INTERVIEW JAVA DEVELOPER MULTITHREADING

Ответы на вопросы на собеседование Multithreading (часть 2).

Vasyl K 7:26:00 2 Комментарии

- **ЧТО ТАКОЕ THREADLOCAL ПЕРЕМЕННАЯ?**

ThreadLocal переменные – специальный вид переменных, доступных Java программисту. Так же, как для состояний есть переменная состояния, для нитей есть ThreadLocal переменные. Это неплохой способ достичь ните-безопасности для затратных-для-создания объектов, например вы можете сделать SimpleDateFormat ните-безопасным, используя ThreadLocal. Так как это затратный класс, его нежелательно использовать в локальной области, которая требует отдельных экземпляров на каждый вызов. Предоставляя каждой нити её собственную копию, вы убиваете двух зайцев. Во-первых, вы уменьшаете количество экземпляров затратных объектов, используя по новой фиксированное количество экземпляров, и во-вторых, вы достигаете ните-безопасности, без потерь синхронизации и неизменяемости. Ещё один хороший пример локальной переменной у нити – класс ThreadLocalRandom, который уменьшает количество экземпляров затратных-для-создания объектов Random в много-нитиевой среде.

- **ЧТО ТАКОЕ FUTURETASK?**

FutureTask представляет собой отменяемое асинхронное вычисление в параллельном Java приложении. Этот класс предоставляет базовую реализацию Future, с методами для запуска и остановки вычисления, методами для запроса состояния вычисления и извлечения результатов. Результат может быть получен только когда вычисление завершено, метод получения будет заблокирован, если вычисление ещё не завершено. Объекты FutureTask могут быть использованы для обёртки объектов Callable и Runnable. Так как FutureTask реализует Runnable, его можно передать Executor'у на выполнение.

- **РАЗЛИЧИЕ МЕЖДУ INTERRUPTED И ISINTERRUPTED?**

Основное различие между interrupted() и isInterrupted() в том, что первый сбрасывает статус прерывания, а второй нет. Механизм прерывания в Java реализован с использованием внутреннего флага, известного как статус прерывания. Прерывание нити вызовом Thread.interrupt() устанавливает этот флаг. Когда прерванная нить проверяет статус прерывания, вызывая статический метод Thread.interrupted(), статус прерывания сбрасывается. Нестатический метод isInterrupted(), который используется нитью для проверки статуса прерывания у другой нити, не изменяет флаг прерывания. Условно, любой метод, который завершается, выкинув InterruptedException сбрасывает при этом флаг прерывания. Однако, всегда существует возможность того, что флаг тут же снова установится, если другая нить вызовет interrupt().

- **ПОЧЕМУ МЕТОДЫ WAIT И NOTIFY ВЫЗЫВАЮТСЯ В СИНХРОНИЗИРОВАННОМ БЛОКЕ?**

Основная причина вызова wait и notify из статического блока или метода в том, что Java API обязательно требует этого. Если вы вызовете их не из синхронизированного блока, ваш код выбросит IllegalMonitorStateException. Более хитрая причина в том, чтобы избежать состояния гонки между вызовами wait и notify.

- **ЧТО ТАКОЕ ПУЛ НИТЕЙ?**

Создание нити затратно в плане времени и ресурсов. Если вы создаёте нить во время обработки запроса, это замедлит время отклика, также процесс может создать только ограниченное число нитей. Чтобы избежать этих проблем, во время запуска приложения создаётся пул нитей и нити повторно используются для обработки запросов. Этот пул нитей называется "thread pool", а нити в нём – рабочая нить. Начиная с Java 1.5 Java API предоставляет фреймворк Executor, который позволяет вам создавать различные пулы нитей, например single thread pool, который обрабатывает только одно задание за единицу времени, fixed thread pool, пул с фиксированным количеством нитей, и cached thread pool, расширяемый пул, подходящий для приложений с множеством недолгих заданий.

- **РАЗЛИЧИЯ МЕЖДУ LIVELOCK И DEADLOCK?**

Livelock схож с deadlock, только в livelock состояния нитей или вовлечённых процессов постоянно изменяются в зависимости друг от друга. Livelock – особый случай нехватки ресурсов. Реальный пример livelock'а – когда два человека

встречаются в узком коридоре и каждый, пытаясь быть вежливым, отходит в сторону, и так они бесконечно двигаются из стороны в сторону.

- **КАК ПРОВЕРИТЬ, УДЕРЖИВАЕТ ЛИ НИТЬ LOCK?**

Я и не подозревал, что можно проверять, удерживает ли нить lock в данный момент, до тех пор, пока не столкнулся с этим вопросом в одном телефонном интервью. В java.lang.Thread есть метод holdsLock(), он возвращает true, тогда и только тогда, когда текущая нить удерживает монитор у определённого объекта.

- **КАК ПОЛУЧИТЬ ДАМП НИТИ?**

Дамп нити позволяет узнать, чем нить занимается в данный момент. Существует несколько способов получения дампа нити, зависящих от операционной системы. В Windows вы можете использовать комбинацию ctrl + Break, в Linux – команду kill -3. Также вы можете воспользоваться утилитой jstack, она оперирует над id процесса, который вы можете узнать с помощью другой утилиты jps.

