拉勾教育
—互联网人实战大学—

《31 讲带你搞懂 SkyWalking》

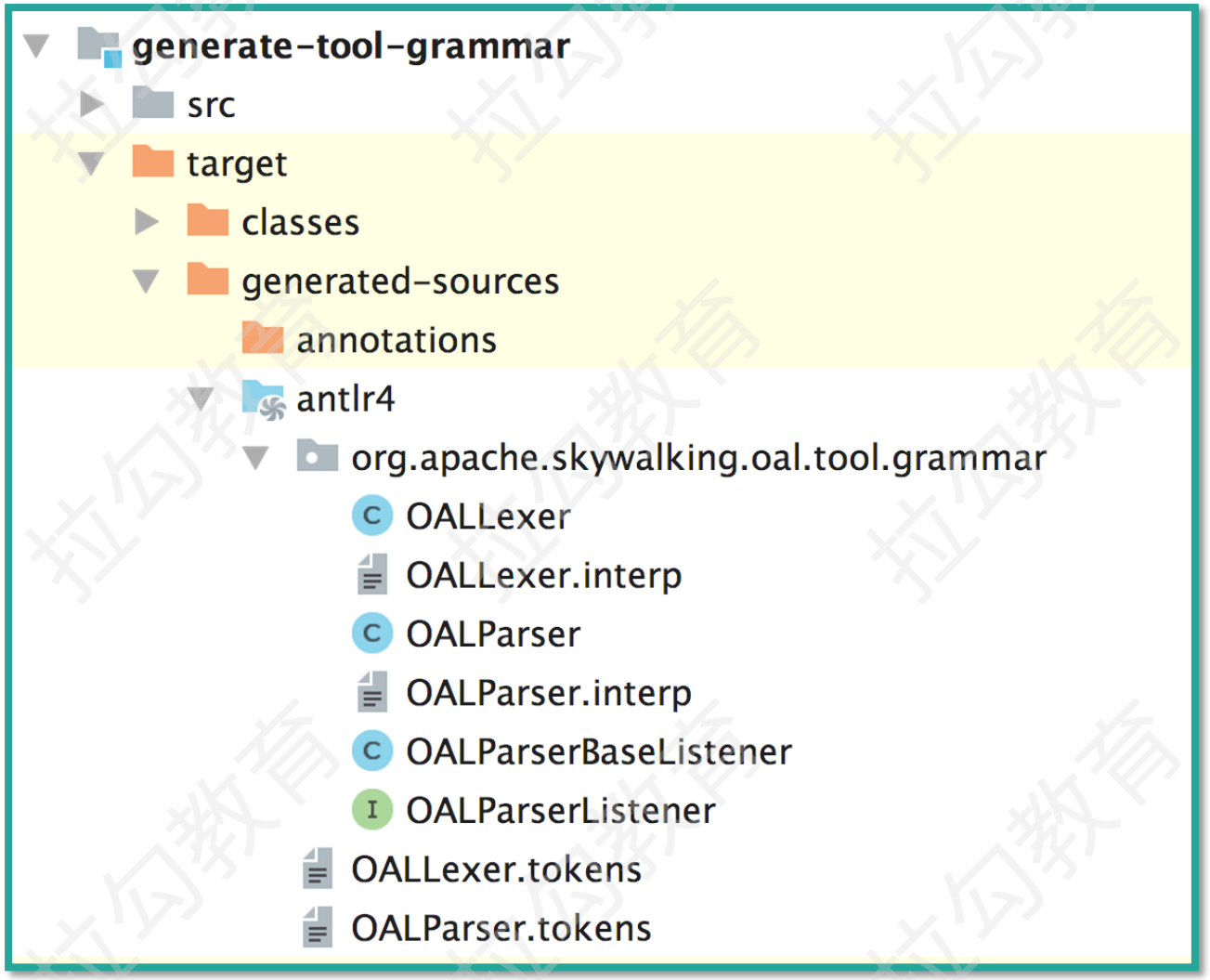
徐郡明 资深技术专家

— 拉勾教育出品 —

拉勾教育
—互联网人实战大学—

第32讲：OAL 语言
原来定义创造一门新语言如此轻松（下）

前言

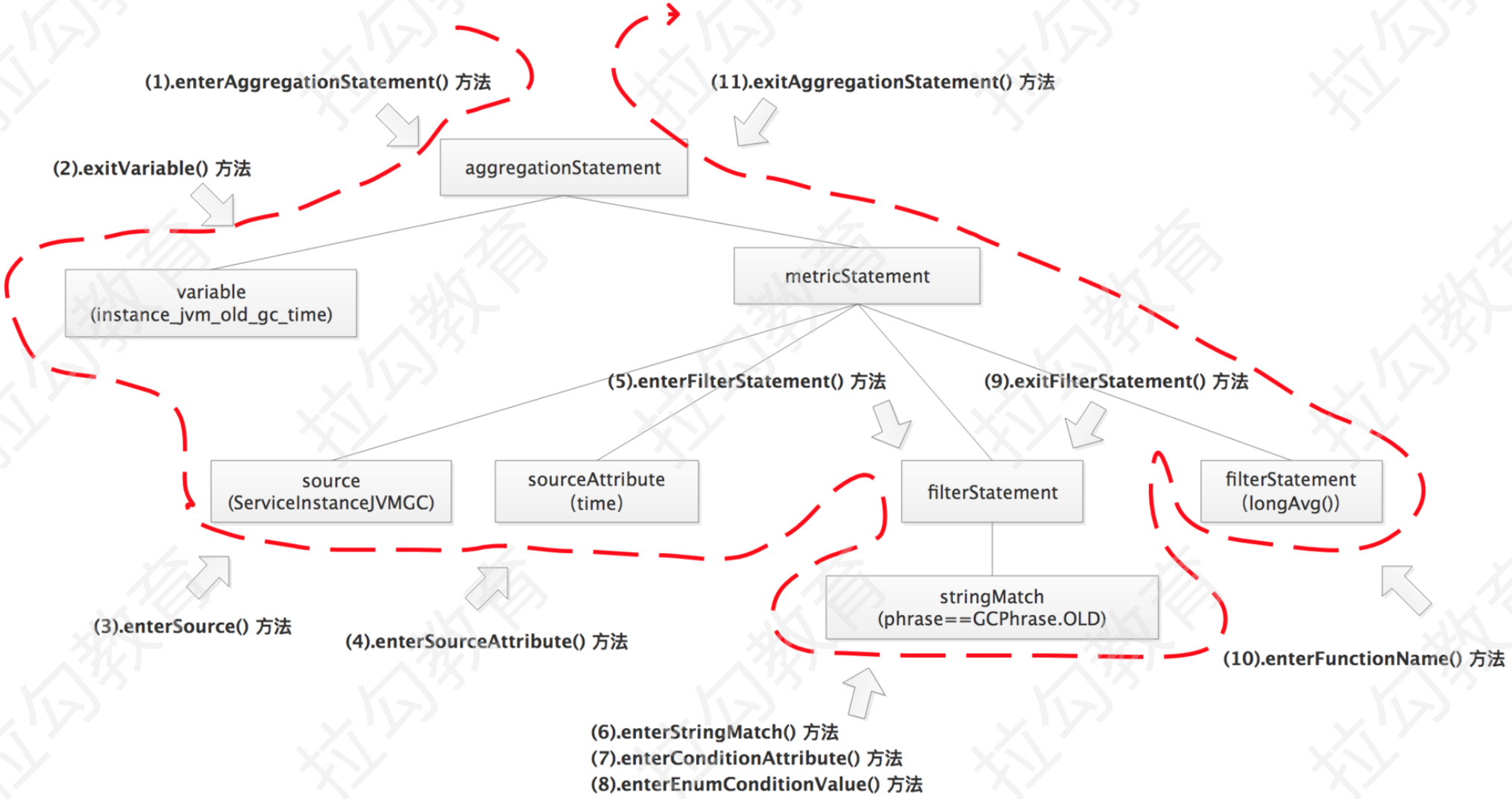


前言

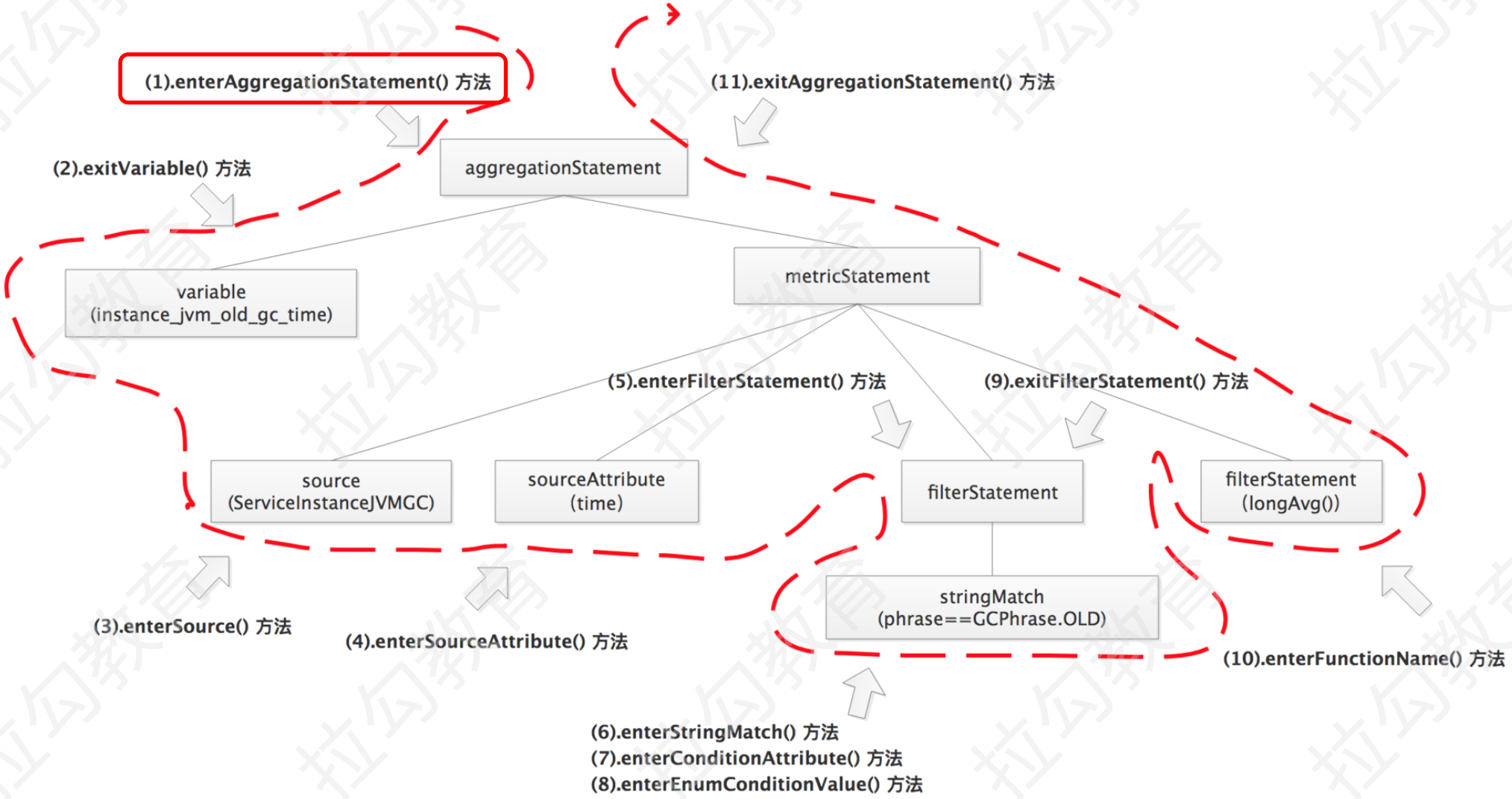
拉勾教育
— 互联网人实战大学 —

```
// 构造 official_analysis.oal 文件的完整路径
String scriptFilePath = StringUtil.join(File.separatorChar,
    modulePath, "src", "main", "resources", "official_analysis.oal");
// 创建 ScriptParser 实例
ScriptParser scriptParser =
    ScriptParser.createFromFile(scriptFilePath);
// 调用 parse()方法识别 official_analysis.oal文件
OALScripts oalScripts = scriptParser.parse();
```

