

# ASMv3 插件开发文档

版本	变更概要	作者	修订日期
V1.0	创建文档	wangtc	2016-11-18
V1.1	增加 <b>action</b> 开发说明	wangtc	2016-11-29

1 环境搭建.....	3
1.1 MAVEN 配置 .....	3
1.2 GIT 地址 .....	3
1.3 环境要求.....	3
2 内部采集插件开发.....	4
2.1 插件定义.....	4
2.1.1 插件创建.....	4
2.1.2 插件运行参数定义.....	5
2.1.3 监控属性定义.....	7
2.1.4 插件模板.....	10
2.2 插件设计.....	11
2.3 运行测试.....	12
2.3.1 监控作业配置.....	12
2.3.2 部署.....	12
2.3.3 单元测试.....	12
3 Exagent 插件开发.....	13
3.1 插件定义.....	13
3.2 插件设计.....	14
3.3 运行测试.....	16
3.3.1 文件上传.....	16
3.3.2 作业运行.....	16
3.3.3 离线测试.....	16
4 监控动作开发.....	17
4.1 新增动作.....	17
4.2 监控动作模板.....	19
4.3 监控动作开发.....	20
4.3.1 程序结构.....	20
4.3.2 运行参数.....	20
4.3.3 模板参数.....	20
4.4 运行测试.....	21

## 1 环境搭建

### 1.1 MAVEN 配置

增加 asm-plugin 和 asm-plugin-test 引用:

```
<dependencies>
  <dependency>
    <groupId>com.techsure</groupId>
    <artifactId>asm-plugin</artifactId>
    <version>3.0.0</version>
  </dependency>
  <dependency>
    <groupId>com.techsure</groupId>
    <artifactId>asm-plugin-test</artifactId>
    <version>3.0.0</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

通过我们的 repository 下载:

```
<repositories>
  <repository>
    <id>nexus-release</id>
    <url>http://repo.techsure.cn:9008/nexus/content/groups/public/</url>
    <releases>
      <enabled>true</enabled>
    </releases>
    <snapshots>
      <enabled>true</enabled>
    </snapshots>
  </repository>
</repositories>
```

### 1.2 GIT 地址

<http://repo.techsure.cn:7001/customer/asm-plugins.git>

### 1.3 环境要求

JDK1.7 以上, 编码 UTF-8

## 2 内部采集插件开发

插件开发过程：



运行测试过程：



### 2.1 插件定义

#### 2.1.1 插件创建

菜单：系统管理-监控-插件管理，添加插件

添加插件

名称:

插件类型:

插件组:

告警动作(手动):

描述:

采集Java类:

Exagent模块:

配置表单:

配置 [帮助](#)

1

采集器【主】:

帮助:

图 2.1.1.1

插件名称：比如 redis 监控

插件类型：选择内部采集

采集 Java 类：自定义的采集类，比如

com.techsure.asm.plugin.demo.DemoDataCollector，需要继承

com.techsure.asm.plugin.DataCollector，实现 execute 方法

配置表单：定义插件运行参数的表单，详见 2.1.2

采集器：选择想要运行该插件的采集器

### 2.1.2 插件运行参数定义

插件运行参数指某个监控插件运行所需要的参数，比如 IP 地址、账号密码等等。

用户在配置监控的时候，系统将数据保存为 json 格式，保存到数据库中；插件运行时，将转换为 JSONObject 对象。

在添加插件/修改插件界面有配置帮助说明：

配置表单： [配置](#) [帮助](#)

帮助：运行配置需要符合以下JSON格式：

字段说明：

- name: 必选属性，名称，最好使用英文字符。
- type: 必选属性，控件类型：
  - block: 控件块，多个控件的集合，需要使用children定义子控件。
  - text: 文本输入框。
  - textarea: 文本域。
  - password: 密码输入框。
  - select: 单选下拉框，可使用children定义级联控件。
  - mselect: 多选下拉框，可使用children定义级联控件。
- cloneable: 可选属性，是否可复制，默认值false。
- children: 可选属性，子控件，值是数组，block、select和mselect可使用此属性。
- data: 可选属性，select和mselect的可选值，值是包含value（必选）、text（必选）和children（可选）属性的数组。
- checktype: 可选属性，校验规则，多重校验用空格分隔，常用值：
  - 必选: required
  - IP: ip
  - 端口: port
  - 整数: integer
  - 正整数: integer\_p

范例：

