# 简析运维监控系统及Open-Falcon - Java之旅 - 博客频道 - CSDN.NET

http://blog.csdn.net/puma\_dong/article/details/51895063

## 前言

监控系统，可以从运营级别（基本配置即可），以及应用级别（二次开发，通过端口进行日志上报），对服务器、[**操作系统**](http://lib.csdn.net/base/operatingsystem)、中间件、应用进行全面的监控，及报警，对我们的系统正常运行的作用非常重要。

1、开源还是商用？十大云运维监控工具横评

<http://www.oschina.net/news/67525/monitoring-tools>

2、Zabbix、Nagios、Open-Falcon这3大开源运维监控工具的比较：



我所在的公司，在去年完成了从Zabbix到Open-Falcon的转换，有从小米过来的运维工程师的大力推动的影响，也是由于对于巨量的数据上报/统计，万+上报节点，Zabbix力不从心了，而Open-Falcon这方面更加强大，转换之后，感觉Open-Falcon使用的比较舒服。

所以本文就结合Falcon，对监控系统的作用，进行一些简单的分析；并结合日常的工作学习过程，对监控参数的意义，进行持续的学习和补充。

## Open-Falcon

小米开发的，小米、金山云、美团、京东金融等公司在用。

Open-Falcon编写的整个脑洞历程：<http://mp.weixin.qq.com/s?__biz=MjM5OTcxMzE0MQ==&mid=400225178&idx=1&sn=c98609a9b66f84549e41cd421b4df74d>

小米开源监控系统OpenFalcon应对高并发7种手段：<http://h2ex.com/894>

官网：

<http://open-falcon.org/>

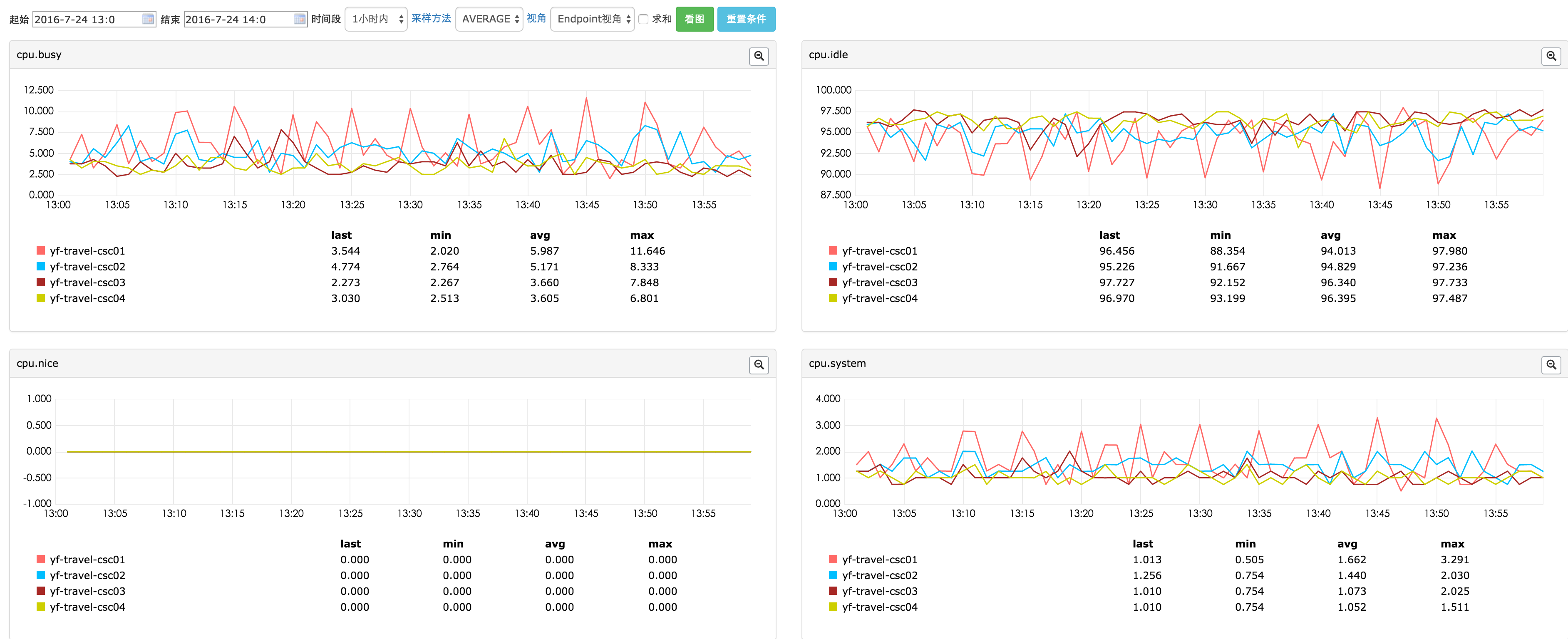
<http://book.open-falcon.org/zh/index.html>

### 基础监控

CPU、Load、内存、磁盘、IO、网络相关、内核参数、ss 统计输出、端口采集、核心服务的进程存活信息采集、关键业务进程资源消耗、NTP offset采集、DNS解析采集，这些指标，都是open-falcon的agent组件直接支持的。

[**Linux**](http://lib.csdn.net/base/linux)运维基础采集项：<http://book.open-falcon.org/zh/faq/linux-metrics.html>

对于这些基础监控选项全部理解透彻的时刻，也就是对Linux运行原理及命令进阶的时刻。



### 第三方监控

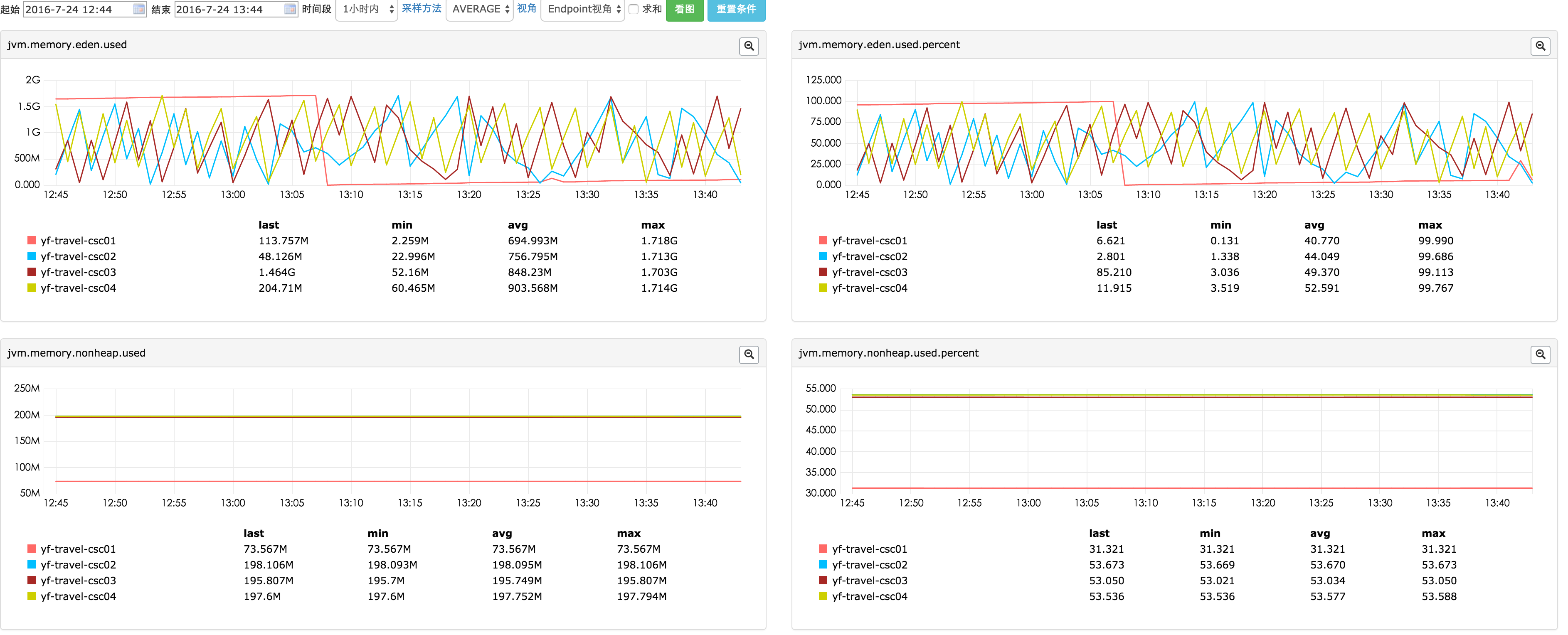
术业有专攻，运行在OS上的应用甚多，Open-Falcon的开发团队不可能把所有的第三方应用的监控全部做完，这个就需要开源社区提供更多的插件，当前对于很多常用的第三方应用都有相关插件了。

### JVM监控

对于[**Java**](http://lib.csdn.net/base/javase)作为主要开发语言的大多数公司，对于JVM的监控不可或缺。

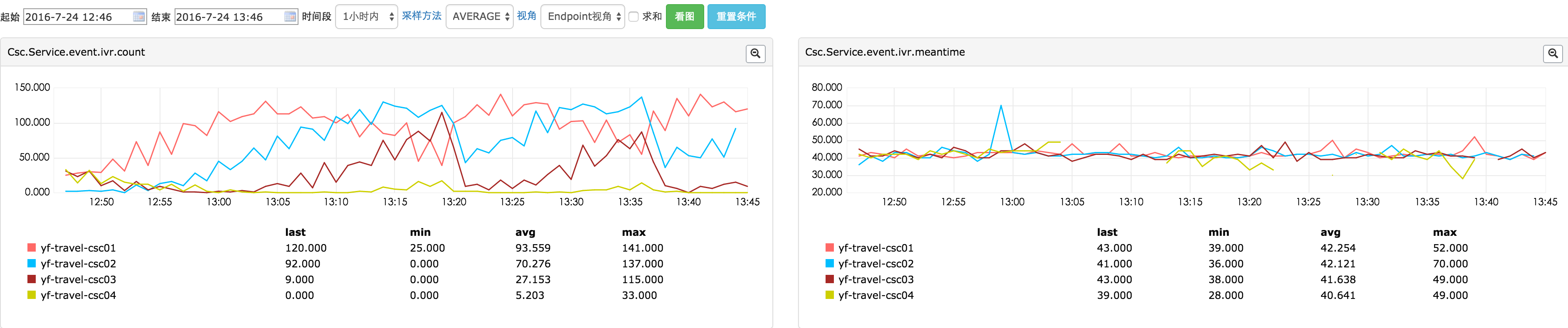
每个JVM应用的参数，比如GC、类加载、JVM内存、进程、线程，都可以上报给Falcon，而这些参数的获得，都可以通过MxBeans实现。

使用 Java 平台管理 bean：<http://www.ibm.com/developerworks/cn/java/j-mxbeans/>



### 业务应用监控

对于业务需要监控的接口，比如响应时间等。可以根据业务的需要，上报相关数据到Falcon，并通过Falcon查看结果。



## 总结

每个公司都应该根据自己的情况，比如使用的开发语言，使用的第三方框架，定义合适的插件，并开发合适的监控上报客户端，比如XMonitor，进行相关的第三方应用及业务应用的数据上报，达到对业务支出系统进行全方位监控的目的，既有系统层面，也有业务层面。