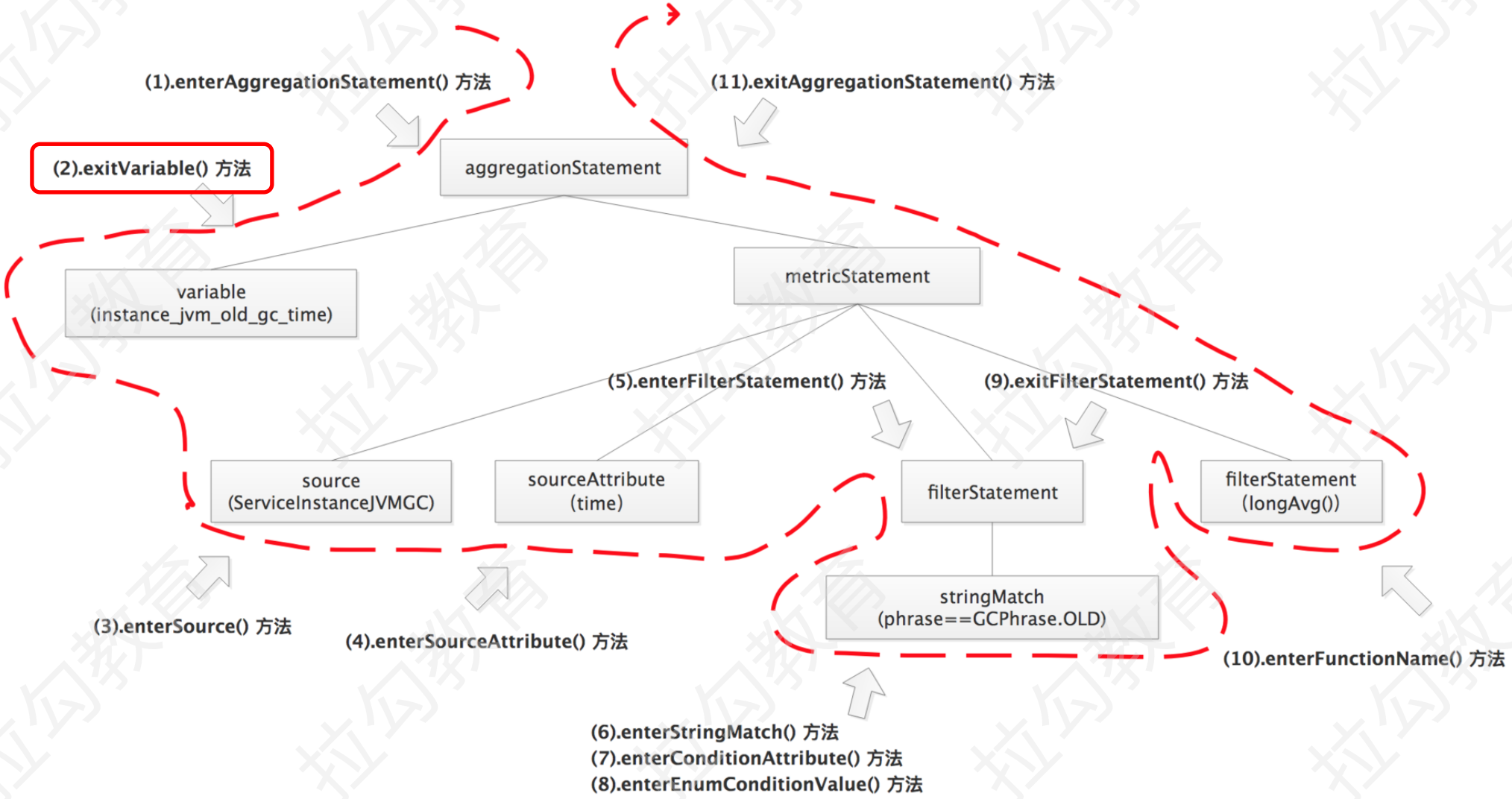
前言



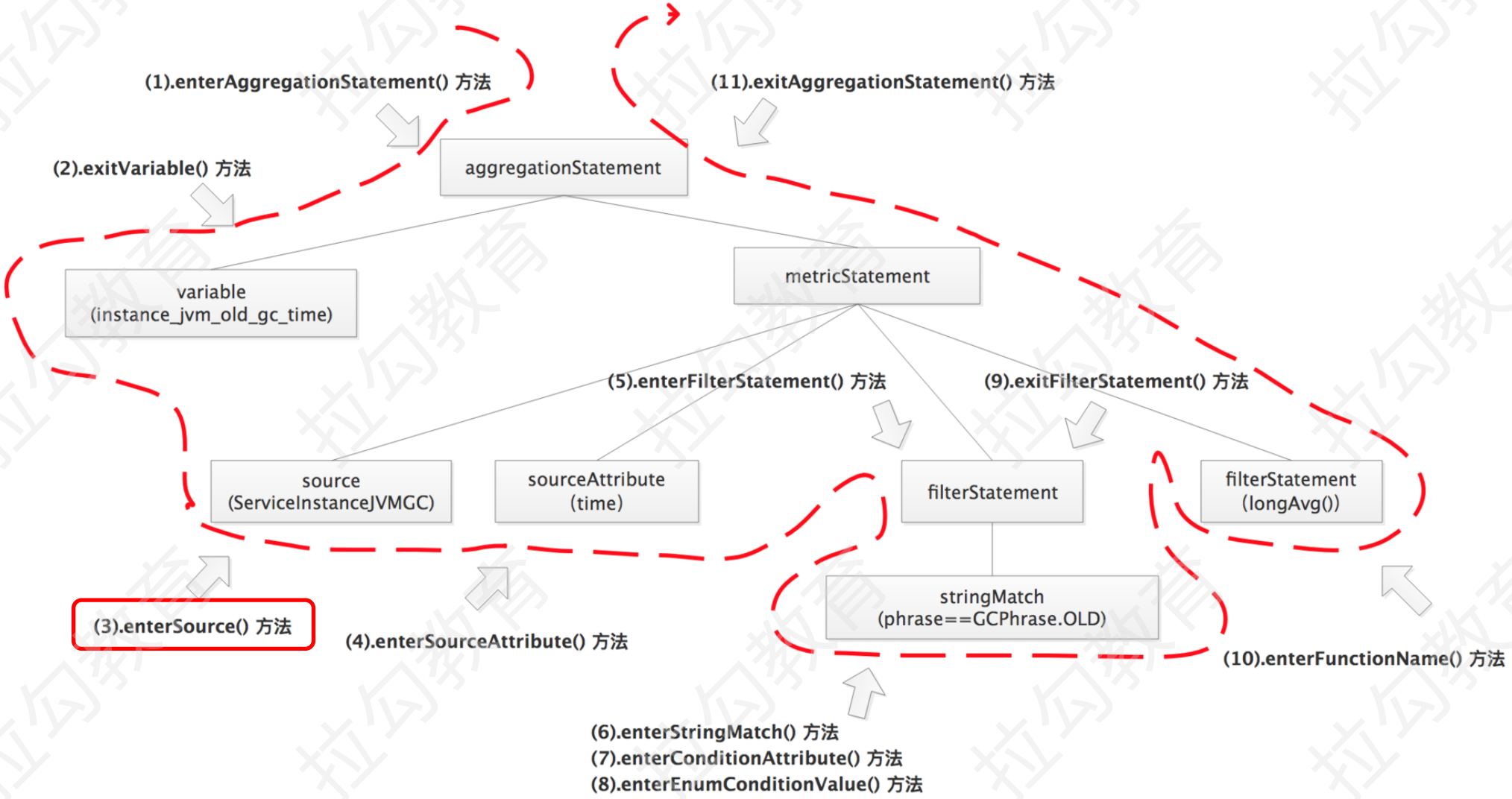
前言



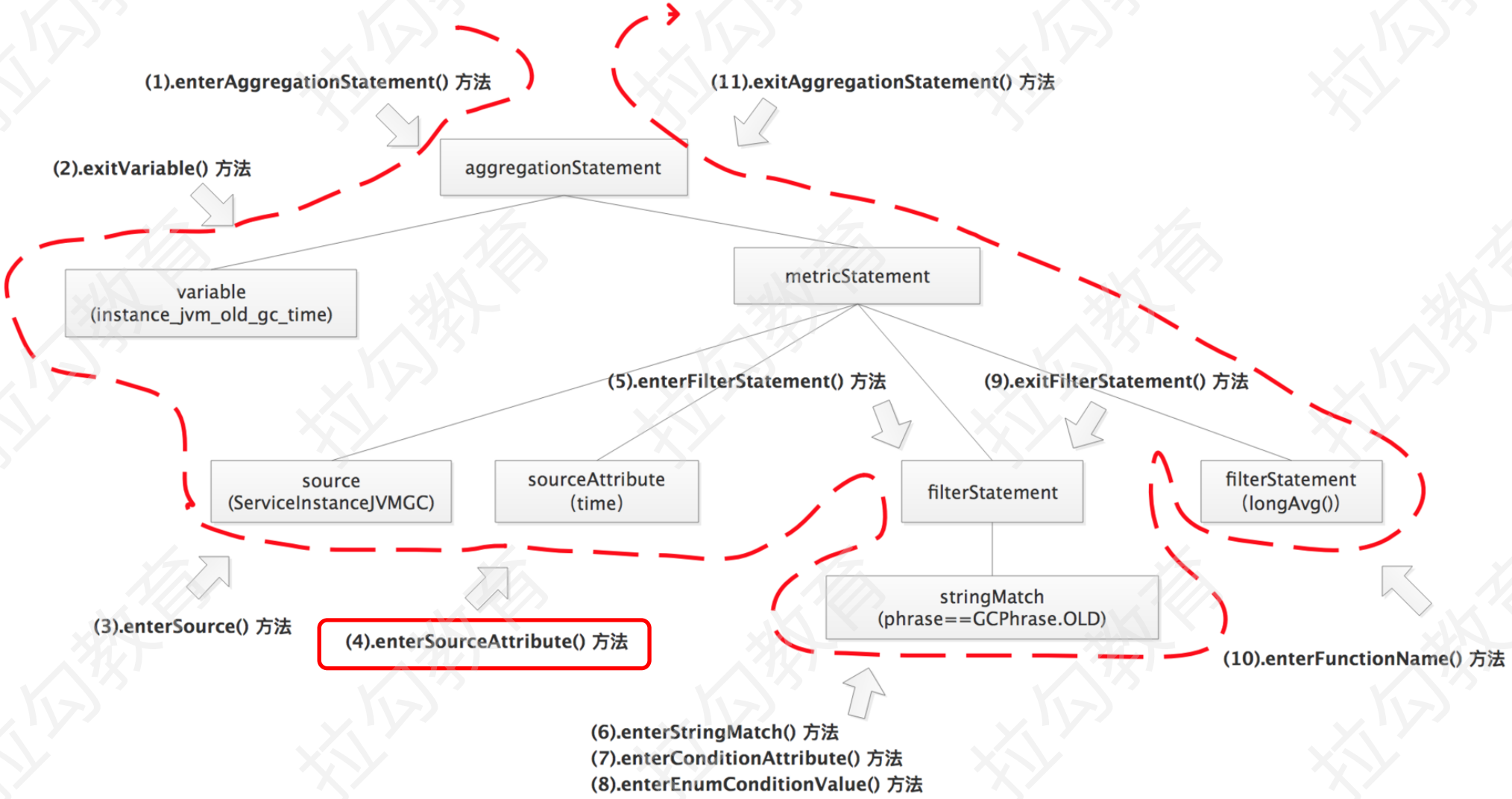
前言



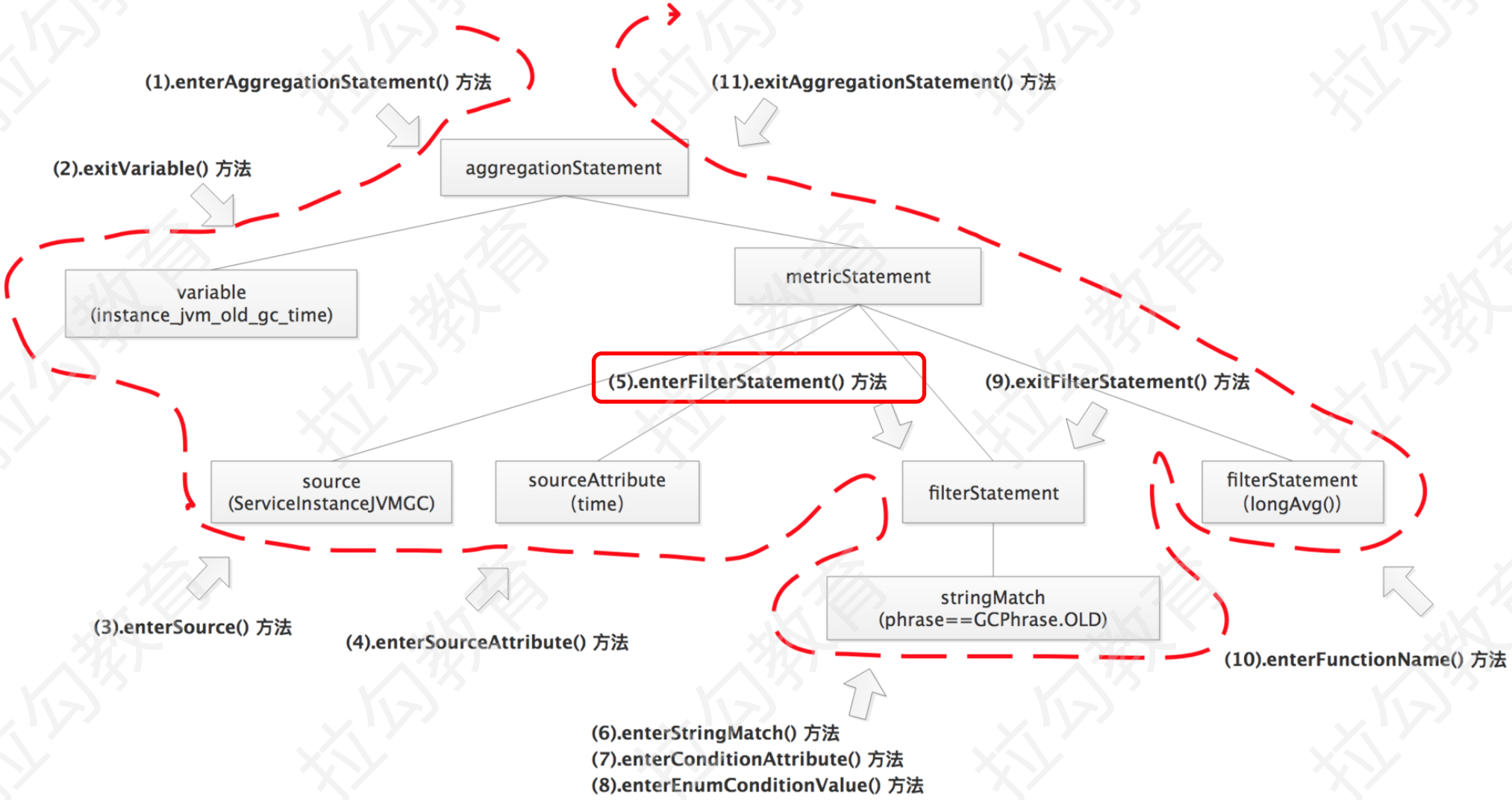
前言



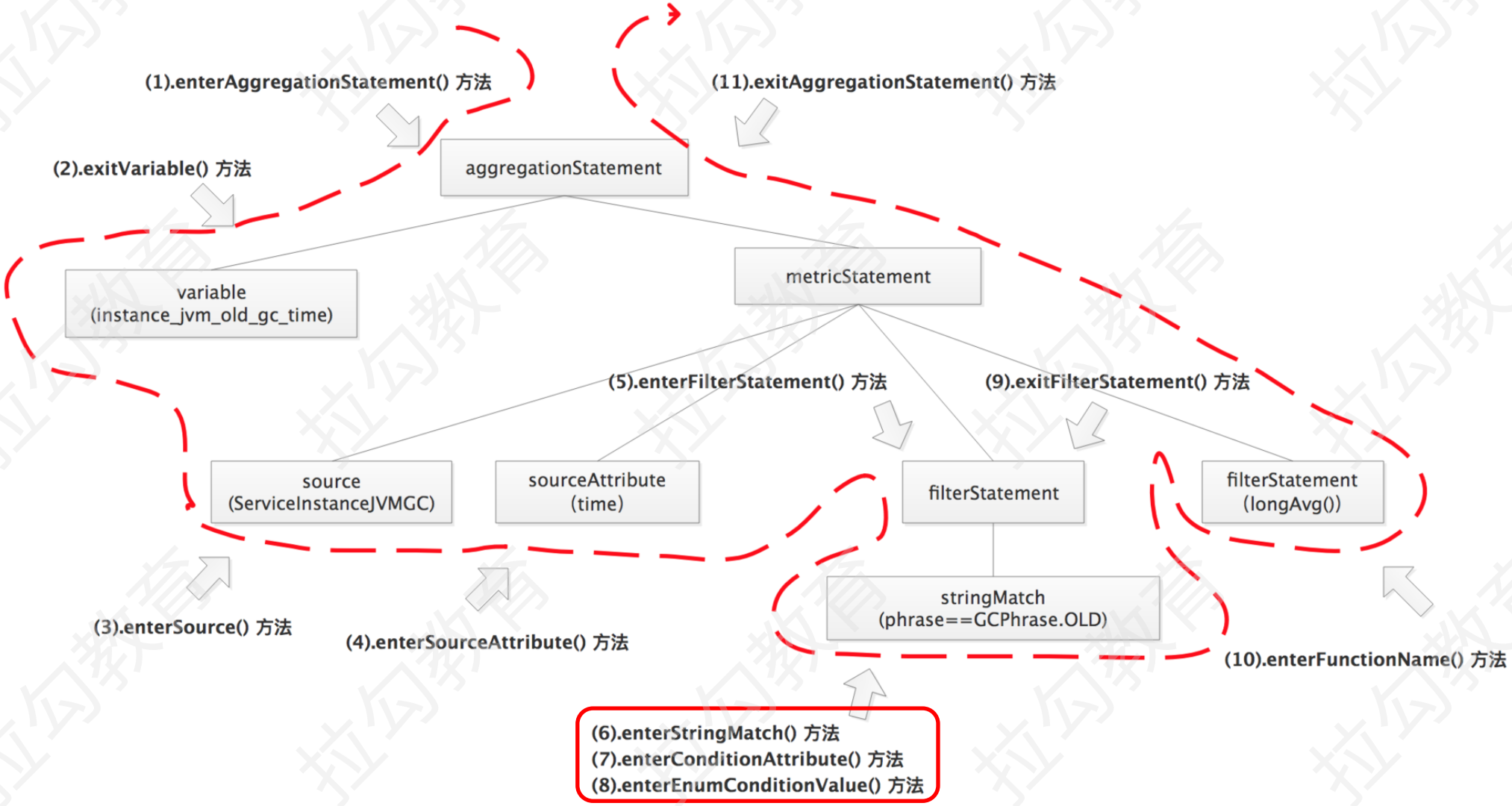
前言



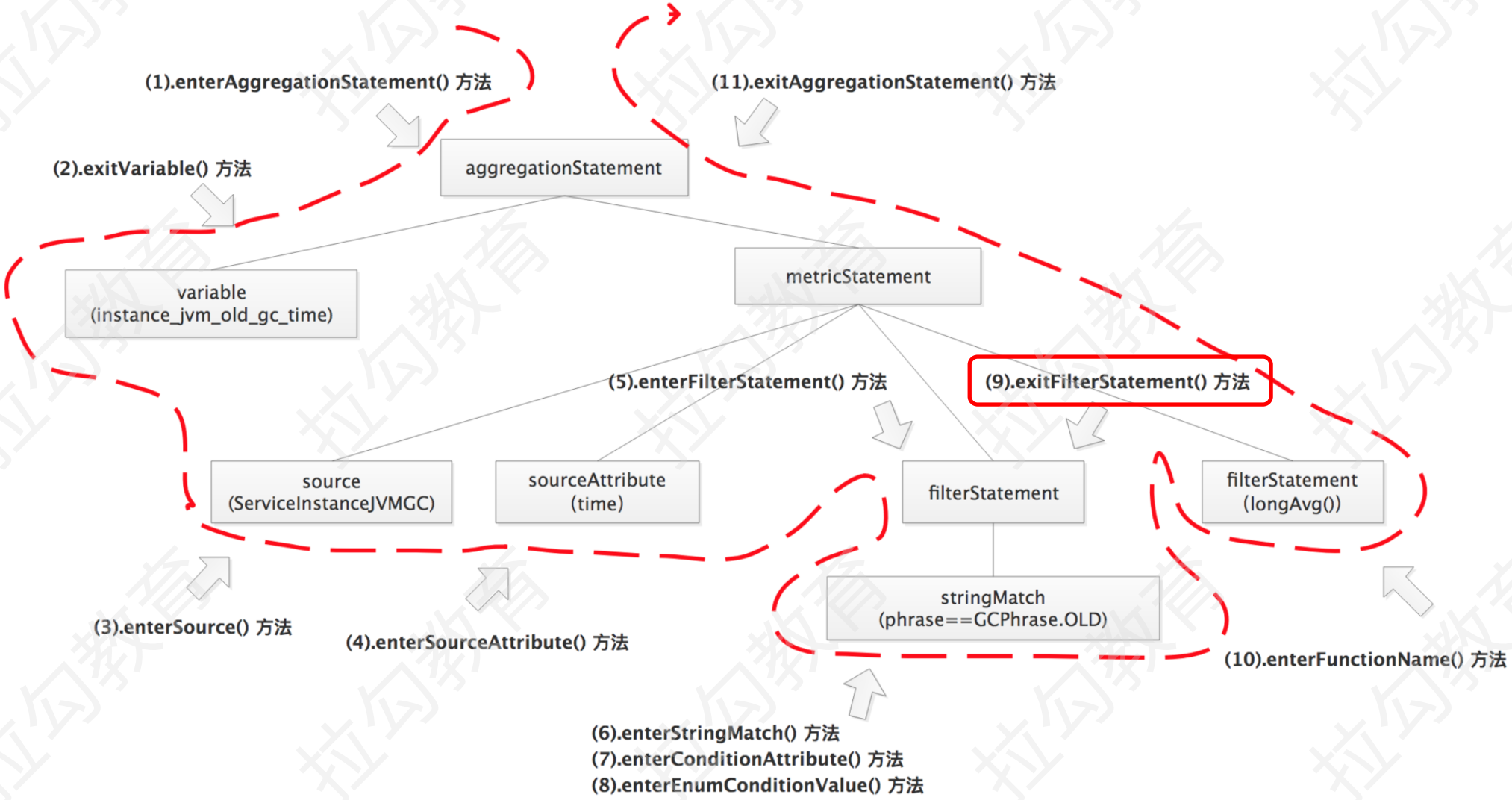
前言



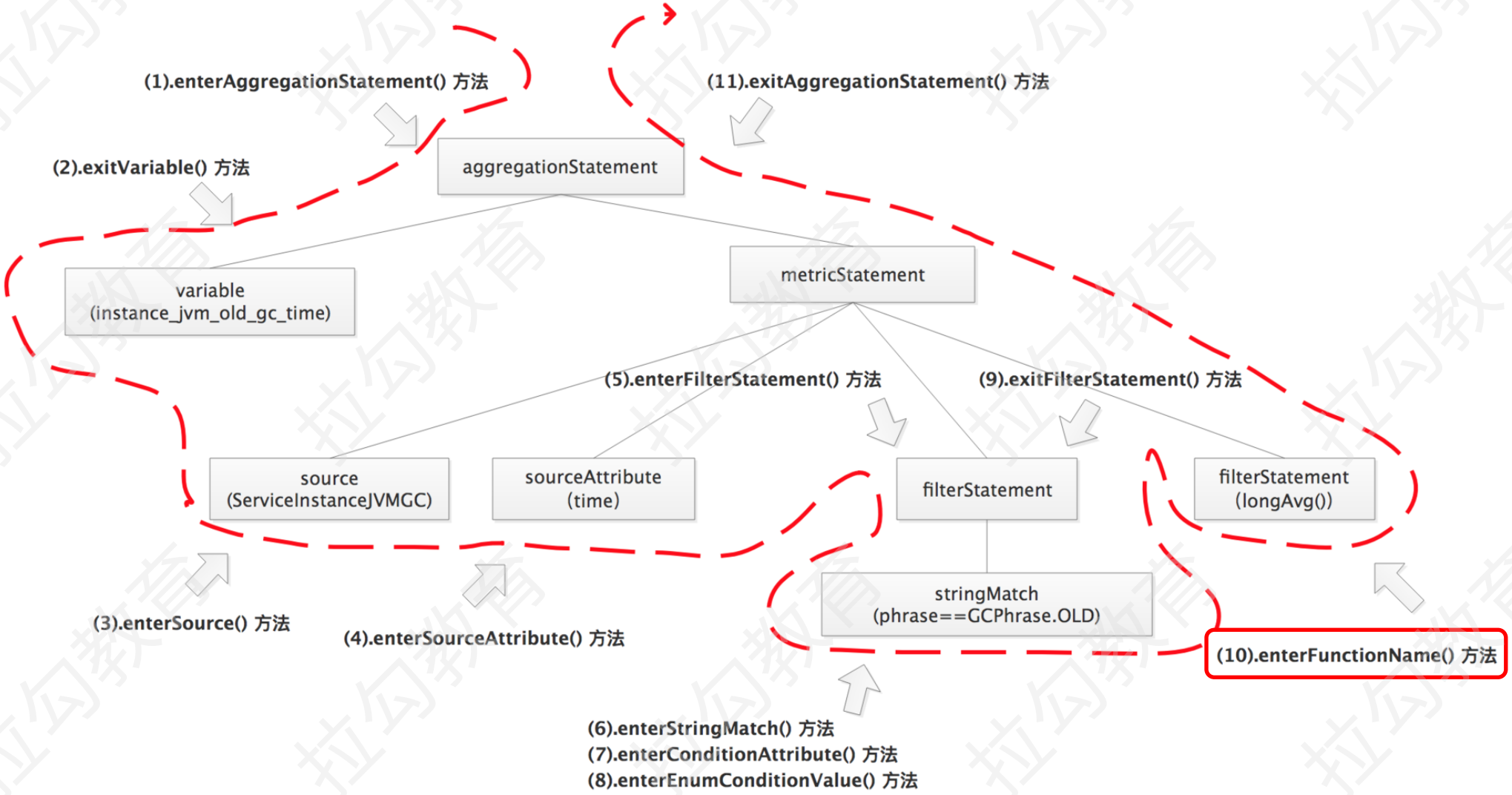
前言



前言



前言

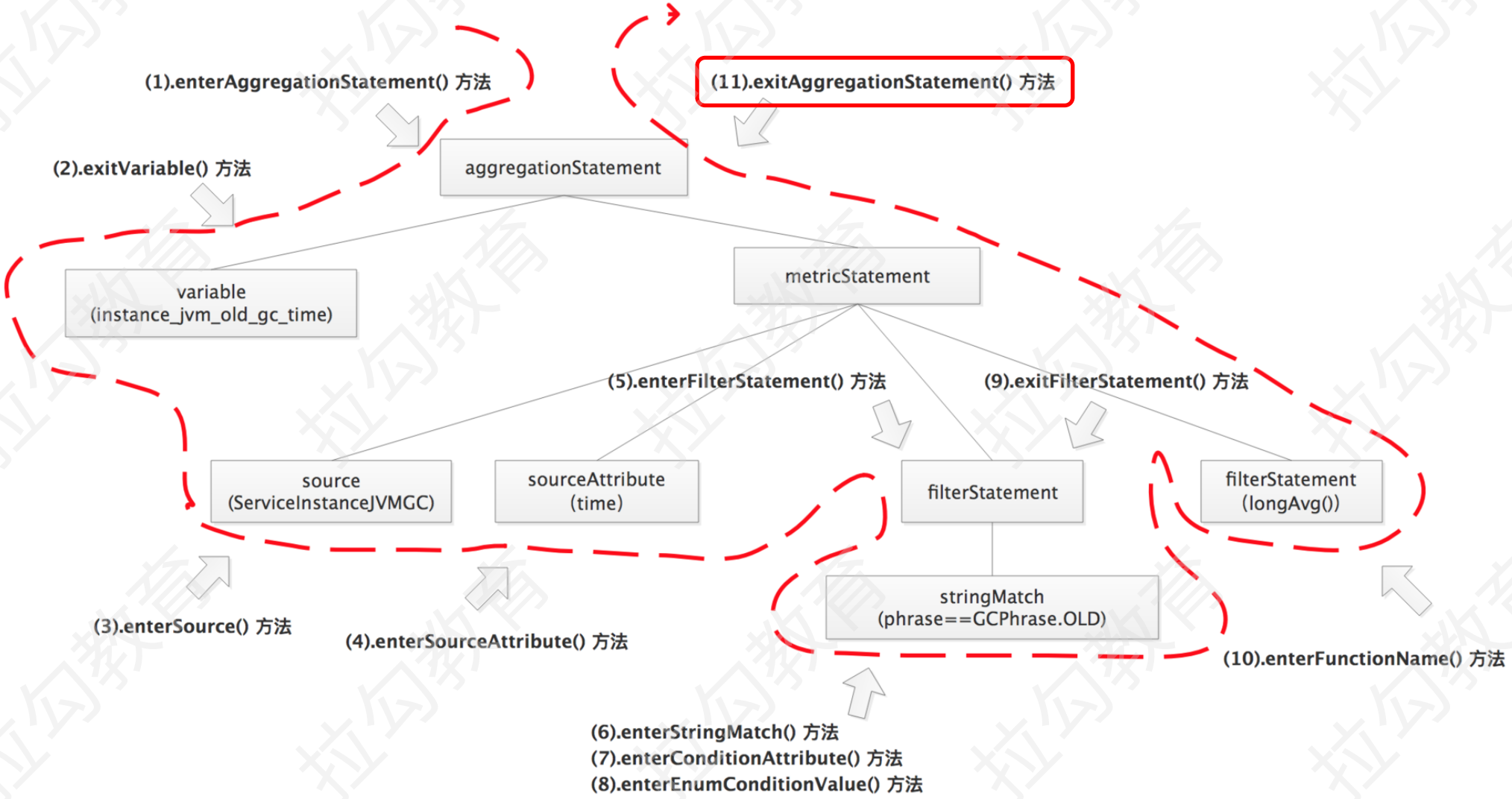


前言

拉勾教育
— 互联网人实战大学 —

```
current = {AnalysisResult@1299}
  varName = "instance_jvm_old_gc_time"
  metricsName = "InstanceJvmOldGcTime"
  tableName = "instance_jvm_old_gc_time"
  sourceName = "ServiceInstanceJVMGC"
  sourceScopeId = 11
  sourceAttribute = "time"
  aggregationFunctionName = "longAvg"
