HZ BOOKS
华章IT

资深Java专家10年经验总结，全程案例式讲解，全面介绍Java并发包相关的技术

以浅白的措辞，结合大量实例模拟实际应用场景，全面解析Java并发包中的核心类、API与并发框架的使用

Java
核心技术
系列

Java 并发编程

核心方法与框架

Java Concurrent Programming
Core Method and Frameworks

高洪岩 著

机械工业出版社
China Machine Press

Java 并发编程

核心方法与框架

Java Concurrent Programming
Core Method and Frameworks

高洪岩 著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

Java 并发编程：核心方法与框架 / 高洪岩著 . —北京：机械工业出版社，2016.5
（Java 核心技术系列）

ISBN 978-7-111-53521-8

I. J… II. 高… III. JAVA 语言 – 程序设计 IV. TP312

中国版本图书馆 CIP 数据核字（2016）第 078593 号

Java 并发编程：核心方法与框架

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）
责任编辑：高婧雅 责任校对：殷 虹
印 刷：北京市荣盛彩色印刷有限公司 版 次：2016 年 5 月第 1 版第 1 次印刷
开 本：186mm × 240mm 1/16 印 张：23
书 号：ISBN 978-7-111-53521-8 定 价：79.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88379426 88361066 投稿热线：（010）88379604
购书热线：（010）68326294 88379649 68995259 读者信箱：hzjsj@hzbook.com

Preface 前　言

为什么要写这本书

早在几年前笔者就曾想过整理一份与 Java 并发包有关的稿件。因为市面上所有的 Java 书籍都是以 1 章或 2 章的篇幅介绍 Java 并发包技术，这就导致对 Java 并发包的讲解并不是非常详尽，包含的知识量远远不够，并没有完整覆盖 Java 并发包技术的知识点。但可惜，苦于当时的时间及精力有限，一直没有如愿。

也许是注定的安排，笔者现所在单位是集技术与教育为一体的软件类企业，学员在学习完 JavaSE/JavaEE 之后想探索更深入的技术，比如大数据、分布式、高并发类的专题，就会立即遇到与 Java 并发包中 API 相关的问题。为了带领学员在技术层面上有更高的追求，所以我将 Java 并发包的技术点以教案的方式进行整理，在课堂上与同学们一起进行学习、交流，同学们反响非常强烈。至此，若干年前的心愿终于了却，同学们也很期待这样一本书能出版发行，那样他们就有真正的纸质参考资料了。若这份资料也被其他爱好 Java 并发的朋友们看到，并通过它学到相关知识，那就是我最大的荣幸了。

本书将给读者一个完整的视角，秉承“大道至简”的主导思想，只介绍 Java 并发包开发中最值得关注的内容，希望能抛砖引玉，以个人的一些想法和见解，为读者拓展出更深入、全面的思路。

本书特色

本书尽量减少“啰嗦”式的文字语言，全部用 Demo 式案例来讲解技术点的实现，使读者看到代码及运行结果后就可以知道此项目要解决的是什么问题。类似于网络中 Blog 的风格，可让读者用最短的时间学会此知识点，明白此知识点如何应用，以及在使用时要避免什么。这就像“瑞士军刀”，虽短小，却锋利。本书的目的就是帮读者快速学习并解决问题。

读者对象

- ❑ Java 初级、中级程序员
- ❑ Java 多线程开发者
- ❑ Java 并发开发者
- ❑ 系统架构师
- ❑ 大数据开发者
- ❑ 其他对多线程技术感兴趣的人员

如何阅读本书

在整理本书时，笔者本着实用、易懂的学习原则整理了 10 个章节来介绍 Java 并发包相关的技术。

第 1 章讲解了 Semaphore 和 Exchanger 类的使用，学完本章后，能更好地控制线程间的同步性，以及线程间如何更好、更方便地传输数据。

第 2 章是第 1 章的延伸，主要讲解了 CountDownLatch、CyclicBarrier 类的使用及在 Java 并发包中对并发访问的控制。本章主要包括 Semaphore、CountDownLatch 和 CyclicBarrier 的使用，它们在使用上非常灵活，所以对于 API 的介绍比较详细，为读者学习控制同步打好坚实的基础。