- **КАКОЙ JVM ПАРАМЕТР ИСПОЛЬЗУЕТСЯ ДЛЯ КОНТРОЛЯ РАЗМЕРА СТЕКА НИТИ?**

Это один из простых, -Xss параметр используется для контроля размера стека нити в Java.

- **РАЗЛИЧИЯ МЕЖДУ SYNCHRONIZED И REENTRANTLOCK?**

Были времена, когда единственный способ достижения взаимного исключения был через ключевое слово synchronized, но он имеет несколько недостатков, например нельзя расширить lock за пределами метода или блока кода и т.д. Java 5 решает эту проблему, предоставляя более утончённый контроль через интерфейс Lock. ReentrantLock – распространённая реализация Lock, которая предоставляет Lock с таким же базовым поведением и семантикой, как у неявного монитора, достигаемый использованием синхронизированных методов, но с расширенными возможностями.

- **ЧТО ТАКОЕ SEMAPHORE?**

Semaphore – это новый тип синхронизатора. Это семафор со счётчиком. Концептуально, семафор управляет набором разрешений. Каждый acquire() блокируется, если необходимо, до того, как разрешение доступно, затем получает его. Каждый release() добавляет разрешение, потенциально освобождая блокирующий

получатель (acquirer). Однако при этом не используются фактические объекты разрешений; Semaphore просто хранит количество доступных и действует соответственно. Semaphore используется для защиты дорогих ресурсов, которые доступны в ограниченном количестве, например подключение к базе данных в пуле.

- **ЧТО БУДЕТ, ЕСЛИ ОЧЕРЕДЬ ПУЛА НИТЕЙ УЖЕ ЗАПОЛНЕНА, А ВЫ ПОДАДИТЕ ЗАДАЧУ?**

Если очередь пула нитей заполнилась, то поданная задача будет "отклонена". Метод submit() у ThreadPoolExecutor'а выкидывает RejectedExecutionException, после чего вызывается RejectedExecutionHandler.

- **РАЗЛИЧИЯ МЕЖДУ МЕТОДАМИ SUBMIT() И EXECUTE() У ПУЛА НИТЕЙ?**

Оба метода являются способами подачи задачи в пул нитей, но между ними есть небольшая разница. Execute(Runnable command) определён в интерфейсе Executor и выполняет поданную задачу в будущем, но, что более важно, ничего не возвращает. С другой стороны submit() – перегруженный метод, он может принимать задачи типов Runnable и Callable и может возвращать объект Future, который можно использовать для отмены выполнения и/или ожидания результата вычислений. Этот метод определён в интерфейсе ExecutorService, который наследуется от интерфейса Executor, и каждый класс пула нитей, например ThreadPoolExecutor или ScheduledThreadPoolExecutor, наследует эти методы.

- **ЧТО ТАКОЕ БЛОКИРУЮЩИЙ МЕТОД?**

Блокирующий метод – метод, который блокируется, до тех пор, пока не выполнится задание, например метод accept() у ServerSocket блокируется в ожидании подключения клиента. Здесь блокирование означает, что контроль не вернётся к вызывающему методу до тех пор, пока не выполнится задание. С другой стороны, существуют асинхронные или не блокирующиеся методы, которые завершаются до выполнения задачи.

- **ЧТО ТАКОЕ READWRITELOCK?**

В целом, ReadWriteLock – это результат техники разбора lock'а для улучшения производительности параллельных приложений. Это интерфейс, который был добавлен в Java 5. Он оперирует парой связанных lock'ов, один для операций чтения, один для записи. Читающий lock может удерживаться одновременно несколькими читающими нитями, до тех пор пока не будет записывающих. Записывающий lock

эксклюзивен. Если хотите, вы можете реализовать интерфейс с вашим набором правил, или вы можете использовать ReentrantReadWriteLock, который поддерживает максимум 65535 рекурсивных записывающих lock'ов и 65535 читающих lock'ов.

- **ЧТО ТАКОЕ DOUBLE CHECKED LOCKING СИНГЛТОНА?**

Это старый способ создания нитe-безопасного синглтона, который пытается оптимизировать производительность, блокируясь только когда экземпляр синглтона создаётся впервые.

- **ЧТО ТАКОЕ ФРЕЙМВОРК FORK/JOIN?**

Фреймворк Fork/Join, представленный в JDK 7, – это мощная утилита, позволяющая разработчику пользоваться преимуществами нескольких процессоров у современных серверов. Он разработан для работы, которую можно рекурсивно разбить на маленькие частицы. Цель – использовать всю доступную вычислительную мощь, для увеличения производительности вашего приложения. Одного значительное преимущество этого фреймворка в том, что он использует work-stealing алгоритм (от work – работа и steal – красть). Рабочие нити, у которых закончились свои задания, могут "своровать" задания у других нитей, которые всё ещё заняты.