When to use CountDownLatch : Java concurrency example tutorial

JULY 18, 2013 LOKESH 16 COMMENTS

As per java docs, **CountDownLatch** is a synchronization aid that allows one or more threads to wait until a set of operations being performed in other threads completes. CountDownLatch concept is very common **interview question** in **java concurrency**, so make sure you understand it well. In this post, I will cover following points related to CountDownLatch in java concurrency.

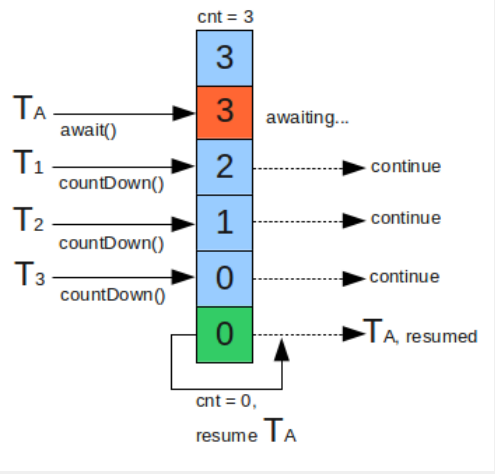
Sections in this post:

What is CountDownLatch?
How CountDownLatch works?
Possible usages in real time applications
Example application
Common interview questions

What is CountDownLatch?

CountDownLatch was **introduced with JDK 1.5 along with other concurrent utilities like CyclicBarrier, Semaphore, ConcurrentHashMap and BlockingQueue** in java.util.concurrent package. This class **enables a java thread to wait until other set of threads completes** their tasks. e.g. Application's main thread want to wait, till other service threads which are responsible for starting framework services have completed started all services.

CountDownLatch works by having a counter initialized with number of threads, which is decremented each time a thread complete its execution. When count reaches to zero, it means all threads have completed their execution, and thread waiting on latch resume the execution.



CountDownLatch Concept

Pseudo code for CountDownLatch can be written like this:

```
//Main thread start
//Create CountDownLatch for N threads
//Create and start N threads
//Main thread wait on latch
//N threads completes there tasks are returns
//Main thread resume execution
```

How CountDownLatch works?

CountDownLatch.java class defines one constructor inside:

```
1  //Constructs a CountDownLatch initialized with the given count.
2  public void CountDownLatch(int count) {...}
```

This **count is essentially the number of threads**, for which latch should wait. This value can be set only once, and CountDownLatch provides **no other mechanism to reset this count**.

The first interaction with CountDownLatch is with main thread which is goind to wait for other threads. This main thread must call, **CountDownLatch.await()** method immediately after starting other threads. The execution will stop on await() method till the time, other threads complete their execution.

Other N threads must have reference of latch object, because they will need to notify the CountDownLatch object that they have completed their task. This notification is done by method : **CountDownLatch.countDown()**; Each invocation of method decreases the initial count set in constructor, by 1. So, when all N threads have call this method, count reaches to

Java executor framework tutorial and best practices

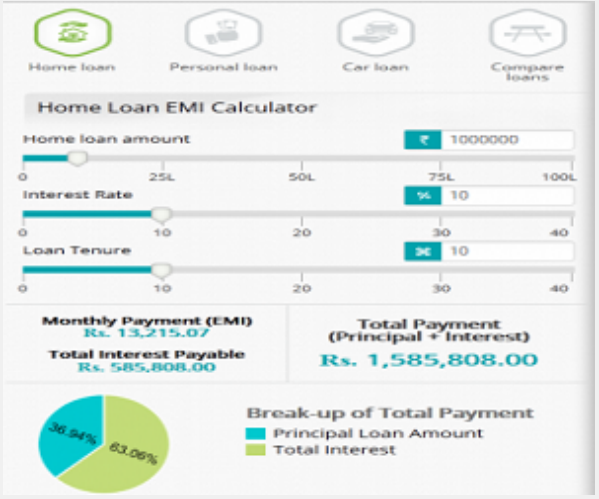
Difference between sleep() and wait()?

Difference between “implements Runnable” and “extends Thread” in java

Difference between yield() and join() in threads in java?

How to use BlockingQueue and ThreadPoolExecutor in java

Thread synchronization, object level locking and class level locking



zero, and main thread is allowed to resume its execution past await() method.

Possible usages in real time applications

Let's try to identify some possible usage of CountDownLatch in real time java applications. I am listing, as much i can recall. If you have any other possible usage, please leave a comment. It will help others.

1. **Achieving Maximum Parallelism** : Sometimes we want to start a number of threads at the same time to achieve maximum parallelism. For example, we want to test a class for being singleton. This can be done easily if we create a CountDownLatch with initial count 1, and make wait all threads to wait of latch. A single call to countDown() method will resume execution for all waiting threads in same time.
2. **Wait N threads to completes before start execution**: For example an application start-up class want to ensure that all N external systems are Up and running before handling the user requests.
3. **Deadlock detection**: A very handy use case in which you can use N threads to access a shared resource with different number of threads in each test phase, and try to create a deadlock.