**监控对于业务开发的影响：**

1、有好的，有坏的，每个人都自己想想，自己想的都是对的；

2、有监控，有报警，出了问题早知道；

3、我们对于我们系统的运行情况，更清晰了，除了问题，到监控系统查出蛛丝马迹，比到OS上通过命令查询，效率更高；同时呢，其实对于开发人员的要求降低了；

4、几年前，想监控某个接口的调用次数，响应时间啥的，low的要自己写，自己出统计结果，其实通过上报到某个监控系统，就完全能解决；

顶

0

踩

0

上一篇[ZooKeeper源码之旅--搭建Eclipse工程](http://blog.csdn.net/puma_dong/article/details/52000283)

下一篇[记一次PermGen持续增长的解决过程](http://blog.csdn.net/puma_dong/article/details/51895055)

我的同类文章

架构运维（16）

•[记一次PermGen持续增长的解决过程](http://blog.csdn.net/puma_dong/article/details/51895055)2016-07-25阅读1527

•[Java之旅--跨域（CORS）](http://blog.csdn.net/puma_dong/article/details/51395976)2016-05-13阅读663

•[How Tomcat Works -- 目录](http://blog.csdn.net/puma_dong/article/details/50101993)2015-12-08阅读370

•[jetty](http://blog.csdn.net/puma_dong/article/details/46595501)2015-06-22阅读1024

•[程序人生--世界观](http://blog.csdn.net/puma_dong/article/details/46225681)2015-07-24阅读642

•[Swagger介绍-一套流行的API框架](http://blog.csdn.net/puma_dong/article/details/51752099)2016-06-24阅读2332

•[程序人生--架构师](http://blog.csdn.net/puma_dong/article/details/50479922)2016-01-12阅读340

•[记一次TcpListenOverflows报警解决过程](http://blog.csdn.net/puma_dong/article/details/46669499)2015-06-28阅读1396

•[Http抓包工具--查尔斯](http://blog.csdn.net/puma_dong/article/details/46591481)2015-06-22阅读1795

•[Java之旅--硬件和Java并发（神之本源）](http://blog.csdn.net/puma_dong/article/details/45439707)2015-05-04阅读696

[更多文章](http://blog.csdn.net/puma_dong/article/category/2403979)

# open-falcon编写的整个脑洞历程

http://mp.weixin.qq.com/s?\_\_biz=MjM5OTcxMzE0MQ==&mid=400225178&idx=1&sn=c98609a9b66f84549e41cd421b4df74d

钟搞定，写稿件的时候越写越多，限于个人表达能力的问题，而又想尽量讲清楚，总是啰啰嗦嗦的，最后发现时间关系还是没法全部讲到，大家见谅。

我们先来讲讲zabbix时代的痛，小米初期是使用zabbix来做监控的。zabbix大家应该是知道的，大名鼎鼎，成熟稳健，很多公司都在用。小米也不例外，初期做运维开发的人比较少，机器量、业务量也少，zabbix可以很好的满足需求。

我们要开发一个项目、产品，肯定是要解决一些问题，zabbix是很好，但是机器量、业务量上来之后，zabbix就有些力不从心了。那我们先来说一下zabbix存在的问题。

小米在大规模使用zabbix的年代，我当时在研究CloudFoundry，直到众位sre受不了zabbix了，希望开发一个新的监控系统，我们开始组建监控团队，开始open-falcon的设计、编码。所以对zabbix的问题可能理解不深入，抛砖引玉……

1. 性能瓶颈。zabbix是使用MySQL来存放监控历史数据的。一台机器假设有100个监控项，2000台机器，就是20w监控项，监控系统的数据采集没有高峰低谷，是持续性的，周期性的，一般是一分钟采集一次。

机器量越来越大，数据量就越来越大，MySQL的写入逐渐成为瓶颈，业界有一些proxy的方案，我们有试用，也尝试把采集周期调长，比如3分钟采集一次，或者5分钟采集一次，但是都治标不治本。

zabbix有些数据采集是通过pull的方式，也就是server端主动探测的方式，当目标机器量大了之后，这些pull任务也经常出现积压。维护zabbix的同学焦头烂额。

2. 管理成本高昂。为了让zabbix压力小一点，我们整个公司搭建了多套zabbix，比如多看自己用一套、电商自己用一套、米聊自己用一套，如果要做一些公司级别的统计，需要去多个数据源拉取数据。每套zabbix都得有运维人员跟进，人力成本上升。

3. zabbix有些易用性问题。比如zabbix的模板是不支持继承的，机器分组也是扁平化的，监控策略不容易复用。zabbix要采集哪些数据，是需要在server端做手工配置的，我们认为这是一个本该省去的操作。

4. 监控系统没有统一化。机器相关的数据采集、监控我们使用zabbix来做，但是我们还需要业务的监控，比如某个thrift rpc接口，每分钟调用的cps，latency，我们希望做监控，某些url的5xx、4xx我们也希望做监控，某个开源软件，比如redis、openstack、mysql的一些状态统计数据，我们也希望做监控。

我们称后面这些数据为performance数据，我同事来炜曾经专门写了一个系统叫perfcounter，用来存放这些performance数据。perfcounter使用rrdtool来绘图，rrd是一个环形数据库，很多监控系统的数据都是使用rrd来存储的，这个就不展开讲了，大家可以google一下rrd相关知识。

perfcounter的绘图做得不错，但是报警功能比较薄弱。zabbix本身有多套，再加上perfcounter，入口比较多，要看监控数据，可能要去不同的地方分别查看，比较麻烦。迫切需要一个大一统的解决方案。

不可否认，zabbix是个很优秀的方案，初期zabbix帮我们解决了很大的问题，不过如上所述，我们在使用过程中也遇到了一些问题，刚开始的时候提到过，zabbix时代我还在做PaaS，所以上面的说法主要是在做open-falcon设计的时候从各位sre了解到的。

下面我们来说说open-falcon的目标制定。我们已经确定要做一个新的监控解决方案了。于是开始讨论，这货应该做成什么样子。每个人可能都会有一些思考，但是每个point基本都比较散乱，我初入监控这个业务，主要是推动大家的讨论，然后整理方案，再讨论，再整理，最终出一个产品需求文档和概要设计。

有人说，我们应该既可以处理机器监控，也可以处理业务监控，还可以处理各种开源软件的监控。

没错，就是要这种大一统，而且不同的系统有不同的采集方式，有不同的监控指标，DBA对MySQL熟悉，知道应该采集哪些指标，云存储团队对HBase熟悉，知道应该采集哪些指标，但是我作为监控系统的开发人员，我对MySQL不熟，对HBase不熟，所以我们无法提供其对应的采集脚本。

下面重点！

所以我们要制定规范，制定推送接口，大家自己去采集自己的系统，完事按照我们的规范，把数据组织成监控系统需要的数据格式，发送给监控系统的接口。这样大家就可以共建监控数据，把各种需要监控的软件、平台都纳入进来。

当然了，对于操作系统的一些指标，比如cpu、内存、IO、网卡、磁盘、负载等等，这个还是需要我们监控系统开发人员去提供采集机制的。

于是，我们仿照zabbix\_agentd，编写了falcon-agent。

这一部分主要是讲监控系统的数据采集机制

这里有哪些思考的点呢？首先，我们不希望用户在server端配置哪个机器应该采集哪个指标，这样做麻烦且没必要。我们尽量把agent做得可以自发现，比如某机器有12块盘，agent应该可以自动探测到，然后采集各块盘的指标push给server；比如某个机器有2块网卡，agent也应该自动探测到，把各块网卡的流量、丢包率等信息采集push给server。

就是说，我们希望在装机的时候就把agent安装好，agent就可以自动去采集相关数据了，省去了服务端的配置。

顺着这个思路继续延伸一下哈，天不遂人愿，我们希望服务端不做任何配置，agent就可以自动去采集，这是我们的设计哲学，但是有的时候还是要打破一下，比如端口存活监控、进程数监控。

我们拿端口存活监控来举例，按照我们自动去采集数据的哲学，agent应该如何处理端口存活性呢？大家稍微思考一下。

刚开始我们的想法是这样的：我们可以把当前机器上监听的所有tcp端口收集到，汇报给server，比如某机器监听了22、80两个端口，ss -tln，获取之。这样的确是可以收集到数据。但是我现在要做80端口的存活性监控，server端应该怎么做呢？

想想cpu.idle，我们通常会在server端做配置，说某个机器的cpu.idle，连续3次<5%，就报警，最多报3次。这个策略要想正常工作，cpu.idle的数据就应该源源不断的上来。每个周期都不能少。

但是端口监控，ss -tln，可以获取当前有哪些tcp端口在监听，假设现在nginx进程挂了，80端口不再监听，ss -tln就只能获取到22端口，获取不到80端口了，于是汇报给server端的数据就少了80端口的数据。

类似cpu.idle的监控方式是需要数据源源不断上来的，于是，端口监控在这种模式下无法完成。怎么办？

我们可以写一个nodata的组件，来判断数据多长时间没上来就报警。但是nodata组件实现起来还有点小麻烦，而且还存在另外两个问题，一个是策略配置方式与cpu.idle等数据的策略配置方式不一样，比如cpu.idle是all(#3)<5报警，端口存活需要配置成类似：nodata(300)，方式不同用户的使用成本会上升；再一点是有些机器可能监听了N多个端口，但是真正需要监控的端口，只有少量几个，自发现端口造成资源浪费。

all(#3) 表示连续3次都

nodata(300) 表示300s没数据上来

换个方式……

端口不再做成自发现的。用户要想对端口做监控，最后肯定是要配置策略的。我们从用户配置的策略中找出所有需要监控的端口，通过hbs模块下发给agent，这样agent就可以只对特定端口做监控了。比如刚才的例子，某主机需要监控22和80两个端口，用户必然在服务端配置策略，我们从策略中分析出22和80，然后下发给对应的agent，agent就可以只监控这俩端口了，端口活着，汇报个1给服务端，端口死了，汇报个0给服务端，比如:net.port.listen/port=22 value=1，net.port.li1，net.port.li1，net.port.listen/port=80 value=0

下发这个动作是通过hbs这个模块来完成的，即heartbeat server。agent每分钟去调用hbs，询问自己应该监听哪些端口、哪些进程、要执行哪些插件。通过这种机制来下发一些状态信息。

OK，数据采集这部分，就是这么多内容：

1. server端制定接口规范，以此接入各种监控数据

2. agent自发现采集各种Linux性能指标，无需server端做配置

3. 进程、端口监控等无法做到自发现的，又不想引入nodata组件，想让策略配置统一化，就需要hbs来下发信息。

继续下个点之前，这里再补充两点不太重要的。

1. agent要采用什么语言来开发？

开发open-falcon这个系统之前，其实我最熟悉的语言是java。但是agent要run在所有目标机器上的，用自己最熟悉的java来开发么？要run agent，每个机器上都要先启动一个java虚拟机，这有点扯……

我们希望找一个资源占用尽量少的语言，C、C++或许是个不错的选择，但是C、C++我并不熟悉，据说写不好还容易内存泄露，呵呵。C/C++相对更底层，对于不同的操作系统可能要写很多分支，简单看过zabbix\_agentd的源码，各种操作系统的分支处理，很麻烦。当然了，zabbix希望兼容各种操作系统，所以写的分支比较多，我们公司的操作系统清一色centos，不会有这么多分支判断。