第 3 章是第 2 章的升级，由于 CountDownLatch 和 CyclicBarrier 类都有相应的弊端，所以在 JDK1.7 中新增加了 Phaser 类来解决这些缺点。

第 4 章中讲解了 Executor 接口与 ThreadPoolExecutor 线程池的使用，可以说本章中的知识也是 Java 并发包中主要的应用技术点，线程池技术也在众多的高并发业务环境中使用。掌握线程池能更有效地提高程序运行效率，更好地统筹线程执行的相关任务。

第 5 章中讲解 Future 和 Callable 的使用，接口 Runnable 并不支持返回值，但在有些情况下真的需要返回值，所以 Future 就是用来解决这样的问题的。

第 6 章介绍 Java 并发包中的 CompletionService 的使用，该接口可以增强程序运行效率，因为可以以异步的方式获得任务执行的结果。

第 7 章主要介绍的是 ExecutorService 接口，该接口提供了若干方法来方便地执行业务，是比较常见的工具接口对象。

第 8 章主要介绍计划任务 ScheduledExecutorService 的使用，学完本章可以掌握如何将计划任务与线程池结合使用。

第 9 章主要介绍 Fork-Join 分治编程。分治编程在多核计算机中应用很广，它可以将大

的任务拆分成小的任务再执行，最后再把执行的结果聚合到一起，完全利用多核 CPU 的优势，加快程序运行效率。

第 10 章主要介绍并发集合框架。Java 中的集合在开发项目时占有举足轻重的地位，在 Java 并发包中也提供了在高并发环境中使用的 Java 集合工具类，读者需要着重掌握 Queue 接口的使用。

勘误和支持

由于笔者的水平有限，加之编写时间仓促，书中难免会出现一些错误或者不准确的地方，恳请读者批评指正。笔者邮箱是 279377921@qq.com，期待能够得到你们的真挚反馈，在技术之路上互勉共进。

本书的源代码可以在华章网站（www.hzbook.com）下载。

致谢

感谢所在单位领导的支持与厚爱，使我在技术道路上更有信心。

感谢机械工业出版社华章公司的编辑们始终支持我的写作，是你们的鼓励和帮助引导我顺利完成全部书稿。

高洪岩

目　录 *Contents*

第 1 章 Chapter 1

Semaphore 和 Exchanger 的使用

本书将介绍并发包中常见的并发类的主要 API 方法，掌握这些 API 方法所提供的功能是掌握并发包技术的主要手段，每一个类所提供的功能都是独有的，控制线程的行为也是不同的，这些都要依赖于类中的方法才可以实现。并发工具类中的方法其实并不算少，但它们之间却有着非常相似的功能，所以在学习上可以增加效率，理解起来并不是非常复杂。

作为本书的第 1 章，我将和大家一起交流一下类 Semaphore 和 Exchanger 的使用及其有关 API，类 Semaphore 所提供的功能完全就是 synchronized 关键字的升级版，但它提供的功能更加的强大与方便，主要的作用就是控制线程并发的数量，而这一点，单纯地使用 synchronized 是做不到的。

