**Windows下双守护进程设计说明文档**

**作者：郑智聪**

**时间：2018年10月**

1.概述

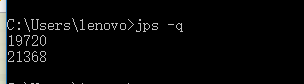
Java的运行环境不同于C等语言开发的程序，Java程序跑在JVM上面。不同于C语言可以直接创建进程，Java创建一个进程等同于使用java -jar xxx.jar启动一个程序。针对于服务器端上线后突然挂掉的解决方案是使用其他进程来守护服务器程序，如果服务器程序挂了，通过守护进程来启动服务器程序。万一守护进程挂了呢？使用双守护来提高稳定性，守护A负责监控服务器程序与守护B，守护B负责监控守护A，任何一方出现问题，都能快速的启动程序，提高服务器程序的稳定性。

2.涉及技术

1. jps命令

jps位于jdk的bin目录下，其作用是显示当前系统的java进程情况，及其id号。 jps相当于Solaris进程工具ps。不象”pgrep java”或”ps -ef grep java”，jps并不使用应用程序名来查找JVM实例。因此，它查找所有的Java应用程序，包括即使没有使用java执行体的那种（例如，定制的启动 器）。另外，jps仅查找当前用户的Java进程，而不是当前系统中的所有进程。

-q 只显示pid，不显示class名称,jar文件名和传递给main 方法的参数。



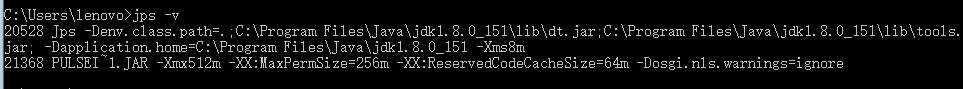
-m 输出传递给main 方法的参数，在嵌入式jvm上可能是null。



-l 输出应用程序main class的完整package名 或者 应用程序的jar文件完整路径名。



-v 输出传递给JVM的参数。



2. java.nio.channels.FileLock文件锁类的使用

FileLock是java 1.4 版本后出现的一个类，它可以通过对一个可写文件(w)加锁，保证同时只有一个进程可以拿到文件的锁，这个进程从而可以对文件做访问；而其它拿不到锁的进程要么选择被挂起等待，要么选择去做一些其它的事情， 这样的机制保证了众进程可以顺序访问该文件。也可以看出，能够利用文件锁的这种性质，在一些场景下，虽然我们不需要操作某个文件， 但也可以通过 FileLock 来进行并发控制，保证进程的顺序执行，避免数据错误。本程序中使用它可以维持在读取文件的同时给文件加上锁，判断文件时候有锁可以判断该文件是否被其他的程序使用。

3. Java Runtime.exec()的使用

用Java编写应用时，有时需要在程序中调用另一个现成的可执行程序或系统命令。比如用法Runtime.getRuntime.exec("notepad")，运行这个Java程序，就会运行记事本程序。同理，只需修改那个参数就可以运行其他的一些程序，也可以进行一些操作，比如关机。本程序中使用这一命令调用批处理文件启动java程序。

3.设计原理

Server：服务器程序

GuardA：守护进程A

GuardB：守护进程B

C:\\java\\A.txt:守护进程A的文件锁

C:\\java\\B.txt:守护进程B的文件锁

A和B之间的守护

1.A判断B是否存活，没有就启动B

2.B判断A是否存活，没有就启动A

3.在运行过程中A与B互相去拿对方的文件锁，如果拿到了，证明对面挂了，则启动对方。

4.A启动的时候，获取C:\\java\\A.txt文件的锁，如果拿到了证明没有A启动，则A运行;如果没有拿到锁，证明A已经启动了，或者是B判断的时候拿到了锁，如果是A已经启动了，不需要再次启动A，如果是B判断的时候拿到了锁，问题不大，反正B会再次启动A。