我们希望找一个更工程化的语言，不容易出错的语言，毕竟要在所有的机器上部署，今天这个core了明天那个core了也让人受不了。

go语言看着还不错哦。首先，go是静态编译的，编译完了一个二进制，扔到相同平台的机器上可以直接跑，不需要安装乱七八糟的lib库，这点特别吸引我，特别是我在上家公司做过相当长一段时间的自动化部署，这种部署友好的静态编译发布，我喜欢。

go的进程不需要java那种虚拟机，资源占用比较少；go的并发支持是语言层面的，server端需要并发的组件用着合适；go的资源回收是defer关键字，也不是传统的try...catch...finally，干净且不容易出错；go的function可以有多个返回值，错误处理通常是一个额外的error类型的返回值，无需抛异常，因为是返回值，不刻意忽略的话就肯定会记得处理，否则编译都过不了；go与github结合，生态建立的比较快；go的模板看起来怪怪的，还好我不需要用……

2. agent怎么扩展？

我们可以在agent中预置一些采集项，但是我们不是神，无法覆盖所有的需求。有些用户需要扩展agent的功能，采集更多的指标。我们需要提供这种机制。这就是插件机制的设计初衷。

zabbix是有一个目录，大家只要把采集脚本放到这个目录，zabbix就去执行。这样做自然是可以解决问题，但是有些问题其实是扔给了使用者，比如脚本的分发管理问题。

那我们来思考，要采集数据发送给server端，应该要做哪些事情呢？

a) 写一个采集数据的脚本

b) 把脚本分发到需要采集数据的机器上

c) 写一个cron每隔一段时间去跑这个脚本

d) 收集脚本的输出，调用server的接口去push数据

e) 根据需求相应的对脚本做上线、下线、升级

插件机制是如何解决这几个过程的呢？首先，脚本肯定还是要用户自己写，但是我们提供一个脚本的管理，采集脚本是代码，代码就需要版本管理，我们内部有个gitlab，要求用户把插件提交到我们指定的git repo。然后agent每隔一段时间去git pull这个git repo，采集脚本就完成了分发。

然后我们在server端提供一个配置，哪些机器应该执行哪些插件，通过hbs把这个信息分发给agent，agent就可以去执行了，脚本按照一个什么周期去跑呢？这个信息写在脚本的名称里，比如60\_ntpoffset.sh，60表示60s执行一次，agent通过对脚本文件名的分析，可以得知执行周期。

执行周期也可以在server端配置，都可以，我们是放到文件名里了

脚本执行完了，把输出打印到stdout，agent截获之后push给server。插件的升级、回滚，就是通过git repo和server端的配置来完成。

OK，数据采集就说这么多，真的是又臭又长，哈。我们回到系统设计阶段，看还有哪些点值得分享一下。

1. tag的设计。这个要好好跟大伙说说。这个设计灵感来自opentsdb，我们说监控数据有很多，如果对每条监控项都配置报警，是一个很繁重的工作量。那我们能否通过某个手段来对采集项做个聚合，一条配置可以搞定多个监控项？

举个例子，比如某台机器有多个分区:

/

/home

/home/work/data1

/home/work/data2

/home/work/data…

/home/work/data12

我们想配置任何一个分区的磁盘剩余量小于5%就报警，怎么办？上例中，假设我们有12块盘，加上/home分区和/分区，就有14个采集项，写14条策略规则？OMG……

这个时候tag机制就派上用场了，采集到的数据在汇报的时候，每条数据组织成这个样子：

{

"endpoint": "qd-sadev-falcon-graph01.hd",

"metric": "df.bytes.free.percent",

"tags": "fstype=ext4,mount=/home",

"value": 10.2,

"timestamp": 1427204756,

"step": 60,

"counterType": "GAUGE"

}

上面表示qd-sadev-falcon-graph01.hd这个机器的/home分区的磁盘剩余百分比（df.bytes.free.percent）

再举个例子

{

"endpoint": "qd-sadev-falcon-graph01.hd",

"metric": "df.bytes.free.percent",

"tags": "fstype=ext4,mount=/home/work/data1",

"value": 10.2,

"timestamp": 1427204756,

"step": 60,

"counterType": "GAUGE"

}

metric（监控项名称）是相同的，只是tag不同，不同的tag表示不同的挂载点。这样一来，我们就可以这么配置策略：说，对于某一批机器而言，df.bytes.free.percent这个采集项，只要value<5就报警。我们没有配置tag，那么这14个挂载点的数据都与这个策略关联，一个策略配置，搞定了14条数据。还有比如网卡也可以用类似的处理方式。

上面的例子还无法完全体现出tag的威力。我们再举个例子，说一个业务监控的例子。小米的好多服务都是java写的，有个团队依据open-falcon提供的接口规范，写了一个通用jar包，所有thrift中间层服务，只要引入这个jar包，就可以自动采集所有rpc接口的调用延迟，于是，产生了N多这样的数据：

{

"endpoint": "qd-sadev-falcon-judge01.hd",

"metric": "latency",

"tags": "department=sadev,project=falcon,module=judge,method=com.mi.falcon.judge.rpc.send",

"value": 10.2,

"timestamp": 1427204756,

"step": 60,

"counterType": "GAUGE"

}

插一句：业务代码中嵌入监控采集逻辑，目前我们的实践来看，真的很好用，嘿

如果我们这么配置：latency/department=sadev all(#2) > 20就报警。就意味着对sadev这个部门的所有rpc接口的latency都做了策略配置。覆盖了N条监控数据。怎么样？有那么点意思吧？

说白了，tag就是一种聚合手段，可以用更少的配置覆盖更多的监控项。

OK，tag的设计就讲这么多。下面我们说说模板继承。zabbix的host有一个扁平的group来管理，模板是无法继承的，这个让我们的管理非常不方便。举个例子。

sadev这个部门的所有机器可以放到一个group，配置load.1min大于20报警，sadev这个部门下有很多项目，比如falcon这个项目，falcon这个项目整体使用机器资源比较狠，于是配置的load.1min大于30报警，falcon这个项目下有很多模块，比如graph模块，graph的机器负载更重，正常的load.1min都是35，我们希望load.1min大于40报警。于是问题来了

qd-sadev-falcon-graph01.hd这个机器是graph的一台机器，它应该处于graph这个组，自然也应该处于falcon这个组和sadev这个组，于是，这个机器与三个模板都有绑定关系，load.1min>20的时候报一次警，>30 >40的时候都会报警，这显然不是我们需要的。

模板继承可以解决这个问题，graph.template继承自falcon.template，falcon.template又继承自sadev.template，sadev.template中配置各种常用监控项，比如cpu.idle < 5报警，df.bytes.free.percent < 5报警，load.1min < 20报警，falcon.template和graph.template中只是load.1min比较特殊，那就只需要对load.1min配置特定的阈值即可。如此一来，graph的机器只有在load.1min>40的时候报警，>20的时候>30的时候都不会报警，大家理解一下。

模板的思考就讲解到这里。下面说说这个机器分组的问题，open-falcon与zabbix一样，都是使用扁平的分组结构。这种情况有个问题，就是机器的维护不方便，比如上面的例子，我们graph机器要扩容，需要把新扩容的机器加入graph这个组，同时也要加入falcon这个组，加入sadev这个组，比较麻烦。

所以小米内部的实践方式是与内部的机器管理系统相结合，那是一个树状的结构，机器加入graph中，也就自动加入了falcon中，自动加入了sadev中。大家如果要在公司内部引入open-falcon，最好也要做个二次开发，与自己公司的CMDB、服务树、机器管理之类的系统结合一下。

那这里有一个点，好多公司把机器管理、服务树、CMDB这种管理机器的系统做成树状结构，原因何在？部署系统有这个需求么？部署系统的最小部署单位通常是模块，一个模块所在的机器完全可以放到一个扁平的group中。树状结构不是部署系统的需求，而是监控系统的需求，监控系统通常在一个大节点上绑定一些常用的策略配置，然后在小节点上绑定一些特殊的策略配置，相互之间有个继承覆盖关系。好多人不理解这个树状结构的设计初衷，我的理解就是这个样子的，不一定对哈，个人观点而已。

我们继续往下讲：架构设计的习惯性

这个阶段有哪些脑洞历程呢？首先，监控系统的数据量估计是不小，你看一个zabbix都扛不住对吧，哈哈……而且没有高峰低谷，周期性的，每时每刻都有大量数据上来。那第一反应是什么？

没错，上集群！一台机器无论如何都是搞不定的，那就必须要搞多个机器协同工作。

监控系统的本质，在我看来就是采集数据并做处理的一个过程。处理方式最重要的有两种，一个是报警，一个是存储历史数据。这两个处理逻辑差异比较大，报警要求速度很快，尽量在内存里完成；存储历史数据主要是写磁盘，读少写多。

那最直观的思路就是把报警做成一个组件，把数据存储做成另一个组件。先说存储，之前说过，我们内部有个perfcounter系统来存放performance数据，这个系统是用rrd来存放的数据。架构设计的习惯性，让我们再次选择使用rrd来做存储，毕竟rrd是很多监控系统的存储介质，随大流，问题不大。

rrd是本地文件，一台机器的磁盘容量或许可以搞定监控系统的所有数据，但是IO肯定不够，perfcounter当时用了5台机器，全部都是ssd做raid10，IO还是比较高。所以，要搞多台机器来存放rrd文件，每台机器只处理一部分，压力就小了。

于是，我们习惯性的使用perfcounter采用的一致性哈希算法，来对数据做分片。数据分片通常有两种方式，一个是使用算法来计算，一个是在中心端记录一个对应关系。监控系统的量会比较大，如果要使用对应关系，还得维护一个存储，当时我个人其实是主张使用一个存储来存放对应关系的，但是团队其他成员都觉得略麻烦，就放弃了，直接使用一致性哈希来计算对应关系，架构上显得简单不少。

大家应该知道，使用算法来计算对应规则，会有一个比较麻烦的问题，就是扩容，一旦扩容，一些本来打到老机器的数据就会打到新机器上，虽然一致性哈希的算法会使迁移的量比较少，但是仍然不可避免会有数据迁移。这个问题怎么办呢？

分享一个观点：遇到一个比较棘手的技术问题，如果能在业务层面做规避，将是最省事的。于是我们就想业务上能否接受数据迁移时候的丢失呢？说实话，不太容易接受，能不丢，尽量还是不要丢。

于是我们退一步，扩容的时候，我们不是立马迁移到新机器列表，而是先做一段时间的migrating，比如做一个月。这样一来，扩容之后，对于每个监控项还是会有至少一个月的数据，这样用户是可以接受的。

说完了存储，我们再来说说报警。报警数据要求的历史点数比较少，比如我们通常会配置说某个机器的cpu.idle连续三次达到阈值就报警，这里的“连续三次”就意味着，只要有最近三个点就可以搞定用户这个策略。数据量少，但是访问频繁，因为每个数据上来，都有可能找到其关联的策略，都需要拉取其历史点做判断，这样的场景，没啥说的，用内存吧。

我们把用来处理报警的这个模块称为judge，judge会有很多实例，当时想，某个实例挂了或者重启都不应该影响正常的报警，为了能够水平扩展，尽量做得无状态，那历史数据就不能存在judge的内存里了。redis吧，有名的内存nosql。于是我们就一次性上线了一堆judge实例，前面架设lvs做负载均衡，数据使用redis存储，走起。

