStreamProcessor` (指标处理器)
- `metrics`：指标类，存储指标数据
- `StorageBuilder`：存储构造器，实现类 `StorageBuilder` 接口，提供 `map` 与 `StorageData` 之间互转的方法

```
oal-rt
├─ dispatcher
│   ├── ServiceInstanceJVMClassDispatcher.class
│   └─ ServiceInstanceJVMThreadDispatcher.class
└─ metrics
    ├── InstanceJvmClassLoadedClassCountMetrics.class
    ├── InstanceJvmClassTotalLoadedClassCountMetrics.class
    ├── InstanceJvmClassUnloadedClassCountMetrics.class
    ├── InstanceJvmThreadDaemonCountMetrics.class
    ├── InstanceJvmThreadDeadlockedMetrics.class
    ├── InstanceJvmThreadLiveCountMetrics.class
    └─ builder
        ├── InstanceJvmClassLoadedClassCountMetricsBuilder.class
        ├── InstanceJvmClassTotalLoadedClassCountMetricsBuilder.class
        ├── InstanceJvmClassUnloadedClassCountMetricsBuilder.class
        ├── InstanceJvmThreadDaemonCountMetricsBuilder.class
        ├── InstanceJvmThreadDeadlockedMetricsBuilder.class
        └─ InstanceJvmThreadLiveCountMetricsBuilder.class
```

指标类

```
package org.apache.skywalking.oap.server.core.source.oal.rt.metrics;

import org.apache.skywalking.oap.server.core.analysis.Stream;
import org.apache.skywalking.oap.server.core.analysis.metrics.LongAvgMetrics;
import org.apache.skywalking.oap.server.core.analysis.metrics.Metrics;
import org.apache.skywalking.oap.server.core.analysis.metrics.MetricsMetaInfo;
import org.apache.skywalking.oap.server.core.analysis.metrics.WithMetadata;
import org.apache.skywalking.oap.server.core.analysis.worker.MetricsStreamProcessor;
import org.apache.skywalking.oap.server.core.remote.grpc.proto.RemoteData;
import org.apache.skywalking.oap.server.core.remote.grpc.proto.RemoteData.Builder;
import
org.apache.skywalking.oap.server.core.source.oal.rt.metrics.builder.InstanceJvmClassLoadedClassCountMetricsBuilde
r;
import org.apache.skywalking.oap.server.core.storage.annotation.Column;

@Stream(
    name = "instance_jvm_class_loaded_class_count",
    scopeId = 11000,
    builder = InstanceJvmClassLoadedClassCountMetricsBuilder.class,
    processor = MetricsStreamProcessor.class
)
public class InstanceJvmClassLoadedClassCountMetrics extends LongAvgMetrics implements WithMetadata {
    @Column(
        columnName = "entity_id",
        length = 512
    )
}
```

存储构造器

```
package org.apache.skywalking.oap.server.core.source.oal.rt.metrics.builder;

import java.util.HashMap;
import java.util.Map;
import org.apache.skywalking.oap.server.core.source.oal.rt.metrics.InstanceJvmClassLoadedClassCountMetrics;
import org.apache.skywalking.oap.server.core.storage.StorageBuilder;
import org.apache.skywalking.oap.server.core.storage.StorageData;

public class InstanceJvmClassLoadedClassCountMetricsBuilder implements StorageBuilder {
    public InstanceJvmClassLoadedClassCountMetricsBuilder() {
    }

    public Map data2Map(StorageData var1) {
        InstanceJvmClassLoadedClassCountMetrics var2 = (InstanceJvmClassLoadedClassCountMetrics)var1;
        HashMap var3 = new HashMap();
        var3.put((Object)"entity_id", var2.getEntityId());
        var3.put((Object)"service_id", var2.getServiceId());
        var3.put((Object)"summation", new Long(var2.getSummation()));
        var3.put((Object)"count", new Long(var2.getCount()));
        var3.put((Object)"value", new Long(var2.getValue()));
        var3.put((Object)"time_bucket", new Long(var2.getTimeBucket()));
        return var3;
    }

    public StorageData map2Data(Map var1) {
        InstanceJvmClassLoadedClassCountMetrics var2 = new InstanceJvmClassLoadedClassCountMetrics();
```

