Threads

---

1. Организация связи между потоками: назначение и работа с методами wait, notify.

Представим ситуацию, что у нас многопоточное приложение. Есть некий класс, выполняющий отправку данных по почте в одном потоке, и есть класс, который эти данные подготавливаются в другом потоке. Перед отправкой данных необходимо как-то связаться с потоком, подготавливающим данные, чтобы узнать, когда данные будут готовы к отправке. Можно, конечно использовать флаг и цикл while, однако это слишком дорого - теряется время CPU.

Для реализации этой задачи Java содержит механизм связи через методы wait(), notify() и notifyAll(). Они реализованы в классе Object, поэтому доступны всем классам.

wait ()	останавливает выполнение текущего потока, переводит его в состояние WAITING и освобождает от блокировки захваченный объект. Вернуть поток в состояние RUNNABLE можно вызовом notify() или notifyAll() из другого потока, либо это произойдет автоматически по истечении срока ожидания, если в режим ожидания он был переведен методом wait(long timeout)
notify ()	возвращает поток, для которого ранее был вызван wait(), в состояние RUNNABLE
notifyAll()	возвращает все потоки для которых ранее был вызван wait(), в состояние RUNNABLE. Первым начнет выполняться самый высокоприоритетный поток.

Методы wait(), notify() и notifyAll() должны обязательно находиться внутри блока synchronized, либо внутри synchronized-метода, иначе будет сгенерирован Exception.

```
public class DataManager {
    private static final Object monitor = new Object();

    public void sendData() {
        synchronized (monitor) {
            System.out.println("Waiting for data...");
            try {
                monitor.wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            // continue execution and sending data
            System.out.println("Sending data...");
        }
    }

    public void prepareData() {
        synchronized (monitor) {
            System.out.println("Data prepared");
            monitor.notifyAll();
        }
    }
}
```

2. Что такое “главный поток”? Как получить ссылку на главный поток выполнения программы?

При запуске программы автоматически создается главный поток — поток, который выполняет метод main(), то есть главный метод программы. В рамках главного потока могут создаваться (запускаться) дочерние потоки (подпотоки), в которых, в свою очередь также могут запускаться потоки, и так далее. Главный поток от прочих потоков отличается тем, что создается первым. Главный поток, как правило, является и последним потоком, завершающим выполнение программы.

Несмотря на то что главный поток исполнения создается автоматически при запуске программы, им можно управлять через объект класса Thread. Для этого достаточно получить ссылку на него, вызвав метод currentThread(). Этот метод возвращает ссылку на тот поток исполнения, из которого он был вызван. Если этот метод вызывать в главном методе программы (инструкция вида Thread.currentThread()), получим ссылку на главный поток. Получив ссылку на главный поток, можно управлять им таким же образом, как и любым другим потоком исполнения.

3. Методы isAlive() и getState() класса Thread

public final boolean isAlive()	возвращает логическое значение true, если поток, для которого он вызван, еще не имеет статус TERMINATED
public Thread.State getState()	возвращает текущее состояние потока

Поток может находиться в одном из состояний, соответствующих элементам статически вложенного перечисления Thread.State:
NEW — поток создан, но еще не запущен;
RUNNABLE — поток выполняется;
BLOCKED — поток блокирован;
WAITING — поток ждет окончания работы другого потока;
TIMED_WAITING — поток некоторое время ждет окончания другого потока;
TERMINATED — поток завершен.

4. Потоки демоны: назначение, создание, случаи применения.

Потоки-демоны используются для работы в фоновом режиме вместе с программой, но не являются неотъемлемой частью логики программы. Примером такого потока может служить сборщик мусора. Если какой-либо процесс может выполняться на фоне работы основных потоков выполнения и его деятельность заключается в обслуживании основных потоков приложения, то такой процесс может быть запущен как поток-демон. С помощью метода setDaemon(boolean value), вызванного вновь созданным потоком до его запуска, можно определить поток-демон. Метод boolean isDaemon() позволяет определить, является ли указанный поток демоном или нет.

Базовое свойство потоков-демонов заключается в возможности основного потока приложения завершить выполнение потока-демона (в отличие от обычных потоков) с окончанием кода метода main(), не обращая внимания на то, что поток-демон еще работает.

5. Что такое синхронизация, понятие монитора

Когда несколько потоков нуждаются в доступе к разделяемому ресурсу, им необходим некоторый способ гарантии того, что ресурс будет использоваться одновременно только одним потоком. Процесс, с помощью которого это достигается, называется синхронизацией. Синхронизация в Java гарантирует, что никакие два потока не

смогут выполнить синхронизированный метод одновременно или параллельно. Базовая синхронизация в Java возможна при использовании синхронизированных методов и синхронизированных блоков с использованием ключевого слова synchronized. Когда какой либо поток входит в синхронизированный метод или блок, он приобретает блокировку, и всякий раз, когда поток выходит из синхронизированного метода или блока, JVM снимает блокировку. Если синхронизированный метод вызывает другой синхронизированный метод, который требует такой же замок, то текущий поток, который держит замок может войти в этот метод не приобретая замок.