```
[ { "name": "username", "type": "text", "label": "用户名", "cloneable": true }, { "name":
```

图 2.1.2.1

示例，如一个 ssh 主机监控插件配置定义：

```
[ {  
  "name": "host",  
  "type": "text",  
  "label": "主机",  
}, {  
  "name": "sshport",  
  "type": "text",  
  "label": "端口",  
}, {  
  "name": "method",  
  "type": "text",
```

```

        "label" : "监控模式"
      }, {
        "name": "sshusername",
        "type" : "text",
        "label" : "用户名"
      }, {
        "name": "sshpassword",
        "type" : "password",
        "label" : "密码"
      }, {
        "name": "sshtimeout",
        "type" : "text",
        "label" : "连接超时（毫秒）",
        "unit" : "ms"
      } ]

```

对应的监控/模板配置界面为：

基本配置

主机:

#{ip}

端口:

#{port}

监控模式:

ssh

用户名:

app

密码:

.....

连接超时（毫秒）:

3000

ms

图 2.1.2.2

注 1：在配置监控/模板运行参数时，可用#{ip}和#{port}变量引用监控目标的 ip 和端口属性，插件运行的时候将自动替换为监控目标的值，避免重复录入。

注 2：如果 name 的名字是 password/psw/pass/sshpassord，保存数据库的时候会做加密，保存密文，运行的时候再做解密

数据库存放的参数为：

```

{
  "host": "192.168.0.1",

```

```
"sshport": 22
"method": "ssh",
"sshusername": "app",
"sshpasword": "hehehe",
"sshtimeout": 3000
}
```

监控插件中，根据配置表单的 name 作为 key 来引用：

```
String host = jobConf.getString("host");
String port = jobConf.getString("sshport");
```

**注 3：**如果 cloneable=true，且配置多个参数时，数据则为 JSONArray（单个参数仍为 JSONObject 对象）如：

```
JSONArray def = jobConf.get("files");
for (int i = 0; i < def.size(); i++) {
    JSONObject jsonObject = def.getJSONObject(i);
    String typeName = jsonObject.getString("type");
    // do something
}
```

### 2.1.3 监控属性定义

## 属性预定义

监控属性指某个监控指标的定义，比如 CPU 使用率，交易量等。监控属性不能跨插件使用。

在插件的属性管理里，可以预定义监控属性，如：

属性类型

CPUCoreUtilization[单核CPU]

CPUUtilization[CPU]

Disk[磁盘]

DiskIOStats[磁盘IO]

InterfaceStats[网卡]

Memory[内存]

ProcStats[进程]

Status[状态]

SystemLoad[系统负载]

ToolSetStatus[采集工具状态]

属性清单

关键字：

每页显示：

Q搜索

ID	名称	显示名	值类型	值单位	说明
823	user	用户CPU使用率	number	%	用户态CPU使用率
822	used	CPU使用率	number	%	总体CPU使用率
821	sys	系统CPU使用率	number	%	CPU系统使用时间占比
820	queue	等待进程数	number	-	在运行队列中等待的进程数
819	iowait	IO等待时间	number	%	IO等待所占用的CPU时间的百分比
818	int	中断数	number	/秒	每秒的中断数，包括时钟中断
817	idle	空闲率	number	%	CPU 空闲时间占比
816	block	非中断睡眠进程数	number	-	非中断睡眠中的进程数

图 2.1.3

## 监控属性说明：

编辑属性

名称：	<input type="text" value="user"/>	
显示名：	<input type="text" value="用户CPU使用率"/>	
数据类型：	<input type="text" value="number"/>	
值类型：	<input type="text" value="GAUGE(原始值)"/>	
原值单位：	<input type="text"/>	<input type="text" value="数值:%"/>
目标值单位：	<input type="text"/>	<input type="text" value="数值:%"/>
时间粒度：	<input type="text" value="原始值(不显示时间单位)"/>	
	原始值类型选择立即数，增量值依据选择进行粒度转换。例如：采集间隔为1分钟，要计算粒度为秒，那么把1分钟的增量值除以60秒。要计算粒度是5分钟，就把1分钟的增量值乘以5	
显示单位：	<input type="text" value="%"/>	
健康方向：	<input type="text" value="越小越好"/>	
是否显示：	<input type="text" value="是"/>	
	是否在曲线图上显示。	
对象过滤：	<input type="text"/>	
	用于采集器过滤对象名称的正则表达式，支持小括号取match group的功能，获取第一个match group作为object名称，例如： name={w+}	
值公式：	<input type="text"/>	
	复合指标公式设置，用现有监控指标计算当前指标，支持加减乘除和括号，设置此类指标，必须保证引用的指标是存在的。例如： {(Memory.FreePhysicalMem)} + {(Memory.Inactive)}/{Metory.Total}	
说明：	<input type="text" value="用户态CPU使用率"/>	

图 2.1.3.2

数据类型：number/string，只有 number 型的才会保存历史数据，string 型仅保存最后一笔采集数据

值类型：GAUGE/COUNTER，COUNTER 型数据每次仅保留增量值，GAUGE 则不做任何差值处理

单位转换：可做常见的单位转换，如 B 转 KB 等

时间粒度：仅在 COUNTER 值类型有效，根据监控作业的采集间隔做数据转换

健康方向：用于做 percentile 数据归档和 SIGMA 计算使用

是否显示：是否在性能曲线图上可选

对象过滤：过滤监控对象，匹配不到的数据丢弃

值公式：用现有监控属性运算得到新的数据，属性引用格式：{属性类型名称.属性名称}

如{(Memory.FreePhysicalMem)} + {(Memory.Inactive)}/{Metory.Total}

注意不能跨插件引用，只能引用本插件下定义的属性。



## 预定义属性在插件里引用

监控属性通过 `tmplVo` 对象根据名字获取：

