程序设计：

要求：

1. 使用Java语言和框架，采集linux系统的总CPU使用率，内存使用率，磁盘(根目录)使用率，网卡的发送(tx)和接受（rx）速率，按照一分钟一次的频率的进行，并且存储在数据库中。
2. 将存储在数据库中的CPU、内存、磁盘和网卡的数据，采用java web程序，将其按照时间顺序，展示出来，可以是文本形式，最好采用曲线进行展现，不需要考虑界面美化。
3. 可以借助google，百度等搜索引擎，可以导入开源的jar包和库，来实现程序，但是不可以采用完整的开源程序如Prometheus, grafana等
4. 时间为24小时，如果最后无法将完整程序写完，可以提交已完成的模块。

问题一答案：

package com.util;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileInputStream;

import java.io.InputStreamReader;

import java.io.LineNumberReader;

import java.lang.management.ManagementFactory;

import java.math.BigDecimal;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import java.util.StringTokenizer;

import org.apache.log4j.Logger;

import com.sun.management.OperatingSystemMXBean;

public class ComputerMonitorUtil {

private static String osName = System.getProperty("os.name");

private static final int CPUTIME = 500;

private static final int PERCENT = 100;

private static final int FAULTLENGTH = 10;

private static final Logger logger = Logger.getLogger(ComputerMonitorUtil.class);

/\*\*

\* 功能：获取Linux和Window系统cpu使用率

\* \*/

public static double getCpuUsage() {

// 如果是window系统

if (osName.toLowerCase().contains("windows")

|| osName.toLowerCase().contains("win")) {

try {

String procCmd = System.getenv("windir")

+ "//system32//wbem//wmic.exe process get Caption,CommandLine,KernelModeTime,ReadOperationCount,ThreadCount,UserModeTime,WriteOperationCount";

// 取进程信息

long[] c0 = readCpu(Runtime.getRuntime().exec(procCmd));//第一次读取CPU信息

Thread.sleep(CPUTIME);//睡500ms

long[] c1 = readCpu(Runtime.getRuntime().exec(procCmd));//第二次读取CPU信息

if (c0 != null && c1 != null) {

long idletime = c1[0] - c0[0];//空闲时间

long busytime = c1[1] - c0[1];//使用时间

Double cpusage = Double.valueOf(PERCENT \* (busytime) \* 1.0 / (busytime + idletime));

BigDecimal b1 = new BigDecimal(cpusage);

double cpuUsage = b1.setScale(2, BigDecimal.ROUND\_HALF\_UP).doubleValue();

return cpuUsage;

} else {

return 0.0;

}

} catch (Exception ex) {

logger.debug(ex);

return 0.0;

}

} else {

try {

Map<?, ?> map1 = ComputerMonitorUtil.cpuinfo();

Thread.sleep(CPUTIME);

Map<?, ?> map2 = ComputerMonitorUtil.cpuinfo();

long user1 = Long.parseLong(map1.get("user").toString());

long nice1 = Long.parseLong(map1.get("nice").toString());

long system1 = Long.parseLong(map1.get("system").toString());

long idle1 = Long.parseLong(map1.get("idle").toString());

long user2 = Long.parseLong(map2.get("user").toString());

long nice2 = Long.parseLong(map2.get("nice").toString());

long system2 = Long.parseLong(map2.get("system").toString());

long idle2 = Long.parseLong(map2.get("idle").toString());

long total1 = user1 + system1 + nice1;

long total2 = user2 + system2 + nice2;

float total = total2 - total1;

long totalIdle1 = user1 + nice1 + system1 + idle1;

long totalIdle2 = user2 + nice2 + system2 + idle2;

float totalidle = totalIdle2 - totalIdle1;

float cpusage = (total / totalidle) \* 100;

BigDecimal b1 = new BigDecimal(cpusage);

Double cpuUsage= b1.setScale(2, BigDecimal.ROUND\_HALF\_UP).doubleValue();

return cpuUsage;

} catch (InterruptedException e) {

logger.debug(e);

}

}

return 0;

