Иногда возникает потребность в приостановке исполнения потоков. Но в любом случае приостановить исполнение потока совсем не трудно. Выполнение приостановленного потока может быть легко возобновлено.

Механизм временной или окончательной остановки потока исполнения, а также его возобновления отличался в ранних версиях jаvа, например, Jаvа 1.0, от современных версий, начиная с Java 2. До версии Java 2 методы suspend( ) и resume( ), определенные в классе Thread, использовались в программах для приостановки и возобновления потоков исполнения. На первый взгляд применение этих методов кажется вполне благоразумным и удобным подходом к управлению выполнением потоков. Тем не менее пользоваться ими в новых программах нa java не рекомендуется по следующей причине: метод suspend( ) из класса Thread несколько лет назад был объявлен не рекомендованным к употреблению, начиная с версии jаvа 2. Это было сделано потому, что иногда он способен порождать серьезные системные сбои. Допустим, что поток исполнения получил блокировки для очень важных структур данных. Если в этот момент приостановить исполнение данного потока, блокировки не будут сняты. Другие потоки исполнения, ожидающие эти ресурсы, могут оказаться взаимно блокированными. Метод resume( ) также не рекомендован к употреблению. И хотя его применение не вызовет особых осложнений, тем не менее им нельзя пользоваться без метода suspend( ) , который его дополняет. Метод stop( ) из класса Thread также объявлен устаревшим с версии Java 2. Это было сделано потому, что он может иногда послужить причиной серьезных системных сбоев. Допустим, поток выполняет запись в критически важную структуру данных и успел произвести лишь частичное ее обновление. Если его остановить в этот момент, структура данных может оказаться в поврежденном состоянии. Дело в том, что метод stop( ) вызывает снятие любой блокировки, устанавливаемой вызывающим потоком исполнения. Следовательно, поврежденные данные могут быть использованы в другом потоке исполнения, ожидающем по той же самой блокировке.

Если методы suspend( ), resume( ) или stop( ) нельзя использовать для управления потоками исполнения, то можно прийти к выводу, что теперь вообще нет никакого механизма для приостановки, возобновления или прерывания потока исполнения. К счастью, это не так. Вместо этого код управления выполнением потока должен быть составлен таким образом, чтобы метод run ( ) периодически проверял, должно ли исполнение потока быть приостановлено, возобновлено или прервано.

Ниже приведены общие формы их объявления.

final void resurne()

final void suspend()

final void stop()

Поток переходит в «пассивное» состояние (TERMINATED), если вызваны методы interrupt(), stop() (deprecated-метод) или метод run() завершил выполнение, и запустить его повторно уже невозможно. После этого, чтобы запустить поток, необходимо создать новый объект потока. Метод interrupt() успешно завершает поток, если он находится в состоянии «работоспособный». Если же поток неработоспособен, например, находится в состоянии TIMED\_WAITING, то метод инициирует исключение InterruptedException. Чтобы это не происходило, следует предварительно вызвать метод isInterrupted(), который проверит возможность завершения работы потока. При разработке не следует использовать методы принудительной остановки потока, так как возможны

проблемы с закрытием ресурсов и другими внешними объектами.

Когда метод вызывается то устанавливается флаг прерывания данного потока. Это состояние устанавливается с помощью признака типа Boolean у каждого потока

На данный момент в Java принят уведомительный порядок остановки потока (хотя JDK 1.0 и имеет несколько управляющих выполнением потока методов, например stop(), suspend() и resume() - в следующих версиях JDK все они были помечены как deprecated из-за потенциальных угроз взаимной блокировки).

Для корректной остановки потока можно использовать метод класса Thread - interrupt(). Этот метод выставляет некоторый внутренний флаг-статус прерывания. В дальнейшем состояние этого флага можно проверить с помощью метода isInterrupted() или Thread.interrupted() (для текущего потока). Метод interrupt() также способен вывести поток из состояния ожидания или спячки. Т.е. если у потока были вызваны методы sleep() или wait() – текущее состояние прервется и будет выброшено исключение InterruptedException. Флаг в этом случае не выставляется.

