基于GMA的网格监控系统实现

杨道杰

(山东中医药高等专科学校，山东 烟台265200)

摘 要： 介绍了Java本地接口方法JNI，重点讨论了利用JNI调用C/C++动态联接库，实时监控系统中硬盘、内存、CPU等资源的运行状态的方法，实现了Java与C/C++的互操作。同时，通过在Web Service服务中调用JNI方法，提取Web Service的远程调用功能。在网格环境中，构建了基于GMA的监控系统模式，并给出了该系统的具体实现方法。通过理论分析和具体应用证实了该系统的有效性。

关键词： 网格；JNI；Web Service；GMA；监控系统

中图分类号： TP393 **文献标识码：** A

Implementation of grid monitoring system based on GMA

Yang Dao Jie

(Shandong College of Traditional Chinese Medicine，Yantai 265200，China)

Abstract: This article firstly introduced the Java native interface method JNI.Focused on the use of the JNI to call C/C++ dynamic link library，monitor running states of system such as the hard disk, memory and CPU resources. It also realized interoperability between Java and C/C++. At the same time, through the web service call service JNI method，extracting the remote web servicecall features. Finally, in the grid environment, built a monitoring system based on GMA model, and gave the specific methods to achieve it. It also confirmed the specific applicationof the system's effectiveness.

Key words: Grid；JNI；web service；GMA；monitoring system

网格是以资源共享为目的，支持对可计算资源的远程和并发访问，用高速互联网络连接地理上分布的可计算资源所组成的一个具有单一系统映像的高性能计算和信息服务环境[1]。

在网格环境下，存在着各种各样异构的计算资源，这些计算资源无论在硬件还是在软件上都存在很大差异。而且，这些计算资源可能分布在世界各地，通过互联网结合在一起。由于这些特点，运行过程中一些节点可能会发生故障，导致网络断开或者出现性能问题。而且，一些节点可能随时会动态地加入或者离开网格环境。虽然各种网格中间件都有一定的容错性，但在某些情况下，人工干预也是不可避免的。由于网格规模巨大，在系统运行时会产生大量的性能数据，手工对网格系统进行状态信息的收集、监控和分析而不借助一定的工具是很困难的，这就使得监控系统在网格中的作用显得尤为突出。

由于资源具有动态性、流动性的特征，网格系统的运行性能、稳定性、可靠性等重要指标，很大程度上依赖于网格系统的实时状态。这就要求在网格中提供一种资源监测机制负责对各种资源进行静、动态监测，收集各种资源及节点的状态变化信息，使用户和应用程序能够及时掌握资源分配与调度、网络带宽、处理器负载、系统吞吐量等信息，以便及时解决网格系统中出现的各种障碍，提高整个网格的性能。

1 JNI本地方法

Java本地接口方法JNI(Java Native Interface)是JDK的一部分，为Java提供一个本地代码的接口，是Java世界和其他语言间的桥梁。JNI允许运行在Java虚拟机JVM(Java Virtual Machine)上的代码调用本地程序和类库，或者被它们调用，这些程序和类库可以是其他语言编写的，比如C、C++或者汇编语言等。

1.1 JNI技术实现步骤

JNI在不同平台上的实现步骤相同。除了生成动态链接库的方法不同外，其他实现方法相同。图1以Java编程中通过JNI方法调用不同平台下的C/C++程序为例，说明其具体

步骤。

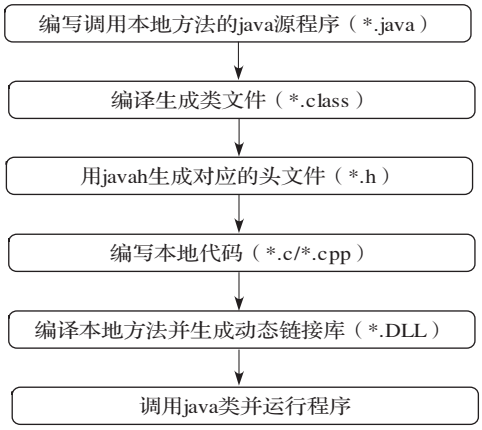

图1 JNI应用流程图

1.2 JNI 提取资源节点系统信息

(1)编写java源程序(MonitorInformation.java)；

```
public class MonitorInformation {
public native double[] Disk( );//硬盘的信息
public native int[] Cpu();//cpu的使用率
public native double[] Memory();//内存、虚拟内存的大小及使用率
public native long pin();//主频
static{
    try{//此处即为本地方法所在链接库名
System.loadLibrary("monitorinformation");
            }catch(UnsatisfiedLinkError e){
        ·····           }
            }
    }
```