Ключом к синхронизации является концепция монитора (также называемая семафором). Монитор — это объект, который используется для взаимоисключающей блокировки, его также называют mutex. Только один поток может захватить и держать монитор в заданный момент. Когда поток получает блокировку, говорят, что он вошел в монитор. Все другие потоки пытающиеся войти блокированный монитор, будут приостановлены, пока первый не вышел из монитора. Говорят, что другие потоки ожидают монитор. При желании поток, владеющий монитором, может повторно захватить тот же самый монитор. В Java каждый объект имеет свой собственный монитор. Статический метод захватывает монитор экземпляра класса Class, того класса, на котором он вызван. Он существует в единственном экземпляре. Нестатический метод захватывает монитор экземпляра класса, на котором он вызван.

---

6. Опишите жизненный цикл потока

Поток может находиться в одном из состояний, соответствующих элементам статически вложенного перечисления Thread.State:
NEW — поток создан, но еще не запущен;
RUNNABLE — поток выполняется;
BLOCKED — поток блокирован;
WAITING — поток ждет окончания работы другого потока;
TIMED_WAITING — поток некоторое время ждет окончания другого потока;
TERMINATED — поток завершен.
При создании потока он получает состояние «новый» (**NEW**) и не выполняется.

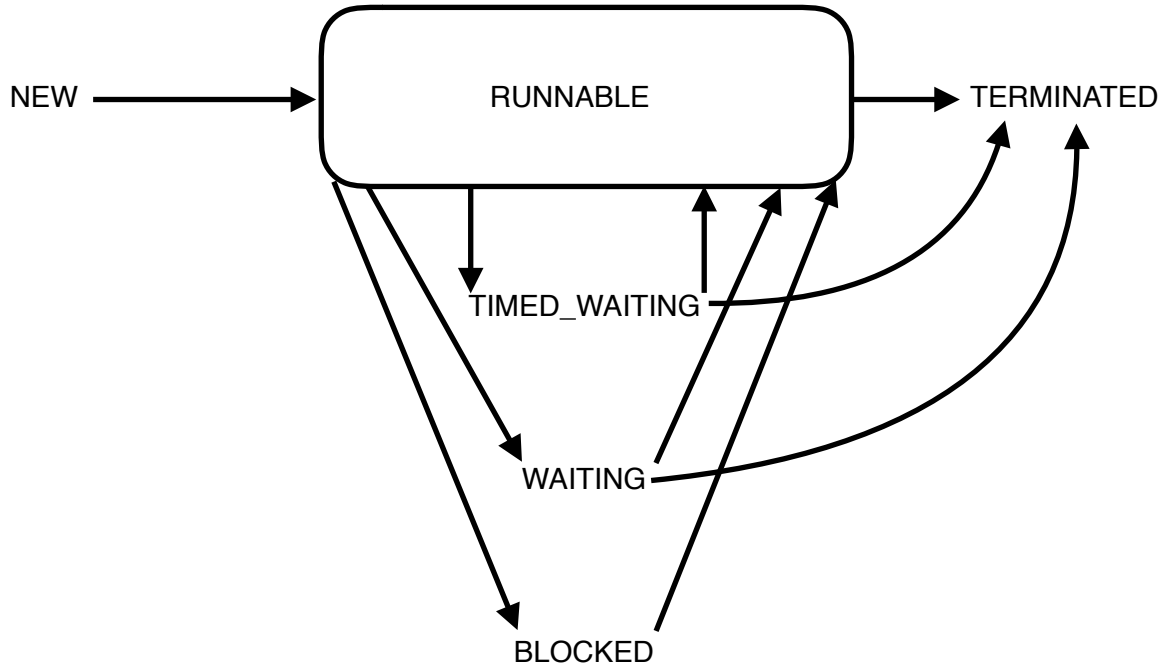
Для перевода потока из состояния «новый» в состояние «работоспособный» (**RUNNABLE**) следует выполнить метод start(), который вызывает метод run() — основной метод потока.

