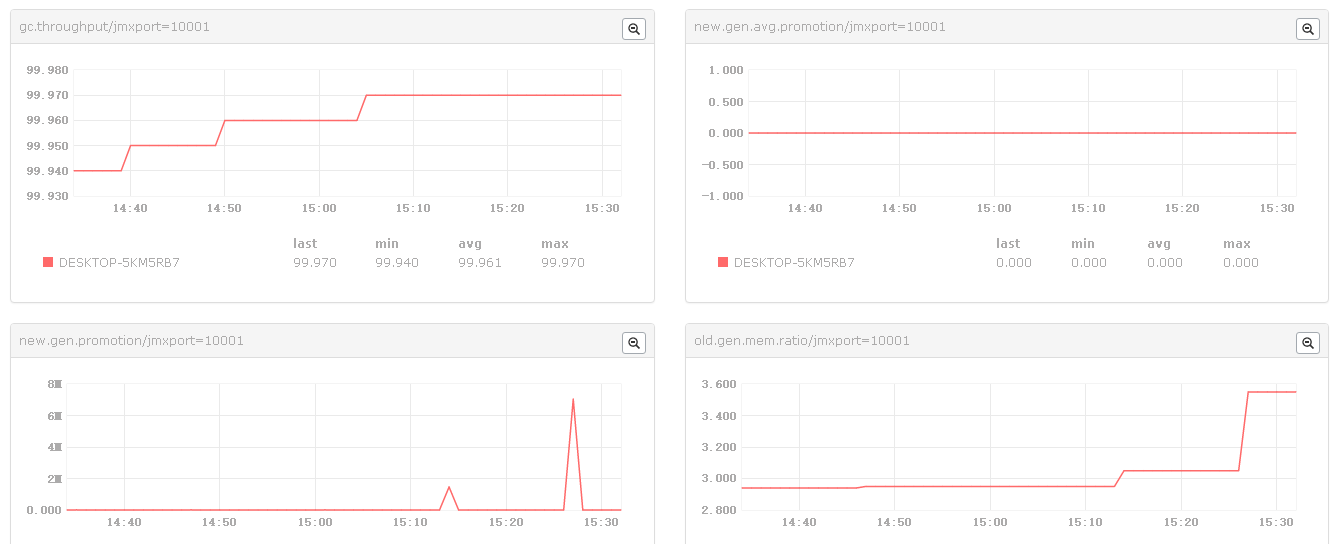
# 上报数据



[{

"metric": "ps marksweep.gc.avg.time",

"endpoint": "DESKTOP-5KM5RB7",

"timestamp": 1489212781,

"step": 60,

"value": 0.0,

"counterType": "GAUGE",

"tags": "jmxport=10001"

}, {

"metric": "ps marksweep.gc.count",

"endpoint": "DESKTOP-5KM5RB7",

"timestamp": 1489212781,

"step": 60,

"value": 0.0,

"counterType": "GAUGE",

"tags": "jmxport=10001"

}, {

"metric": "ps scavenge.gc.avg.time",

"endpoint": "DESKTOP-5KM5RB7",

"timestamp": 1489212781,

"step": 60,

"value": 0.0,

"counterType": "GAUGE",

"tags": "jmxport=10001"

}, {

"metric": "ps scavenge.gc.count",

"endpoint": "DESKTOP-5KM5RB7",

"timestamp": 1489212781,

"step": 60,

"value": 0.0,

"counterType": "GAUGE",

"tags": "jmxport=10001"

}, {

"metric": "gc.throughput",

"endpoint": "DESKTOP-5KM5RB7",

"timestamp": 1489212781,

"step": 60,

"value": 99.87,

"counterType": "GAUGE",

"tags": "jmxport=10001"

}, {

"metric": "old.gen.mem.used",

"endpoint": "DESKTOP-5KM5RB7",

"timestamp": 1489212781,

"step": 60,

"value": 4.1669024E7,

"counterType": "GAUGE",

"tags": "jmxport=10001"

}, {

"metric": "old.gen.mem.ratio",

"endpoint": "DESKTOP-5KM5RB7",

"timestamp": 1489212781,

"step": 60,

"value": 2.95,

"counterType": "GAUGE",

"tags": "jmxport=10001"

}, {

"metric": "new.gen.promotion",

"endpoint": "DESKTOP-5KM5RB7",

"timestamp": 1489212781,

"step": 60,

"value": 0.0,

"counterType": "GAUGE",

"tags": "jmxport=10001"

}, {

"metric": "new.gen.avg.promotion",

"endpoint": "DESKTOP-5KM5RB7",

"timestamp": 1489212781,

"step": 60,

"value": 0.0,

"counterType": "GAUGE",

"tags": "jmxport=10001"

