Библиотека Java Concurrency

JSP & Servlets
19 уровень, 0 лекция

ОТКРЫТА

Многопоточность в Java

Java Virtual Machine поддерживает **параллельные вычисления**. Все вычисления могут быть выполнены в контексте одного или нескольких потоков. Мы легко можем настроить доступ к одному ресурсу или объекту для нескольких потоков, а также настроить поток на выполнения отдельного блока кода.

Любому разработчику необходимо синхронизировать работу с потоками при операциях чтения и записи для ресурсов, на которые выделены несколько потоков.

Это важно, чтобы на момент обращения к ресурсу у тебя были актуальные данные, чтобы другой поток мог изменить их и ты получил самую обновленную информацию. Даже если взять пример банковского счета, пока на него не пришли деньги, пользоваться ими ты не можешь, поэтому важно всегда иметь актуальные данные. В Java есть специальные классы для синхронизации потоков и управления ими.

Объекты потока

Все начинается с главного (основного) потока, то есть минимально в твоей программе уже есть один выполняемый поток. Основной поток может создавать другие потоки с помощью **Callable** или **Runnable**. Создание отличается только возвращаемым результатом, **Runnable** не возвращает результата и не может выбросить проверяемое исключение. Поэтому у тебя получается хорошая возможность построить эффективную работу с файлами, но это очень опасно и нужно быть аккуратным.

Также есть возможность планировать выполнения потока на отдельном ядре центрального процессора. Система может легко перемещаться между потоками и выполнять определенный поток при правильных настройках: то есть выполняется сначала поток, который читает данные, как только у нас появились данные, далее мы передаем их потоку, который отвечает за валидацию, после этого передаем потоку для выполнения какой-то бизнес-логики и новым потоком записываем их обратно. В такой ситуации 4 потока поочередно обрабатывают данные и все будет работать быстрее, чем один поток. Каждый такой поток преобразуется в нативный поток ОС, а вот, каким способом его будут преобразовывать, зависит от реализации JVM.

Класс `Thread` служит для создания потоков и работы с ними. В нем есть стандартные механизмы управления, так и абстрактные, например, классы и коллекции из **java.util.concurrent**.

Синхронизация потоков в Java

Коммуникация обеспечивается за счет разделения доступа к объектам. Это весьма эффективно, но в то же время очень легко допустить ошибку при работе. Ошибки бывают двух случаев: thread interference — когда другой поток вмешивается в твой поток, и memory consistency errors — консистентности памяти. Для решения и предотвращения этих ошибок у нас есть разные методы синхронизации.

Синхронизацией потоков в Java занимаются мониторы, — это высокоуровневый механизм, позволяющий единовременно только одному потоку выполнять блок кода, защищённый этим же монитором. Поведение мониторов рассмотрено в терминах блокировок; один монитор — одна блокировка.

Синхронизация имеет несколько важных моментов, на которых нужно обратить внимание. Первый момент — это взаимное исключение (mutual exclusion) — только один поток может владеть монитором, таким образом, синхронизация на мониторе подразумевает, что как только один поток входит в synchronized-блок, защищённый монитором, никакой другой поток не может войти в блок, защищенный этим монитором, пока первый поток не выйдет из synchronized-блока. То есть несколько потоков не могут обратится в один блок synchronized одновременно.

Но синхронизация — это не только взаимное исключение. Синхронизация гарантирует, что данные, записанные в память до или внутри синхронизированного блока, становятся видимыми для других потоков, которые синхронизируются на том же мониторе. После выхода из блока мы освобождаем монитор и другой поток может захватить его и начать выполнения этого блока кода.

Когда новый поток захватывает монитор, мы получаем доступ и возможность к исполнению этого блока кода, и в этот момент времени переменные будут загружены из основной памяти. Тогда мы сможем увидеть все записи, сделанные видимым предыдущим освобождением (release) монитора.

Чтение-запись в поле — это атомарная операция, если поле объявлено volatile, либо защищено уникальной блокировкой, получаемой перед любым чтением-записью. Но если ты все-таки столкнулся с ошибкой, то получаешь ошибку о переупорядочивании (изменение порядка следования, reordering). Она проявляется в некорректно синхронизированных многопоточных программах, где один поток может наблюдать эффекты, которые производятся другими потоками.

Эффект взаимного исключения и синхронизации потоков, то есть их корректная работа достигается только вхождением в synchronized-блок или метод, неявно получающий блокировку, или получением блокировки явным образом. Мы поговорим об этом ниже. Оба способа работы влияют на твою память и важно не забывать о работе с volatile-переменными.

Volatile поля в Java

Если переменная помечена, как volatile, она доступна глобально. Это значит то, что если поток обращается к volatile переменной, то получит его значение перед тем, чтобы использовать значение из кэша.