  filterExpressionsParserResult = {LinkedList@13
    0 = {ConditionExpression@1310}
      expressionType = "stringMatch"
      attribute = "phrase"
      value = "GCPhrase.OLD"
```

前言



前言

拉勾教育
— 互联网人实战大学 —

```
REGISTER = {HashMap@1332}  size = 14
  0 = {HashMap$Node@1361} "cpm" -> "class  CPMMetrics"
  1 = {HashMap$Node@1333} "longAvg" -> "class LongAvgMetrics"
```

该集合总共有 14对KV，这里只展示了2对，且省略了包名

前言

拉勾教育
— 互联网人实战大学 —

```
@Entrance
public void combine(@SourceFrom long summation, @ConstOne int count) {
  this.summation += summation;
  this.count += count;
}
```

前言

拉勾教育
— 互联网人实战大学 —

```
EntryMethod entryMethod = new EntryMethod();
result.setEntryMethod(entryMethod);
// @Entrance注解标注的入口方法名
entryMethod.setMethodName(entranceMethod.getName());

//根据入口方法的参数设置参数代码
for (Parameter parameter : entranceMethod.getParameters()) {
  Annotation[] parameterAnnotations = parameter.getAnnotations();
  Annotation annotation = parameterAnnotations[0];
  if (annotation instanceof SourceFrom) {
    entryMethod.addArg("source." + ClassMethodUtil
        .toGetMethod(result.getSourceAttribute()) + "()");
  } else if (annotation instanceof ConstOne) {
    entryMethod.addArg("1");
  }
  //还有针对其他注解的处理，例如 @Expression、@ExpressionArg0等，不再展开

}
```

前言

拉勾教育
— 互联网人实战大学 —

```
▼ entryMethod = {EntryMethod@1338}
  ▶ f methodName = "combine"
  ▼ f argsExpressions = {LinkedList@1350}  size = 2
