Модификатор volatile

В компьютере есть два вида памяти – глобальная (обычная) и встроенная в процессор. Встроенная в процессор делится на регистры, затем кэш первого уровня (L1), кэш второго уровня (L2) и третьего уровня (L3).

Эти виды памяти отличаются по скорости работы. Самая быстрая и самая маленькая память – это регистры, затем идет кэш процессора (L1, L2, L3) и, наконец, глобальная память (самая медленная).

Скорость работы глобальной памяти и кэша процессора сильно отличаются, поэтому Java-машина позволяет каждой нити хранить самые часто используемые переменные в локальной памяти нити (в кэше процессора).

Есть одна маленькая проблемка. Когда две нити работают с одной и той же переменной, каждая из них может сохранить ее копию в своем внутреннем локальном кэше. И тогда может получится такая ситуация, что одна нить переменную меняет, а вторая не видит этого изменения, т.к. по-прежнему работает со своей копией переменной.

На этот случай разработчики Java предусмотрели специальное ключевое слово – volatile. Если есть переменная, к которой обращаются из разных нитей, ее нужно пометить модификатором volatile, чтобы Java-машина не помещала ее в кэш. Вот как это обычно выглядит:

```
public volatile int count = 0;
```

Модификатор volatile гарантирует только безопасное чтение/запись переменной, но не ее изменение.

Как изменяется переменная:

Код	Что происходит на самом деле:	Описание

`count++`	`register = count;`
	`register = register+1;`
	`count = register;`
		Этап 1.
		Значение переменной count копируется из глобальной памяти в регистровую память процессора.
		
		Этап 2
		Внутри процессора регистровая переменная увеличивается на 1.
		
		Этап 3
		Значение переменной копируется из процессора в глобальную память.

Модификатор volatile, гарантирует, что при обращении к переменной count она будет прочитана из памяти (этап 1). А если какая-то нить захочет присвоить ей новое значение , то оно обязательно окажется в глобальной памяти (этап 3). Но Java-машина не гарантирует, что не будет переключения нитей между этапами 1 и 3.

Если две нити попытаются обратиться к volatile переменной, то может произойти такое:

Нить 1	Нить 2	Результат

`register1 = count;`
`register1++;`
`count = register1;`
	`register2 = count;`
	`register2++;`
	`count = register2;`
		`register1 = count;`
		`register2 = count;`
		`register2++;`
		`count = register2;`
		`register1++;`
		`count = register1;`

Следовательно, обращаться к переменной можно без опасений, а вот при изменении необходимо использовать synchronized.

Атомарные типы данных и операции

В Java операции чтения и записи полей всех типов, кроме long и double, являются атомарными.

Что такое атомарность? Ну, например, если ты в одном потоке меняешь значение переменной int, а в другом потоке читаешь значение этой переменной, ты получишь либо ее старое значение, либо новое — то, которое получилось после изменения в потоке 1. Никаких «промежуточных вариантов» там появиться не может.

Однако с long и double это не работает из-з кроссплатформенности.

Так как принцип Java — «написано однажды — работает везде». Это и есть кроссплатформенность. То есть Java-приложение запускается на абсолютно разных платформах. Например, на операционных системах Windows, разных вариантах Linux или MacOS, и везде это приложение будет стабильно работать.

long и double — самые «тяжеловесные» примитивы в Java: они весят по 64 бита. И в некоторых 32-битных платформах просто не реализована атомарность чтения и записи 64-битных переменных. Такие переменные читаются и записываются в две операции. Сначала в переменную записываются первые 32 бита, потом еще 32. Соответственно, в этих случаях может возникнуть проблема. Один поток записывает какое-то 64-битное значение в переменную X, и делает он это «в два захода». В то же время второй поток пытается прочитать значение этой переменной, причем делает это как раз посередине, когда первые 32 бита уже записаны, а вторые — еще нет. В результате он читает промежуточное, некорректное значение, и получается ошибка.

Решается эта проблема с помощью ключевого слова volatile.

Если мы объявляем переменную со словом volatile, то это означает, что:

1. Она всегда будет атомарно читаться и записываться. Даже если это 64-битные double или long.
2. Java-машина не будет помещать ее в кэш. Так что ситуация, когда 10 потоков работают со своими локальными копиями исключена.