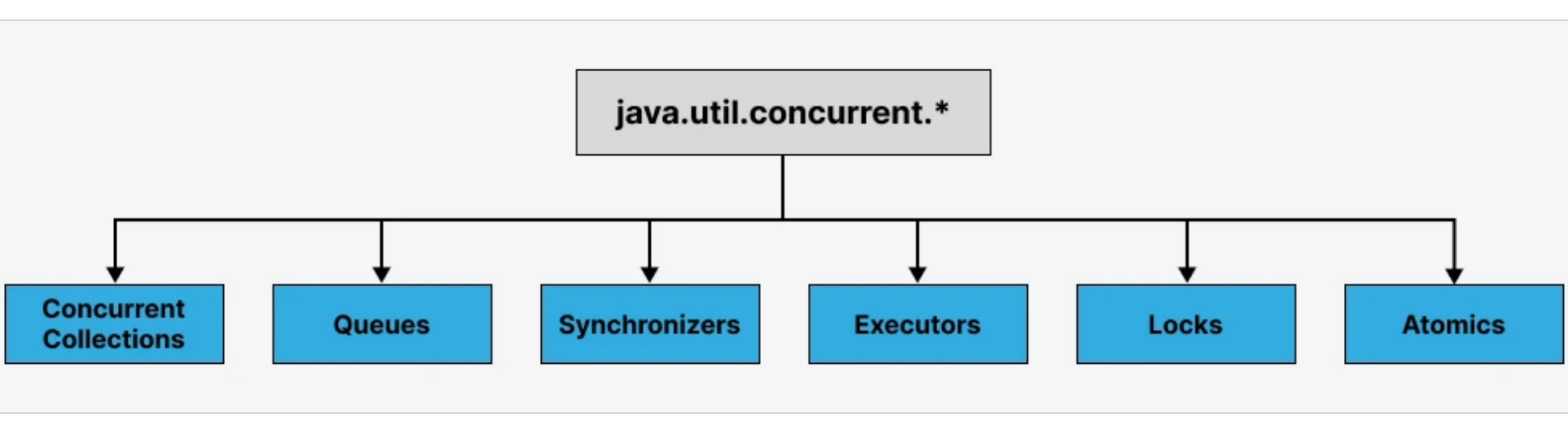
Запись работает как освобождение монитора, а чтение — как захват монитора. Доступ осуществляется в отношении по типу “выполняется прежде”. Если разобраться, то все, что будет видно для потока A, когда он обращался к volatile переменной, — это переменная для потока B. То есть вы гарантированно не потеряете ваши изменения из других потоков.

Volatile-переменные атомарны, то есть при чтении такой переменной используется такой же эффект, как и при получении блокировки — данные в памяти объявляются недействительными или некорректными и значение volatile переменной снова читается из памяти. При записи используется эффект для памяти, как и при освобождении блокировки — volatile-поле записывается в память.

Java Concurrent

Если ты хочешь сделать суперэффективное и многопоточное приложение, необходимо использовать классы из библиотеки **JavaConcurrent**, которые находятся в пакете **java.util.concurrent**.

Библиотека очень объемная и имеет разный функционал, поэтому давайте разберем, что есть внутри и поделим на некоторые модули:



Concurrent Collections — набор коллекций для работы в многопоточной среде. Вместо базового враппера Collections.synchronizedList с блокированием доступа ко всей коллекции используются блокировки по сегментам данных или используются wait-free алгоритмы для параллельного чтения данных.

Queues — неблокирующие и блокирующие очереди для работы в многопоточной среде. Неблокирующие очереди сосредоточены на скорости и работе без блокирования потоков. Блокирующие очереди подходят для работы, когда нужно “притормозить” потоки **Producer** или **Consumer**. Например, в той ситуации, когда не выполнены какие-то из условий, очередь пуста или переполнена, или же нет свободного **Consumer**'а.

Synchronizers — вспомогательные утилиты для синхронизации потоков. Представляют собой мощное оружие в “параллельных” вычислениях.

Executors — фреймворк для более удобного и легкого создания пулов потоков, легко настроить планирование работы асинхронных задач с получением результатов.

Locks — много гибких механизмов синхронизации потоков по сравнению с базовыми `synchronized`, `wait`, `notify`, `notifyAll`.

Atomics — классы, которые могут поддерживать атомарные операции над примитивами и ссылками.

‹ Предыдущая лекция

Следующая лекция ›

JavaCoder

Введите текст комментария

Anton Rantsev Уровень 68 EXPERT 11 января, 22:35

интересная лекция

Ответить 0

RUSH

JavaRush — это интерактивный онлайн-курс по изучению Java-программирования с нуля. Он содержит 1200 практических задач с проверкой решения в один клик, необходимый минимум теории по основам Java и мотивирующие фишки, которые помогут пройти курс до конца: игры, опросы, интересные проекты и статьи об эффективном обучении и карьере Java-девелопера.

ПОДПИСЫВАЙТЕСЬ

ЯЗЫК ИНТЕРФЕЙСА

Русский

СКАЧИВАЙТЕ НАШИ ПРИЛОЖЕНИЯ

ДОСТУПНО В Google Play

Загрузите в App Store