```
AttrVo attrVo = tmplVo.getAttrVo("Test", "test");//属性类型
名+属性名
```

## 监控数据对象创建

获取监控属性后，通过 `MetricValue` 构造方法创建监控数据对象。

```
MetricValueVo metricValueVo = new MetricValueVo(attrVo,
testObj, testValue, curTime);
```

需要参数：

`AttrVo` 对象

监控对象名称，比如：不同的 mount 点，不同的交易名称，不同的网卡名称等，如只有一个对象，可用“-”替代

采集值：Number 或者 String 类型对象

采集时间：毫秒时间戳

## 属性自动创建

监控属性也可以在监控插件中自动创建

```
//jobContext 对象从 ThreadLocal 里获取
```

```
JobContext jobContext = CurrentContext.get();
```

```
JobExecHelper jobExecHelper = jobContext.getHelper();
```

```
//获取属性类型，没有则创建,参数：模板 vo，属性类型名
```

```
AttrTypeVo statusTypeVo =
```

```
jobExecHelper.getAttrTypeVo(tmplVo, "Status");
```

```
//获取监控属性，没有则创建,参数：模板 vo，属性类型 vo，属性名，属性类型
（0 表示数值型，1 表示字符型）
```

```
AttrVo responseTimeAttrVo = jobExecHelper.getAttrVo(tmplVo,
statusTypeVo, "ResponseTime", 0);
```

注 1：属性自动创建目前只能区分 **number** 和 **string** 类型，默认为 GAUGE(原始值)，其它如单位转换、增量值等需要在属性创建完后再自行修改。

注 2：自动创建属性默认会自动加到当前选择的监控模板里，不能通过模板的属性选择进行属性过滤。

## 2.1.4 插件模板

监控模板主要包括监控运行参数、调度参数和监控属性等。  
每个插件至少需要定义一个监控模板，监控插件将监控属性数据都封装到 TmplVo 对象里。

## 监控模板继承

监控作业基于监控模板配置，其中运行参数默认是继承监控模板里的定义

<input checked="" type="checkbox"/> 继承模板	主机:	<input data-bbox="670 654 1010 698" type="text" value="#{ip}"/>
<input type="checkbox"/> 继承模板	端口:	<input data-bbox="670 736 1010 781" type="text" value="22"/>
<input checked="" type="checkbox"/> 继承模板	监控模式:	<input data-bbox="670 819 1010 864" type="text" value="ssh"/>
<input type="checkbox"/> 继承模板	用户名:	<input data-bbox="670 902 1010 947" type="text" value="root"/>
<input type="checkbox"/> 继承模板	密码:	<input data-bbox="670 985 1010 1030" type="password" value="....."/>
<input checked="" type="checkbox"/> 继承模板	连接超时 (毫秒):	<input data-bbox="670 1068 1010 1113" type="text" value="3000"/> <span>ms</span>

运行配置		
<input checked="" type="checkbox"/> 继承模板	调度方式:	<input data-bbox="670 1265 1010 1310" type="text" value="周期执行"/>
	执行间隔:	<input data-bbox="670 1335 1010 1379" type="text" value="60"/> <span>秒</span>
<input checked="" type="checkbox"/> 继承模板	告警重复方式:	<input data-bbox="670 1417 1010 1462" type="text" value="不重复"/>