5.B启动的时候原理与A一致。

6.运行中如果A挂了，B判断到A已经挂了，则启动A。B同理。

守护服务器程序

1.A用于守护B和Server，B用于守护A。

2.原理与Step 1 一致，只是A多个一个守护Serer的任务。

3.当A运行的时候，使用进程pid检测到Server已经挂了，就启动Server

4.如果Server与A都挂了，B会启动A，然后A启动Server

5.如果Server与B挂了，A启动Server与B

6.如果A与B都挂了，守护结束

4.类的实现

1.GuardA

**public** **class** GuardA {

// GuardA用于维持自己的锁

**private** File fileGuardA;

**private** FileOutputStream fileOutputStreamGuardA;

**private** FileChannel fileChannelGuardA;

**private** FileLock fileLockGuardA;

// GuardA用于检测B的锁

**private** File fileGuardB;

**private** FileOutputStream fileOutputStreamGuardB;

**private** FileChannel fileChannelGuardB;

**private** FileLock fileLockGuardB;

**public** GuardA() **throws** Exception {

fileGuardA = **new** File(Configure.*GUARD\_A\_LOCK*);

**if** (!fileGuardA.exists()) {

fileGuardA.createNewFile();//文件不存在，创建新文件

}

//获取文件锁，拿不到证明GuardA已启动则退出

fileOutputStreamGuardA = **new** FileOutputStream(fileGuardA);

fileChannelGuardA = fileOutputStreamGuardA.getChannel();

//tryLock() 是非阻塞式的，它设法获取锁，但如果不能获得，例如因为其他一些进程已经持有相同的锁，

//而且不共享时，抛出文件重叠锁异常【OverlappingFileLockException】

fileLockGuardA = fileChannelGuardA.tryLock();

**if** (fileLockGuardA == **null**) {

System.*exit*(0);//终止jvm，正常退出java程序

}

fileGuardB = **new** File(Configure.*GUARD\_B\_LOCK*);

**if** (!fileGuardB.exists()) {

fileGuardB.createNewFile();

}

fileOutputStreamGuardB = **new** FileOutputStream(fileGuardB);

fileChannelGuardB = fileOutputStreamGuardB.getChannel();

}

/\*\*

\* 检测B是否存在

\*

\* **@return** true B已经存在

\*/

**public** **boolean** checkGuardB() {

**try** {

fileLockGuardB = fileChannelGuardB.tryLock();

**if** (fileLockGuardB == **null**) {

**return** **true**;

} **else** {

fileLockGuardB.release();

**return** **false**;

}

} **catch** (IOException e) {

System.*exit*(0);

// never touch

**return** **true**;

}

}

}

类GuardA实现在守护进程A启动时拿到A.txt的文件锁以及去检测是否拿到B.txt文件的文件锁的功能。

2.配置类condifure

**public** **class** Configure {

**public** **static** **final** String *GUARD\_B\_LOCK* = "C:\\java\\A.txt";

**public** **static** **final** String *GUARD\_A\_LOCK* = "C:\\java\\B.txt";

//得到被守护进程的进程名

**public** String getServername() {

// **TODO** Auto-generated method stub

String serverName = **null**;

//守护web服务器

//springmvc版本

//serverName = "org.apache.catalina.startup.Bootstrap";

//springboot版本

serverName = "ROOM.jar";//

**return** serverName;

}

//运行被守护进程的路径

**public** String getStartserver() {

// **TODO** Auto-generated method stub

//服务器

//String Cmd = "cmd /c C:\\java\\tamcat\\apache-tomcat-8.5.32-windows-x64\\apache-tomcat-8.5.32\\bin\\startup.bat";

//本地

//String Cmd = "cmd /c G:\\tamcat\\apache-tomcat-8\\apache-tomcat-8.0.51\\bin\\startup.bat";

String file = "C:\\java\\Webserver.bat";

String Cmd = "cmd /c start "+file.replaceAll(" ", "\" \"");

**return** Cmd;

}

//睡眠时间

**public** **long** getInterval() {

// **TODO** Auto-generated method stub

**return** 5000;

}

**public** String getStartguardb() {

String file = "C:\\java\\DaemonB.bat";

String cmd ="cmd /c start "+file.replaceAll(" ", "\" \"");

**return** cmd;

}

}

该类实现文件锁，守护的进程名，服务器进程和守护进程B的启动方法，以及睡眠时间的配置

3.主类GuardAMain

**public** **class** GuardAMain {

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

System.*out*.println("GuardA已启动..."+*getStringDate*());

GuardA guardA = **new** GuardA();

Configure configure = **new** Configure();

GuardServer server = **new** GuardServer(configure.getServername());