    ▶ 0 = "source.getTime()"
    ▶ 1 = "1"
```

前言

拉勾教育
— 互联网人实战大学 —

generator-scope-meta.yml

Document 1/1 › scopes: › Item 4/16 › columns: ›

```yaml
79    - name: ServiceInstanceJVMGC
80      columns:
81      - fieldName: entityId
82        columnName: entity_id
83        typeName: java.lang.String
84        ID: true
85      - fieldName: serviceId
86        columnName: service_id
87        typeName: int
88        ID: false
```

MetricsImplementor 模板

拉勾教育
— 互联网人实战大学 —

```
void generateMetricsImplementor(AnalysisResult result,Writer output) {
  configuration.getTemplate("MetricsImplementor.ftl")
      .process(result, output);
}
```

MetricsImplementor 模板

拉勾教育
— 互联网人实战大学 —

```
<!--直接获取 AnalysisResult中相应的字段值，生成的@Stream注解-->
@Stream(name = "${tableName}",
      scopeId = ${sourceScopeId},
      builder = ${metricsName}Metrics.Builder.class,
      processor = MetricsStreamProcessor.class)
<!--填充类名以及父类名称-->
public class ${metricsName}Metrics extends ${metricsClassName}
  implements WithMetadata {

<!--遍历 AnalysisResult中的 fieldsFromSource集合，生成相应的字段 -->
<#list fieldsFromSource as sourceField>
  <!– 设置 @Column注解的名称 -->
  @Setter @Getter @Column(columnName = "${sourceField.columnName}")
  <!– 根据配置是否添加 @IDColumn注解 -->
  <#if sourceField.isID()>@IDColumn</#if>
  private ${sourceField.typeName} ${sourceField.fieldName};
</#list>
```

