Вопросы по теме Multithreading:

1. Что такое модель памяти Java (JMM)?
2. Что такое потокобезопасность?
3. В чем разница между конкуренцией и параллелизмом?
4. Какой тип многозадачности использует Java?
5. Что такое процесс и поток? Разница между потоком и процессом?
6. Что такое sequential consistency, as-if-serial?
7. Что такое happens-before?

Модель памяти JMM.

Процесс – абстрактная совокупность кода и взаимосвязанных с выполнением этого кода физических и виртуальных процессов (выделение процессорного времени, памяти, адресного пространства).

Обычно одна программа – одни процесс (есть исключения). Под каждый процесс операционная система выделяется свое виртуальное адресное пространство (которое проецируется на физическую память), поэтому процессы выполняются независимо друг от друга (каждый процесс использует свою область памяти). Если процессу нужно получить информацию от другого процесса, то используется различные специальные средства (например, передача данных через файлы, потоки ввода / вывода, специальные библиотеки).

При запуске программы операционная система создает процесс, загружая в его адресное пространство код и данные программы, а затем запускает главный поток созданного процесса. Процесс не может существовать без потоков, поэтому если существует процесс, в нём существует хотя бы один поток.

Поток – наименьшая единица исполнения кода. Каждый поток выполняет инструкции процесса, которому он принадлежит, параллельно с другими потоками этого процесса. Поток вне процесса существовать не может.

Модель памяти Java (JMM) – описывает поведение потоков в JVM, а именно:

1. правила выполнения многопоточных программ (набор действий межпоточного взаимодействия);
2. правил, по которым потоки могут взаимодействовать друг с другом посредством основной памяти.

Существует два основных нюанса, имеющих отношение к JMM:

Visibility (область видимости) – поток может временно сохранять значения некоторых полей не в основную память, а в регистры или локальный кэш процессора (для ускорения производительности, т.к. обращаться к основной памяти значительно дороже). В то же время второй поток, читая данные из основной памяти, может не увидеть последних изменений этого поля (т.к. они сохранены в кэш, а второй поток этого не знает). В результате второй поток в дальнейшей своей работе будет использовать неверные данные, что в итоге приведет к неприемлемым результатам.

Виталий Жуков
unicraft1989@gmail.com
Ссылка на курс Stepik

Reordering (изменение порядка) – для увеличения производительности Java позволяет компилятору или процессору для увеличения производительности выполнять оптимизации (переставлять местами некоторые операции). В коде с неправильной синхронизацией это может привести к парадоксальным последствиям. Например,

int A = 10; int B = 20;

Thread 1: 1. Sasha.age = A; 2. B = 30; // ожидаем Sasha.age = 10, затем B = 30

Thread 1: 3. Olga.age = B; 4. A = 40; // ожидаем Olga.age = 20, затем A = 40

Т.к. компиляторам разрешено переупорядочивать инструкции в любом потоке, если это не влияет на выполнение этого потока в отдельности, то он имеет право менять местами инструкции 1 и 2, 3 и 4. В это случае возможен результат Sasha.age = 40, Olga.age = 30.

Свойства (поведение) сущностей (фрагментов) программы:

(это абстрактные понятия, которые характеризуют поведение программы, например, можно сказать: код обладает свойством as-if-serial или метод A и метод B связаны отношением happens-before)

as-if-serial (как будто последовательный) – это свойство программы подразумевающее, что код выполняется как будто последовательно (хотя в действительности операции могут меняться местами). Т.е. замена операций местами не повлияет на конечный результат выполнения кода, все будет работать так, как будто программы выполняется последовательно.

sequential consistency – механизм (гарантия поведения программы), при котором компилятор и процессор не могут переупорядочивать операции, которые имеют зависимости в своих данных. Т.е. если два потока используют общую переменную, то компилятор / процессор не должны производить reordering кода, касающегося этой переменной.

happens-before (происходит до) – отношение между потоками (операциями) гарантирующее, что событие A точно произойдет до событие B, но не наоборот. Например, работа методов run() и join() связано отношением happens-before: метод run() объекта A должен отработать до вызова метода join() объекта B и все изменения объекта A в этом случае буду видны объекту B.

Понятия для работы с многопоточностью:

Потокобезопасность – свойство объекта или кода, которое гарантирует, что при исполнении несколькими потоками, код будет вести себя так, как предполагается.

Атомарность – это действие или совокупность действий, которые происходят одновременно, единой командой. Атомарное действие не может остановиться на середине: оно либо происходит полностью, либо не происходит вовсе. В Java атомарными являются операции:

1. чтение и запись для ссылочных переменных;
2. чтение и запись для примитивных переменных (кроме long и double);
3. чтение и запись для всех *volatile* переменных (включая long и double).

Типы long и double занимают 64 бита. На 32-х битных платформах эти типы данных состоят из двух 32-х битных частей. Поэтому может возникнуть проблема, когда в переменную один поток записал первую часть числа, а второй поток эту часть прочитал. В итоге второй поток получает неверную информацию. Для устранения подобной ситуации существует ключевое слово **volatile**.

Ключевое слово **volatile** дает переменной следующие свойства:

1. переменная будет читаться и записываться атомарно;
2. переменную нельзя поместить в кэш, только в общую память (см. понятие visibility);

Ключевое слово **final** (в контексте многопоточности), установленное на полях класса, после создания объекта этого класса позволяет видеть значения его final полей для всех потоков без дополнительной синхронизации. Единственный нюанс, объект должен быть полностью инициализирован, т.е. его конструктор должен завершить свою работу.

Ключевое слово **static** (в контексте многопоточности), установленное на полях класса, имеет свои нюансы при работе с несколькими потоками. Дело в том, что каждый поток может иметь свою собственную копию такого поля в своем собственном локальном пространстве памяти / кэше. Поэтому без ключевого слова volatile нет никаких гарантий относительно порядка операций со статическими полями и того, когда обновления будут переданы другим потокам. Поэтому рекомендуется использовать локальные переменные вместо статических полей.

Concurrency (конкуренция) – способ одновременного решения множества разных задач.

Parallel (параллелизм) – способ одновременного выполнения разных частей одной программы

Кооперативная многозадачность – это способ деления процессорного времени между потоками, при котором каждый поток обязан отдавать управление следующему добровольно.

Вытесняющую многозадачность – способ деления процессорного времени, при котором решение о переключении между потоками процесса принимает операционная система. Этот тип многозадачности использует Java.

Решение принимается в соответствии с приоритетами задач. Управление операционной системе передаётся вне зависимости от состояния работающих приложений, благодаря чему зависшие приложения, не «подвешивают» операционную систему. За счёт регулярного переключения задач также улучшается отзывчивость системы, оперативность освобождения ресурсов системы, которые больше не используются задачей.

Виталий Жуков
unicraft1989@gmail.com
Ссылка на курс Stepik