**while** (**true**) {

// 如果GuardB未运行 运行GuardB

**if** (!guardA.checkGuardB()) {

System.*out*.println("GuardB挂掉了,启动GuardB...."+*getStringDate*());

Process process = Runtime.*getRuntime*().exec(configure.getStartguardb());

*printMessage*(process.getInputStream());

*printMessage*(process.getErrorStream());

**int** value = process.waitFor();

System.*out*.println(value);

}

// 检测服务器存活

**if** (server.checkServer() <= 0) {

**boolean** isServerDown = **true**;

// trip check

**for** (**int** i = 0; i < 4; i++) {

// 如果服务是存活着

**if** (server.checkServer() > 0) {

isServerDown = **false**;

**break**;

}

}

**if** (isServerDown)

System.*out*.println("web服务器挂掉了,启动web"+*getStringDate*());

server.startServer(configure.getStartserver());

}

Thread.*sleep*(configure.getInterval());

}

}

/\*

\* 开两个线程分别去处理标准输出流和错误输出流

\* 处理缓冲区的信息，防止进程的输出信息量很大导致程序阻塞

\*/

**private** **static** **void** printMessage(**final** InputStream input) {

**new** Thread (**new** Runnable() {

@Override

**public** **void** run() {

Reader reader = **new** InputStreamReader(input);

BufferedReader bf = **new** BufferedReader(reader);

String line = **null**;

**try** {

**while**((line=bf.readLine())!=**null**){

System.*out*.println(line);

}

} **catch** (Exception e) {

e.printStackTrace();

}

}

}).start();

}

/\*\*

\* 获取现在时间

\*

\* **@return返回字符串格式** yyyy-MM-dd HH:mm:ss

\*/

**public** **static** String getStringDate() {

Date currentTime = **new** Date();

SimpleDateFormat formatter = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

String dateString = formatter.format(currentTime);

**return** dateString;

}

}

该类实现检测服务器进程和守护进程 B是否存活的功能，如果挂掉了，就启动他们。为了防止进程的输出信息量很大导致程序阻塞，printMessage开两个线程分别去处理标准输出流和错误输出流

4.类GuardServer

**public** **class** GuardServer {

**private** String servername;

**public** GuardServer(String servername) {

**this**.servername = servername;

}

**public** **void** startServer(String cmd) **throws** Exception {

System.*out*.println("Start Server : " + cmd);

Process process = Runtime.*getRuntime*().exec(cmd);

*printMessage*(process.getInputStream());

*printMessage*(process.getErrorStream());

**int** value = process.waitFor();

System.*out*.println(value);

Thread.*sleep*(10000);

}

/\*\*

\* 检测服务是否存在

\*

\* **@return** 返回配置的java程序的pid

\* **@return** pid >0 返回的是 pid <=0 代表指定java程序未运行

\* \*\*/

**public** **int** checkServer() **throws** Exception {

**int** pid = -1;

Process process = **null**;

BufferedReader reader = **null**;

process = Runtime.*getRuntime*().exec("jps -l");

reader = **new** BufferedReader(**new** InputStreamReader(process.getInputStream()));

String line;

**while** ((line = reader.readLine()) != **null**) {

String[] strings = line.split("\\s{1,}");

**if** (strings.length < 2)

**continue**;

**if** (strings[1].contains(servername)) {

pid = Integer.*parseInt*(strings[0]);

**break**;

}

}

reader.close();

process.destroy();

**return** pid;

}

/\*

\* 开两个线程分别去处理标准输出流和错误输出流

\* 处理缓冲区的信息，防止进程的输出信息量很大导致程序阻塞

\*/

**private** **static** **void** printMessage(**final** InputStream input) {

**new** Thread (**new** Runnable() {

@Override

**public** **void** run() {

Reader reader = **new** InputStreamReader(input);

BufferedReader bf = **new** BufferedReader(reader);

String line = **null**;

**try** {

**while**((line=bf.readLine())!=**null**){

System.*out*.println(line);

}

} **catch** (Exception e) {

e.printStackTrace();

}

}

}).start();

