

# Умри, замри, воскресни...

Андрей Терешко

Нити или треды (threads). Работать с ними трудно, понять их работу ещё труднее, но, вошедши во вкус — за уши не оттянешь. Хотя есть программисты, считающие, что надо так проектировать программу, чтобы нити были нужны как можно реже. А если ещё обойтись и без объектов, без ООП — то, вообще, будет хорошо. И, надо признаться, поначалу, когда пытаешься отладить программу со многими нитями, то желание все бросить и взять обыкновенный, простой язык, типа ... (Вы поняли какой), возникало у меня неоднократно — особенно это касалось взаимодействия между тредами.

Нет-нет, в книгах и статьях все очень просто описано. Тред может "впасть в ожидание извещения о событии" вызовом метода wait любого(!) объекта. Ожидающий тред можно известить вызовом метода notify() (notifyAll()) для объекта, у которого тред и ждет. Но чтобы разобраться в этом подробнее, чтобы уяснить разность между notify() и notifyAll(), мне пришлось придумать свою модель "жизни" тредов.

Я представил себе, что объект — это комната со змеями-тредами. Внутри комнаты расположены клетки-методы с белыми мышами (поля объекта и (или) его локальные переменные). В одних клетках-методах дверцы постоянно распахнуты — и в них могут вползать любые змеи-треды в любое время и в любом количестве — это public методы. Другие клетки-методы, имеют такой механизм дверей, что при попадании любой змеи-треда в одну из этих клеток, двери остальных клеток автоматически закрываются, никого не впуская, и открываются только тогда, когда змея-тред покинет клетку. Такие клетки отмечены словом synchronized. Дверцы у них синхронизированы между собой.

И вот, в такой серпентарии кипит жизнь: змеи-треды постоянно передвигаются по клеткам за мышами или даже достают мышей вне клеток (свободные мыши вне клеток-методов — это public поля объекта, если они есть, конечно). И для чего это я все придумал? Сейчас будет разъяснение.

Кроме клеток и мышей в этой комнате-объекте находится мешок, куда змеи-треды вползают, чтобы "дождаться извещения о событии". В этот мешок змеи-треды могут попасть только из клеток-методов, отмеченных словом `synchronized`. Из остальных клеток — ну ни как! Так что, если змея-тред хочет "впасть в ожидание извещения о событии" (для того, чтобы дождаться чего-либо, не тратя попусту жизненные силы на постоянную проверку свершения какого либо события), то она должна попасть в клетку-метод `synchronized`, а оттуда уже — в мешок, где, возможно, уже находятся другие "ждущие извещения о событии" змеи-треды.

Так как одновременно в любой из клеток-`synchronized` может находиться только одна змея-тред, то возможна конкуренция между змеями-тредами, желающими попасть в клетку-`synchronized`. Часть из этих змей-тредов мечтает через клетку-`synchronized` попасть в мешок и ждать там извещения о событии, а другая часть — просто желает "покрутиться" в клетке-`synchronized` и вылезть из нее, не залезая в мешок. Таким образом, змеи-треды выстраиваются в одну общую(!) для объекта очередь на доступ к любой из клеток-`synchronized` объекта.

Змея-тред может находиться в нескольких состояниях:

Первое — активное состояние — нескончаемый поиск и пожирание белых мышей.

Второе — "общая очередь ожидающих доступа для одного объекта" (не путать со "спячкой" и "ожиданием события") — на доступ в одну из клеток-методов-`synchronized`. Из этого состояния змею-тред может вывести только попадание в клетку-`synchronized` и ничто другое! Если какая нибудь змея-тред займет любую из клеток-`synchronized` объекта "навечно" (например, по тем или иным причинам, войдет в "вечный" цикл), то остальные змеи-треды, стоящие в очередь на доступ в любую из клеток- `synchronized` "навечно" и будут ждать. Я не знаю методов и способов, способных заставить змею-тред покинуть эту "очередь ожидающих доступа", кроме как впустить ее в клетку-метод-`synchronized`. Или вызвать `Deprecated` метод треда `stop` (бр-р).

Третье состояние — "ожидание извещения о событии", в которое змея-тред впадает, когда вызывает метод `wait()` для объекта. В отличии от "очереди ожидающих доступа", змея-тред может выйти из состояния "ожидания извещения о событии" двумя способами: либо получив извещение об событии методом `notify` (`notifyAll`), либо по

## *Умри, замри, воскресни...*

истечения времени "ожидания извещения о событии", которое она же себе и установила, вызвав метод `wait(long timeout)`.

И другие состояния — состояние "сна" (не путать с "ожиданием доступа"), вызывается методом треда `sleep(long millis)`; состояние приостановки треда, вызывается `Deprecated` методом треда `suspend()`; состояние ожидания завершения другого треда, вызывается методом треда `join()` или `join(long millis)`.

Как только змея-тред забралась в мешок "ожидающих извещения о событии", то тем самым клетка-`synchronized` освобождается и доступ ко всем клеткам-`synchronized` объекта становится свободным для тредов все еще стоящих в "очереди ожидающих доступа".

Любая(!) змея-тред, находясь в любой(!) клетке-`synchronized` объекта (и только в клетке типа `synchronized`), может "укусить", то есть известить о событии либо одну змею-тред из мешка (ту, которая дольше всех "ожидает извещения о событии"), вызвав метод `notify()`, либо может известить о событии всех змей-тредов "ожидающих извещения о событии" в мешке — методом `notifyAll()`.