<input checked="" type="checkbox"/> 继承模板	告警动作:	<div data-bbox="670 1500 1010 1641"> <div>无动作</div> <div>[微信企业号]techsure企业号</div> <div>[发送邮件]testEmail</div> <div>[wechat]微信网关</div> </div>

图 2.1.4.1

需监控作业配置时选择不继承模板，则监控作业运行的时候会优先使用监控作业定义的参数，如上图的用户名/密码参数。

```
public MetricValueList execute(JobVo jobVo, JSONObject jobConf, TmplVo tmpVo);
```

监控插件运行时的 jobConf 对象是**合并后**的参数（监控作业的参数优先）。

## 2.2 插件设计

监控插件开发只需要自行实现 `execute` 方法，返回 `MetricValueList` 即可，`moncollector` 自动完成作业调度、数据的入口、阈值判断、告警产生等操作。

### 参数说明

```
/**
 * 插件执行入口
 * @param jobVo job 基本信息
 * @param jobConf 合并后的运行参数
 * @param tmplVo 模板数据，包含监控属性定义
 * @return
 */
@Override
public MetricValueList execute(JobVo jobVo, JSONObject
jobConf, TmplVo tmplVo) {
    MetricValueList dataList = collect(jobVo, jobConf,
tmplVo);
    return dataList;
}
```

`jobVo` 包含监控目标数据，如目标名称、IP、扩展属性等。  
`jobConf` 合并后的监控运行参数 `JSONObject` 对象。  
`tmplVo` 监控模板，主要是监控属性的集合。

### 程序结构

参考 `DemoDataCollector.java` 的 `collect` 方法：

```
//根据 jobId 初始化 metricDataList
MetricValueList dataList = new MetricValueList(jobId);
//获取运行参数
String host = jobConf.getString(PluginConstants.HOST);
//执行采集方法
...
...
//获取到监控对象和值后，封装 MetricValueVo 对象
Double testValue = 99.999D;
String testObj = "testObj";
AttrVo attrVo = tmplVo.getAttrVo("Test", "test");//属性类型
名 + 属性名
```

```

if(attrVo!=null) {
    MetricValueVo metricValueVo = new MetricValueVo(attrVo,
testObj, curTime, testValue);
    //放入 datalist 里
    dataList.add(metricValueVo);
}
//返回数据
return dataList;

```

## 2.3 运行测试

### 2.3.1 监控作业配置

在监控中心-监控目标界面，点击“添加目标”，录入一个新的监控目标，选择上述步骤创建的监控插件和模板，插件一个监控作业。

### 2.3.2 部署

执行 maven install，将生成的 plugin 插件（如 asm-plugin-demo-3.0.0.jar）放到 /app/systems/moncollector/plugins 目录，重启 moncollector 采集器。

命令：

```
deployadmin -s moncollector -a restart moncollector1
```

### 2.3.3 单元测试

继承 asm-plugin-test 的 CollectorTestBase 类，可以在开发环境加载作业运行，测试前可调用 initTest 方法，把上述的作业加载进来执行。

注：需要开发主机能连接到 bsm 配置数据库，修改 config.properties 文件的连接串地址，并且在 classpath 引用 config 目录。

