

ImportNew

- [首页](#)
- [所有文章](#)
- [资讯](#)
- [Web](#)
- [架构](#)
- [基础技术](#)
- [书籍](#)
- [教程](#)
- [Java小组](#)
- [工具资源](#)

- 导航条 - ▾

什么时候使用CountDownLatch

2015/04/14 | 分类: [基础技术](#) | [11 条评论](#) | 标签: [CountDownLatch](#), [工具类](#)

分享到:

本文由 [ImportNew](#) - [张涛](#) 翻译自 [howtodoinjava](#)。欢迎加入[翻译小组](#)。转载请见文末要求。

正如每个Java文档所描述的那样, [CountDownLatch](#)是一个同步工具类, 它允许一个或多个线程一直等待, 直到其他线程的操作执行完后再执行。在[Java并发](#)中, countdownlatch的概念是一个常见的[面试题](#), 所以一定要确保你很好的理解了它。在这篇文章中, 我将会涉及到在Java并发编程中跟CountDownLatch相关的以下几点:

目录

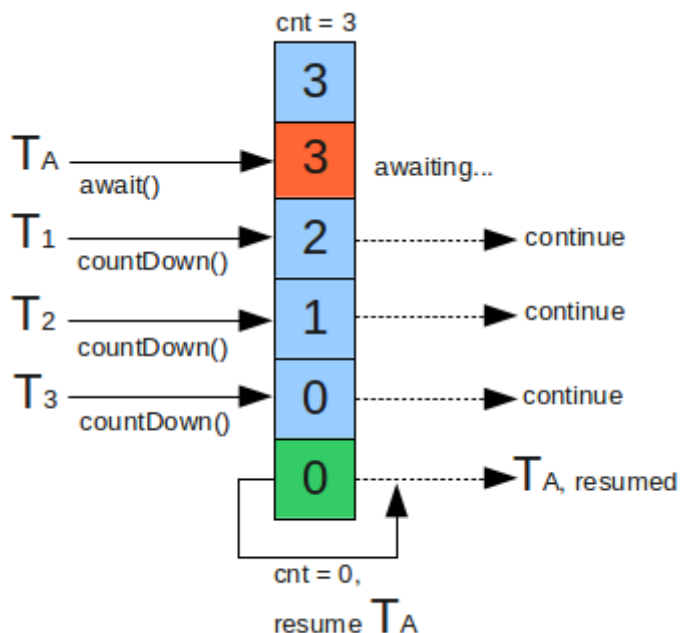
- CountDownLatch是什么?
- CountDownLatch如何工作?

- 在实时系统中的应用场景
- 应用范例
- 常见的面试题

CountDownLatch是什么

CountDownLatch是在java1.5被引入的，跟它一起被引入的并发工具类还有CyclicBarrier、Semaphore、[ConcurrentHashMap](#)和[BlockingQueue](#)，它们都存在于java.util.concurrent包下。CountDownLatch这个类能够使一个线程等待其他线程完成各自的工作后再执行。例如，应用程序的主线程希望在负责启动框架服务的线程已经启动所有的框架服务之后再执行。

CountDownLatch是通过一个计数器来实现的，计数器的初始值为线程的数量。每当一个线程完成了自己的任务后，计数器的值就会减1。当计数器值到达0时，它表示所有的线程已经完成了任务，然后在闭锁上等待的线程就可以恢复执行任务。



CountDownLatch的伪代码如下所示：

```
1 //Main thread start
2 //Create CountDownLatch for N threads
3 //Create and start N threads
4 //Main thread wait on latch
```

```
5 //N threads completes there tasks are returns  
6 //Main thread resume execution
```

CountDownLatch如何工作

CountDownLatch.java类中定义的构造函数：

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

构造器中的**计数值 (count)** 实际上就是闭锁需要等待的线程数量。这个值只能被设置一次，而且CountDownLatch没有提供任何机制去重新设置这个计数值。

与CountDownLatch的第一次交互是主线程等待其他线程。主线程必须在启动其他线程后立即调用**CountDownLatch.await()**方法。这样主线程的操作就会在这个方法上阻塞，直到其他线程完成各自的任務。

