Any way to "reboot" the JVM?

Asked 16 years, 1 month ago Modified 8 years, 4 months ago

Viewed 109k times

35

Is there any way to reboot the JVM? As in don't actually exit, but close and reload all classes, and run main from the top?

java jvm

Share

Improve this question

Follow

edited Feb 25, 2011 at 15:54

rob199

25 • 6

asked Nov 3, 2008 at 17:20

Alex

4,190 • 5 • 35 • 40

You might want to give a little more background. What are you trying to accomplish? – James Van Huis Nov 3, 2008 at 17:45

@HotLicks yes because we are all born experts on everything, right? – ununiform Dec 16, 2014 at 16:56

1 @ununiform - Because it requires fairly intimate knowledge of the JVM and it's interface with the OS to do it right. – Hot Licks Dec 16, 2014 at 17:01

dzone.com/articles/programmatically-restart-java
– Koekiebox Aug 21, 2017 at 10:41

7 Answers

Sorted by: Highest score (default)

18

Your best bet is probably to run the java interpreter within a loop, and just exit. For example:

```
#!/bin/sh
while true
do
    java MainClass
done
```

If you want the ability to reboot or shutdown entirely, you could test the exit status:

```
#!/bin/sh
STATUS=0
while [ $STATUS -eq 0 ]
do
    java MainClass
    STATUS=$?
done
```

Within the java program, you can use System.exit(0) to indicate that you want to "reboot," and System.exit(1) to indicate that you want to stop and stay stopped.

Share Improve this answer

Follow

answered Nov 3, 2008 at 18:12

Andru Luvisi

25.3k ● 6 ● 55 ● 66

7

IBM's JVM has a feature called "resettable" which allows you to effectively do what you are asking.

http://publib.boulder.ibm.com/infocenter/cicsts/v3r1/index.jsp?topic=/com.ibm.cics.ts31.doc/dfhpj/topics/dfhpje9.htm

Other than the IBM JVM, I don't think it is possible.

Share Improve this answer

Follow

answered Nov 3, 2008 at 18:08

DustinB

11.3k ● 5 ● 49 ● 54

2

Not a real "reboot" but:

You can build your own class loader and load all your classes (except a bootstrap) with it. Then, when you want to "reboot", make sure you do the following:

1. End any threads that you've opened and are using your classes.
2. Dispose any Window / Dialog / Applet you've created (UI application).
3. Close / dispose any other GC root / OS resources hungry peered resource (database connections, etc).
4. Throw away your customized class loader, create another instance of it and reload all the classes. You can probably optimize this step by pre-processing the classes from files so you won't have to access the codebase again.
5. Call your main point of entry.

This procedure is used (to some extent) while "hot-swapping" webapps in web servers.

Note though, static class members and JVM "global" objects (ones that are accessed by a GC root that isn't under your control) will stay. For example, Locale.setLocale() effects a static member on Locale. Since the Locale class is loaded by the system class loader, it will not be "restarted". That means that the old Locale object that was used in Locale.setLocale() will be available afterward if not explicitly cleaned.

Yet another route to take is instrumentation of classes. However, since I know little of it, I'm hesitant to offer advice.

[Explanation about hot deploy with some examples]

Share Improve this answer

Follow

answered Nov 4, 2008 at 5:41

Ran Biron

6,365 ● 7 ● 38 ● 70

1

If you're working in an application server, they typically come with built-in hot deployment mechanisms that'll reload all classes in your application (web app, enterprise app) when you redeploy it.

Otherwise, you'll have to look into commercial solutions. Java Rebel (http://www.zeroturnaround.com/javarebel/) is one such option.

Share Improve this answer

Follow

answered Nov 3, 2008 at 18:12

Jack Leow

22.5k ● 4 ● 53 ● 59

AFAIK there is no such way.

Notice that if there were a way to do that, it would highly depend on the current loaded code to properly release all held resources in order to provide a graceful restart (think about files, socket/tcp/http/database connections, threads, etc).

Some applications, like Jboss AS, capture Ctrl+C on the console and provide a graceful shutdown, closing all resources, but this is application-specific code and not a JVM feature.

Share Improve this answer

Follow

answered Nov 3, 2008 at 18:07

Miguel Ping

18.3k ● 23 ● 91 ● 137

0

I do something similar using JMX, I will 'unload' a module using JMX and then 'reload' it. Behind the scenes I am sure they are using a different class loader.

Share Improve this answer

Follow

answered Nov 3, 2008 at 20:28

Javamann

2,902 ● 2 ● 25 ● 22

0

Well, I currently have this, it works perfectly, and completely OS-independent. The only thing that must

0

work: executing the java process without any path/etc, but I think this can also be fixed.

The little code pieces are all from stackoverflow except RunnableWithObject and restartMinecraft() :)

You need to call it like this:

```
restartMinecraft(getCommandLineArgs());
```

So what it basically does, is:

1. Spawns a new Process and stores it in the p variable
2. Makes two RunnableWithObject instances and fills the process object into their data value, then starts two threads, they just print the inputStream and errorStream when it has available data until the process is exited
3. Waits for the process to exit
4. prints debug message about process exit
5. Terminates with the exit value of the process(not necessary)

And yes it is directly pulled from my minecraft project:)

The code:

Tools.isProcessExited() method:

```
public static boolean isProcessExited(Process p) {
    try {
```

```
            p.exitValue();
        } catch (IllegalThreadStateException e) {
            return false;
        }
        return true;
    }
```

Tools.restartMinecraft() method:

```
    public static void restartMinecraft(String args) t
InterruptedException {
//Here you can do shutdown code etc
        Process p = Runtime.getRuntime().exec(args);
        RunnableWithObject<Process> inputStreamPrinter
RunnableWithObject<Process>() {

            @Override
            public void run() {
                // TODO Auto-generated method stub
                while (!Tools.isProcessExited(data)) {
                    try {
                        while (data.getInputStream().a
                            System.out.print((char)
data.getInputStream().read());
                        }
                    } catch (IOException e) {
                    }
                }
            }
        };
        RunnableWithObject<Process> errorStreamPrinter
RunnableWithObject<Process>() {

            @Override
            public void run() {
                // TODO Auto-generated method stub
                while (!Tools.isProcessExited(data)) {
                    try {
                        while (data.getErrorStream().a
                            System.err.print((char)
data.getErrorStream().read());
                        }
```

```
                    } catch (IOException e) {
                    }
                }
            }
        };

        inputStreamPrinter.data = p;
        errorStreamPrinter.data = p;

        new Thread(inputStreamPrinter).start();
        new Thread(errorStreamPrinter).start();
        p.waitFor();
        System.out.println("Minecraft exited. (" + p.e
        System.exit(p.exitValue());
    }
```

Tools.getCommandLineArgs() method:

```
public static String getCommandLineArgs() {
    String cmdline = "";
    List<String> l = ManagementFactory.getRuntimeMXBea
    cmdline += "java ";
    for (int i = 0; i < l.size(); i++) {
        cmdline += l.get(i) + " ";
    }
    cmdline += "-cp " + System.getProperty("java.class
System.getProperty("sun.java.command");

    return cmdline;
}
```

Aaaaand finally the RunnableWithObject class:

```
package generic.minecraft.infinityclient;

public abstract class RunnableWithObject<T> implements
    public T data;
}
```

Good luck :)

Share Improve this answer

Follow

edited Jun 1, 2014 at 13:59

answered Jun 1, 2014 at 13:45

2xsaiko

1,071 ● 10 ● 15