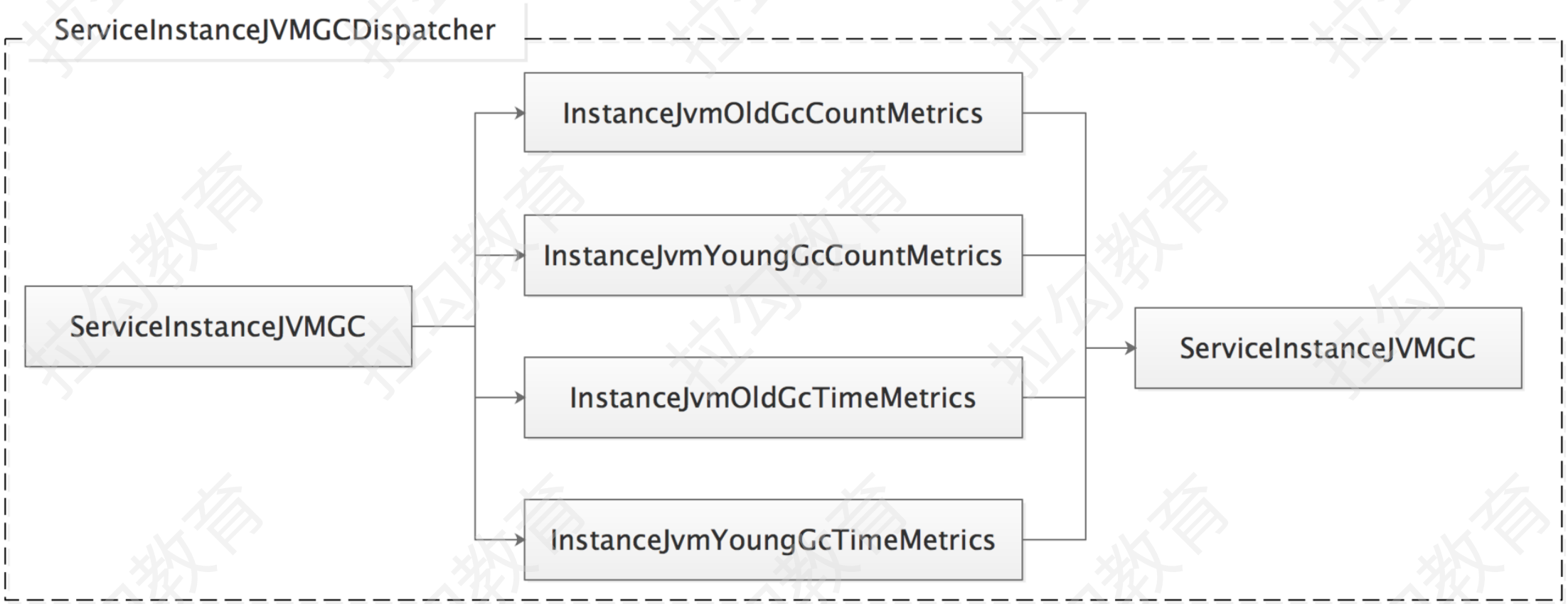
MetricsImplementor 模板

拉勾教育
— 互联网人实战大学 —

```
  @Override public String id() {
    String splitJointId = String.valueOf(getTimeBucket());
<!– 遍历 AnalysisResult中的 fieldsFromSource集合 -->
<#list fieldsFromSource as sourceField>
  <#if sourceField.isID()> <!– 根据ID配置决定是否参与构造Document Id-->
    <#if sourceField.getTypeName() == "java.lang.String">
      splitJointId += Const.ID_SPLIT + ${sourceField.fieldName};
    <#else>
      splitJointId += Const.ID_SPLIT +
          String.valueOf(${sourceField.fieldName});
    </#if>
  </#if>
</#list>
    return splitJointId;
  }
  <!– 省略后续其他方法 -->
}
```

DispatcherTemplate 模板

拉勾教育
— 互联网人实战大学 —



DispatcherTemplate 模板

拉勾教育
— 互联网人实战大学 —

```
private String source; // Source 名称
private String packageName; // Dispatcher所在包名
//该 Source所有衍生 Metrics对应的 AnalysisResult对象集合
private List<AnalysisResult> metrics = new ArrayList<>();
```

DispatcherTemplate 模板

拉勾教育
— 互联网人实战大学 —

```
void generateDispatcher(AnalysisResult result, Writer output) {
  String scopeName = result.getSourceName();
  // 根据 Source名称查找相应的 DispatcherContext
  DispatcherContext context =
      allDispatcherContext.getAllContext().get(scopeName);
  // 生成 Dispatcher实现类的代码并写入到指定文件中
configuration.getTemplate("DispatcherTemplate.ftl")
      .process(context, output);
}
```

DispatcherTemplate 模板

拉勾教育
— 互联网人实战大学 —

```
<#list metrics as metrics> <!--遍历 DispatcherContext.metrics 集合 -->
  <!– 填充 do*() 方法签名 -->
  <!– 示例中对应 doInstanceJvmOldGcTime(ServiceInstanceJVMGC)方法-->
  private void do${metrics.metricsName}(${source} source) {
    <!– 创建相应Metrics实例 -->
    ${metrics.metricsName}Metrics metrics =
        new ${metrics.metricsName}Metrics();
  <#if metrics.filterExpressions??>
    <!—根据 OAL语句中 filter表达式生成对source过滤的代码(略) -->
  </#if>
    <!– 下面开始填充 Metrics对象-->
    metrics.setTimeBucket(source.getTimeBucket());
  <#list metrics.fieldsFromSource as field>
    metrics.${field.fieldSetter}(source.${field.fieldGetter}());
  </#list>
    <!– 根据 AnalysisResult.entryMethod 生成 -->
    <!-- doInstanceJvmOldGcTime() 方法中调用的是 combine() 方法 -->
```

DispatcherTemplate 模板

拉勾教育
— 互联网人实战大学 —

```
    <!—根据 OAL语句中 filter表达式生成对source过滤的代码(略) -->
  </#if>
    <!– 下面开始填充 Metrics对象-->
    metrics.setTimeBucket(source.getTimeBucket());
  <#list metrics.fieldsFromSource as field>
    metrics.${field.fieldSetter}(source.${field.fieldGetter}());
  </#list>
    <!– 根据 AnalysisResult.entryMethod 生成 -->
    <!-- doInstanceJvmOldGcTime() 方法中调用的是 combine() 方法 -->
    metrics.${metrics.entryMethod.methodName}(
     <!– 生成入口方法的参数 -->
      <#list metrics.entryMethod.argsExpressions as arg>
       ${arg}<#if arg_has_next>, </#if></#list>
    );
    MetricsStreamProcessor.getInstance().in(metrics);
  }
</#list>
```

内置 oal 引擎

拉勾教育
— 互联网人实战大学 —

```
oalEngine = OALEngineLoader.get();
oalEngine.setStreamListener(streamAnnotationListener);
oalEngine.setDispatcherListener(receiver.getDispatcherManager());
oalEngine.start(getClass().getClassLoader());
```

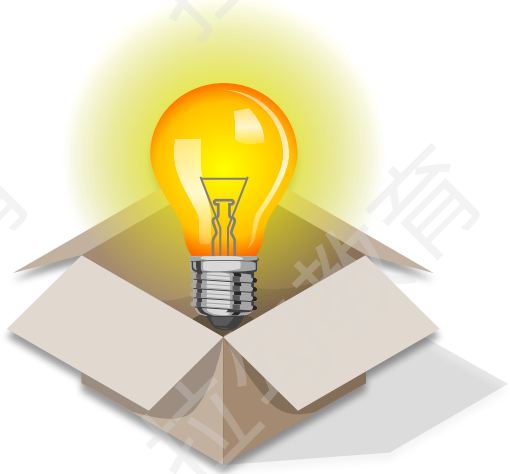
内置 oal 引擎

拉勾教育
— 互联网人实战大学 —

- 6.3 版本之后的 OAL 语法略有改动，但改动很小，并不影响理解
- 6.3 版本之后在运行时生成代码，而 6.2 版本是在编译期生成
- 6.3 版本之后生成代码时使用了 Javassist 和 FreeMarker，6.2 版本只使用了 FreeMarker
- 6.3 版本之后生成的代码默认不会保存到磁盘中

  可以在环境变量中设置 SW_OAL_ENGINE_DEBUG=Y 参数保存运行时生成的 Java 文件

  如果感兴趣可以对比 6.2 和 6.3 生成的 Java 代码，会发现两者区别不大



Next：第33讲《优化 Trace 上报性能，让你的 OAP 集群轻松抗住百万流量》

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