}

/\*\*

\* 功能：Linux CPU使用信息

\* \*/

public static Map<?, ?> cpuinfo() {

InputStreamReader inputs = null;

BufferedReader buffer = null;

Map<String, Object> map = new HashMap<String, Object>();

try {

inputs = new InputStreamReader(new FileInputStream("/proc/stat"));

buffer = new BufferedReader(inputs);

String line = "";

while (true) {

line = buffer.readLine();

if (line == null) {

break;

}

if (line.startsWith("cpu")) {

StringTokenizer tokenizer = new StringTokenizer(line);

List<String> temp = new ArrayList<String>();

while (tokenizer.hasMoreElements()) {

String value = tokenizer.nextToken();

temp.add(value);

}

map.put("user", temp.get(1));

map.put("nice", temp.get(2));

map.put("system", temp.get(3));

map.put("idle", temp.get(4));

map.put("iowait", temp.get(5));

map.put("irq", temp.get(6));

map.put("softirq", temp.get(7));

map.put("stealstolen", temp.get(8));

break;

}

}

} catch (Exception e) {

logger.debug(e);

} finally {

try {

buffer.close();

inputs.close();

} catch (Exception e2) {

logger.debug(e2);

}

}

return map;

}

/\*\*

\* 功能：Linux 和 Window 内存使用率

\* \*/

public static double getMemUsage() {

if (osName.toLowerCase().contains("windows")

|| osName.toLowerCase().contains("win")) {

try {

OperatingSystemMXBean osmxb = (OperatingSystemMXBean) ManagementFactory

.getOperatingSystemMXBean();

// 总的物理内存+虚拟内存

long totalvirtualMemory = osmxb.getTotalSwapSpaceSize();

// 剩余的物理内存

long freePhysicalMemorySize = osmxb.getFreePhysicalMemorySize();

Double usage = (Double) (1 - freePhysicalMemorySize \* 1.0 / totalvirtualMemory) \* 100;

BigDecimal b1 = new BigDecimal(usage);

double memoryUsage = b1.setScale(2, BigDecimal.ROUND\_HALF\_UP).doubleValue();

return memoryUsage;

} catch (Exception e) {

logger.debug(e);

}

} else {

Map<String, Object> map = new HashMap<String, Object>();

InputStreamReader inputs = null;

BufferedReader buffer = null;

try {

inputs = new InputStreamReader(new FileInputStream("/proc/meminfo"));

buffer = new BufferedReader(inputs);

String line = "";

while (true) {

line = buffer.readLine();

if (line == null)

break;

int beginIndex = 0;

int endIndex = line.indexOf(":");

if (endIndex != -1) {

String key = line.substring(beginIndex, endIndex);

beginIndex = endIndex + 1;

endIndex = line.length();

String memory = line.substring(beginIndex, endIndex);

String value = memory.replace("kB", "").trim();

map.put(key, value);

}

}

long memTotal = Long.parseLong(map.get("MemTotal").toString());

long memFree = Long.parseLong(map.get("MemFree").toString());

long memused = memTotal - memFree;

long buffers = Long.parseLong(map.get("Buffers").toString());

long cached = Long.parseLong(map.get("Cached").toString());

double usage = (double) (memused - buffers - cached) / memTotal \* 100;

BigDecimal b1 = new BigDecimal(usage);

double memoryUsage = b1.setScale(2, BigDecimal.ROUND\_HALF\_UP).doubleValue();

return memoryUsage;

} catch (Exception e) {

logger.debug(e);

} finally {

try {

buffer.close();

inputs.close();

} catch (Exception e2) {

logger.debug(e2);

}

}

}

return 0.0;

}

/\*\*

\* Window 和Linux 得到磁盘的使用率

\*

\* @return

\* @throws Exception

\*/

public static double getDiskUsage() throws Exception {

double totalHD = 0;

double usedHD = 0;

if (osName.toLowerCase().contains("windows")

|| osName.toLowerCase().contains("win")) {

long allTotal = 0;

long allFree = 0;

for (char c = 'A'; c <= 'Z'; c++) {

String dirName = c + ":/";