在本章将介绍 Semaphore 类中的常用 API，方法列表如图 1-1 所示。

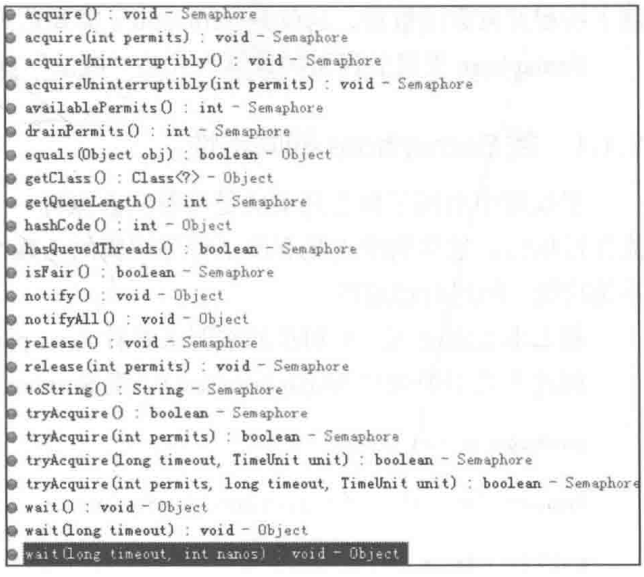


图 1-1 类 Semaphore 中的 API

类 Exchanger 的主要作用可以使 2 个线程之间互相方便地进行通信，它的常用 API 如图 1-2 所示。

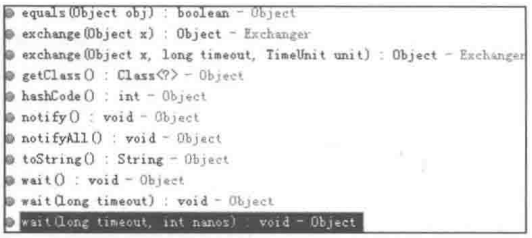


图 1-2 类 Exchanger 中的 API

1.1 Semaphore 的使用

本章将对 Semaphore 类中的全部方法进行案例式的实验，这样可以全面地了解此类提供了哪些核心功能。

单词 Semaphore['seməfɔ:(r)] 的中文含义是信号、信号系统。此类的主要作用就是限制线程并发的数量，如果不限制线程并发的数量，则 CPU 的资源很快就被耗尽，每个线程执行的任务是相当缓慢，因为 CPU 要把时间片分配给不同的线程对象，而且上下文切换也要耗时，最终造成系统运行效率大幅降低，所以限制并发线程的数量还是非常有必要的。

在生活中也存在这种场景，比如一个生产键盘的生产商，发布了 10 个代理销售许可，所以最多只有 10 个代理商来获得其中的一个许可，这样就限制了代理商的数量，同理也限制了线程并发数的数量，这就是 Semaphore 类要达到的目的。

Semaphore 类发放许可的计算方式是“减法”操作。

1.1.1 类 Semaphore 的同步性

多线程中的同步概念其实就是排着队去执行一个任务，执行任务是一个一个执行，并不能并行执行，这样的优点是有助于程序逻辑的正确性，不会出现非线程安全问题，保证软件系统功能上的运行稳定性。

那么本节就使用一个初步的案例来看看 Semaphore 类是如何实现限制线程并发数的。

创建实验用的项目 SemaphoreTest1，类 Service.java 代码如下：

```
package service;

import java.util.concurrent.Semaphore;

public class Service {

    private Semaphore semaphore = new Semaphore(1);

    public void testMethod() {
        try {
            semaphore.acquire();
            System.out.println(Thread.currentThread().getName()
```

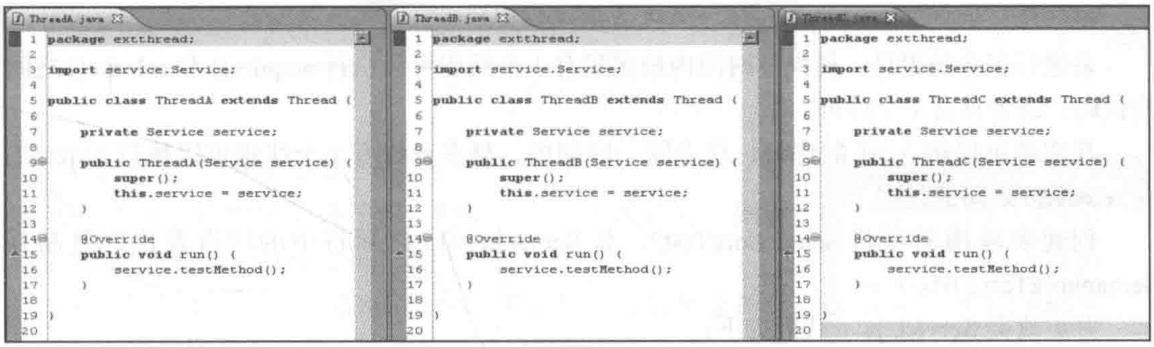
```
                    + " begin timer=" + System.currentTimeMillis());
            Thread.sleep(5000);
            System.out.println(Thread.currentThread().getName()
                    + "    end timer=" + System.currentTimeMillis());
            semaphore.release();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

}
```