Поток переходит в состояние «неработоспособный» в режиме ожидания (**WAITING**) вызовом методов join(), wait(), suspend() (deprecated-метод) или методов ввода/вывода, которые предполагают задержку. Для задержки потока на некоторое время (в миллисекундах) можно перевести его в режим ожидания по времени (**TIMED_WAITING**) с помощью методов yield(), sleep(long millis), join(long timeout) и wait(long timeout), при выполнении которых может генерироваться прерывание InterruptedException. Вернуть потоку работоспособность после вызова метода suspend() можно методом resume() (deprecated метод), а после вызова метода wait() — методами notify() или notifyAll().

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.

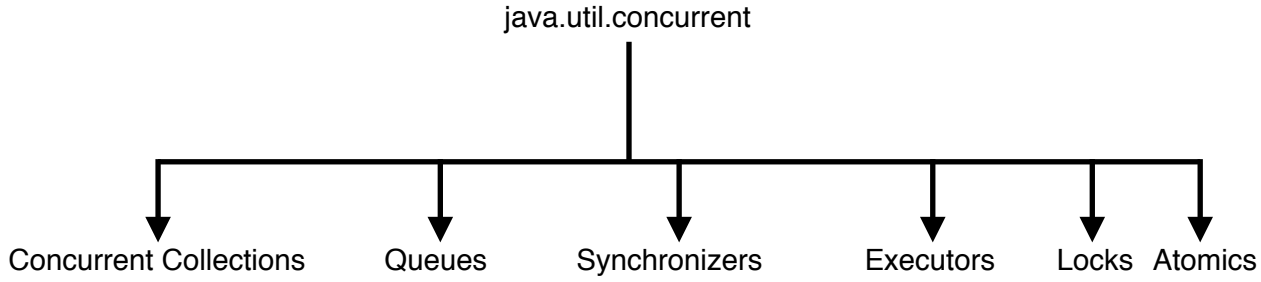
Методы suspend(), resume() и stop() являются deprecated-методами и запрещены к использованию, так как они не являются в полной мере «потокобезопасными».

Получить текущее значение состояния потока можно вызовом метода getState().



7. Классы синхронизированных коллекций java.util.concurrent

В версии Java 5 добавлен пакет java.util.concurrent, классы которого обеспечивают высокую производительность при построении потокобезопасных приложений.



Concurrent Collections - набор коллекций, которые работает намного эффективней в многопоточной среде нежели стандартные коллекции из java.util пакета. Вместо блокирования доступа ко всей коллекции используются блокировки по сегментам данных или же оптимизируется работа для параллельного чтения данных по wait-free алгоритмам. Некоторые из них:

ConcurrentHashMap	аналог Hashtable, данные представлены в виде сегментов, доступ блокируется по сегментам
ConcurrentLinkedQueue	аналог LinkedList
CopyOnWriteArrayList	аналог ArrayList
CopyOnWriteArraySet	имплементация интерфейса Set, за основу взят CopyOnWriteArrayList
ConcurrentSkipListMap	аналог TreeMap
ConcurrentSkipListSet	аналог TreeSet

Queues - неблокирующие и блокирующие очереди с поддержкой многопоточности: ConcurrentLinkedQueue, ConcurrentLinkedDeque, BlockingQueue, ArrayBlockingQueue, BlockingDeque и др.

Synchronizers - вспомогательные утилиты для синхронизации потоков:

Semaphore	предлагает потоку ожидать завершения действий в других потоках

Exchanger	обмен объектами между двумя потоками

Executors - содержит в себе фрейморки для создания пулов потоков, планирования работы асинхронных задач с получением результатов:

Future	интерфейс для получения результатов работы асинхронной операции
ExecutorService	интерфейс, который описывает сервис для запуска Runnable или Callable задач
Executor	организует запуск пула потоков и службы из планирования

Locks - представляет собой альтернативные и более гибкие механизмы синхронизации потоков по сравнению с базовыми synchronized, wait, notify, notifyAll: Lock, ReadWriteLock.

Atomics - классы с поддержкой атомарных операций над примитивами и ссылками.

---

8. Метод join() класса Thread

В Java существует механизм, который позволяет одному потоку ожидать завершения выполнения другого потока. Этот механизм "включается" при помощи метода join(). Вызывающий поток дожидается, пока указанный поток присоединится к нему. Для того, чтобы заставить главный поток дождаться завершения работы побочного потока необходимо в главном потоке вызвать этот метод для побочного потока:

```
newThread.join();
```

Как только поток newThread закончит свою работу, он присоединится к главному потоку, и главный поток сможет продолжить выполнение. Если поток будет прерван при помощи метода interrupt(), будет выброшен InterruptedException.

Существуют дополнительные формы метода join(), которые позволяют указывать максимальный промежуток времени, в течение которого требуется ожидать завершения указанного потока исполнения:

```
public final void join(long millis) throws InterruptedException
public final void join(long millis, int nanos) throws InterruptedException
```