File win = new File(dirName);

if (win.exists()) {

long total = (long) win.getTotalSpace();

long free = (long) win.getFreeSpace();

allTotal = allTotal + total;

allFree = allFree + free;

}

}

Double precent = (Double) (1 - allFree \* 1.0 / allTotal) \* 100;

BigDecimal b1 = new BigDecimal(precent);

precent = b1.setScale(2, BigDecimal.ROUND\_HALF\_UP).doubleValue();

return precent;

} else {

Runtime rt = Runtime.getRuntime();

Process p = rt.exec("df -hl");// df -hl 查看硬盘空间

BufferedReader in = null;

try {

in = new BufferedReader(new InputStreamReader(p.getInputStream()));

String str = null;

String[] strArray = null;

while ((str = in.readLine()) != null) {

int m = 0;

strArray = str.split(" ");

for (String tmp : strArray) {

if (tmp.trim().length() == 0)

continue;

++m;

if (tmp.indexOf("G") != -1) {

if (m == 2) {

if (!tmp.equals("") && !tmp.equals("0"))

totalHD += Double.parseDouble(tmp.substring(0, tmp.length() - 1)) \* 1024;

}

if (m == 3) {

if (!tmp.equals("none") && !tmp.equals("0"))

usedHD += Double.parseDouble(tmp.substring(0, tmp.length() - 1)) \* 1024;

}

}

if (tmp.indexOf("M") != -1) {

if (m == 2) {

if (!tmp.equals("") && !tmp.equals("0"))

totalHD += Double.parseDouble(tmp.substring(0, tmp.length() - 1));

}

if (m == 3) {

if (!tmp.equals("none") && !tmp.equals("0"))

usedHD += Double.parseDouble(tmp.substring(0, tmp.length() - 1));

}

}

}

}

} catch (Exception e) {

logger.debug(e);

} finally {

in.close();

}

// 保留2位小数

double precent = (usedHD / totalHD) \* 100;

BigDecimal b1 = new BigDecimal(precent);

precent = b1.setScale(2, BigDecimal.ROUND\_HALF\_UP).doubleValue();

return precent;

}

}

// window读取cpu相关信息

private static long[] readCpu(final Process proc) {

long[] retn = new long[2];

try {

proc.getOutputStream().close();

InputStreamReader ir = new InputStreamReader(proc.getInputStream());

LineNumberReader input = new LineNumberReader(ir);

String line = input.readLine();

if (line == null || line.length() < FAULTLENGTH) {

return null;

}

int capidx = line.indexOf("Caption");

int cmdidx = line.indexOf("CommandLine");

int rocidx = line.indexOf("ReadOperationCount");

int umtidx = line.indexOf("UserModeTime");

int kmtidx = line.indexOf("KernelModeTime");

int wocidx = line.indexOf("WriteOperationCount");

long idletime = 0;

long kneltime = 0;//读取物理设备时间

long usertime = 0;//执行代码占用时间

while ((line = input.readLine()) != null) {

if (line.length() < wocidx) {

continue;

}

// 字段出现顺序：Caption,CommandLine,KernelModeTime,ReadOperationCount

String caption = substring(line, capidx, cmdidx - 1).trim();

String cmd = substring(line, cmdidx, kmtidx - 1).trim();

if (cmd.indexOf("wmic.exe") >= 0) {

continue;

}

String s1 = substring(line, kmtidx, rocidx - 1).trim();

String s2 = substring(line, umtidx, wocidx - 1).trim();

List<String> digitS1 = getDigit(s1);

List<String> digitS2 = getDigit(s2);

if (caption.equals("System Idle Process") || caption.equals("System")) {

if (s1.length() > 0) {

if (!digitS1.get(0).equals("") && digitS1.get(0) != null) {

idletime += Long.valueOf(digitS1.get(0)).longValue();

}

}

if (s2.length() > 0) {

if (!digitS2.get(0).equals("") && digitS2.get(0) != null) {

idletime += Long.valueOf(digitS2.get(0)).longValue();