(2)编译生成类文件(MonitorInformation.class)；

在 Eclipse+Myeclipse 开发环境下，这一步可以省略，因为 Myeclipse 会自动编译java源程序为.class文件，若不是，则可使用javac MonitorInform-ation.java进行编译，生成MonitorInformation.class文件。

(3)用javah生成头文件(sys_MonitorInformation.h)；

用javah sys.MonitorInformation 为native方法生成sys_MonitorInformation.h头文件。

(4)编写native方法(monitorinformation.cpp)；

JNI函数名称分为3部分：①Java关键字，供Java虚拟机识别；②调用者类名称；③对应的方法名称，各段名称之间用下划线分割。JNI函数的参数也由3部分组成：第1个是JNIEnv *，它是一个指向JNI运行环境的指针；第2个参数随本地方法是否静态而不同，非静态本地方法的第2个参数是对对象的引用，而静态本地方法的第2个参数是对其Java类的引用；其余的参数通常对应Java方法的参数，参数类型需要根据一定规则进行映射。注意：JNI函数返回值类型与Java函数返回值类型的相互转化。

```
JNIEXPORT jdoubleArray JNICALL Java_sys_MonitorInformation_Disk(JNIEnv *,
jobject){ //提取资源节点硬盘数据 }
JNIEXPORT jlong JNICALL Java_sys_MonitorInformation_pin(JNIEnv *, jobject){ //提取资源节点CPU主频 }
JNIEXPORT jintArray JNICALL Java_sys_MonitorInformation_Cpu (JNIEnv *, jobject){ //提取资源节点使用CPU的动态数据 }
JNIEXPORT jdoubleArray JNICALL Java_sys_MonitorInformation_Memory (JNIEnv *, jobject){ //提取资源节点内存数据(包括虚拟内存)}
```

(5)编译native方法并生成动态链接库(monitorinformation.dll)；

最后，将动态链接库放在Windows->System32文件夹下，调用并运行java程序。

2 Web Service

Web Service是建立在开放的Internet基础上的新的分布式计算模型[3]。Web Service组件是一套开放的技术规范，其组件的基本组成部分为HTTP、XML&XSD、WSDL、UDDI和SOAP。其系统构架基于TCP/IP、HTTP、XML等协议和规范，可以实现事务之间的通信、链接文档的浏览、事务的自动调用、服务的动态发现和发布等。其体系结构由Service provider、Service requester和Service broker 3个角色及Publish、Bind和Find 3个动作构建而成，如图2所示。

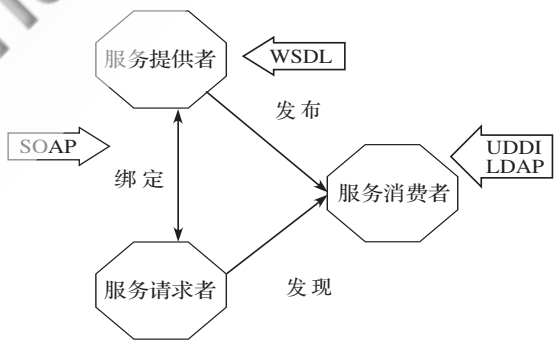

图2 面向服务的体系结构

服务提供者首先使用WSDL协议编制服务描述文件，并将其发布到UDDI注册中心，UDDI利用SOAP消息机制(标准的XML/HTTP)来发布、编辑、浏览以及查找注册信息。服务消费者在UDDI注册中心发现感兴趣的服务描述后，需要启动消息通信，消息和通信进程可以绑定到基于HTTP上的SOAP上，服务提供者根据SOAP的绑定参数，为服务请求者实施相应的服务。同时，在发布、发现、绑定服务的过程中，服务请求者和服务提供者对SOAP规范全力支持，从而实现了良好的跨平台、无缝互操作性[4]。

3 基于GMA的网格监控系统

3.1 基于GMA的网格监控模式

为了有效减少网格中的数据传输，本监控系统采用基于生产者/消费者/注册模式的GMA监控体系结构[4]，使用JNI、Web Service、目录服务等技术，构建面向服务的基于GMA的监控系统。作为一种类型的消费者，它实现了GMA定义的消费者接口，并以Web页面的方式向用户显示实时动态的性能信息。监控系统采用Servlet从生产者订阅数据、利用实时的性能数据动态生成图片，然后利用JSP页面与用户进行交互，图3显示了网格监控系统模式。