}, {

"metric": "thread.active.count",

"endpoint": "DESKTOP-5KM5RB7",

"timestamp": 1489212781,

"step": 60,

"value": 36.0,

"counterType": "GAUGE",

"tags": "jmxport=10001"

}, {

"metric": "thread.peak.count",

"endpoint": "DESKTOP-5KM5RB7",

"timestamp": 1489212781,

"step": 60,

"value": 37.0,

"counterType": "GAUGE",

"tags": "jmxport=10001"

}

]

# 屏蔽javadoc生成

mvn clean package -DskipTests

<!--plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-javadoc-plugin</artifactId>  
 <version>2.9</version>  
 <executions>  
 <execution>  
 <phase>package</phase>  
 <goals>  
 <goal>jar</goal>  
 </goals>  
 </execution>  
 </executions>  
 <configuration>  
 <encoding>UTF-8</encoding>  
 <charset>UTF-8</charset>  
 <docencoding>UTF-8</docencoding>  
 </configuration>  
</plugin-->

# JMX监控 | Open-Falcon

https://book.open-falcon.org/zh/usage/jmx.html



jmxmon 简介

jmxmon是一个基于open-falcon的jmx监控插件，通过这个插件，结合open-falcon agent，可以采集任何开启了JMX服务端口的java进程的服务状态，并将采集信息自动上报给open-falcon服务端

## 主要功能

通过jmx采集java进程的jvm信息，包括gc耗时、gc次数、gc吞吐、老年代使用率、新生代晋升大小、活跃线程数等信息。

对应用程序代码无侵入，几乎不占用系统资源。

## 采集指标

| **Counters** | **Type** | **Notes** |
| --- | --- | --- |
| parnew.gc.avg.time | GAUGE | 一分钟内，每次YoungGC(parnew)的平均耗时 |
| concurrentmarksweep.gc.avg.time | GAUGE | 一分钟内，每次CMSGC的平均耗时 |
| parnew.gc.count | GAUGE | 一分钟内，YoungGC(parnew)的总次数 |
| concurrentmarksweep.gc.count | GAUGE | 一分钟内，CMSGC的总次数 |
| gc.throughput | GAUGE | GC的总吞吐率（应用运行时间/进程总运行时间） |
| new.gen.promotion | GAUGE | 一分钟内，新生代的内存晋升总大小 |
| new.gen.avg.promotion | GAUGE | 一分钟内，平均每次YoungGC的新生代内存晋升大小 |
| old.gen.mem.used | GAUGE | 老年代的内存使用量 |
| old.gen.mem.ratio | GAUGE | 老年代的内存使用率 |
| thread.active.count | GAUGE | 当前活跃线程数 |
| thread.peak.count | GAUGE | 峰值线程数 |

## 建议设置监控告警项

不同应用根据其特点，可以灵活调整触发条件及触发阈值

| **告警项** | **触发条件** | **备注** |
| --- | --- | --- |
| gc.throughput | all(#3)<98 | gc吞吐率低于98%，影响性能 |
| old.gen.mem.ratio | all(#3)>90 | 老年代内存使用率高于90%，需要调优 |
| thread.active.count | all(#3)>500 | 线程数过多，影响性能 |

使用帮助

详细的使用方法常见：[jmxmon](https://github.com/toomanyopenfiles/jmxmon)

# eclipse

-Djava.rmi.server.hostname=localhost -Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false -Dcom.sun.management.jmxremote.port=10001

# toomanyopenfiles/jmxmon: 基于open-falcon的jmx监控插件

https://github.com/toomanyopenfiles/jmxmon

jmxmon 简介

jmxmon是一个基于open-falcon的jmx监控插件，通过这个插件，结合open-falcon agent，可以采集任何开启了JMX服务端口的java进程的服务状态，并将采集信息自动上报给open-falcon服务端

## 主要功能

通过jmx采集java进程的jvm信息，包括gc耗时、gc次数、gc吞吐、老年代使用率、新生代晋升大小、活跃线程数等信息。

对应用程序代码无侵入，几乎不占用系统资源。

## 环境需求

Linux

JDK>=1.6

Open-Falcon>=0.0.5

目标java进程开启jmx端口

## jmxmon部署

1. 安装并启动open-falcon agent
2. 下载并解压编译好的 [release包](https://github.com/toomanyopenfiles/jmxmon/releases/latest) 到目标安装目录下
3. cp conf.example.properties conf.properties
4. 修改conf.properties配置文件，一般情况下只需要将jmx.ports的端口号配置上就可以了
5. sh control start
6. sh control tail查看日志，或者cat var/app.log以确认程序是否正常启动