其他N 个线程必须引用闭锁对象，因为他们需要通知CountDownLatch对象，他们已经完成了各自的任務。这种通知机制是通过**CountDownLatch.countDown()**方法来完成的；每调用一次这个方法，在构造函数中初始化的count值就减1。所以当N个线程都调用了这个方法，count的值等于0，然后主线程就能通过await()方法，恢复执行自己的任务。

在实时系统中的使用场景

让我们尝试罗列出在java实时系统中CountDownLatch都有哪些使用场景。我所罗列的都是我所能想到的。如果你有别的可能的使用方法，请在留言里列出来，这样会帮助到大家。

1. **实现最大的并行性**：有时我们想同时启动多个线程，实现最大程度的并行性。例如，我们想测试一个单例类。如果我们创建一个初始计数为1的CountDownLatch，并让所有线程都在这个锁上等待，那么我们可以很轻松地完成测试。我们只需调用一次countDown()方法就可以让所有的等待线程同时恢复执行。
2. **开始执行前等待n个线程完成各自任务**：例如应用程序启动类要确保在处理用户请求前，所有N个外部系统已经启动和运行了。
3. **死锁检测**：一个非常方便的使用场景是，你可以使用n个线程访问共享资源，在每次测试阶段的线程数目是不同的，并尝试产生死锁。



CountDownLatch使用例子

在这个例子中，我模拟了一个应用程序启动类，它开始时启动了n个线程类，这些线程将检查外部系统并通知闭锁，并且启动类一直在闭锁上等待着。一旦验证和检查了所有外部服务，那么启动类恢复执行。

BaseHealthChecker.java: 这个类是一个Runnable，负责所有特定的外部服务健康的检测。它删除了重复的代码和闭锁的中心控制代码。

```

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, thread can countDown() the latch
8      public BaseHealthChecker(String serviceName, CountDownLatch latch)
9      {
10         super();
11         this._latch = latch;
12         this._serviceName = serviceName;
13         this._serviceUp = false;
14     }
15
16     @Override
17     public void run() {
18         try {
19             verifyService();
20             _serviceUp = true;
21         } catch (Throwable t) {
22             t.printStackTrace(System.err);
23             _serviceUp = false;
24         } finally {
25             if(_latch != null) {
26                 _latch.countDown();
27             }
28         }
29     }
30
31     public String getServiceName() {
32         return _serviceName;
33     }
34
35     public boolean isServiceUp() {
36         return _serviceUp;
37     }
38     //This methos needs to be implemented by all specific service checker
39     public abstract void verifyService();
40 }

```



NetworkHealthChecker.java: 这个类继承了BaseHealthChecker，实现了verifyService()方法。 **DatabaseHealthChecker.java**和 **CacheHealthChecker.java**除了服务名和休眠时间外，与NetworkHealthChecker.java是一样的。

```

1  public class NetworkHealthChecker extends BaseHealthChecker
2  {
3      public NetworkHealthChecker (CountDownLatch latch) {
4          super("Network Service", latch);
5      }
6  }

```

```

7  @Override
8  public void verifyService()
9  {
10     System.out.println("Checking " + this.getServiceName());
11     try
12     {
13         Thread.sleep(7000);
14     }
15     catch (InterruptedException e)
16     {
17         e.printStackTrace();
18     }
19     System.out.println(this.getServiceName() + " is UP");
20 }
21 }

```

ApplicationStartupUtil.java: 这个类是一个主启动类，它负责初始化闭锁，然后等待，直到所有服务都被检测完。

```

1  public class ApplicationStartupUtil
2  {
3      //List of service checkers
4      private static List<BaseHealthChecker> _services;
5
6      //This latch will be used to wait on
7      private static CountDownLatch _latch;
8
9      private ApplicationStartupUtil()
10     {
11     }
12
13     private final static ApplicationStartupUtil INSTANCE = new ApplicationStartupUtil();
14
15     public static ApplicationStartupUtil getInstance()
16     {
17         return INSTANCE;
18     }
19
20     public static boolean checkExternalServices() throws Exception
21     {
22         //Initialize the latch with number of service checkers
23         _latch = new CountDownLatch(3);
24
25         //All add checker in lists
26         _services = new ArrayList<BaseHealthChecker>();
27         _services.add(new NetworkHealthChecker(_latch));
28         _services.add(new CacheHealthChecker(_latch));
29         _services.add(new DatabaseHealthChecker(_latch));
30
31         //Start service checkers using executor framework
32         Executor executor = Executors.newFixedThreadPool(_services.size());
33
34         for(final BaseHealthChecker v : _services)
35         {