类 Semaphore 的构造函数参数 permits 是许可的意思，代表同一时间内，最多允许多少个线程同时执行 acquire() 和 release() 之间的代码。

无参方法 acquire() 的作用是使用 1 个许可，是减法操作。

创建 3 个线程类如图 1-3 所示。



```
ThreadA.java
1  package extthread;
2
3  import service.Service;
4
5  public class ThreadA extends Thread {
6
7      private Service service;
8
9      public ThreadA(Service service) {
10         super();
11         this.service = service;
12     }
13
14     @Override
15     public void run() {
16         service.testMethod();
17     }
18
19 }
20
```

```
ThreadB.java
1  package extthread;
2
3  import service.Service;
4
5  public class ThreadB extends Thread {
6
7      private Service service;
8
9      public ThreadB(Service service) {
10         super();
11         this.service = service;
12     }
13
14     @Override
15     public void run() {
16         service.testMethod();
17     }
18
19 }
20
```

```
1  package extthread;
2
3  import service.Service;
4
5  public class ThreadC extends Thread {
6
7      private Service service;
8
9      public ThreadC(Service service) {
10         super();
11         this.service = service;
12     }
13
14     @Override
15     public void run() {
16         service.testMethod();
17     }
18
19 }
20
```

图 1-3　线程数量为 3

运行类 Run.java 代码如下：

```
package test;

import service.Service;
import extthread.ThreadA;
import extthread.ThreadB;
import extthread.ThreadC;

public class Run {

    public static void main(String[] args) {
        Service service = new Service();
        ThreadA a = new ThreadA(service);
        a.setName("A");
        ThreadB b = new ThreadB(service);
        b.setName("B");
        ThreadC c = new ThreadC(service);
        c.setName("C");
        a.start();
        b.start();
```

```
            c.start();
        }

    }
```

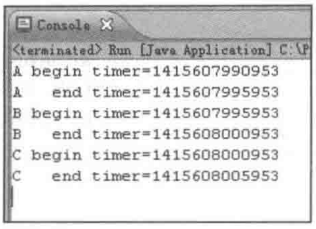


图 1-4 同步运行了

程序运行后的效果如图 1-4 所示。

说明使用代码：

```
private Semaphore semaphore = new Semaphore(1);
```

来定义最多允许 1 个线程执行 acquire() 和 release() 之间的代码，所以打印的结果就是 3 个线程是同步的。

1.1.2 类 Semaphore 构造方法 permits 参数作用

参数 permits 的作用是设置许可的个数，前面已经使用过代码：

```
private Semaphore semaphore = new Semaphore(1);
```

来进行程序的设计，使同一时间内最多只有 1 个线程可以执行 acquire() 和 release() 之间的代码，因为只有 1 个许可。

其实还可以传入 >1 的许可，代表同一时间内，最多允许有 x 个线程可以执行 acquire() 和 release() 之间的代码。

创建实验用的项目 SemaphoreTest2，将 SemaphoreTest1 项目中的所有源代码复制到 SemaphoreTest2 中。

并更改类 Service.java 代码如下：

```
package service;

import java.util.concurrent.Semaphore;

public class Service {

    private Semaphore semaphore = new Semaphore(2);

    public void testMethod() {
        try {
            semaphore.acquire();
            System.out.println(Thread.currentThread().getName()
                    + " begin timer=" + System.currentTimeMillis());
            Thread.sleep(5000);
            System.out.println(Thread.currentThread().getName()
                    + "   end timer=" + System.currentTimeMillis());
            semaphore.release();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

}
```