}

}

continue;

}

if (s1.length() > 0) {

if (!digitS1.get(0).equals("") && digitS1.get(0) != null) {

kneltime += Long.valueOf(digitS1.get(0)).longValue();

}

}

if (s2.length() > 0) {

if (!digitS2.get(0).equals("") && digitS2.get(0) != null) {

kneltime += Long.valueOf(digitS2.get(0)).longValue();

}

}

}

retn[0] = idletime;

retn[1] = kneltime + usertime;

return retn;

} catch (Exception ex) {

logger.debug(ex);

} finally {

try {

proc.getInputStream().close();

} catch (Exception e) {

logger.debug(e);

}

}

return null;

}

/\*\*

\* 从字符串文本中获得数字

\*

\* @param text

\* @return

\*/

private static List<String> getDigit(String text) {

List<String> digitList = new ArrayList<String>();

digitList.add(text.replaceAll("\\D", ""));

return digitList;

}

/\*\*

\* 由于String.subString对汉字处理存在问题（把一个汉字视为一个字节)，因此在 包含汉字的字符串时存在隐患，现调整如下：

\*

\* @param src

\* 要截取的字符串

\* @param start\_idx

\* 开始坐标（包括该坐标)

\* @param end\_idx

\* 截止坐标（包括该坐标）

\* @return

\*/

private static String substring(String src, int start\_idx, int end\_idx) {

byte[] b = src.getBytes();

String tgt = "";

for (int i = start\_idx; i <= end\_idx; i++) {

tgt += (char) b[i];

}

return tgt;

}

public static void main(String[] args) throws Exception {

double cpuUsage = ComputerMonitorUtil .getCpuUsage();

//当前系统的内存使用率

double memUsage = ComputerMonitorUtil .getMemUsage();

//当前系统的硬盘使用率

double diskUsage = ComputerMonitorUtil .getDiskUsage();

System.out.println("cpuUsage:"+cpuUsage);

System.out.println("memUsage:"+memUsage);

System.out.println("diskUsage:"+diskUsage);

}

}

问题2答案：

/\*\*

\* @Title RuntimeTest.java

\* @Description 将存储在数据库中的CPU、内存、磁盘和网卡的数据，采用java web程序，将其按照时间顺序，展示出来

\* @date 2020-06-01

\*/

package osinfo;

import java.net.InetAddress;

import java.net.UnknownHostException;

import java.util.Map;

import java.util.Properties;

import org.hyperic.sigar.CpuInfo;

import org.hyperic.sigar.CpuPerc;

import org.hyperic.sigar.FileSystem;

import org.hyperic.sigar.FileSystemUsage;

import org.hyperic.sigar.Mem;

import org.hyperic.sigar.NetFlags;

import org.hyperic.sigar.NetInterfaceConfig;

import org.hyperic.sigar.NetInterfaceStat;

import org.hyperic.sigar.OperatingSystem;

import org.hyperic.sigar.Sigar;

import org.hyperic.sigar.SigarException;

import org.hyperic.sigar.Swap;

import org.hyperic.sigar.Who;

public class RuntimeTest {

public static void main(String[] args) {

try {

// System信息，从jvm获取

property();

System.out.println("----------------------------------");

// cpu信息

cpu();

System.out.println("----------------------------------");

// 内存信息

memory();

System.out.println("----------------------------------");

// 操作系统信息

os();

System.out.println("----------------------------------");

// 用户信息

who();

System.out.println("----------------------------------");

// 文件系统信息

file();

System.out.println("----------------------------------");

// 网络信息

net();

System.out.println("----------------------------------");

// 以太网信息

ethernet();

System.out.println("----------------------------------");

} catch (Exception e1) {

e1.printStackTrace();

}

}

private static void property() throws UnknownHostException {

Runtime r = Runtime.getRuntime();

Properties props = System.getProperties();

InetAddress addr;

addr = InetAddress.getLocalHost();

String ip = addr.getHostAddress();

Map<String, String> map = System.getenv();