稍总结一下，对于server的数据处理，这个属于产品实现细节了，我们因为团队人少，刚开始只有俩人，后来三个，所以力求简单，粗暴的解决问题。数据通过一致性哈希对rrd做一个分片，确实可以大幅缩减开发时间，不过也存在一些问题，大家发现问题在哪里了么？

下面我们先说试用之后立马发现的问题。

当时第一版上线，用了7台机器56个redis实例，给judge存放少量历史数据，发现每个redis的qps都是4K、5K的样子，第一版只是部署了少量agent（具体的量记不清了），结果qps就这么大，当时agent比较少，redis的压力让我们不能接受。

怎么办？大家有思路么？

外部内存的速度如果跟不上，那最直观的想法就是使用进程自身的内存，这里也就是judge组件的内存。那如果用了judge的内存，judge就有状态了，就无法水平扩展了呢。

仔细想想其实问题不大，judge使用内存的话，仍然需要使用算法做分片，既然绘图组件用了一致性哈希，那可以复用一下。同一个监控指标的数据，当前这分钟上来的数据打到某个judge，下一分钟上来的数据，同样应该打到这个judge，一致性哈希可以做到，没问题。

每个judge实例只处理一部分数据，比如50w，放到内存里量也不大，只是Go的GC不知道好不好用，如果Go的GC不好用，我们就在单机多搞一些judge实例，做进程级隔离，先想好退路，呵呵。

judge是数据驱动型的。就是说，数据来到我这个实例，我就去处理，数据没来，我就不处理，judge前面有个组件叫transfer来做转发，做一致性哈希计算，可以保证数据分发规则，看起来都没啥问题。最惹人烦的是什么呢？是judge重启，大家可以想想judge重启会带来哪些问题

judge一旦重启，就丢掉了历史数据，比如用户配置说某个机器的cpu.idle连续3次达到阈值就报警。前两次都达到阈值了，然后，然后judge重启，这俩点就丢了。judge重启完成，又上来一个达到阈值的新点，已经满足连续3次达到阈值了，但是，却无法报警。如果后面继续有两个点达到阈值那还好，还是会报警，如果后面的点都正常了，那这次本该有的报警就永远都不会报了。

这个问题业务上可以接受么？好吧，我们又来问业务了。首先，judge重启的几率比较小，只有升级的时候才会重启，代码写得注意一些，理论上是不会挂的；再一点，机器一旦出了问题，如果sre不介入处理，通常采集到的点都是持续性达到阈值，所以总是会报出来的，否则基本就可以不用关心。

这么看来，这个改造是可以进行的。于是，代码开始重构，上线之后效果大大好转，处理能力是原来的8倍左右。

本来想单开一节来说说一些折中考虑，发现折中都穿插在前面讲完了，那我们就不再单独讲系统的折中处理了。

时间已经比较久了，其他一些小的点我们就先不讲了，最后我们说说系统的问题以及未来的改造。

最大的问题是什么？相信大家已经想到了，就是每个graph（绘图组件）实例都是单点。如果某一台绘图机器磁盘坏，那这部分数据就丢了。怎么解决呢？

最简单的，硬件解决，我们每个机器都是ssd做raid10，稳定性比较高。然后，我们做了一个双写机制，但是现在的绘图机器IO已经比较高了，再搞个双写，势必要加机器，老板最近又在搞什么控制成本云云，就算了吧。监控数据虽然不是丢了会死人那么严重，但是如果不解决仍然让我们如鲠在喉。

于是，我们开始求助于一些分布式存储，我们做数据分片，做双写备份，很大程度上其实就是在解决分布式存储领域的问题，那我们为什么不直接使用分布式存储呢？

opentsdb，嗯，看着还不错，就是用来存放时间序列的数据。我们组一个同事聂安对opentsdb做了一些调研，觉得不能满足需求，两点：1. opentsdb不提供归档，要查看一年的历史数据，opentsdb就真的去把一年的所有点load出来，速度慢不说，浏览器非卡死不可； 2. opentsdb有tag数目限制，tag数不能多于8个，但我们在用的过程发现，好多业务数据的tag都是9个10个，没法减少。

大家看rrd的时候顺便看一下它提供的归档机制，很强大

通过改代码，或许可以调整tag数不能多于8个的限制，但性能可能会急剧下降。另外在很早的时候，同事来炜曾简单尝试过opentsdb，当时只用了10来台opentsdb前端，一打就挂一打就挂，可能是我们机器用的太少，也可能是某些参数没有做调优，时间关系，当时没有深入去研究，但是先入为主的觉得opentsdb不够强。再加上聂安这次调研，我们最终就放弃了opentsdb。

小米有好几个hbase的committer，团队强大，我们现在准备直接使用hbase做存储，云存储团队给我们技术支持，自己做归档，用hbase做后端，实现一个分布式rrd。目前基本开发完成，测试中……如果OK的话，以后就不用担心graph单点数据丢失的问题了。

第二个问题：agent挂了无从知晓

这个问题的本质其实是一个nodata的需求，也就是数据多久没上来应该有个报警机制。开源版本的open-falcon目前没有提供解决方案。应该如何设计这个nodata组件呢？

nodata组件的开源工作应该正在进行中

我们还是不希望改变用户的使用习惯，比如现在每个agent每个周期都会push一个agent.alive=1的数据，用户只要在portal上配置说all(#3) agent.alive != 1就报警。如果没有nodata组件，这个策略是没法工作的，因为agent挂了，不会有agent.alive=0的数据上来。那我们只要写一个组件，在发现没有agent.alive数据的时候就自动push一个agent.alive=0的数据不就可以了么。

O了，这就是nodat组件的工作逻辑，我们可以配置一些关键的指标的default值（当server端发现没有对应数据上来的时候，就自动push一个default值），注意，这里只是配置了一些关键指标，不是所有指标都配置，如果所有指标都配置，工作量不可接受，而且也没必要，比如agent会采集cpu、内存、磁盘、io等很多数据，我们没必要为这些数据都配置default值，只要配置agent.alive的default值是0，就可以了，就可以监测到agent是否挂掉。

最后一个问题：如何处理集群监控

比如我们有一个集群，挂一两台机器没啥问题，但是挂的机器超过10%就有问题了。另一个例子：我们有个集群，失败请求率小于5%没有问题，大于5%了就要尽快接入处理了。

这种从集群视角去判断阈值的情形，应该如何支持呢？

这个问题如果从简单着手，可以直接做成：对一批机器的某个指标做一个相加。比如某集群有30台机器，要对集群的整体错误率做一个统计，只要每台机器都计算一个本机的错误率：query\_fail\_rate=query\_fail/query\_total，然后把所有机器的这个query\_fail\_rate相加求平均即可。

但是，这样做显然是不太准确，更准确的做法应该是把所有的query\_fail相加，把所有的query\_total相加，然后做除法。

所以最后可以抽象为一批机器+一个除法表达式。一批机器就是指某个集群的所有机器，一个除法表达式，在上例中，其分子是query\_fail，分母是query\_total。集群指标聚合的这个组件姑且称之为aggregator，这个aggregator组件得到机器列表和分子、分母之后，就可以去计算了。把每个机器的query\_fail查询到并全部相加，把每个机器的所有query\_total查询到并全部相加，最后做个除法即可

这种做法是否可以解决集群机器挂的数量达到某个值就报警的需求？没问题的，分子是agent.alive，分母是机器总量，比如30（这个可以动态计算），agent.alive正常来说都是1，如果挂了就是0（我们的nodata组件的成果），比如有3台挂了，分子之和计算出来应该是27，分母是30，27/30<90%? 符合阈值就报警。

这种抽象看起来问题不大，最后一点要说明的是，aggregator模块去计算的集群聚合结果应该是每个周期都要计算，计算结果要重新push给falcon，这样这个集群的聚合结果就可以像一个普通的监控项一样，可以用来绘图，可以用来报警。

OK，已经挺长时间了，整个open-falcon的脑洞历程就讲解这么多，希望对大家有帮助，谢谢大家。

Q1 和elk或者stasd加graphite比是否使用方便

A1. 这个方便程度很难讲了，falcon侧重的是做一个监控框架，纳入各种数据，然后做好报警，还有历史数据归档，和elk之类的可能目标不是完全一致，不是一类产品

Q2. 最后说的 求 集群监控 query\_fail\_rate 的部分应该也会有 单点问题的吧、怎么规避的呢

A2. aggregator模块的确是会有单点问题的，目前还没有去解决，可以做个选主之类的逻辑，多个实例一起上，但是每次只有一个主在工作

Q3.我想请教一下，open-falcon将大多数监控处理放到了agent，部署升级得关注版本一致性，如果使用snmp协议监控资源，服务端每次采集资源时通过go的goroutine并发采集入库，资源报警和其他程序通过trap反馈给服务端，感觉是否会更方便一些，问的比较弱，见笑了。

A3. snmp没有agent强大，agent作为一个常驻进程运行在机器上，可以干的事情更多一些，而且snmp去pull，server端的压力会大一些，处理起来略麻烦

Q4.监控在告警时存在误报么、怎么保证每条发送出去的告警都是有用的？

A4. hbs只是用来分发agent要执行的插件、要监控的进程、端口，不做报警相关的处理，这样逻辑结构上更清晰一些:)

Q5. 目前业务层也接入了监控系统，会出现监控系统单点故障导致全局的应用全部不可用吗？

A5. 业务系统中嵌入监控的逻辑，就要很小心了，这个监控逻辑不能阻塞主要业务处理，只是一个旁路，一个单独的线程，监控系统整个挂了，不会影响到各个业务：）

Q6.监控在告警时存在误报么、怎么保证每条发送出去的告警都是有用的？

A6. 误报这个是不存在的，都是按照用户的策略去报警，当然，有的时候用户不知道自己的阈值设置多少合适，随便设置了一个，这种情况报的警可能确实没啥用处

Q7、agent端数据采集点有多少，性能压力是怎样的？

A7. agent在所有目标机器上部署，比如有1w台机器，就要部署1w个agent，每个agent大约采集200各监控项，现在小米的falcon应用情况是每个周期5000w左右的数据上来，有几十台监控server端协同工作

Q8、jduge中的transfer做转发，这个是单点处理么？如果挂了怎么办？

A8. transfer只是一个转发组件，是无状态的，transfer挂个一两台没有问题

Q9、看到agent运行脚本的时候，采集的是stdout的数据，stderr的数据采集么？如果不小心采集了二进制数据，是怎么处理？不小心调试的时候采集数据量比较大，阻塞了网络的情况有么？

A9. 只采集stdout是一种约定，需要汇报的数据写入stdout，脚本运行出错了写入stderr，如果采集的脚本不能decode成要求的数据格式，直接扔掉。目前没有出现调试的时候数据量大阻塞网络的情况：）

Q10、agent是打包到你们的镜像文件中，系统安装完后自动启动，然后上报么？如果是这样，那么什么时候填写的服务器地址？

A10. 我们的系统由系统组统一装机，然后统一初始化，初始化的时候安装一些基础组件，agent就是在这个时机安装，之后各个业务线再做针对自己业务的初始化工作：）

Q11. 有没有相应的监控报表之类的？是否可以对接kibana

A11. 现在基本没有报表性质的东西，只有趋势图，针对各个监控项的历史趋势图。哦，还有一个未恢复的报警列表，各个产品线工程师可以在下班的时候看一眼这个未恢复的报警列表，看是否有忘记处理的。其他就没了，不懂kibana：）

