

Во многих случаях у программистов возникает необходимость выполнить некоторые действия в момент, когда пользователь заканчивает работу с приложением...

Но проблема состоит в том, что пользователи не всегда пользуются рекомендованными и правильными способами выхода из приложения. Java предоставляет элегантный подход к выполнению какого-либо кода в середине процесса выгрузки процесса вашего приложения, таким образом гарантируя, что этот код, который, например, занимается какими-либо "очистительными" операциями, будет обязательно выполнен. Эта статья рассказывает о том, каким образом можно вешать обработчик прерывания работы приложения для гарантированного выполнения завершающего кода независимо от того, каким образом пользователь завершил работу с вашим приложением.

Очень часто бывает нужно выполнять какие-то операции по завершению приложения. Например, когда вы пишете текстовый редактор с использованием Swing, и это ваше приложение создает временный файл при начале своей работы. Временный файл должен быть удален, как только пользователь закроет ваше приложение. Если же вы пишете приложение, состоящее из множества сервлетов, встраиваемых в сервлет-контейнер (например, Tomcat или Jetty), то вы должны вызывать метод `destroy` для каждого из загруженных вами сервлетов до того, как завершится работа приложения.

Во многих случаях вы надеетесь на то, что пользователь закроет приложение приемлемым для вас способом. Например, в первом случае вы можете предоставить ему компонент JButton, после клика на который выполняются необходимые завершающие операции и осуществляется непосредственно выход из приложения. Как альтернативный вариант вы можете повесить обработчик события окна, который бы обрабатывал событие windowClosing. Tomcat же использует специальный batch-файл, который может быть выполнен при правильном завершении работы с приложением. Однако хорошо известно, что пользователи далеко не так часто корректно завершают работу с приложениями. Они могут делать с приложениями все что пожелаю. Помимо этого, пользователь может просто-напросто закрыть консоль или завершить свой сеанс работы с операционной системой, оставив при этом ваше приложение незакрытым.

В Java виртуальная машина завершает работу в двух случаях: во-первых, когда из приложения вышли нормальным способом, т.е. был вызван метод `System.exit`, или же

когда остался последний поток, не являющийся демоном. Во-вторых, когда пользователь внезапно прерывает работу виртуальной машины, например, нажимая комбинацию клавиш Ctrl+C или же выходя из системы, не закрыв предварительно работающее Java-приложение.

К счастью, виртуальная машина следует следующей двухфазной последовательности действий, прежде чем выгрузить себя:

1. Виртуальная машина запускает все зарегистрированные shutdown-ловушки, если таковые были установлены. Shutdown-ловушки - это нити (threads), которые регистрируются с помощью класса Runtime. Все эти ловушки будут запущены и будут работать параллельно до тех пор, пока все они не завершат своей работы.
2. Виртуальная машина вызывает все определенные finalize-операции (если есть подходящие).

В этой статье мы рассмотрим первый пункт, поскольку он позволяет программисту озадачить виртуальную Java-машину выполнением необходимых операций по завершению приложения. Shutdown-ловушки - это просто экземпляры классов-наследников класса Thread. Чтобы создать такую ловушку, нужно выполнить следующую последовательность действий:

1. Описать класс, наследующий класс Thread.
2. Осуществить реализацию метода run этого нового класса. Этот метод содержит код, который и будет выполняться для завершения работы виртуальной машины вне зависимости от того, нормально или нет было завершено приложение.
3. Связать класс shutdown-ловушки с вашим приложением.

4. Зарегистрировать ловушку с помощью метода addShutdownHook текущего экземпляра класса Runtime.

Как вы уже могли заметить, вам не нужно запускать только что созданную нить ловушки, как вы бы запускали другой класс, унаследовавший Thread. Забота о запуске этой нити ложится на виртуальную машину, которая, подойдя к выполнению своей shutdown-последовательности, запустит все зарегистрированные нити ловушек.

Код Листинга 1 представляет простой класс Shutdown HookDemo и подкласс класса Thread - ShutdownHook. Учтите, что метод run класса ShutdownHook просто выводит строку Shutting down на консоль. Конечно, вы можете вставить абсолютно любой код, который вам необходимо выполнить во время завершения вашего приложения.

После запуска public-класса вызывается метод start. Метод start создает shutdown-ловушку и регистрирует ее в текущем экземпляре Runtime-класса.

```
ShutdownHook shutdownHook = new ShutdownHook();
Runtime.getRuntime().addShutdownHook(shutdownHook);
```

После этого программа ждет нажатия пользователем клавиши Enter. System.in.read();

Когда пользователь нажимает Enter, осуществляется выход из программы. Однако перед выходом виртуальная машина запускает зарегистрированную shutdown-ловушку, которая в свою очередь печатает строчку "Shutting down".

Листинг 1

```
package test;
public class ShutdownHook Demo {
    public void start() {System.out.println("Demo");
                        ShutdownHook
    shutdown Hook = new ShutdownHook();
    Runtime.getRuntime(). addShutdownHook(shutdownHook);
}
public static void main (String[] args) {
    ShutdownHookDemo demo = new ShutdownHookDemo();
    demo.start();
```

```
try {
    System.in.read();
} catch (Exception e) {
    ;
}
}
}
}
}

class ShutdownHook extends Thread {
public void run() {
    System.out.println ("Shutting down");
}
}
```