```



```
36         executor.execute(v);
37     }
38
39     //Now wait till all services are checked
40     _latch.await();
41
42     //Services are file and now proceed startup
43     for(final BaseHealthChecker v : _services)
44     {
45         if( ! v.isServiceUp())
46         {
47             return false;
48         }
49     }
50     return true;
51 }
52 }
```

现在你可以写测试代码去检测一下闭锁的功能了。

```
1 public class Main {
2     public static void main(String[] args)
3     {
4         boolean result = false;
5         try {
6             result = ApplicationStartupUtil.checkExternalServices();
7         } catch (Exception e) {
8             e.printStackTrace();
9         }
10        System.out.println("External services validation completed !! Result was :: "+ result);
11    }
12 }
```

1 Output in console:

```
2
3 Checking Network Service
4 Checking Cache Service
5 Checking Database Service
6 Database Service is UP
7 Cache Service is UP
8 Network Service is UP
9 External services validation completed !! Result was :: true
```



常见面试题

可以为你的下次面试准备以下一些CountDownLatch相关的问题：

- 解释一下CountDownLatch概念？
- CountDownLatch 和CyclicBarrier的不同之处？

- 给出一些CountDownLatch使用的例子?
- CountDownLatch 类中主要的方法?

下载上述例子的源代码，请点击如下链接：

- [源码下载](#)

原文链接: [howtodoinjava](#) 翻译: [ImportNew.com](#) - [张涛](#)

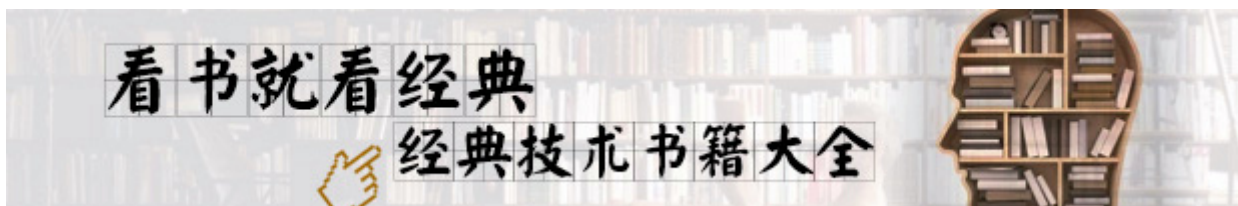
译文链接: <http://www.importnew.com/15731.html>

[**转载请保留原文出处、译者和译文链接。**]

关于作者: 张涛

(新浪微博: [@zthenry](#))

[查看张涛的更多文章 >>](#)



相关文章

- [排名Top 16的Java实用类库](#)
- [工具类与函数编程毫不相干](#)
- [Java8中的java.util.Random类](#)
- [自己编写Java Web框架: Takes框架的Web App架构](#)
- [Netty那点事 \(3\) Channel与Pipeline](#)
- [Java对象的内存布局](#)



- [java提高篇之抽象类与接口](#)
- [深入并发包 ConcurrentHashMap](#)
- [程序员你为什么这么累【续】：编码习惯之 Controller 规范](#)
- [高并发风控技术解密（下）](#)

发表评论

Comment form

Name*

姓名

邮箱*

请填写邮箱

网站 (请以 http://开头)

请填写网站地址

评论内容*

请填写评论内容



(*) 表示必填项

[提交评论](#)

11 条评论

1. *tosswang@163.com* 说道:

[2016/09/19 下午 5:20](#)

挺好的文章

Well-loved. Like or Dislike:  26  2

[回复](#)

o *1东* 说道:

[2017/04/12 下午 9:59](#)

executor未shutdown

Well-loved. Like or Dislike:  22  2

[回复](#)

2. *Threeboys33* 说道:

[2017/07/15 下午 5:25](#)

写的好

 1  0

[回复](#)

3. *reetmoon* 说道:

[2017/08/26 下午 9:47](#)

写的很不错!