String userName = map.get("USERNAME");// 获取用户名

String computerName = map.get("COMPUTERNAME");// 获取计算机名

String userDomain = map.get("USERDOMAIN");// 获取计算机域名

System.out.println("用户名: " + userName);

System.out.println("计算机名: " + computerName);

System.out.println("计算机域名: " + userDomain);

System.out.println("本地ip地址: " + ip);

System.out.println("本地主机名: " + addr.getHostName());

System.out.println("JVM可以使用的总内存: " + r.totalMemory());

System.out.println("JVM可以使用的剩余内存: " + r.freeMemory());

System.out.println("JVM可以使用的处理器个数: " + r.availableProcessors());

System.out.println("Java的运行环境版本： " + props.getProperty("java.version"));

System.out.println("Java的运行环境供应商： " + props.getProperty("java.vendor"));

System.out.println("Java供应商的URL： " + props.getProperty("java.vendor.url"));

System.out.println("Java的安装路径： " + props.getProperty("java.home"));

System.out.println("Java的虚拟机规范版本： " + props.getProperty("java.vm.specification.version"));

System.out.println("Java的虚拟机规范供应商： " + props.getProperty("java.vm.specification.vendor"));

System.out.println("Java的虚拟机规范名称： " + props.getProperty("java.vm.specification.name"));

System.out.println("Java的虚拟机实现版本： " + props.getProperty("java.vm.version"));

System.out.println("Java的虚拟机实现供应商： " + props.getProperty("java.vm.vendor"));

System.out.println("Java的虚拟机实现名称： " + props.getProperty("java.vm.name"));

System.out.println("Java运行时环境规范版本： " + props.getProperty("java.specification.version"));

System.out.println("Java运行时环境规范供应商： " + props.getProperty("java.specification.vender"));

System.out.println("Java运行时环境规范名称： " + props.getProperty("java.specification.name"));

System.out.println("Java的类格式版本号： " + props.getProperty("java.class.version"));

System.out.println("Java的类路径： " + props.getProperty("java.class.path"));

System.out.println("加载库时搜索的路径列表： " + props.getProperty("java.library.path"));

System.out.println("默认的临时文件路径： " + props.getProperty("java.io.tmpdir"));

System.out.println("一个或多个扩展目录的路径： " + props.getProperty("java.ext.dirs"));

System.out.println("操作系统的名称： " + props.getProperty("os.name"));

System.out.println("操作系统的构架： " + props.getProperty("os.arch"));

System.out.println("操作系统的版本： " + props.getProperty("os.version"));

System.out.println("文件分隔符： " + props.getProperty("file.separator"));

System.out.println("路径分隔符： " + props.getProperty("path.separator"));

System.out.println("行分隔符： " + props.getProperty("line.separator"));

System.out.println("用户的账户名称： " + props.getProperty("user.name"));

System.out.println("用户的主目录： " + props.getProperty("user.home"));

System.out.println("用户的当前工作目录： " + props.getProperty("user.dir"));

}

private static void memory() throws SigarException {

Sigar sigar = new Sigar();

Mem mem = sigar.getMem();

// 内存总量

System.out.println("内存总量: " + mem.getTotal() / 1024L + "K av");

// 当前内存使用量

System.out.println("当前内存使用量: " + mem.getUsed() / 1024L + "K used");

// 当前内存剩余量

System.out.println("当前内存剩余量: " + mem.getFree() / 1024L + "K free");

Swap swap = sigar.getSwap();

// 交换区总量

System.out.println("交换区总量: " + swap.getTotal() / 1024L + "K av");

// 当前交换区使用量

System.out.println("当前交换区使用量: " + swap.getUsed() / 1024L + "K used");

// 当前交换区剩余量

System.out.println("当前交换区剩余量: " + swap.getFree() / 1024L + "K free");

}

private static void cpu() throws SigarException {

Sigar sigar = new Sigar();

CpuInfo infos[] = sigar.getCpuInfoList();

CpuPerc cpuList[] = null;

cpuList = sigar.getCpuPercList();

for (int i = 0; i < infos.length; i++) {// 不管是单块CPU还是多CPU都适用

CpuInfo info = infos[i];