Q12 如此多的监控项和图表如何展示，如何快速找到需要的东西

A12. 一般我们要看什么数据是有很强目的性的，比如某个服务挂了，你可能要看这个服务所在机器的情况，于是就可以拿着机器名之类的去搜索，也可以根据metric（监控项）去搜索，等等。也可以提前做screen，把经常要看的图表放在一个页面

Q13. 是否有监控到异常，然后自动对服务器进行后续操作的？

A13 有的，异常了之后通常的处理是报警，也可以回调一个业务的接口，把相关的参数都传递个这个接口，用户就可以在这个接口中写自己的业务逻辑，比如去重启某个机器

# 使用 Java 平台管理 bean

https://www.ibm.com/developerworks/cn/java/j-mxbeans/

使用 Java 平台管理 bean

**监视 Java SE 5.0 应用程序**

Java™ 平台的最新版本包含许多新的系统监视和管理特性。在本文中，三位来自 IBM Java 技术中心团队的开发人员一起带您开始使用这个 API。在快速概述了 java.lang.management 包之后，他们将指导您经历大量短小的实践场景，探测运行的 JVM 性能。

1https://dw1.s81c.com/developerworks/i/v17/dw-cmts-arrow.png [评论](https://www.ibm.com/developerworks/cn/java/j-mxbeans/#icomments)

[May Glover Gunn](https://www.ibm.com/developerworks/cn/java/j-mxbeans/#authorN10019) ([mglovergunn@uk.ibm.com](mailto:mglovergunn@uk.ibm.com?subject=%E4%BD%BF%E7%94%A8%20Java%20%E5%B9%B3%E5%8F%B0%E7%AE%A1%E7%90%86%20bean)), 软件工程师, IBM

[George Harley](https://www.ibm.com/developerworks/cn/java/j-mxbeans/#authorN1002B) ([gharley@uk.ibm.com](mailto:gharley@uk.ibm.com?subject=%E4%BD%BF%E7%94%A8%20Java%20%E5%B9%B3%E5%8F%B0%E7%AE%A1%E7%90%86%20bean)), 软件开发人员, IBM

[Caroline Gough](https://www.ibm.com/developerworks/cn/java/j-mxbeans/#authorN1003B)([goughc@uk.ibm.com](mailto:goughc@uk.ibm.com?subject=%E4%BD%BF%E7%94%A8%20Java%20%E5%B9%B3%E5%8F%B0%E7%AE%A1%E7%90%86%20bean)), 软件工程师, IBM

2006 年 4 月 28 日

* [expand](https://www.ibm.com/developerworks/cn/java/j-mxbeans/#toggle)**内容**



在 IBM Bluemix 云平台上开发并部署您的下一个应用。

[**开始您的试用**](https://developer.ibm.com/sso/bmregistration?lang=zh_CN&ca=dwchina-_-bluemix-_-j-mxbeans-_-sidebar)

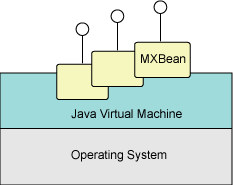
在 Java 2 平台 5.0 版引入的众多新特性中，有一个 API 可以让 Java 应用程序和允许的工具监视和管理 Java 虚拟机（JVM）和虚拟机所在的本机操作系统。在本文中，将学习这个新的平台管理 API 的功能，这个 API 包含在 java.lang.management 包中。本文将让您迅速掌握在未来的 Java 平台版本中将变得更重要的一套新的强大特性。

## 监视和管理 5.0 虚拟机

Java 5.0 提供了监视和管理正在运行的虚拟机的新功能。开发人员和系统管理员能够监视 5.0 虚拟机的性能，而且对于某些属性还可以进行调优。以前有过使用 Java Management Extensions（JMX）经验的人都会熟悉执行这些活动的机制。通过 JMX 技术，一套需要控制的平台资源可以被当作简单的、定义良好的对象，对象的属性映射到给定资源的更低级特征上。

在平台管理 API 中，这些定义良好的对象叫做 MXBean。如果觉得 MXBean 听起来更像某种可能更熟悉的 MBean，那么就对了。这些 MXBean（或 平台 MBean）在效果上，就是封装了 5.0 平台内部特定部分的管理 bean。图 1 展示了 MXBean 在更大系统中的位置：

##### 图 1. MXBean 提供了 Java 平台的管理接口



在运行着的 5.0 兼容的虚拟机中可以发现和定制许多不同种类的功能；例如，可以得到正在使用的即时编译系统的行为细节或者获得垃圾收集服务的进展情况。

任何 Java 应用程序只要获得需要的 bean 引用（使用我们本文中介绍的技术），就能利用平台 bean，然后调用适当的方法调用。在最简单的场景中，bean 客户机可以发现客户机所在的平台的信息。但是客户机还能监视完全独立的 JVM 的行为。这是有可能的，因为 MXBean 是一种 MBean，可以用 Java 5.0 可用的标准 JMX 服务远程地进行管理。

[**回页首**](https://www.ibm.com/developerworks/cn/java/j-mxbeans/#ibm-pcon)

## JConsole

bean 客户机的一个示例就是与 Java SE 5.0 SDK 一起提供的 JConsole 工具。这是一个图形界面，连接到 JVM 并显示 JVM 的信息。GUI 中的选项卡与 JVM 的特定方面相关；有 Memory、Threads 和 Classes 等选项卡。JConsole 工具还提供了一个整体的 Summary 选项卡，一个VM 选项卡（提供虚拟机启动时的环境信息）和一个 MBean 选项卡（用户可以用它更详细地监视平台 MBean 的状态）。

### 运行 JConsole

在命令提示符下输入 jconsole 就可以启动 JConsole（假设 SDK 的 bin 目录在当前路径中）。请输入运行要监视的 JVM 的主机名，以及侦听管理请求的端口号 —— 以及其他需要的认证细节 —— 然后点击 **Connect**。用默认值 localhost 和端口 0 点击 **Connect**，监视的是用于运行 JConsole 自己的 JVM（因为 JConsole 是一个 Java 进程）。这称作自监视模式。图 2 展示了 JConsole 的启动情况：

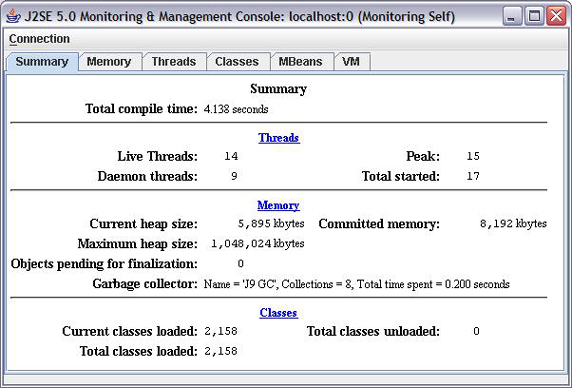
##### 图 2. JConsole 启动



### JConsole 在工作

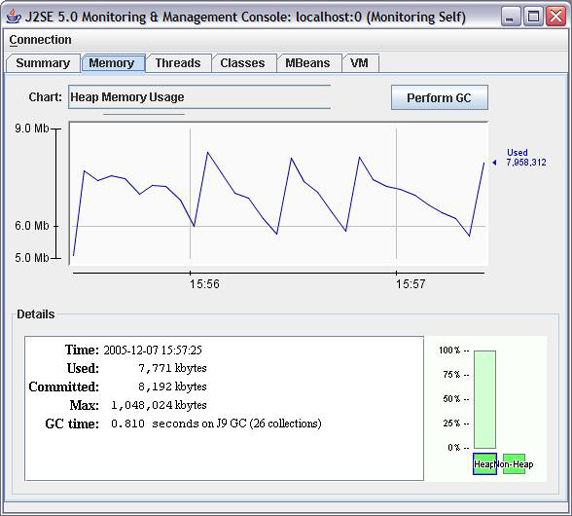
到 JVM 的连接建立之后，JConsole 先显示 Summary 选项卡，如图 3 所示：

##### 图 3. JConsole 的 Summary 选项卡



从这开始，可以选择其他选项卡。例如 Memory 选项卡，如图 4 所示，展示了 JVM 中每个内存池的使用历史：

##### 图 4. JConsole 的 Memory 选项卡



请注意面板右上角的 **Perform GC** 按钮。这是使用平台 MBean 可以在 JVM 上执行的众多操作中的一个示例。

[**回页首**](https://www.ibm.com/developerworks/cn/java/j-mxbeans/#ibm-pcon)

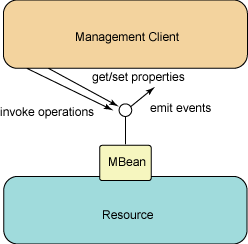
## 工作方式

到此为止所阅读的基础内容是托管 bean 或 MBean 的概念。可以把 MBean 当成资源的管理接口 的可编程表示。用更简单的术语来说，可以把它们当成围绕在受控实体周围的 Java 包装器。而用更实际的术语来说，MBean 就是 Java 类，这些类的 public 方法是根据定义良好的规则集编写的；这些规则要求把受管理的应用程序或资源的那些特征进行完整的封装。最终，资源（不论是什么以及在网络的什么位置上）的管理者为了控制的目的定位并使用对应的 MBean。

通过 API，MBean 提供以下信息，如图 5 所示：

* 资源的当前状态，通过资源的属性 提供
* 管理代理能够在资源上执行的操作
* 能够发送到有兴趣的合作方的可能的事件通知

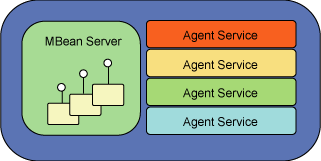
##### 图 5. MBean 客户机利用属性、操作和事件



MBean 创建好之后，需要注册到 MBean 服务器上。除了充当 MBean 的注册表，MBean 服务器还提供了让管理系统发现和利用已注册 MBean 的方式。管理已注册 MBean 的附加功能，由 JMX 代理服务 执行。这类服务包括：监视 MBean 的属性值，把 MBean 的改变通知给有兴趣的合作方，周期性地把 MBean 的特定信息通知给侦听器，维持 MBean 之间的关系。JMX 的代理服务通常是 MBean 本身。

MBean 服务器与必需的 JMX 代理服务的结合，被称作 JMX 代理，如图 6 所示：

##### 图 6. JMX 代理



JMX 代理可以让它的托管资源 —— 也就是说，目前注册到它的 MBean 服务器上的 MBean 集 —— 对其他远程代理可用。

在 Java 5.0 发行之前，javax.management API 是 Java 平台的可选扩展，用户可以通过独立的下载获得，并通过 Java 代码把它用作管理和监视资源的手段。在这个上下文中，资源 可以是应用程序、运行业务关键型应用程序的 J2EE 服务器、普通的旧式 Java 对象（POJO）、甚至于硬件实体（例如网络设备、机顶盒、电信设备，或者类似的东西）。资源如果可以从 Java 代码中引用，那么它就可以潜在地成为托管资源。

虽然在这里我们实际上只是涉及了 JMX 的表面，但对于认识 MXBean 来说，介绍的已经足够多了。关于 JMX 的设计和功能的全面讨论超出了本文的范围。要了解 JMX 在网络管理应用程序中负责的那部分功能的精彩概述，可以阅读 Sing Li 关于这一主题的系列（请参阅 [参考资料](https://www.ibm.com/developerworks/cn/java/j-mxbeans/#resources)）。