```

public class DemoDataCollectorTest extends
CollectorTestBase{
    @Test
    public void executeTest(){
        //作业 ID，监控目标 IP、端口
        initTest(410L, "192.168.0.88", "80");
        DemoDataCollector collector = new
DemoDataCollector();
        MetricValueList metricValueList =
collector.execute(jobVo, jobConf, tmpVo);
        System.out.println(metricValueList);
    }
}

```

注：单元测试暂不能自动创建监控属性

## 3 Exagent 插件开发

### 3.1 插件定义

Exagent 插件定义与 2.1.1 类似，插件类型需要选择 EXAgent 类型，Exagent 模块写入新增的 Python 文件名，采集 Java 类留空。

#### 编辑插件

名称:	<input type="text" value="Linux监控"/>
插件类型:	<input type="text" value="EXAgent"/>
插件组:	<input type="text" value="操作系统"/>
告警动作(手动):	<input type="text" value="请选择..."/>
描述:	<input type="text"/>
采集Java类:	<input type="text"/>
Exagent模块:	<input type="text" value="LinuxMonitorPlugin"/>
配置表单:	<div><div>配置</div><div>帮助</div></div> <div>1</div>

图 3.1.1

## 3.2 插件设计

Exagent 插件只需要自行实现 plugin.py 的 collectData 方法即可。示例 (DemoPlugin.py):

```
class DemoPlugin(plugin.plugin):

    def __init__(self, jobConf):
        plugin.plugin.__init__(self, jobConf)

    #执行入口
    def collectData(self):
        try:
            #初始化数据
            startTime=self.getCurTime()
            self.intStatus()
            data = {}
            data["jobId"] = str(self.jobId)
            metricList = []

            logger.info(self.runConf)

            #获取参数
            host = self.runConf['host']

            #自行实现数据采集
            platForm = util.getOsPlatform() #示例
            #创建数据对象
            testMetric = self.getMetricValue("Test.platform",
            "Demo", startTime, platForm)
            metricList.append(testMetric)

            #可用性属性，自动创建属性
            metricAvail=self.getMetricValue("Status.Availability", "De
            mo",startTime,"OK", None, 1)
            metricList.append(metricAvail)

        except:
            logger.error(self.logHeader +
            traceback.format_exc())
```

```

metricAvail=self.getMetricValue("Status.Availability","De
mo",startTime,traceback.format_exc())
        metricList.append(metricAvail)

        #记录耗时
        endTime=long(self.getCurTime())

metricResponse=self.getMetricValue("Status.ResponseTime",
"Demo", startTime, str(endTime-long(startTime)))
        metricList.append(metricResponse)

        data["data"]=metricList
        #返回数据
        self.addMonData(data)

```

说明：

运行参数通过 `self.runConf` 获取，运行参数为 json 格式，定义方法参考 2.1.2

如果需要自动创建属性，需要调用方法(auto=1)：