## 配置说明

配置文件默认文件名为conf.properties，内容说明如下：

# 工作目录用来存放jmxmon的临时缓存文件，注意不要修改此目录下的文件

workDir=./

# 需要监听的本地jmx端口，支持监听多个端口，多端口用逗号分隔

jmx.ports=10000,10001,10002,10003

# 本地agent的上报url，如果使用open-falcon的默认配置，则这里不需要改变

agent.posturl=http://localhost:1988/v1/push

# 可选项：上报给open-falcon的endpoint，默认值为本机hostname。不建议修改

#hostname=

# 可选项：上报给open-falcon的上报间隔，默认值60，单位秒。不建议修改

#step=

## 采集指标

| **Counters** | **Type** | **Notes** |
| --- | --- | --- |
| parnew.gc.avg.time | GAUGE | 一分钟内，每次YoungGC(parnew)的平均耗时 |
| concurrentmarksweep.gc.avg.time | GAUGE | 一分钟内，每次CMSGC的平均耗时 |
| parnew.gc.count | GAUGE | 一分钟内，YoungGC(parnew)的总次数 |
| concurrentmarksweep.gc.count | GAUGE | 一分钟内，CMSGC的总次数 |
| gc.throughput | GAUGE | GC的总吞吐率（应用运行时间/进程总运行时间） |
| new.gen.promotion | GAUGE | 一分钟内，新生代的内存晋升总大小 |
| new.gen.avg.promotion | GAUGE | 一分钟内，平均每次YoungGC的新生代内存晋升大小 |
| old.gen.mem.used | GAUGE | 老年代的内存使用量 |
| old.gen.mem.ratio | GAUGE | 老年代的内存使用率 |
| thread.active.count | GAUGE | 当前活跃线程数 |
| thread.peak.count | GAUGE | 峰值线程数 |

## 建议设置监控告警项

不同应用根据其特点，可以灵活调整触发条件及触发阈值

| **告警项** | **触发条件** | **备注** |
| --- | --- | --- |
| gc.throughput | all(#3)<98 | gc吞吐率低于98%，影响性能 |
| old.gen.mem.ratio | all(#3)>90 | 老年代内存使用率高于90%，需要调优 |
| thread.active.count | all(#3)>500 | 线程数过多，影响性能 |

# JMXMonitor

E:\workspace\open-falcon\jmxmon\src\main\java\com\stephan\tof\jmxmon\JMXMonitor.java

## main

Config.I.init("conf.properties");初始化脚本

使用定时线程每隔1分钟上报数据。

public static void main(String[] args) throws IOException, ConfigurationException {  
  
 Config.*I*.init("conf.properties");  
   
 ScheduledExecutorService executor = Executors.*newScheduledThreadPool*(1);  
 executor.scheduleAtFixedRate(new Runnable() {  
 @Override  
 public void run() {  
 *runTask*();  
 }  
 }, 0, Config.*I*.getStep(), TimeUnit.*SECONDS*);  
   
 }

## runTask

根据配置中端口遍历，也就是说监听不同的应用程序

Config.I.getJmxPorts()

private static void runTask() {  
 try {  
 List<FalconItem> items = new ArrayList<FalconItem>();  
   
 for (int jmxPort : Config.*I*.getJmxPorts()) {  
 // 从JMX中获取JVM信息  
 ProxyClient proxyClient = null;  
 try {  
 proxyClient = ProxyClient.*getProxyClient*(Config.*I*.getJmxHost(), jmxPort, null, null);  
 proxyClient.connect();  
   
 JMXCall<Map<String, GCGenInfo>> gcGenInfoExtractor = new JVMGCGenInfoExtractor(proxyClient, jmxPort);  
 Map<String, GCGenInfo> genInfoMap = gcGenInfoExtractor.call();  
 items.addAll(gcGenInfoExtractor.build(genInfoMap));  
   
 JMXCall<Double> gcThroughputExtractor = new JVMGCThroughputExtractor(proxyClient, jmxPort);  
 Double gcThroughput = gcThroughputExtractor.call();  
 items.addAll(gcThroughputExtractor.build(gcThroughput));  
   
 JMXCall<MemoryUsedInfo> memoryUsedExtractor = new JVMMemoryUsedExtractor(proxyClient, jmxPort);  
 MemoryUsedInfo memoryUsedInfo = memoryUsedExtractor.call();  
 items.addAll(memoryUsedExtractor.build(memoryUsedInfo));  
   
 JMXCall<ThreadInfo> threadExtractor = new JVMThreadExtractor(proxyClient, jmxPort);  
 ThreadInfo threadInfo = threadExtractor.call();  
 items.addAll(threadExtractor.build(threadInfo));  
 } finally {  
 if (proxyClient != null) {  
 proxyClient.disconnect();  
 }  
 }  
 }  
   
 // 发送items给Openfalcon agent  
 String content = JacksonUtil.*writeBeanToString*(items, false);  
 HttpResult postResult = HttpClientUtils.*getInstance*().post(Config.*I*.getAgentPostUrl(), content);  
 *logger*.info("post status=" + postResult.getStatusCode() +   
 ", post url=" + Config.*I*.getAgentPostUrl() + ", content=" + content);  
 if (postResult.getStatusCode() != HttpClientUtils.*okStatusCode* ||  
 postResult.getT() != null) {  
 throw postResult.getT();   
 }  
   