System.out.println("第" + (i + 1) + "块CPU信息");

System.out.println("CPU的总量MHz: " + info.getMhz());// CPU的总量MHz

System.out.println("CPU生产商: " + info.getVendor());// 获得CPU的卖主，如：Intel

System.out.println("CPU类别: " + info.getModel());// 获得CPU的类别，如：Celeron

System.out.println("CPU缓存数量: " + info.getCacheSize());// 缓冲存储器数量

printCpuPerc(cpuList[i]);

}

}

private static void printCpuPerc(CpuPerc cpu) {

System.out.println("CPU用户使用率: " + CpuPerc.format(cpu.getUser()));// 用户使用率

System.out.println("CPU系统使用率: " + CpuPerc.format(cpu.getSys()));// 系统使用率

System.out.println("CPU当前等待率: " + CpuPerc.format(cpu.getWait()));// 当前等待率

System.out.println("CPU当前错误率: " + CpuPerc.format(cpu.getNice()));//

System.out.println("CPU当前空闲率: " + CpuPerc.format(cpu.getIdle()));// 当前空闲率

System.out.println("CPU总的使用率: " + CpuPerc.format(cpu.getCombined()));// 总的使用率

}

private static void os() {

OperatingSystem OS = OperatingSystem.getInstance();

// 操作系统内核类型如： 386、486、586等x86

System.out.println("操作系统: " + OS.getArch());

System.out.println("操作系统CpuEndian(): " + OS.getCpuEndian());//

System.out.println("操作系统DataModel(): " + OS.getDataModel());//

// 系统描述

System.out.println("操作系统的描述: " + OS.getDescription());

// 操作系统类型

// System.out.println("OS.getName(): " + OS.getName());

// System.out.println("OS.getPatchLevel(): " + OS.getPatchLevel());//

// 操作系统的卖主

System.out.println("操作系统的卖主: " + OS.getVendor());

// 卖主名称

System.out.println("操作系统的卖主名: " + OS.getVendorCodeName());

// 操作系统名称

System.out.println("操作系统名称: " + OS.getVendorName());

// 操作系统卖主类型

System.out.println("操作系统卖主类型: " + OS.getVendorVersion());

// 操作系统的版本号

System.out.println("操作系统的版本号: " + OS.getVersion());

}

private static void who() throws SigarException {

Sigar sigar = new Sigar();

Who who[] = sigar.getWhoList();

if (who != null && who.length > 0) {

for (int i = 0; i < who.length; i++) {

// System.out.println("当前系统进程表中的用户名" + String.valueOf(i));

Who \_who = who[i];

System.out.println("用户控制台: " + \_who.getDevice());

System.out.println("用户host: " + \_who.getHost());

// System.out.println("getTime(): " + \_who.getTime());

// 当前系统进程表中的用户名

System.out.println("当前系统进程表中的用户名: " + \_who.getUser());

}

}

}

private static void file() throws Exception {

Sigar sigar = new Sigar();

FileSystem fslist[] = sigar.getFileSystemList();

for (int i = 0; i < fslist.length; i++) {

System.out.println("分区的盘符名称" + i);

FileSystem fs = fslist[i];

// 分区的盘符名称

System.out.println("盘符名称: " + fs.getDevName());

// 分区的盘符名称

System.out.println("盘符路径: " + fs.getDirName());

System.out.println("盘符标志: " + fs.getFlags());//

// 文件系统类型，比如 FAT32、NTFS

System.out.println("盘符类型: " + fs.getSysTypeName());

// 文件系统类型名，比如本地硬盘、光驱、网络文件系统等

System.out.println("盘符类型名: " + fs.getTypeName());

// 文件系统类型

System.out.println("盘符文件系统类型: " + fs.getType());

FileSystemUsage usage = null;

usage = sigar.getFileSystemUsage(fs.getDirName());

switch (fs.getType()) {

case 0: // TYPE\_UNKNOWN ：未知

break;

case 1: // TYPE\_NONE

break;

case 2: // TYPE\_LOCAL\_DISK : 本地硬盘

// 文件系统总大小

System.out.println(fs.getDevName() + "总大小: " + usage.getTotal() + "KB");