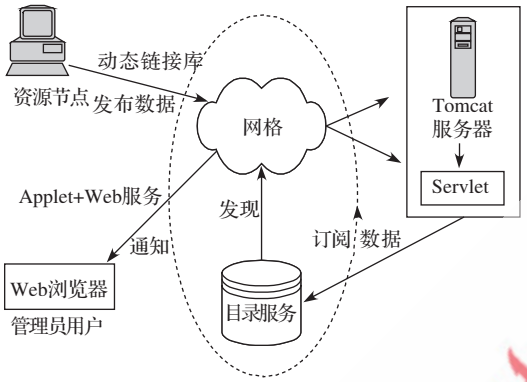

图3 网格监控模式图

网格监控流程：采用JNI技术对网格中每个活动资源节点进行动态实时监测，获取监控参数，通过Web服务技术进行远程数据传输，然后以applet形式显示到Web浏览器中。这样，用户首先以Web页面的方式登录管理节点，登录后可以查看该管理节点辖域内所有的资源节点，然后点击相应的资源节点即可查看该资源节点的具体配置信息。

由于网格监控的特点，监控的数据必须及时地传送到需要的地点，这就要求系统具有较小的延时以及较大的吞吐量。另外，监控系统要尽量减小给网格资源本身带来的负载。因此，GGF[5]性能工作组认为，将数据收集和数据传送分离有利于实现以上2点。GMA体系结构设计了一种独立的生产者/消费者模型，能够根据协商做到“需求匹配”，并且可以根据系统负载以一种更精确并且更分布的方式来控制数据流量，从而达到网格监控系统所要求的监控数据传输的低延迟、高传输率、低负载和安全性，使系统非常易于扩展。

3.2 实现

该资源监控系统采用了面向服务的GMA体系结构，在网格中部署分布式的资源监控服务。在实现过程中运用了Java技术调用VC++动态链接库的JNI技术以及在Java Applet小程序中封装Web服务技术，通过接口统一发布Web服务，以便及时获取所需要的信息。

(1)创建Web服务：monitorinformation

部分实现代码如下：

```
public interface Imonitorinformation {
public double[] dispdisk()，int[] dispcpu()，double[] dispmemory()，long disppin();
}
```

monitorinformationImpl.java部分代码如下：

```
public class monitorinformationImpl implements Imonitorinformation {
public double[] dispdisk(){
MonitorInformation m=new MonitorInformation();
        double[] disk=m.Disk( );
                return disk;
    }
public int[]dispcpu(){ //原理同上，……}
public double[] dispmemory(){ //原理同上，……}
public long disppin(){ //原理同上，…… }
    }
```

MApplet.java部分代码如下：

```
public class MApplet extends JApplet implements Runnable {
private int[] cpu，double[] memory，double[] disk;
private long pin;
    private Service srvcModel;
    private XFireProxyFactory factory;
    private String serviceURL;
    private Imonitorinformation srvc;
    public MApplet() throws Exception {
srvcModel = new ObjectServiceFactory().create(Imonitorinformation.cl-ass);
// 创建服务对象
factory = new XFireProxyFactory(XFireFactory.newInstance().getXFire());
// 使用XFire的服务工厂，生成创建实例
serviceURL=“http://”+ip+”:8080/project/services/monitorinformation";
// 指定服务的地址
try { //初始化请求一次
        srvc = (Imonitorinformation)factory.create(srvcModel， serviceURL);
        cpu = srvc.dispcpu();
        memory = srvc.dispmemory();
        disk = srvc.dispdisk();
        pin=srvc.disppin();
    } catch (MalformedURLException e){
        e.printStackTrace(); }
…… }
```

(2)如果没有经过数字签名，访问客户端程序下载后会受到安全限制。因此，将Web服务工程打包以后，对Applet小程序进行数字签名，签名工程如下：

第1步：创建证书keytool -genkey -alias <keyname>

-keystore <url>。

这里keyname是要给出的密钥别名，例如"mykeyname"；url是存放宇航局钥的文件位置，通常就是cacerts文件，在{java.home}/lib/security/cacerts，这里的java.home是指jre的路径，在jdk里，本系统的jre路径是：C:\jdk1.5.0_04\jre\lib\security\cacerts。

第2步：签名。

jarsigner -keystore <url> <jarfile> <keyname>

本系统的签名路径是：jarsigner -keystore C:\jdk1.5.0_04\jre\lib\security\cacerts G:\workspace\Graph\Graph_fat.jar zhu2008。