 // 将context数据回写文件  
 Config.*I*.flush();  
 } catch (Throwable e) {  
 *logger*.error(e.getMessage(), e);  
 }  
}

# ProxyClient

E:\workspace\open-falcon\jmxmon\src\main\java\com\stephan\tof\jmxmon\jmxutil\ProxyClient.java

## Connect()

public void connect() {  
 setConnectionState(ConnectionState.*CONNECTING*);  
 try {  
 tryConnect();  
 setConnectionState(ConnectionState.*CONNECTED*);  
 } catch (Exception e) {  
 setConnectionState(ConnectionState.*DISCONNECTED*);  
 throw new IllegalStateException(e);  
 }  
}

## setConnectionState

// The SwingPropertyChangeSupport will fire events on the EDT  
private SwingPropertyChangeSupport propertyChangeSupport =  
 new SwingPropertyChangeSupport(this, true);

private void setConnectionState(ConnectionState state) {  
 ConnectionState oldState = this.connectionState;  
 this.connectionState = state;  
 propertyChangeSupport.firePropertyChange(*CONNECTION\_STATE\_PROPERTY*,  
 oldState, state);  
}

关联属性，也称之为绑定属性。绑定属性会在属性值发生变化时，通知所有相关的监听器。为了实现一个绑定属性，必须实现两个机制。

1）  无论何时，只要属性的值发生变化，该bean必须发送一个PropertyChange事件给所有已注册的监听器。该变化可能发生在调用set方法时，或者程序的用户做出某种动作时。

2）  为了使感兴趣的监听器能够进行注册，bean必须实现以下两个方法：

|  |
| --- |
| **void** addPropertyChangeListener(PropertyChangeListener listener);  **void** removePropertyChangeListener(PropertyChangeListener listener); |

// The SwingPropertyChangeSupport will fire events on the EDT  
private SwingPropertyChangeSupport propertyChangeSupport =  
 new SwingPropertyChangeSupport(this, true);

public void addPropertyChangeListener(PropertyChangeListener listener) {  
 propertyChangeSupport.addPropertyChangeListener(listener);  
}

public void removePropertyChangeListener(PropertyChangeListener listener) {  
 propertyChangeSupport.removePropertyChangeListener(listener);  
}

## getGarbageCollectorMXBeans

public synchronized Collection<GarbageCollectorMXBean> getGarbageCollectorMXBeans()  
 throws IOException {  
  
 // *TODO: How to deal with changes to the list??* if (garbageCollectorMBeans == null) {  
 ObjectName gcName = null;  
 try {  
 gcName = new ObjectName(*GARBAGE\_COLLECTOR\_MXBEAN\_DOMAIN\_TYPE* + ",\*");  
 } catch (MalformedObjectNameException e) {  
 // should not reach here  
 assert(false);  
 }  
 Set mbeans = server.queryNames(gcName, null);  
 if (mbeans != null) {  
 garbageCollectorMBeans = new ArrayList<GarbageCollectorMXBean>();  
 Iterator iterator = mbeans.iterator();  
 while (iterator.hasNext()) {  
 ObjectName on = (ObjectName) iterator.next();  
 String name = *GARBAGE\_COLLECTOR\_MXBEAN\_DOMAIN\_TYPE* +  
 ",name=" + on.getKeyProperty("name");  
  
 GarbageCollectorMXBean mBean =  
 *newPlatformMXBeanProxy*(server, name,  
 GarbageCollectorMXBean.class);  
 garbageCollectorMBeans.add(mBean);  
 }  
 }  
 }  
 return garbageCollectorMBeans;  
}

# JVMGCGenInfoExtractor

E:\workspace\open-falcon\jmxmon\src\main\java\com\stephan\tof\jmxmon\JVMGCGenInfoExtractor.java

## 构造器

public JVMGCGenInfoExtractor(ProxyClient proxyClient, int jmxPort) throws IOException {  
 super(proxyClient, jmxPort);  
}

# JVMDataExtractor

E:\workspace\open-falcon\jmxmon\src\main\java\com\stephan\tof\jmxmon\JVMDataExtractor.java

## 构造器

### 获取所有的数据

public JVMDataExtractor(ProxyClient proxyClient, int jmxPort) throws IOException {  
 super(proxyClient, jmxPort);  
 gcMXBeanList = proxyClient.getGarbageCollectorMXBeans();  
 runtimeMXBean = proxyClient.getRuntimeMXBean();  
 memoryPoolList = proxyClient.getMemoryPoolProxies();  
 threadMXBean = proxyClient.getThreadMXBean();  
}