Змеи-треды, извещенные о событии, вытряхиваются и вываливаются из "мешка-ожидания", но стоп(!), они еще не могут начать активную "жизнь". Известить всех о событии — это еще не значит перевести их в активное состояние! Все извещенные змеи-треды сразу же попадают в "очередь ожидающих доступа" — так как, раз они стали ожидать события в клетке-`synchronized`, то должны(!) в клетке-`synchronized` и получить известие о событии(а то где?). А так как в любой из клеток-`synchronized` в одно и то же время может находиться только одна(!) змея-тред, то все получившие известие о событии неизбежно попадают в "очередь ожидающих доступа" к клетке-`synchronized` (если эта очередь, конечно существует, то есть, если получили известие о событии сразу несколько змей-тредов или несколько змей-тредов уже находятся в этой очереди).

Даже, если только одна змея-тред будет извещена о событии (методом `notify()`), то вернуться к активному состоянию (а для этого надо обязательно(!) попасть в клетку-`synchronized`) ей может помешать уже имеющаяся "очередь ожидающих доступа" в клетки-`synchronized`. Точно так, как проснувшись по звонку будильника, нам мешает начать полноценную жизнь... — очередь в ванную.

И даже, если имеется только две змеи-треда, использующие клетки-методы одного объекта, и при этом, одна змея-тред, проникнув в клетку-метод-synchronized известила о событии другую змею-тред, находящуюся в мешке-ожидания, но затем не покинула клетку-метод-synchronized (а, например, обратилась к длительному установлению связи с удаленным узлом сети) — то, до тех, пока эта змея-тред не покинет клетку-метод-synchronized, до тех пор извещенная змея-тред будет так и ждать доступа в клетку-метод-synchronized, пускай даже для того, чтобы эту клетку-метод-synchronized просто покинуть!

Уже получившую извещение о событии, но еще не активную, змею-тред не надо повторно извещать, вызывая метод notify(). Подойдет ее очередь в клетку-synchronized, вот тогда и заживет змея-тред активной своей жизнью.

Имеется возможность закрыть доступ другим змеям-тредам в клетки-методы-synchronized объекта, не заходя ни в одну из них(!). Для этого змея-тред должна использовать оператор synchronized( ссылка на объект) {...}. Использование этого оператора фактически приводит к запиранию всех(!) клеток-методов-synchronized объекта, до тех пор, пока змея-тред не покинет блок-кода оператора synchronized.

Естественно, использование оператора synchronized () {...} (не путать с атрибутом метода!), не приводит к полному "блокированию объекта", ибо доступ к public полям объекта всегда(!) свободен, также как и вызов методов объекта не имеющих атрибута synchronized. То есть, использование оператора synchronized(){...} влияет только (и только!) на доступ к методам объекта с атрибутом synchronized.

Вообще, термин "блокирования объекта" надо употреблять с пониманием, ибо заблокировать можно только тред, путем блокирования доступа к методам объекта имеющих атрибут synchronized (и только к этим методам!). Или, иначе говоря, заблокировать можно доступ для треда к методу-synchronized. Все остальные методы объекта, не имеющие атрибут synchronized, а также поля объекта, всегда доступны (в пределах описанных для них атрибутов, типа public или private), не зависимо ни от каких манипуляций!

Однако, если, говоря "объект заблокирован", имеют в виду только и только(!) блокировку доступа тредов к методам-synchronized этого объекта, то вполне

## Умри, замри, воскресни...

допустимо и "блокировать объект" — главное, чтобы Вы понимали и Вас понимали — о чем идет речь. То есть, когда говорят: "дом заблокирован", то подразумевают, что "доступ в дом через эту дверь заблокирован". Но это еще не значит, что в дом нельзя попасть через окно по пожарной лестнице, если, конечно, в доме есть окна без решеток.

Использование термина "блокировать объект", вместо правильного "блокирования доступа к методам объекта имеющих атрибут synchronized" — на практике оказывается вполне допустимым, приемлемым и повсеместно применяемым. Приемлемо именно из-за краткости.

Если программист-разработчик класса, не положил мешок для "ожидающих извещения о событии" в какой-либо из клеток-методов-synchronized, то всегда можно любому(!) объекту "навесить" этот мешок. Не имеет значения — есть ли у этого объекта методы с атрибутом synchronized или нет. Для этого надо использовать оператор synchronized(ссылка на объект){...} , в блок-коде которого применить метод wait():

```
Object myObject = ...
...
synchronized(myObject) {
    ...
    myObject.wait();
    ...
}
```

Аналогично, чтобы известить всех ожидающих события в мешке:

```
synchronized(myObject) {
    ...
    myObject.notifyAll();
    ...
}
```

Таким образом, даже для объекта, у которого нет методов с атрибутом synchronized можно сказать — "объект заблокирован" оператором synchronized. Но, по большому счету, блокировать в таком объекте — нечего, ведь методов с атрибутом synchronized нет! И "блокировка" оператором нужна только для того, чтобы влезть в мешок тредов "ожидающих извещения о событии" или покинуть этот мешок — поскольку только мешок и находится у "блокируемого объекта".

Вот так и "ловят мышей" змеи-треды, переходя из одной клетки-метода в другую, выстраиваясь в "очередь тредов ожидающих доступа" к клеткам- `synchronized`, переползая из клеток-`synchronized` в мешок тредов "ожидающих извещения о событии", получающих извещение о событии, вновь занимающих "очередь тредов ожидающих доступа" к клеткам-`synchronized` и т.д.

Я надеюсь, что читателю, запутавшемуся в `wait-notify`, поможет моя модель "серпентария" в решении его проблем — ведь главное, чтобы Вы меня понимали.