Example application using CountDownLatch

In this example, I have simulated an application startup class which starts N threads that will check for external systems and report back to latch, on which startup class is waiting. As soon as all services are verified and checked, startup proceeds.

BaseHealthChecker.java : This class is a Runnable and parent for all specific external service health checkers. This remove the code duplicacy and central control over latch.

```
1  public abstract class BaseHealthChecker implements Runnable {
2
3      private CountDownLatch _latch;
4      private String _serviceName;
5      private boolean _serviceUp;
6
7      //Get latch object in constructor so that after completing the task, threa
8      public BaseHealthChecker(String serviceName, CountDownLatch latch)
9      {
```

```
        super();
        this._latch = latch;
        this._serviceName = serviceName;
        this._serviceUp = false;
    }

    @Override
    public void run() {
        try {
            verifyService();
            _serviceUp = true;
        } catch (Throwable t) {
            t.printStackTrace(System.err);
            _serviceUp = false;
        } finally {
            if(_latch != null) {
                _latch.countDown();
            }
        }
    }

    public String getServiceName() {
        return _serviceName;
    }

    public boolean isServiceUp() {
        return _serviceUp;
    }
    //This methos needs to be implemented by all specific service checker
    public abstract void verifyService();
}
```

NetworkHealthChecker.java : This class extends BaseHealthChecker and needs to provide only implementation of verifyService() method. **DatabaseHealthChecker.java** and **CacheHealthChecker.java** are same as NetworkHealthChecker.java apart from their service names and sleep time.

```
public class NetworkHealthChecker extends BaseHealthChecker
{
    public NetworkHealthChecker (CountDownLatch latch)  {
        super("Network Service", latch);
    }

    @Override
    public void verifyService()
    {
        System.out.println("Checking " + this.getServiceName());
        try
        {
```

```
            Thread.sleep(7000);
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
        System.out.println(this.getServiceName() + " is UP");
    }
}
```

ApplicationStartupUtil.java : This clas is main startup class which initilizes the latch and wait of this latch till all services are checked.

```
public class ApplicationStartupUtil
{
    //List of service checkers
    private static List<BaseHealthChecker> _services;

    //This latch will be used to wait on
    private static CountDownLatch _latch;

    private ApplicationStartupUtil()
    {
    }

    private final static ApplicationStartupUtil INSTANCE = new ApplicationStar

    public static ApplicationStartupUtil getInstance()
    {
        return INSTANCE;
    }

    public static boolean checkExternalServices() throws Exception
    {
        //Initialize the latch with number of service checkers
        _latch = new CountDownLatch(3);

        //All add checker in lists
        _services = new ArrayList<BaseHealthChecker>();
        _services.add(new NetworkHealthChecker(_latch));
        _services.add(new CacheHealthChecker(_latch));
        _services.add(new DatabaseHealthChecker(_latch));

        //Start service checkers using executor framework
        Executor executor = Executors.newFixedThreadPool(_services.size());

        for(final BaseHealthChecker v : _services)
        {
            executor.execute(v);
```

```
        }

        //Now wait till all services are checked
        _latch.await();

        //Services are file and now proceed startup
        for(final BaseHealthChecker v : _services)
        {
            if( ! v.isServiceUp())
            {
                return false;
            }
        }
        return true;
    }
}
```

Now you can write any test class to check the functionality of latch.

```
public class Main {
    public static void main(String[] args)
    {
        boolean result = false;
        try {
            result = ApplicationStartupUtil.checkExternalServices();
        } catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println("External services validation completed !! Result w
    }
}

Output in console:

Checking Network Service
Checking Cache Service
Checking Database Service
Database Service is UP
Cache Service is UP
Network Service is UP
External services validation completed !! Result was :: true
```

Common interview questions

Be prepared to face these questions related to CountDownLatch in your next interview.

- Explain the concept of CountDownLatch?
- Difference between CountDownLatch and CyclicBarrier?
- Give some example uses of CountDownLatch?
- What are main methods CountDownLatch class has?

To Download the sourcecode of above example application follow given link.

Sourcecode Download

Reference

Java Docs

Related Posts:

1. **Java executor framework tutorial and best practices**
2. **Difference between sleep() and wait()?**
3. **Difference between “implements Runnable” and “extends Thread” in java**
4. **Difference between yield() and join() in threads in java?**
5. **How to use BlockingQueue and ThreadPoolExecutor in java**
6. **Thread synchronization, object level locking and class level locking**

CORE JAVA COUNTDOWNLATCH INTERVIEW QUESTION JAVA CONCURRENCY MULTI THREADING

PREVIOUS POST

Automatic resource management with try-with-resources in java 7

NEXT POST

5 popular java development frameworks

16 THOUGHTS ON "WHEN TO USE COUNTDOWNLATCH : JAVA CONCURRENCY EXAMPLE TUTORIAL"

Ranjeet Singh

AUGUST 31, 2014 AT 7:46 AM

Hi Lokesh,
Thanks for making us understaing multi threading concepts with so ease.
I have one multi threading problem. Could you suggest me a solution.
Question : There are multiple threads(can be more than thousand). Each thread has a numeric value i.e thread1 has '1' , thread2 has '2' and so on. I want these numbers(1,2,3,,……) to be printed in ascending order. How to do this. It looks to me that countdownlatch will be used here but don't know how to implement this.