}

}

该类实现检测服务器进程是否存活以及启动服务器进程的方法。

5.类GuardB

**public** **class** GuardB {

//GuardB用于维持自己的锁

**private** File fileGuardB;

**private** FileOutputStream fileOutputStreamGuardB;

**private** FileChannel fileChannelGuardB;

**private** FileLock fileLockGuardB;

//GuardB用于检测A的锁

**private** File fileGuardA;

**private** FileOutputStream fileOutputStreamGrardA;

**private** FileChannel fileChannelGrardA;

**private** FileLock fileLockGrardA;

**public** GuardB() **throws** Exception{

fileGuardB = **new** File(Configure.*GUARD\_B\_LOCK*);

**if**(!fileGuardB.exists()){

fileGuardB.createNewFile();

}

//获取文件锁，拿不到证明GuardA已启动则退出

fileOutputStreamGuardB = **new** FileOutputStream(fileGuardB);

fileChannelGuardB = fileOutputStreamGuardB.getChannel();

fileLockGuardB = fileChannelGuardB.tryLock();

**if**(fileLockGuardB == **null**){

System.*exit*(0);

}

fileGuardA = **new** File(Configure.*GUARD\_A\_LOCK*);

**if**(!fileGuardA.exists()){

fileGuardA.createNewFile();

}

fileOutputStreamGrardA = **new** FileOutputStream(fileGuardA);

fileChannelGrardA = fileOutputStreamGrardA.getChannel();

}

/\*\*

\* 检测B是否存在

\*

\* **@return** true B已经存在

\*/

**public** **boolean** checkGuardA() {

**try** {

fileLockGrardA = fileChannelGrardA.tryLock();

**if**(fileLockGrardA == **null**){

**return** **true**;

} **else** {

fileLockGrardA.release();

**return** **false**;

}

} **catch** (Exception e) {

System.*exit*(0);

**return** **true**;

}

}

}

与类GuardA同理，实现在守护进程B启动时拿到B.txt的文件锁以及去检测是否拿到A.txt文件的文件锁的功能。

6.GuardB的配置类configre

**public** **class** Configure {

**public** **static** **final** String *GUARD\_B\_LOCK* = "C:\\java\\A.txt";

**public** **static** **final** String *GUARD\_A\_LOCK* = "C:\\java\\B.txt";

//睡眠时间

**public** **long** getInterval() {

// **TODO** Auto-generated method stub

**return** 5000;

}

**public** String getStartguarda() {

String file = "C:\\java\\DaemonA.bat";

String cmd = "cmd /c start "+file.replaceAll(" ", "\" \"");

**return** cmd;

}

}

该类仅定义了文件锁，，睡眠时间以及启动A的方法。

7.GuardB的主类GuardBMain

**public** **class** GuardBMain {

/\*\*

\* **@param** args

\*/

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

System.*out*.println("GuardB已启动...."+*getStringDate*());

GuardB guardB = **new** GuardB();

Configure configure = **new** Configure();

**while**(**true**){

//如果GuardA挂掉了，运行GuardA

**if**(!guardB.checkGuardA()){

System.*out*.println("GuardA挂掉了,启动GuardA...."+*getStringDate*());

Process process = Runtime.*getRuntime*().exec(configure.getStartguarda());

*printMessage*(process.getInputStream());

*printMessage*(process.getErrorStream());

**int** value = process.waitFor();

System.*out*.println(value);

}

Thread.*sleep*(configure.getInterval());

}

}

/\*

\* 开两个线程分别去处理标准输出流和错误输出流

\* 处理缓冲区的信息，防止进程的输出信息量很大导致程序阻塞

\*/

**private** **static** **void** printMessage(**final** InputStream input) {

**new** Thread (**new** Runnable() {

@Override

**public** **void** run() {

Reader reader = **new** InputStreamReader(input);

BufferedReader bf = **new** BufferedReader(reader);

String line = **null**;

**try** {

**while**((line=bf.readLine())!=**null**){

System.*out*.println(line);

}

} **catch** (Exception e) {

e.printStackTrace();

}

}

}).start();

}

/\*\*

\* 获取现在时间

\*

\* **@return返回字符串格式** yyyy-MM-dd HH:mm:ss

\*/

**public** **static** String getStringDate() {

Date currentTime = **new** Date();

SimpleDateFormat formatter = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

String dateString = formatter.format(currentTime);

**return** dateString;

}

}

该类实现了监测A是否运行以及启动A的方法。