 0  0

[回复](#)

4. *gouerli* 说道:



[2017/10/23 下午 3:30](#)

写的很清楚明了。谢谢。

 1  0

[回复](#)

5. 123说道:

[2017/10/25 上午 9:14](#)

写的挺不错的,就是线程池没有关闭

 0  0

[回复](#)

6. tc 说道:

[2017/12/02 下午 11:31](#)

这个在我本机运行的效果 和你这个不太一样呀

 0  0

[回复](#)

◦ 唐尤华说道:

[2017/12/03 上午 9:11](#)

可能和环境有关

 1  0

[回复](#)

7. 态度xiaomi说道:

[2017/12/04 下午 9:31](#)

看起来有点复杂

 1  0



[回复](#)

8. 第三次说道:

[2018/01/05 下午 5:05](#)

最小值

 0  0[回复](#)9. *bricks* 说道:[2018/03/08 下午 12:04](#)

死锁检测那段能具体点吗?

 0  0[回复](#)

« [Guava 教程\(4\): 条件, 多重映射和分片](#)
[工具类与函数编程毫不相干](#) »

Search for:






- [本周热门文章](#)
- [本月热门](#)
- [热门标签](#)

0 [oracle 中 print_table 存储过程...](#)

- 1 [按钮条件逻辑配置化的可选技术方案](#)
- 2 [Map大家族的那点事儿\(3\) : TreeM...](#)
- 3 [一次Java内存泄漏调试的有趣经历](#)
- 4 [Java异常处理的9个最佳实践](#)
- 5 [做一次面向对象的体操：将JSON字...](#)
- 6 [SpringBoot | 第十五章：基于Postm...](#)
- 7 [Map大家族的那点事儿\(1\) : Map](#)
- 8 [Map大家族的那点事儿\(2\) : Abstrac...](#)
- 9 [SpringBoot | 第十六章：web ...](#)



最新评论

- 
Re: [Minor GC、Major GC和Fu...](#)
应该说成：许多 Major GC 是由 Minor GC 引发的 和煜
- 
Re: [SpringBoot | 第十二章：Rab...](#)
@RabbitListener 注解是方法级别的，不能发到Consumer类上面，如果放到上面，则第... jiangnk
- 



Re: [Spring IOC 容器源码分析](#)

请教一下,bean完成初始化后,怎么注册到ioc容器的? 单例实例化对象,作为beandefiniti... 官先森



Re: [探究 Java 虚拟机栈](#)

写得很好 Cxw



Re: [Spring Security_\(二\)...](#)

大神,我有两个问题不解。1,在其他页面跳转到/login,是为了到login页面的,但是在login... wjjxjava



Re: [Spring Security_\(二\)...](#)

好 wjjxjava



Re: [SpringBoot | 第九章: Mybatis...](#)

有时间试试 www.wuliaokankan.cn



Re: [ConcurrentHashMap总结](#)

jdk1.7 rehash那块的“rehash方法中会定位第一个后续所有节点在扩容后index都保持... 罗晴



关于ImportNew

ImportNew 专注于 Java 技术分享。于2012年11月11日 11:11正式上线。是的,这是一个很特别的时刻:)

ImportNew 由两个 Java 关键字 import 和 new 组成,意指:Java 开发者学习新知识的网站。import 可认为是学习和吸收, new 则可认为是新知识、新技术圈子和新朋友.....



联系我们

Email: ImportNew.com@gmail.com

新浪微博: [@ImportNew](#)

推荐微信号



ImportNew 数据分析与开发 算法爱好者

反馈建议: ImportNew.com@gmail.com

广告与商务合作QQ: 2302462408

推荐关注

[小组](#) – 好的话题、有启发的回复、值得信赖的圈子

[头条](#) – 写了文章? 看干货? 去头条!

[相亲](#) – 为IT单身男女服务的征婚传播平台

[资源](#) – 优秀的工具资源导航

[翻译](#) – 活跃 & 专业的翻译小组

[博客](#) – 国内外的精选博客文章

[设计](#) – UI, 网页, 交互和用户体验

[前端](#) – JavaScript, HTML5, CSS

[安卓](#) – 专注Android技术分享

[iOS](#) – 专注iOS技术分享

[Java](#) – 专注Java技术分享

[Python](#) – 专注Python技术分享



