Многопоточность в Java, процессы и потоки, жизненный цикл потока

Многопоточность в Java — это возможность выполнения нескольких потоков (threads) в рамках одного процесса для параллельного выполнения задач.

Процесс — это экземпляр запущенной программы, который имеет своё адресное пространство, системные ресурсы и отдельный набор переменных.

Поток (thread) — это более лёгкий компонент процесса. Все потоки одного процесса имеют общее адресное пространство и могут совместно использовать ресурсы процесса.

Жизненный цикл потока включает следующие состояния:

1. New (новый): Поток создан, но ещё не запущен.

2. Runnable (готовый к выполнению): Поток готов к выполнению, но в данный момент может не выполняться.

3. Running (выполняющийся): Поток в данный момент выполняется.

4. Blocked (заблокирован): Поток заблокирован и ожидает освобождения ресурса.

5. Waiting (ожидающий): Поток ожидает другого потока для выполнения определённого действия.

6. Timed Waiting (ожидание с таймаутом): Поток ожидает в течение определённого времени.

7. Terminated (завершён): Поток завершил выполнение.

Race Condition и Deadlock

Race Condition — это состояние, когда результат работы программы зависит от порядка выполнения потоков, что может привести к некорректным данным.

Deadlock (взаимная блокировка) — это ситуация, когда два или более потоков ждут друг друга для освобождения ресурсов, что приводит к бесконечному ожиданию.

Monitor, Mutex и Semaphore

Мьютекс - обеспечивает чтобы доступ к объекту в определенное время был только у одного потока. (Семафор со счетчиком 1)

Монитор - В блоке кода, который помечен словом synchronized, доступ к коду был только у одного потока.

Семафор - какмьютекс, но используется счетчик потоков, и доступ к ресурсу могут получить сразу несколько из них.

Atomic Action

Atomic Action — это операция, которая выполняется полностью или не выполняется вовсе, без промежуточных состояний. В Java это можно реализовать с помощью класса AtomicInteger, AtomicLong и других атомарных классов из пакета java.util.concurrent.atomic.

Создание потока

Способы создания потока

1. Реализация интерфейса Runnable

В этом подходе вы создаете класс, который реализует интерфейс Runnable. Интерфейс Runnable содержит один метод run(), который вы должны реализовать. Этот метод содержит код, который будет выполняться в новом потоке. Затем вы создаете объект Thread, передавая ему объект Runnable, и вызываете метод start() для запуска нового потока.

2. Наследование от класса Thread

Другой способ создания потока — это наследование от класса Thread. Вы создаете класс, который наследуется от Thread, и переопределяете его метод run(), где размещаете код для выполнения в новом потоке. После этого вы создаете экземпляр вашего класса и вызываете метод start(), чтобы запустить поток.

3. Лямбда-выражения (с Java 8)

Если вы используете Java 8 или выше, вы можете создать поток с помощью лямбда-выражений, поскольку Runnable — это функциональный интерфейс. Вы создаете объект Thread и передаете лямбда-выражение, которое реализует метод run(). Затем вы вызываете start(), чтобы запустить поток.

Запуск и управление потоком

После создания потока необходимо запустить его. Для этого используется метод start(), который вызывает метод run() в новом потоке. Важно не путать методы start() и run() — вызов run() напрямую не создаёт новый поток, он просто выполняет метод в текущем потоке.

Volatile, Synchronized, Join, Wait, Notify, NotifyAll, Sleep, Yield

- volatile: Ключевое слово, которое гарантирует, что значение переменной будет читаться из основной памяти и запись в неё будет происходить напрямую, избегая кэширования.

- synchronized: Ключевое слово для методов и блоков, обеспечивающее, что только один поток может выполнить синхронизированный блок кода в определённый момент времени.

- join(): Метод, который позволяет одному потоку дождаться завершения другого потока.

- wait(): Метод заставляет поток ждать, пока другой поток не вызовет notify() или notifyAll() на том же объекте.

- notify(): Метод, который пробуждает один из потоков, ожидающих на данном объекте.

- notifyAll(): Метод, который пробуждает все потоки, ожидающие на данном объекте.

- sleep(): Метод, который приостанавливает выполнение потока на определённое время.

- yield(): Метод, который позволяет текущему потоку уступить выполнение другим потокам, если они готовы к выполнению.

- start(): Запускает поток, вызывая метод run() в новом потоке.

Нагрузочное тестирование, ELK стек

Нагрузочное тестирование — это процесс тестирования системы или компонента под высокой нагрузкой для проверки её поведения и производительности.

ELK стек — это набор инструментов для централизованного логирования и анализа:

Elasticsearch: Поисковая система для хранения и индексирования данных.

Logstash: Инструмент для сбора, обработки и передачи логов.

Kibana: Веб-интерфейс для визуализации данных из Elasticsearch.

JMeter и Selenium

JMeter — это инструмент для нагрузочного тестирования веб-приложений. Он позволяет моделировать различные сценарии пользователей и измерять производительность.

Selenium — это набор инструментов для автоматизации веб-браузеров, используемый для автоматизированного тестирования веб-приложений.