

Re: Understanding java.lang.Object.wait()

Source: <http://coding.derkeiler.com/Archive/Java/comp.lang.java.programmer/2007-06/msg01938.html>

- *From:* Aria <maps.this.address@xxxxxxxxx>
 - *Date:* Wed, 27 Jun 2007 13:45:52 -0000
-

On Jun 27, 1:21 am, Owen Jacobson <angrybald...@xxxxxxxxx> wrote:

On Jun 26, 7:01 pm, Aria <maps.this.addr...@xxxxxxxxx> wrote:

Hi,

I am a bit confused as how wait() actually works. I know the purpose of this method but what I wanted to know is when a few threads waiting on a certain object lock and they are signaled via notifyAll(), how does one of the threads on the waiting list for that object lock gets the lock and what would happen to the other threads?

The other threads wait for the first thread to release the lock, and then one of them wins it. This repeats until all of the threads have been resumed, exactly like any other lock contention scenario.

Now, I understand that notifyAll() ONLY notifies other threads blocking on this object that the current thread is done with the object is "ready" to release the lock. That is the actual release of the lock occurs AFTER the code is out of synchronized block.

Having said that, what exactly happens when the waiting threads receive such notification? That is:

```
// Other threads running block
public void run() {
    synchronized (obj) {
        while (some condition not being met) {
```

Re: Understanding java.lang.Object.wait()

```
try {
    obj.wait();
    // (1)
} catch (InterruptedException ie) { }
// Some code to process
}
}
```

When "ALL THE THREADS" receive such signal, do they all "get out" of wait() method and then whoever wins the contention would gets the lock and the rest are put on the waiting list once they "failed" the condition?

It's a little hard to interpret this, but it sounds right.

When you call notifyAll on an object, every thread that's wait()ing on that object resumes and tries to reacquire the monitors they held, blocking if necessary until another thread releases those monitors. The contract for a monitor is that at most one thread can hold it at a time; this is enforced by the JVM.

If several threads that held the same monitor are awoken at the same time, they will run "one at a time" through the parts of their executions that hold that monitor. None of them will be put back to waiting on the object.

Owen

So according to your statement, every time blocking threads are signaled, ALL of them gets out of obj.wait(), "try" to acquire the lock on the object, and only one would prevail. Which brings us to the question I asked, they ALL get to their line (1) after each notification and only put on the waiting list due to the while (condition not being met). Is that a correct interpretation?

If that is the case, this would be a bit of waste of effort on JVM part because this concludes that after each notification, JVM needs to move all the blocking threads out of the waiting list, choose one as the winner, let the rest go through their while loop and put on the waiting list again. couldn't the scheduler select one thread, give it a lock and "keep" the rest on the waiting list without getting them ALL out of wait()?

Re: Understanding java.lang.Object.wait()

Re: Understanding java.lang.Object.wait()

So my question pretty much boils down to this: When a notification occurs, do ALL waiting threads execute line [b](1)[/b] regardless of whether win the contention and only to be put back on the waiting list after failing the while (condition not being met)?

.