注意：签名时的<keyname>要与创建证书时的<keyname>相同，若不同，签名不通过。

启动Tomcat服务器后，在初次在浏览器运行时会出现如下提示对话框，必须选中“始终信任此发行者的内容(A)”，以便保证系统的安全性，再点击“运行”即可。如图4所示。

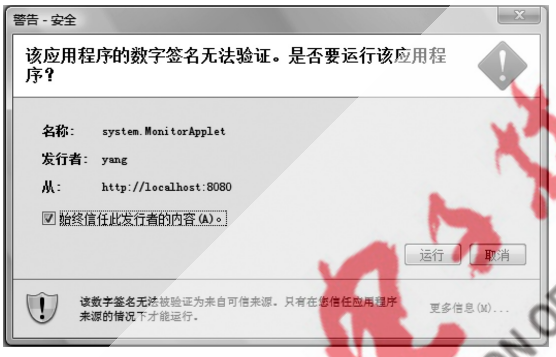

图4 警告–安全对话框

3.3 监控服务

网格监控系统都有自己的监控服务，注册中心[6]、监控信息提供者和监控服务代理等，可以实时监控当前资源的基本信息和实时状态信息。监控服务直接面向监控事件消费者与上层应用，为消费者访问提供统一的信息服务访问接口，它对应面向服务的GMA体系结构的生产者/消费者复合组件[7]，最初运行时需要向目录服务中心进行注册。监控服务主要负责维护资源监控系统内相应监控服务代理的注册信息以及与它相邻监控服务的相关信息，实时获取各个资源节点的数据，定期对所监控的所有资源的静动态监测信息生成详细监测报告，为用户提供监控视图。例如，图5中显示了处理器、内存、硬盘部分监控信息。

因为网格中资源状态信息和其他监控组件的变化都依赖于CPU的改变。如果CPU利用率几乎没有改变，那其他的资源状态也不会有大的改变。反之，意味着资源的状态将会有一个较大的改变，应该立即监控。通常情况下，CPU占用比例大于 70%的时间超过 1/3时，应该加强对CPU监控， 找出 CPU消耗的主要进程，分析进程高 CPU占用率的原因。CPU 占用比例大于80%的时间超过 1/2

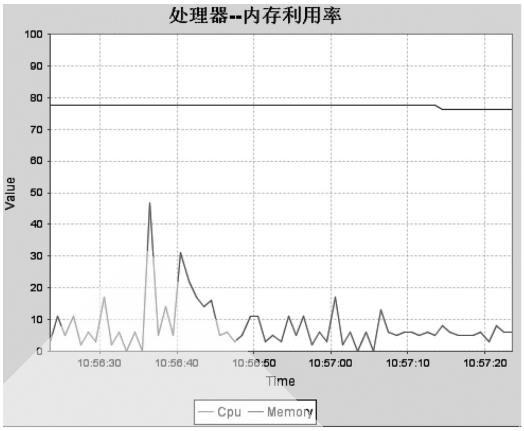

图5 信息监控图示

时，应同时加强内存监控， 考虑升级设备。从而，不必要的系统开销明显降低，事件的准确率也得到满足。

本文基于GMA体系结构以及Web Services、JNI、Applet、目录服务等技术，构建的网格监控系统，可以灵活地将静态和动态信息结合在一起，通过监控可以发现故障的资源节点，分析系统瓶颈，帮助用户在最短的时间内恢复和调整系统；通过监控可以了解计算资源或者存储资源等的负载情况，为调度程序提供信息，以实现节点间的负载平衡。

参考文献

[1] 黄达明，李国东，张德富.网格监控系统研究[J].计算机科学，2003，30(9):144-146.

[2] ROB G ，ALAN E. JNI，Java Native Interface. Prentice Hall，1998.

[3] 范凤岐，熊聪聪.基于网格的Web Service实现[J].计算机与数字工程，2007，35(2):110-112.

[4] 桂小林.网格技术导论[M].北京：北京邮电大学出版社，2006.

[5] IAN F， CARL K. The grid2 blueprint for a new computing infrastructure，2003.

[6] 廖剑伟，蔡洪斌，蒋攀登，等.基于Java的网格监控系统的设计与实现[J].计算机应用研究，2005，25(12):234-236.

[7] 张宏海.网格监控系统[J].超级计算通讯，2006，4(4):43-45.

(收稿日期：2008-12-22)