调度器

```
package org.apache.skywalking.oap.server.core.source.oal.rt.dispatcher;

import org.apache.skywalking.oap.server.core.analysis.SourceDispatcher;
import org.apache.skywalking.oap.server.core.analysis.worker.MetricsStreamProcessor;
import org.apache.skywalking.oap.server.core.source.ServiceInstanceJVMClass;
import org.apache.skywalking.oap.server.core.source.Source;
import org.apache.skywalking.oap.server.core.source.oal.rt.metrics.InstanceJvmClassLoadedClassCountMetrics;
import org.apache.skywalking.oap.server.core.source.oal.rt.metrics.InstanceJvmClassTotalLoadedClassCountMetrics;
import org.apache.skywalking.oap.server.core.source.oal.rt.metrics.InstanceJvmClassUnloadedClassCountMetrics;

public class ServiceInstanceJVMClassDispatcher implements SourceDispatcher<ServiceInstanceJVMClass> {
    private void doInstanceJvmClassLoadedClassCount(ServiceInstanceJVMClass var1) {
        InstanceJvmClassLoadedClassCountMetrics var2 = new InstanceJvmClassLoadedClassCountMetrics();
        var2.setTimeBucket(var1.getTimeBucket());
        var2.setEntityId(var1.getEntityId());
        var2.setServiceId(var1.getServiceId());
        var2.combine(var1.getLoadedClassCount(), (long)1);
        MetricsStreamProcessor.getInstance().in(var2);
    }

    private void doInstanceJvmClassUnloadedClassCount(ServiceInstanceJVMClass var1) {
        InstanceJvmClassUnloadedClassCountMetrics var2 = new InstanceJvmClassUnloadedClassCountMetrics();
        var2.setTimeBucket(var1.getTimeBucket());
        var2.setEntityId(var1.getEntityId());
        var2.setServiceId(var1.getServiceId());
        var2.combine(var1.getUnloadedClassCount(), (long)1);
```

分享并记录所学所见



阅读 70 • 发布于 8 月 26 日



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？

代码写得越急，程序跑得越慢。

注册登录

继续阅读

聊聊skywalking的log4j2-activation

skywalking-6.6.0/apm-sniffer/apm-toolkit-activation/apm-toolkit-log4j-2.x-activation/src/main/resources/skywalking-plugin.def
codecraft · 阅读 891

Cglib 和 Mica Bean copy 生成字节码对比

1. 前言 距离上上篇【mica cglib 增强——【01】cglib bean copy 介绍】已经过去一个月八一天。距离上一篇【Java Bean Copy 性...
如梦技术 · 阅读 1.1k

Skywalking-01：Skywalking介绍

Application performance monitor tool for distributed systems, especially designed for microservices, cloud native and container-b...
Switch · 阅读 197

Java 动态代理 理解

吐槽自己一下，设计的类，接口名不是很好。。 anyway，大致就是这样。根据规范，Mama类实现InvocationHandler接口，实现in...
curlevel2 · 阅读 882

java动态代理及原理

一般情况下的代理就是有一个接口，接口中定义了一些方法需要被实现，实现了那些方法的类被称为实现类，但是当我们需要在调...
演绎梦幻舞步 · 阅读 979

动态代理实现原理

装饰模式 vs (静态)代理模式中提到,在静态代理模式中，针对每一个需要被代理的类都要在编译前就提前写好一个代理类，这样做...

[geeker leon](#) · 阅读 2.6k

聊聊skywalking的metric-exporter

[skywalking-6.6.0/oap-server/exporter/src/main/proto/metric-exporter.proto](#)

[codecraft](#) · 阅读 969

动态代理-Proxy

主要目的是记录java动态代理的实现，为rpc的学习做铺垫。什么是动态代理？动态代理就是在java运行时为某个类生成代理，即...

[芙兰泣露](#) · 阅读 531

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](#)

Copyright © 2011-2021 SegmentFault. 当前呈现版本 21.09.09



[浙ICP备15005796号-2](#) [浙公网安备33010602002000号](#) ICP 经营许可 [浙B2-20201554](#)

杭州堆栈科技有限公司版权所有