```
self.getMetricValue(metricName,objectName,collectTime,value,error=None,auto=1)
```

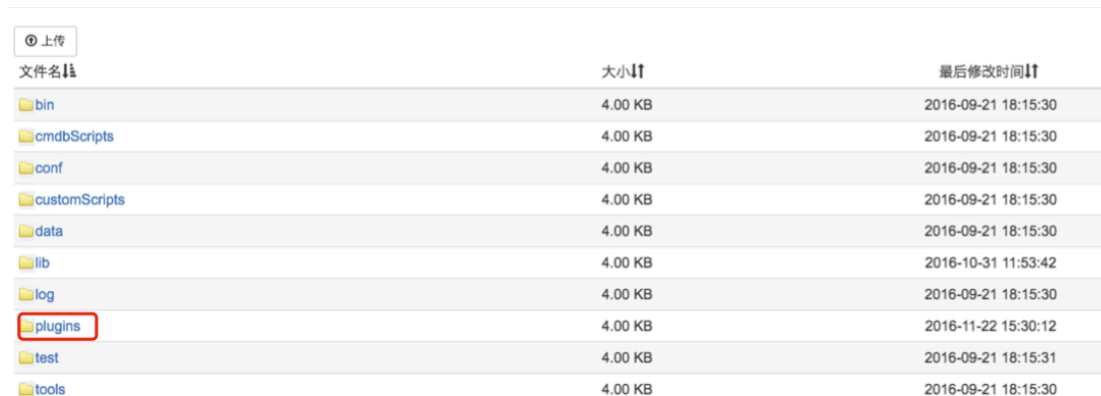
插件的运行日志记录在 `log/exagent.log` 文件里。

如需要引入第三方 lib，可放入 lib 目录，通过 3.3.1 文件上传方式传到服务器，并需要升级客户端版本（系统管理-exagent-配置属性管理）。

## 3.3 运行测试

### 3.3.1 文件上传

菜单：系统管理-文件管理



文件名	大小	最后修改时间
bin	4.00 KB	2016-09-21 18:15:30
cmdbScripts	4.00 KB	2016-09-21 18:15:30
conf	4.00 KB	2016-09-21 18:15:30
customScripts	4.00 KB	2016-09-21 18:15:30
data	4.00 KB	2016-09-21 18:15:30
lib	4.00 KB	2016-10-31 11:53:42
log	4.00 KB	2016-09-21 18:15:30
plugins	4.00 KB	2016-11-22 15:30:12
test	4.00 KB	2016-09-21 18:15:31
tools	4.00 KB	2016-09-21 18:15:30

图 3.3.1.1

进入 plugins 目录，点击上传文件。

### 3.3.2 作业运行

将插件上传到 web 端后，需要完成监控模板、监控目标和监控作业的配置，Exagent 客户端会自动通过 http 方式将上传的 python 文件下载到本地执行。

**注：exagent 插件暂不兼容 crontab 形式的调度方式。**

### 3.3.3 离线测试

默认 exagent 客户端需要和服务端校验 Plugin 文件的 md5 值，确保不被客户端恶意篡改。

如需要离线测试，不通过 web 端更新插件，可修改 exagent 的 conf/agent.ini 文件，将 autoupdate 设置为 False，重启 exagent 即可(可执行 bin/exagent.sh restart)。



## 4 监控动作开发

监控动作定义

### 4.1 新增动作

菜单：系统管理-监控-监控动作管理

编辑动作类型

×

类型:

通知

类型名称:

发送邮件

执行动作java类:

com.techsure.asm.action.plugin.email.EmailAction

存档目录名:

MAIL

配置表单:

配置

帮助