[**回页首**](https://www.ibm.com/developerworks/cn/java/j-mxbeans/#ibm-pcon)

## 什么是 MXBean？如何使用它们？

既然知道了什么是 MBean，现在可以看看在 java.lang.management 包中定义的与它们名称类似的 MXBean。好消息是：MXBean 并没有偏离我们在讨论 MBean 时介绍的概念。这个包中的大多数类型都是符合命名规范的接口，命名规范与标准 MBean 使用的规范类似：平台资源的名称加上后缀 MXBean。（对于标准的 MBean，当然使用后缀 MBean。）

表 1 描述了通过 java.lang.management 包中提供的 MXBean 接口可以使用的平台资源：

##### 表 1. 可以通过 MBean 管理的平台资源

| **平台资源** | **对应的 MXBean** | **可使用的数量** |
| --- | --- | --- |
| 编译 | CompilationMXBean | 0 或 1 |
| 垃圾收集系统 | GarbageCollectorMXBean | 至少 1 |
| 内存 | MemoryMXBean | 恰好是 1 |
| 内存管理器 | MemoryManagerMXBean | 至少 1 |
| 线程 | ThreadMXBean | 恰好是 1 |
| 操作系统 | OperatingSystemMXBean | 恰好是 1 |
| 运行时系统 | RuntimeMXBean | 恰好是 1 |
| 类装入系统 | ClassLoadingMXBean | 恰好是 1 |
| 内存资源 | MemoryPoolMXBean | 至少 1 |

对于每个 MXBean，客户必须编程的接口都在 Java 5.0 规范中做了严格的设置。目前客户还无法定制这样的接口，即它公开平台的任何更加可管理的属性。

在表 1 的第三列中指出的每个 MXBean 类型可能有的实例数量，严重依赖于被管理的具体的平台系统。例如，虽然 JVM 规范允许实现者选择所使用的垃圾收集算法，但是完全有理由使用任意数量的垃圾收集器，所以在任意时间内，就会有任意数量的 GarbageCollectionMXBean 实例在活动。请把这个与 OperatingSystemMXBean 对比，后者只有一个实例可用，因为管理的虚拟机显然在指定时间内只能运行在一个操作系统上。

客户机代码可以安全地把一次性的 MXBean 当成虚拟机中真正的单体。任何时间，只要引用请求的是这些一次性类型，得到的回答总是同一个实例，而不论引用请求从何而来或者在虚拟机生命周期中什么时候发生。即使多个客户机都在监视一个虚拟机的时候，也符合这种情况。

## MBeanServerConnection

接口 javax.management.MBeanServerConnection是 javax.management.MBeanServer 接口的超类型，如果 MBean 服务器与客户机代码运行在同一个 JVM 中（即管理客户机和 JMX 代理共同位于同一个虚拟机中），就可以用这个接口调用 MBean 服务器。因为在MBeanServerConnection 和 MBeanServer 之间有父子关系，所以客户可以用相同的方法调用与远程或本地 MBean 服务器交互。

对 Java 客户机代码来说，MXBean 实例的行为就像任何 POJO 一样。可以直接调用对象来取得信息，不需要其他参与者。当然，这种情况需要 Java 客户机已经直接得到了对本地 bean （与管理应用程序运行在同一个虚拟机上）的引用，或者对于封装远程虚拟机的 bean 已经请求了对它的代理。在两种情况下，引用都是从平台的单体 ManagementFactory 获得的。

也可以通过 javax.management.MBeanServerConnection 访问平台 bean，但是在这种情况下，在会话中多出了额外的一级间接。在 [通过平台服务器监视远程虚拟机](https://www.ibm.com/developerworks/cn/java/j-mxbeans/#8) 一节中将看到，在这种场景中，客户机总是请求 MBeanServerConnection 代表自己来定位指定的远程 bean，并进行调用。这实际还是让 JMX（前面提到过）发出对远程 MBean 的调用，而远程客户机必须与注册 MBean 的 MBean 服务器通信。

### MXBean 不是 JavaBean

为了避免混淆，应当记住：虽然把 MXBean 当成有助于监视和控制 JVM 的 MBean 完全没错，但如果把 MXBean 当成一种 JavaBean，就肯定不对 了。JavaBean 技术是 Java 平台的组件模型，它的设计目的是提供用图形化工具从可重用 Java 组件构造应用程序的能力。虽然 JavaBean 的一些特性（例如用有意义的命名规范帮助工具发现属性）在 MBean 和 MXBean 领域中都存在，但是两种技术是完全不同的，不要把它们混淆了。

### 额外的 MXBean

在本文开始时，我们提到过 java.lang.management 包容纳了平台管理 API。现在，我们对这句话稍做修正，因为不是所有的 MXBean 都包含在这个包中。因为 LoggingMXBean 与 Java 平台 的日志功能捆绑得如此紧密，所以把它放在 java.util.logging 包中更有意义。顾名思义，这类 MXBean 提供了运行虚拟机的日志功具的管理接口。使用对这个 bean 的引用，客户可以获得在平台上注册的所有日志程序的名称和它们彼此之间的关系。还可以获得和设置指定平台日志程序的级别。

就像 OperatingSystemMXBean 和 ThreadMXBean（另两个示例）一样，LoggingMXBean 在运行的虚拟机中也以单体方式存在。对它的已公开属性的任何 get 和 set，不论通过任何通信方式，最后都路由到同一个对象实例。

[**回页首**](https://www.ibm.com/developerworks/cn/java/j-mxbeans/#ibm-pcon)

## 获得 MXBean

客户机代码访问 MXBean 有三种方式：通过工厂方法、通过平台服务器 或作为代理。

### 工厂方法

检索 MXBean 最简单的方式就是使用 java.lang.management.ManagementFactory 类提供的静态方法。但是，用这种方式得到的 MXBean 只能用来监视本地虚拟机。ManagementFactory 类为每种 MXBean 都定义了一个检索方法。有些方法返回 MXBean 的单一实例，有些方法返回 MXBean 实例的强类型 List。

在指定类型只有一个 MXBean 时，检索它的代码很简单。清单 1 展示了检索 ThreadMXBean 的代码：

##### 清单 1. 检索平台惟一的 ThreadMXBean 的引用

ThreadMXBean threadBean = ManagementFactory.getThreadMXBean();

对于可能存在多个 MXBean 实例的那些 MXBean 类型来说，存在着工厂方法，可以在 List 中返回 MXBean，如清单 2 所示：

##### 清单 2. 检索平台上所有已知的 MemoryPoolMXBean 的强类型列表

List<MemoryPoolMXBean> memPoolBeans = ManagementFactory.getMemoryPoolMXBeans();

for (MemoryPoolMXBean mpb : memPoolBeans) {

System.out.println("Memory Pool: " + mpb.getName());

}

LoggingMXBean 是 java.util.logging 包的一部分，所以，要用 LogManager 类而不是 ManagementFactory 类来访问它，如清单 3 所示：

##### 清单 3. 从 LogManager 得到 LoggingMXBean 引用

LoggingMXBean logBean = LogManager.getLoggingMXBean();

记住，这些方法只允许访问属于本地 虚拟机的 MXBean。如果想把客户机代码扩展到能够一同处理位于同一台机器或不同结点上的远程 JVM，那么需要使用下面介绍的两种方法中的一种。

### 通过平台服务器

组织代码，以对远程虚拟机的 MBean 服务器的连接进行调用，是一种可行的选择。要让这个选择成功，首先需要用关键的命令行选项启动远程虚拟机。这些选项设置虚拟机的相关 JMX 代理侦听请求的端口，以及起作用的安全级别。例如，以下选项启动的虚拟机，其代理会在 1234 端口上侦听，且没有安全性：

-Dcom.sun.management.jmxremote.port=1234

-Dcom.sun.management.jmxremote.authenticate=false

-Dcom.sun.management.jmxremote.ssl=false

本文后面的 [安全性](https://www.ibm.com/developerworks/cn/java/j-mxbeans/#11) 一节将介绍对虚拟机的安全访问。

有了远程代理侦听，就可以使用清单 4 中的代码段获得相关 MBean 服务器连接的引用：

##### 清单 4. 用 JMXConnectorFactory 连接不同虚拟机的 MBean 服务器

try {

// connect to a separate VM's MBeanServer, using the JMX RMI functionality

JMXServiceURL address =

new JMXServiceURL( "service:jmx:rmi:///jndi/rmi://localhost:1234/jmxrmi");

JMXConnector connector = JMXConnectorFactory.connect(address);

MBeanServerConnection mbs = connector.getMBeanServerConnection();

} catch ...

一旦检索到 MBeanServerConnection，就可以使用 JMX 方法 getAttribute()、setAttribute() 和 invoke() 操作 MXBean。这将在 [通过平台服务器监视远程虚拟机](https://www.ibm.com/developerworks/cn/java/j-mxbeans/#8) 中介绍。

### 作为代理

访问平台 bean API 的第三种方法与前面介绍的两种方法有共同之处。像以前一样，需要检索到被监视虚拟机的 JMX 代理的MBeanServerConnection。然后，通过使用 ManagementFactory 类的静态助手方法，客户机代码可以请求注册到远程虚拟机的 MBean 服务器上的一个指定 MXBean 的代理实例。清单 5 展示了一个示例：

##### 清单 5. 到远程 MBean 服务器的引用能够获得远程 MXBean 的代理

try {

ThreadMXBean threadBean = ManagementFactory.newPlatformMXBeanProxy

(mbs, ManagementFactory.THREAD\_MXBEAN\_NAME, ThreadMXBean.class);

} catch ...

对于所有的单体 MXBean（除了 LoggingMXBean），在 ManagementFactory 类的公共静态字段中可以得到用来进行服务器注册的完整字符串名称。例如，ThreadMXBean 的 javax.management.ObjectName 的字符串表示就保存在 THREAD\_MXBEAN\_NAME 字段中，而 LoggingMXBean 的注册名称则保存在 java.util.logging.LogManager 类的静态字段中。清单 6 展示了对 LoggingMXBean 代理实例的请求：

##### 清单 6. LoggingMXBean 的字符串名称是 java.util.logging.LogManager 类的常量

try {

LoggingMXBean logBean = ManagementFactory.newPlatformMXBeanProxy

(mbs, LogManager.LOGGING\_MXBEAN\_NAME, LoggingMXBean.class);

} catch ...

对于可能在虚拟机中存在不止一个实例的 MXBean 类型，事情变得略微麻烦一些。在这种情况下，首先需要用 MBeanServerConnection 获得指定类型的全部已注册 MXBean 的名称。为了方便，每个单体 MXBean 的 ObjectName 的域部分保存在 ManagementFactory 中的公共静态字段中。一旦检索到这些名称，就能用每个名称构造独立的代理实例。清单 7 展示了一个示例：

##### 清单 7. 为属于远程虚拟机的每个 MemoryManagerMXBean 创建代理

try {

// Get the names of all the Memory Manager MXBeans in the server

Set srvMemMgrNames = mbs.queryNames(new ObjectName(

ManagementFactory.MEMORY\_MANAGER\_MXBEAN\_DOMAIN\_TYPE + ",\*"), null);

// Get a MXBean Proxy for each name returned

for (Object memMgrName : srvMemMgrNames){

// Cast Object to an ObjectName

ObjectName memMgr = (ObjectName) memMgrName;

// Call newPlatformMXBeanProxy with the complete object name

// for the specific MXBean

MemoryManagerMXBean memMgrBean =

ManagementFactory.newPlatformMXBeanProxy(

mbs, memMgr.toString(), MemoryManagerMXBean.class);

// memMgrBean is a proxy to the remote MXBean. We can use it

// just as if it was a reference to a local MXBean.