Thanks in advance.

REPLY

Viswanath

JUNE 11, 2014 AT 6:03 AM

Hi Lokesh,
Does the count in the constructor denotes the number of threads for which the latch should wait or is it the number of times countDown() method needs to be called before the main thread proceeds?

REPLY

★ **Lokesh**

JUNE 11, 2014 AT 7:15 AM

Ideally both. Because each thread should call countDown() method exactly once.

REPLY

Mayank

FEBRUARY 2, 2014 AT 5:55 PM

Is CountDownLatch is doing any different work than Thread's join() ?

REPLY

★ **Lokesh**

FEBRUARY 3, 2014 AT 1:56 AM

Using intelligently, you can achieve any desired result with basic method calls e.g. wait(), notify(), sleep() or join(). CountDownLatch provides you the ease so that you can concentrate on business, not on other tiny implementation details.

REPLY

Shobs

JANUARY 9, 2014 AT 8:53 AM

request you to post on cyclic barrier as well.

REPLY

★ **Lokesh Gupta**

JANUARY 10, 2014 AT 2:56 PM

I will soon.

REPLY

Ajit

DECEMBER 31, 2013 AT 9:49 AM

This sample is a perfect solution for one of our requirement.I tried your source code, the main thread never gets terminated after execution of the program.What could be the reason for that?

REPLY

Mangpal Singh

JANUARY 5, 2014 AT 8:41 PM

use ExecutorService instead of Executor interface , after execute () call shutdown() to gracefully shutdown the ExecutorService and finally the main() will close.

```
//Start service checkers using executor framework
ExecutorService executor = Executors.newFixedThreadPool(_services.size());

for(final BaseHealthChecker v : _services)
{
executor.execute(v);
}

executor.shutdown();

//Now wait till all services are checked
_latch.await();
```

REPLY

Santosh Singh

OCTOBER 2, 2013 AT 3:56 PM

Hi Lokesh ,

Can you explain CountDownlatch with simple program , i mean please do not use Executer framework.

REPLY

★ Lokesh Gupta

OCTOBER 2, 2013 AT 9:34 PM

I will try to find some time.

REPLY

Nitya

AUGUST 20, 2013 AT 11:39 PM

Please post some tutorial on java.concurrent package classes like CyclicBarrier,ReentrantLock,Semaphore etc…

REPLY

Luke

JULY 20, 2013 AT 10:48 PM

```
List verifyServices = new ArrayList
for (Service service : services) {
verifyServices.add(executor.submit(service));
}
for (Future verifyService : verifyServices) {
try {
verifyService.get();
} catch (Exception e) {
return false;
}
}
return true;
```

REPLY

vijay

JULY 19, 2013 AT 5:22 PM

Hello Lokesh ,

Superb Explaination.
Thank you for such good explaination

REPLY

subbareddy

JULY 18, 2013 AT 4:32 PM

Hi'
i have small doubt in hibernate please give me solution
how to access last updated records in the data base?

REPLY

★ Lokesh Gupta

JULY 18, 2013 AT 4:52 PM

If table colum have any timestamp or date field, then sort on this using criteria.
session.createCriteria(Customer.class).addOrder(Order.desc("lastUpdated")).list();

REPLY

Note:- In comment box, please put your code inside **[java] ... [/java]** OR **[xml] ... [/xml]** tags otherwise it may not appear as intended.

LEAVE A REPLY

Your email address will not be published. Required fields are marked *

Name *

Email *

Website

Are you Human and NOT robot? Solve this to prove !! *

five × = 25

Comment

You may use these HTML tags and attributes:

```
<a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code> <del datetime=""> <em> <i> <q cite=""> <strike> <strong>
```

POST COMMENT

☐ Notify me of follow-up comments by email.

▣ Notify me of new posts by email.

SUBSCRIBE TO BLOG VIA EMAIL

Join 3,668 other subscribers

Email Address

SUBSCRIBE

REFERENCES

Oracle Java Docs

Spring 3 Reference

Spring 4 References

Jboss RESTEasy JAX-RS

Hibernate Developer Guide

Junit Wiki

Maven FAQs

Dzone Dev Links

JSP Homepage

ANT Homepage

META LINKS

Advertise

Share your knowledge

Privacy policy

Contact Us

About Us

POPULAR POSTS

Spring 3 and hibernate integration tutorial with example

Reading/writing excel files in java : POI tutorial

How hashmap works in java

Singleton design pattern in java

Useful java collection interview questions

Understanding hibernate first level cache with example

How hibernate second level cache works?

Working with hashCode and equals methods in java

How to make a java class immutable

4 ways to schedule tasks in Spring 3 : @Scheduled example