```

1 [ { "name": "fromAddress", "type": "text", "label": "发件人地址", "checktype":
2 { "name": "smtpserver", "type": "text", "label": "SMTP服务器", "checktype": "1
3 { "name": "smtpport", "type": "text", "label": "SMTP端口", "checktype": "req
4 { "name": "smtpusername", "type": "text", "label": "SMTP用户", "checktype":
5 { "name": "smtppassword", "type": "password", "label": "SMTP密码", "checktyp
6 { "name": "toAddress", "type": "text", "label": "收件人地址", "checktype": "r
7 { "name": "subject", "type": "text", "label": "主题", "checktype": "required
8 { "name": "mssage", "type": "text", "label": "附加信息", "checktype": "requir
9 { "name": "emailCharset", "type": "text", "label": "emianCharset", "checkty
10 ]

```

图 4.1.1

执行动作 java 类: java 类全路径, 如 com.techsure.asm.action.plugin.email.EmailAction

存档目录: 监控动作执行结果保存目录, 如 java/email 等

配置表单: 监控动作运行参数定义, 如短信网关地址等, 参考 2.1.2 节

添加完动作类型后，点击动作标题，进入监控动作实例管理。

#### 编辑动作

✕

动作名称:	<input type="text" value="testEmail"/>
告警级别:	<input checked="" type="checkbox"/> warn <input checked="" type="checkbox"/> critical <input checked="" type="checkbox"/> disaster <input type="checkbox"/> info
关联模板:	<input type="text" value="告警邮件模板"/>
发件人地址:	<input type="text" value="test@techsure.com.cn"/>
SMTP服务器:	<input type="text" value="smtp.exmail.qq.com"/>
SMTP端口:	<input type="text" value="25"/>
SMTP用户:	<input type="text" value="test@techsure.com.cn"/>
SMTP密码:	<input type="text" value="*****"/>
收件人地址:	<input type="text" value="test@techsure.com.cn"/>
主题:	<input type="text" value="告警动作测试"/>
附加信息:	<input type="text" value="监控报警"/>
emianCharset:	<input type="text" value="UTF-8"/>

图 4.1.2

告警级别：选中的级别才会触发执行动作，其中 info 为告警恢复对应的级别

关联模板：选择告警动作执行时所需的 freemaker 模板

动作参数：根据类型定义的配置表单确定，通常比如邮件服务器地址、收件人等

## 4.2 监控动作模板

菜单：系统管理-监控动作模板管理

监控动作执行可定义内容模板，比如发送邮件的模板，基于 freemaker 实现。

可以分别定义标题和内容模板，可以使用 freemaker 模板语法。

模板里可引用的对象有（封装为 Map<String, Object>）：

对象 KEY	值类型	属性
alertVoList	List<AlertVo>	AlertVo 对象： alertId: 告警 ID jobId: 作业 ID metricId: 指标 ID attrName: 属性名 attrTypeName: 属性类型名 objName: 对象名 alertLevel: 告警级别（数字） alertTime: 告警产生时间（毫秒时间戳） alertTimeStr: 告警产生时间 yyyy-MM-dd HH:mm:ss alertEndTimeStr: 结束时间 yyyy-MM-dd HH:mm:ss(恢复才有) alertMsg: 告警消息 alertThreshold: 告警阈值匹配结果 alertValue: 最近采集值
jobVo	JobVo	jobId: 作业 ID ownerId: 监控目标 ID ip: 监控目标 IP port: 监控目标端口 targetName: 监控目标名称 pluginName: 插件名称 targetExtend: 监控目标扩展属性（JSONObject 对象）
jobConf	JSONObject	<由监控作业和模板定义，参考 2.1.2 说明>
actionConf	JSONObject	<监控动作实例定义> 固定属性： dumpCatalogName: 动作执行结果存档目录名（相对目录） _actionTemlTitile: 标题模板（freemaker） _actionTemlContent: 内容模板（freemaker）
alertLevel	Int	当前作业 alertVoList 里的最高告警级别（同时产生多个告警时）
alertLevelConf	AlertLevel	alertLevel 对应的级别对象，包含的属性： level: 告警级别（告警恢复时为 0） name: 告警级别名称（告警恢复时为 INFO） color: 告警级别对应的颜色（告警恢复时为 green）

## 4.3 监控动作开发

### 4.3.1 程序结构

监控动作开发只需要继承 `com.techsure.asm.plugin.Action` 类，实现 `execute` 和 `myExecute` 方法即可。其中 `execute` 可用于在 web 界面手工执行，比如执行 jmx 的 thread dump（**没有告警信息！**），可不实现；`myExecute` 方法为告警触发，返回执行结果摘要（比如发送给 xxx）。

```
import com.techsure.asm.plugin.Action;
import com.techsure.asm.vo.AlertVo;
import com.techsure.asm.vo.JobVo;
import net.sf.json.JSONObject;

import java.util.List;
import java.util.Map;

public class DemoAction extends Action {

    @Override
    public String myExecute(List<AlertVo> list, JSONObject
actionConf, JobVo jobVo, JSONObject jobConf, Map<String, Object>
map) {
        return null;
    }

    @Override
    public String execute(JSONObject actionConf, JobVo jobVo,
JSONObject jobConf) {
        return null;
    }
}
```

### 4.3.2 运行参数

监控动作执行参数在监控动作实例里定义，封装成 `JSONObject` 对象。其它可用的参数有 `alertVoList`、`jobVo` 等，可参看 4.2 说明。

### 4.3.3 模板参数

获取标题模板：

```
String _actionTemlTitile = actionConf.getString("_actionTemlTitile");
```

模板渲染：

```
String subject = Util.getFreemarkerContent(paramMap, _actionTemlTitile);
```

获取内容模板:

```
String _actionTemlContent = actionConf.getString("_actionTemlContent");
```

内容渲染:

```
String content = Util.getFreemarkerContent(paramMap, _actionTemlContent);
```

获取动作执行结果保存路径:

```
String dumpCatalogName = actionConf.getString("dumpCatalogName ");
```

## 4.4 运行测试

执行 maven install, 将生成在 target 目录下的 jar 包放到 moncollector 的 plugins 目录, 重启 moncollector。