// 文件系统剩余大小

System.out.println(fs.getDevName() + "剩余大小: " + usage.getFree() + "KB");

// 文件系统可用大小

System.out.println(fs.getDevName() + "可用大小: " + usage.getAvail() + "KB");

// 文件系统已经使用量

System.out.println(fs.getDevName() + "已经使用量: " + usage.getUsed() + "KB");

double usePercent = usage.getUsePercent() \* 100D;

// 文件系统资源的利用率

System.out.println(fs.getDevName() + "资源的利用率: " + usePercent + "%");

break;

case 3:// TYPE\_NETWORK ：网络

break;

case 4:// TYPE\_RAM\_DISK ：闪存

break;

case 5:// TYPE\_CDROM ：光驱

break;

case 6:// TYPE\_SWAP ：页面交换

break;

}

System.out.println(fs.getDevName() + "读出： " + usage.getDiskReads());

System.out.println(fs.getDevName() + "写入： " + usage.getDiskWrites());

}

return;

}

private static void net() throws Exception {

Sigar sigar = new Sigar();

String ifNames[] = sigar.getNetInterfaceList();

for (int i = 0; i < ifNames.length; i++) {

String name = ifNames[i];

NetInterfaceConfig ifconfig = sigar.getNetInterfaceConfig(name);

System.out.println("网络设备名: " + name);// 网络设备名

System.out.println("IP地址: " + ifconfig.getAddress());// IP地址

System.out.println("子网掩码: " + ifconfig.getNetmask());// 子网掩码

if ((ifconfig.getFlags() & 1L) <= 0L) {

System.out.println("!IFF\_UP...skipping getNetInterfaceStat");

continue;

}

NetInterfaceStat ifstat = sigar.getNetInterfaceStat(name);

System.out.println(name + "接收的总包裹数:" + ifstat.getRxPackets());// 接收的总包裹数

System.out.println(name + "发送的总包裹数:" + ifstat.getTxPackets());// 发送的总包裹数

System.out.println(name + "接收到的总字节数:" + ifstat.getRxBytes());// 接收到的总字节数

System.out.println(name + "发送的总字节数:" + ifstat.getTxBytes());// 发送的总字节数

System.out.println(name + "接收到的错误包数:" + ifstat.getRxErrors());// 接收到的错误包数

System.out.println(name + "发送数据包时的错误数:" + ifstat.getTxErrors());// 发送数据包时的错误数

System.out.println(name + "接收时丢弃的包数:" + ifstat.getRxDropped());// 接收时丢弃的包数

System.out.println(name + "发送时丢弃的包数:" + ifstat.getTxDropped());// 发送时丢弃的包数

}

}

private static void ethernet() throws SigarException {

Sigar sigar = null;

sigar = new Sigar();

String[] ifaces = sigar.getNetInterfaceList();

for (int i = 0; i < ifaces.length; i++) {

NetInterfaceConfig cfg = sigar.getNetInterfaceConfig(ifaces[i]);

if (NetFlags.LOOPBACK\_ADDRESS.equals(cfg.getAddress()) || (cfg.getFlags() & NetFlags.IFF\_LOOPBACK) != 0

|| NetFlags.NULL\_HWADDR.equals(cfg.getHwaddr())) {

continue;

}

System.out.println(cfg.getName() + "IP地址:" + cfg.getAddress());// IP地址

System.out.println(cfg.getName() + "网关广播地址:" + cfg.getBroadcast());// 网关广播地址

System.out.println(cfg.getName() + "网卡MAC地址:" + cfg.getHwaddr());// 网卡MAC地址

System.out.println(cfg.getName() + "子网掩码:" + cfg.getNetmask());// 子网掩码

System.out.println(cfg.getName() + "网卡描述信息:" + cfg.getDescription());// 网卡描述信息

System.out.println(cfg.getName() + "网卡类型" + cfg.getType());

}

}

}