Схема действия при этом получается следующей:

* Реализовать поток.
* В потоке периодически проводить проверку статуса прерывания через вызов isInterrupted().
* Если состояние флага изменилось или было выброшено исключение во время ожидания/спячки, следовательно поток пытаются остановить извне.
* Принять решение – продолжить работу (если по каким-то причинам остановиться невозможно) или освободить заблокированные потоком ресурсы и закончить выполнение.

Возможная проблема, которая присутствует в этом подходе – блокировки на потоковом вводе-выводе. Если поток заблокирован на чтении данных - вызов interrupt() из этого состояния его не выведет. Решения тут различаются в зависимости от типа источника данных. Если чтение идет из файла – долговременная блокировка крайне маловероятна и тогда можно просто дождаться выхода из метода read(). Если же чтение каким-то образом связано с сетью – стоит использовать неблокирующий ввод-вывод из Java NIO.

Второй вариант реализации метода остановки (а также и приостановки) – сделать собственный аналог interrupt(). Т.е. объявить в классе потока флаги – на остановку и/или приостановку и выставлять их путем вызова заранее определённых методов извне. Методика действия при этом остаётся прежней – проверять установку флагов и принимать решения при их изменении. Недостатки такого подхода:

* Во-первых, потоки в состоянии ожидания таким способом не «оживить».
* Во-вторых, выставление флага одним потоком совсем не означает, что второй поток тут же его увидит. Для увеличения производительности виртуальная машина использует кеш данных потока, в результате чего обновление переменной у второго потока может произойти через неопределенный промежуток времени (хотя допустимым решением будет объявить переменную-флаг как volatile).

**В чем разница между interrupted() и isInterrupted()?**

Механизм прерывания работы потока в Java реализован с использованием внутреннего флага, известного как статус прерывания. Прерывание потока вызовом Thread.interrupt() устанавливает этот флаг. Методы Thread.interrupted() и isInterrupted() позволяют проверить, является ли поток прерванным.

Когда прерванный поток проверяет статус прерывания, вызывая статический метод Thread.interrupted(), статус прерывания сбрасывается.

Нестатический метод isInterrupted() используется одним потоком для проверки статуса прерывания у другого потока, не изменяя флаг прерывания.

Для наглядности протестируйте эти три варианта кода:

Пример Test1 бесконечный цикл

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

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

@Override

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

**while** (1 == 1) {

System.out.println(Thread.currentThread().isInterrupted());

Thread.currentThread().interrupt();

System.out.println(Thread.currentThread().isInterrupted())

}

}

}).start();

}

Не проверяем в цикле interrupted поток или нет (а он таковым становится при первом же шаге цикла, но поток продолжит работать бесконечно)

Пример Test2 проверка флага прерывания

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

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

@Override

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

**while** (!Thread.currentThread().isInterrupted()) {

System.out.println(Thread.currentThread().isInterrupted());

Thread.currentThread().interrupt();

System.out.println(Thread.currentThread().isInterrupted());

}

}

}).start();

}

Здесь interrupted - условие цикла, соответственно как только поток становится interrupted цикл завершается, run тоже, поток заканчивает работу.

Пример Test3 проверка флага прерывания с его сбросом в цикле методом interrupted()

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

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

@Override

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

**while** (!Thread.currentThread().isInterrupted()) {

System.out.println(Thread.currentThread().isInterrupted());

Thread.currentThread().interrupt();

// изменение состояния флага

System.out.println(Thread.currentThread().interrupted());

}

}

}).start();

}

Хотя здесь interrupted - условие цикла, соответственно, как только поток становится interrupted цикл должен завершится, однако метод , interrupted() при проверке соcтаяния меняет флаг прерывания на false, поток заканчивает работу.