System.out.println("Memory Manager Name = " +

memMgrBean.getName());

}

} catch ...

[**回页首**](https://www.ibm.com/developerworks/cn/java/j-mxbeans/#ibm-pcon)

## 使用 MXBean

java.lang.management 文档中列出了每个 MXBean 接口定义的操作。通过这些操作，用户可以管理和监视虚拟机。例如，MemoryMXBean 上的操作允许打开内存系统的详细输出、请求垃圾收集以及检索当前堆和非堆内存池使用的内存的详细信息。所以，如果关心 Java 应用程序所使用的内存数量或者希望调整堆的尺寸，可以容易地用 java.lang.management API 编写管理客户机，连接到应用程序并监视内存的使用情况。

类似地，ThreadMXBean 也提供了 Java 应用程序挂起时会有用的功能。findMonitorDeadlockedThreads() 方法返回被它标识为死锁的线程的 ID。然后就可以用这些 ID 来检索线程的详细信息，包括它们的堆栈跟踪、它们的状态、它们是否在执行本机代码，等等。

这个线程信息是在 ThreadInfo 类的实例中提供的，这个类是 java.lang management 包中提供的由 MXBean 用来向用户返回数据快照的三个类的中一个 —— 其他两个是 MemoryUsage 和 MemoryNotificationInfo 类。这三个类中的每个类都是一个复杂的 数据类型，包含用于描述特定平台性质的结构化信息。

现在来看两个示例场景，演示一下上面讨论的概念如何转变成 Java 代码。

[**回页首**](https://www.ibm.com/developerworks/cn/java/j-mxbeans/#ibm-pcon)

## 示例 1：通过 MXBean 或代理监视虚拟机

正如前面讨论过的，MXBean 的方法既可以直接在本地 MXBean 上调用，也可以通过代理调用。清单 8 展示了如何使用 ThreadMXBean 的 getter 和 setter 操作。这个示例中的 threadBean 变量既可以是从本机虚拟机检索的 MXBean，也可以是从远程虚拟机检索的 MXBean 的代理。一旦得到了引用，那么对于调用者来说就是透明的。

##### 清单 8. 获取和设置 ThreadMXBean 的值

try {

// Get the current thread count for the JVM

int threadCount = threadBean.getThreadCount();

System.out.println(" Thread Count = " + threadCount);

// enable the thread CPU time

threadBean.setThreadCpuTimeEnabled(true);

} catch ...

清单 8 中使用的 setThreadCpuTimeEnabled() 方法在 5.0 兼容的虚拟机中是可选支持的。在使用清单 9 所示的可选功能时，需要进行检查：

##### 清单 9. 在尝试使用可选属性之前，检查是否支持可选属性

if (threadBean.isThreadCpuTimeSupported()) {

threadBean.setThreadCpuTimeEnabled(true);

}

CompilationMXBean 类型的 getTotalCompilationTime() 方法也包含不必在每个 5.0 兼容虚拟机实现中都必须有的功能。就像清单 9 中的setThreadCpuTimeEnabled() 一样，也有相关的方法用来检查支持是否存在。不利用这些检测方法的代码需要处理可选方法可能抛出的任何java.lang.UnsupportedOperationException。

清单 10 展示了如何访问虚拟机中运行的所有线程的信息。每个线程的信息都保存在独立的专用 ThreadInfo 对象中，随后可以查询这个对象。

##### 清单 10. 获得虚拟机中运行的所有线程的名称

try {

// Get the ids of all the existing threads

long[] threadIDs = threadBean.getAllThreadIds();

// Get the ThreadInfo object for each threadID

ThreadInfo[] threadDataset = threadBean.getThreadInfo(threadIDs);

for (ThreadInfo threadData : threadDataset) {

if (threadData != null) {

System.out.println(threadData.getThreadName());

}

}

} catch ...

记住，像 ThreadInfo、MemoryUsage 和 MemoryNotificationInfo 这样的复杂类型中包含的信息，仅仅是请求调用执行的时刻的系统快照。这些对象在您得到对它们的引用之后，不会动态更新。所以，如果应用程序需要刷新 被管理的虚拟机上这些方面的数据，需要再做另一个调用，得到更新的 ThreadInfo 或 MemoryUsage 对象。MemoryNotificationInfo 对象在这方面略有不同，因为它们不是由管理应用程序拉动的，而是在事件通知中推动的（这点 [将很快详细进行讨论](https://www.ibm.com/developerworks/cn/java/j-mxbeans/#10)）。

[**回页首**](https://www.ibm.com/developerworks/cn/java/j-mxbeans/#ibm-pcon)

## 示例 2：通过平台服务器监视远程虚拟机

用 MBeanServerConnection 访问远程 JVM 的 ThreadMXBean 不像清单 1 中的示例那样直接。首先，需要 ThreadMXBean 的一个javax.management.ObjectName 实例。可以用与 MXBean 代理对象相同的名称创建这个实例，如清单 11 所示：

##### 清单 11. 构建 ThreadMXBean 的 ObjectName

try {

ObjectName srvThrdName = new ObjectName(ManagementFactory.THREAD\_MXBEAN\_NAME);

...

} catch ...

可以用 ObjectName 实例来标识特定的远程 ThreadMXBean，以调用 MBeanServerConnection 的 getAttribute()、setAttribute() 和invoke()，如清单 12 所示：

##### 清单 12. 将 ObjectName 用于对远程 MBean 服务器的调用

try {

// Get the current thread count for the JVM

int threadCount =

((Integer)mbs.getAttribute( srvThrdName, "ThreadCount")）。intValue();

System.out.println(" Thread Count = " + threadCount);

boolean supported =

((Boolean)mbs.getAttribute(srvThrdName, "ThreadCpuTimeSupported")）。booleanValue();

if (supported) {

mbs.setAttribute(srvThrdName,

new Attribute("ThreadCpuTimeEnabled", Boolean.TRUE));

...

}

} catch ...

清单 13 展示了通过 MBean 服务器连接来访问虚拟机中所有当前线程的信息。使用这种方法访问的 MXBean 返回复杂的数据类型，这些复杂数据类型包装在 JMX 开放类型 —— 例如 javax.management.openmbean.CompositeData 对象中。

为什么要把复杂数据包装在中间类型中呢？记住，MXBean 可能潜在地由实际上不是用 Java 语言编写的远程应用程序来管理，也有可能由这样的 Java 应用程序来管理，它们不能访问所有用于描述托管资源的不同性质的复杂类型。虽然可以安全地假定到平台 JMX 代理的连接的两端都能理解简单类型（例如 boolean、long 和 string），还可以把它们映射到各自的实现语言中的对应类型，但是要假定每个可能的管理应用程序都能正确地解释 ThreadInfo 或 MemoryUsage 这样的复杂类型，那是不现实的。像 CompositeData 这样的开放类型可以用更基本的类型来代表复杂的（即非基本的或结构化的）数据。

如果 5.0 MXBean 的远程调用要求传递复杂类型的实例，那么对象就被转换成等价的 CompositeData。虽然这可以让信息发送到尽可能广泛的客户，却也有不足之处：实际上能够解析 ThreadInfo 和 MemoryUsage 类型的接收方 Java 应用程序仍然需要从开放类型转换到复杂类型。但即便这样，也不算是太麻烦的步骤，因为 java.lang.management 中定义的所有支持的复杂数据类型，都有静态的方便方法做这件事。

在清单 13 中，threadDataset 属性包含一组 CompositeData 对象，这些对象直接映射到 ThreadInfo 对象。对于每个线程，ThreadInfo 的静态方法 from() 被用来从 CompositeData 构建等价的 ThreadInfo 对象。可以用这个对象访问每个线程的信息。

##### 清单 13. CompositeData 类型在网络上传输复杂数据结构

try {

// Get the ids of all the existing threads

long[] threadIDs = (long[])mbs.getAttribute(srvThrdName, "AllThreadIds");

// Get the ThreadInfo object for each threadID. To do this we need to

// invoke the getThreadInfo method on the remote thread bean. To do

// that we need to pass the name of the method to run together with the

// argument and the argument type. It's pretty ugly we know.

CompositeData[] threadDataset =

(CompositeData[]) (mbs.invoke(srvThrdName, "getThreadInfo",

new Object[]{threadIDs}, new String[] {"[J"}));

// Recover the ThreadInfo object from each received CompositeData using

// the static helper from() method and then use it to print out the

// thread name.

for (CompositeData threadCD : threadDataset) {

ThreadInfo threadData = ThreadInfo.from(threadCD);

if (threadData != null) {

System.out.println(threadData.getThreadName());

}

}

} catch ...

[**回页首**](https://www.ibm.com/developerworks/cn/java/j-mxbeans/#ibm-pcon)

## API 支持

如果要开发检查虚拟机线程状态的平台管理代码，那么可能会遇到一些有趣的行为，与我们在撰写本文时遇到的一样。这会对代码有影响么？这取决于应用程序使用 5.0 版 Java 平台新机制保护代码块不受并发访问影响的程度。

5.0 中的新包 java.util.concurrent.locks 引入了 ReentrantLock 类，顾名思义，可以用它构建一个可重入锁，以保护代码的关键部分。它与现有的 synchronized 关键字的隐式锁定机制非常类似，但是有一些额外的功能，这些功能对于微调控制显式锁会非常有用。清单 14 展示了它的使用示例：

##### 清单 14. ReentrantLock 非常简单的使用

private Lock myLock = new ReentrantLock();

...

void myMethod() {

// Acquire the lock

myLock.lock();

try {

... do work in critical section ...

} finally {

// Relinquish the lock

myLock.unlock();

}// end finally

...

}

在进入关键部分之前，调用 ReentrantLock 对象的 lock() 方法，尝试并获得锁。只有在其他线程不拥有锁的情况下才会成功，如果其他线程拥有锁，当前线程就被阻塞。在 5.0 版 Java 平台之前，可能要用清单 15 那样的代码才能编写清单 14 的功能。（当然，现在还可以这样写，因为 synchronized 还没过时。）

##### 清单 15. 同步的方法

synchronized void myMethod() {

... do work in critical section ...

}

在这些简单的使用中，不会看到代码行为上的差异。但是，如果利用 ThreadMXBean 和 ThreadInfo 类型来检查您知道在运行的程序中会因为进入关键部分而阻塞的线程的阻塞计数，那么结果会根据使用的阻塞方法而不同。通过编写一些简单的代码，其中有两个不同的线程，试图调用同一个 myMethod()，并强迫一个线程总在另一个线程之后到达，您可以自己对这个问题进行演示。这个线程显然会被阻塞，而且应当有一个恰好为 1 的阻塞计数。在 myMethod() 上使用 synchronized 关键字时，会看到与线程关联的 ThreadInfo 对象有一个大于 0 的阻塞计数。但是，使用新的 ReentrantLock 方式，会看到一个为 0 的阻塞计数。可以肯定地说，随着 ThreadMXBean 的虚拟机监视功能采用新的并发包，我们观察到的这种行为上的差异在未来的 Java 平台版本中会被清除。

[**回页首**](https://www.ibm.com/developerworks/cn/java/j-mxbeans/#ibm-pcon)

## 通知

MemoryMXBean 在 MXBean 之间是惟一的，因为它能够把内存使用情况以事件的方式向客户机对象动态地发送通知。对于内存使用超过一些预设阈值的问题，即时通信的好处显而易见，因为可能是应用程序级上的问题征兆，或者表明需要对虚拟机进行进一步的调整。

MemoryMXBean 使用的通知模型来自 JMX MBean 规范，该规范与 Java 编程中使用的事件通知模型很相似。作为通知的广播者，MemoryMXBean实现了 JMX 接口 javax.management.NotificationBroadcaster，这个相对小的接口允许 bean 注册有兴趣的合作方或取消注册。然后，每个有兴趣的合作方（对象）都必须实现 javax.management.NotificationListener 接口。这个接口只包含一个操作，在事件发生的时候，由发出 MXBean 的事件调用。

侦听器可以在虚拟机生命周期的任何时候注册（或取消注册）到 MemoryMXBean 上。通知只广播到当前 已注册的合作方。

侦听器的处理器方法在被调用时，会接收到 javax.management.Notification 类的实例。这是 JMX 事件通知模型中的通用事件信号类型。它被设置成容纳造成它生成的事件的相当数量的信息。对于 MemoryMXBean，目前有两类通知：

* 虚拟机中的内存资源（有时叫做内存池）增长超过了预先设置的阈值。这种事件由 MemoryNotificationInfo 常量MEMORY\_THRESHOLD\_EXCEEDED 表示。
* 垃圾收集之后 内存资源的大小超过了预先设置的阈值。这由 MemoryNotificationInfo 常量 MEMORY\_COLLECTION\_THRESHOLD\_EXCEEDED 表示。

在处理器方法中接收到 Notification 对象后，注册的侦听器可以查询 Notification 的类型，根据 MemoryNotificationInfo 的两个值检查生成的字符串，从而判断出发生的事件类型。

要向侦听器传递事件的详细信息，MemoryMXBean 会用代表 MemoryNotificationInfo 对象 javax.management.openmbean.CompositeData的具体实例正确地设置发出的 Notification 对象的用户数据（实际上，就是广播者包含任何想要的信息的方法）。就像在 [通过平台服务器监视远程虚拟机](https://www.ibm.com/developerworks/cn/java/j-mxbeans/#8) 中解释的一样，在 JMX 开放数据类型中封装事件数据，可以让最广泛的侦听器都能理解数据。

[**回页首**](https://www.ibm.com/developerworks/cn/java/j-mxbeans/#ibm-pcon)

## 安全性

到目前为止都还不错。现在是面对被甩在一边的重要问题 —— 安全性 —— 的时候了。不如果不想让谁的应用程序代码访问和修改虚拟机，该怎么办？有什么选项可用么？ 可以设置一些系统属性，来控制访问级别和虚拟机数据从 JMX 代理向管理客户机传递虚拟机数据的方式。这些属性分成两类：口令认证 和安全套接字层（SSL）。

### 使用命令行选项

为了让 5.0 兼容的虚拟机可以被监视和管理，需要用以下命令行选项设置平台 JMX 代理的端口号：

-Dcom.sun.management.jmxremote.port=<number>

如果不介意谁通过这个端口访问虚拟机，也可以添加以下两个选项，关闭口令认证和 SSL 加密（这两项默认都是开启的）：

-Dcom.sun.management.jmxremote.authenticate=false

-Dcom.sun.management.jmxremote.ssl=false

在开发 java.lang.management 客户机代码，而且想方便地监视另一台虚拟机时，一起使用这三个选项会很方便。在生产环境中，则需要设置口令控制或 SSL（或者两者都要设置）。

### 口令认证

在 5.0 SDK 的 jre/lib/management 目录中，可以找到一个叫做 jmxremote.password.template 的文件。这个文件定义了两个角色的用户名和口令。第一个是监视 角色，允许访问只读的管理函数；第二个是控制 角色，允许访问读写函数。取决于需要的访问级别，客户可以用monitorRole 或 controlRole 用户名进行认证。为了确保只有认证的用户才能访问，需要做以下工作：

1. 把 jmxremote.password.template 的内容拷贝到叫做 jmxremote.password 的文件中，并取消掉文件末尾定义用户名和口令部分的注释，根据需要修改口令。
2. 修改 jmxremote.password 的许可，只让所有者能够读取和修改它。（在 UNIX 和 UNIX 类的系统上，把许可设置成 600。在 Microsoft Windows 上，请按照 “How to secure a password file on Microsoft Windows systems” 一文中的说明操作，可以在 [参考资料](https://www.ibm.com/developerworks/cn/java/j-mxbeans/#resources) 中找到这篇文章的链接。）
3. 在启动虚拟机时，用以下命令行选项指定要使用的口令文件的位置：

-Dcom.sun.management.jmxremote.password.file=<file-path>

从管理客户的角度来说，需要提供正确的用户名/口令组合来访问开启了认证的虚拟机。如果客户是 JConsole，这很简单：在初始的 Connection 选项卡中提供了用户名和口令字段。要编写向远程虚拟机提供认证细节的代码，需要把清单 16 所示的修改添加到清单 4 中提供的连接代码：

##### 清单 16. 连接到要求用户认证的远程虚拟机

try {

// provide a valid username and password (e.g., via program arguments)

String user = "monitorRole";

String pw = "password";

// place the username and password in a string array of credentials that

// can be used when making the connection to the remote JMX agent

String[] credentials = new String[] { user, pw };

// the string array of credentials is placed in a map keyed against the

// well-defined credentials identifier string

Map<String, String[]> props = new HashMap<String, String[]>();

props.put("jmx.remote.credentials", credentials);

// supply the map of credentials to the connect call

JMXServiceURL address =

new JMXServiceURL("service:jmx:rmi:///jndi/rmi://localhost:1234/jmxrmi");

JMXConnector connector = JMXConnectorFactory.connect(address, props);

// it is a trivial matter to get a reference for the MBean server

// connection to the remote agent

MBeanServerConnection mbs = connector.getMBeanServerConnection();

} catch ...

对要求认证的虚拟机提供错误的用户名或口令会造成 java.lang.SecurityException。类似地，以 monitorRole 进行认证，然后想调用读写操作 —— 例如试图请求垃圾收集 —— 也会造成抛出 SecurityException。

### 使用 SSL

可以用 SSL 对从平台 JMX 代理传递到监视平台管理客户的信息进行加密。传递的数据使用公钥（非对称）加密算法加密，所以只有对应私钥的持有者才能解密数据。这就防止了数据包侦听应用程序偷听通信。要使用这个特性，需要在连接的两端都配置 SSL，其中包括生成一对密钥和一个数字证书。具体细节超出了本文的范围，请阅读 Greg Travis 的精彩教程 “Using JSSE for secure socket communication” 了解更多内容（请参阅 [参考资料](https://www.ibm.com/developerworks/cn/java/j-mxbeans/#resources)）。

好消息是，一旦设置好了密钥对和证书，使用 SSL 时并不需要修改管理客户机代码。只要使用一些命令行选项，就可以开启加密。首先，想要监视或管理的 Java 应用程序必须用以下选项启动：

-Dcom.sun.management.jmxremote.ssl.need.client.auth=true

-Djavax.net.ssl.keyStore=<keystore-location>

-Djavax.net.ssl.trustStore=<truststore-location>

-Djavax.net.ssl.keyStoreType=<keystore-type>

-Djavax.net.ssl.keyStorePassword=<keystore-password>

-Djavax.net.ssl.trustStoreType=<truststore-type>

-Djavax.net.ssl.trustStorePassword=<truststore-password>

希望与平台通信的管理客户需要用以上选项的子集启动：可以用第一和第二行。如果客户是 JConsole，可以在启动 GUI 时，用 -J 命令行语法传递这些选项，该语法会把 Java 选项传递给 JVM。

同样，教程 “Using JSSE for secure socket communication” 可以提供这些单个选项的更多细节。

[**回页首**](https://www.ibm.com/developerworks/cn/java/j-mxbeans/#ibm-pcon)

## 结束语

我们希望我们已经吸引您去寻找关于 Java 5.0 平台的管理 API 的更多内容。因为 Java 管理扩展的出现为 Java 企业开发人员和管理人员提供了监视和控制他们部署的标准化方式，所以对 Java 5.0 中 java.lang.management API 的介绍，提供了对应用程序运行的平台进行检查的机制。

不论是要监视本地运行的应用程序的线程池还是安全地检查在 intranet 上其他位置需要注意的任务关键型程序的内存使用情况，MXBean 的获得和使用都非常简单。MXBean 可以起到中心作用，让您更多地了解代码运行的环境，既可以用非侵入性的方式探查和了解不熟悉的 Java 实现的特征，也可以构建自己的检测和性能监视工具。

因为 “诊断、监视和管理” 是 Java 平台即将推出的 6.0 发行版的一个关键主题，所以这个 API 肯定会在未来的 Java 技术中承担起更重要的角色。

## 参考资料

### 学习

* 您可以参阅本文在 developerWorks 全球站点上的 [英文原文](http://www.ibm.com/developerworks/java/library/j-mxbeans/?S_TACT=105AGX52&S_CMP=cn-a-j)。
* [java.lang.management API](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/management/package-summary.html)：请更详细地探索完整的 API。
* “[将jsse用于安全套接字通信](http://www.ibm.com/developerworks/cn/views/java/tutorials.jsp?cv_doc_id=84960)”（Greg Travis，developerWorks，2002 年 4 月）：了解关于安全套接字配置的更多内容。
* “[从黑箱到企业](http://www.ibm.com/developerworks/cn/java/j-jmx1/)”（Sing Li，developerWorks，2002 年 9 月-12 月）：在这个三部分的系列中了解关于 JMX 的更多内容。
* “[How to Secure a Password File on Microsoft Windows systems](http://java.sun.com/j2se/1.5.0/docs/guide/management/security-windows.html)”（Sun Microsystems）：学习如何在 Windows 平台上保护敏感的 Java 资源，例如 jmxremote.password 文件。
* [WebSphere 应用服务器信息中心](http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp?topic=/com.ibm.websphere.base.doc/info/aes/ae/cxml_javamanagementx.html)：了解 WebSphere 应用服务器版本 6 如何使用 JMX 技术。
* [驯服 Tiger](http://www.ibm.com/developerworks/cn/views/java/articles.jsp?view_by=search&search_by=%E9%A9%AF%E6%9C%8D+Tiger%3A)：John Zukowski 在 developerWorks 上的系列是转移到 Java 5 上的必读文章。
* [Java 技术专区](http://www.ibm.com/developerworks/java/cn/?S_TACT=105AGX52&S_CMP=cn-a-j)：数百篇 Java 编程各方面的文章。

### 获得产品和技术

* [MBeanInspector](http://www.alphaworks.ibm.com/tech/mbeaninspector)：WebSphere 5 用户可以下载这个技术，研究注册在应用服务器上的 MBean。

### 讨论

* [developerWorks blogs](http://www.ibm.com/developerworks/blogs/?S_TACT=105AGX52&S_CMP=cn-a-j)：加入 developerWorks 社区。