

[首页](#) [资讯](#) [精华](#) [论坛](#) [问答](#) [博客](#) [专栏](#) [群组](#) [更多 ▼](#)

[您还未登录！](#) [登录](#) [注册](#)

无量的IT生活

- [博客](#)
- [微博](#)
- [相册](#)
- [收藏](#)
- [留言](#)
- [关于我](#)

JAVA进阶----ThreadPoolExecutor机制

博客分类：

- [JAVA进阶](#)
- [java-多线程框架](#)

[javaExecutors并发线程池ThreadPoolExecutor](#)

ThreadPoolExecutor机制

一、概述

- 1、ThreadPoolExecutor作为java.util.concurrent包对外提供基础实现，以内部线程池的形式对外提供管理任务执行，线程调度，线程池管理等服务；
- 2、Executors方法提供的线程服务，都是通过参数设置来实现不同的线程池机制。
- 3、先来了解其线程池管理的机制，有助于正确使用，避免错误使用导致严重故障。同时可以根据自己的需求实现自己的线程池

二、核心构造方法讲解

下面是ThreadPoolExecutor最核心的构造方法

Java代码 ☆

```
1. public ThreadPoolExecutor(int corePoolSize,
2.                           int maximumPoolSize,
3.                           long keepAliveTime,
4.                           TimeUnit unit,
5.                           BlockingQueue<Runnable> workQueue,
6.                           ThreadFactory threadFactory,
7.                           RejectedExecutionHandler handler) {
8.     if (corePoolSize < 0 ||
9.         maximumPoolSize <= 0 ||
10.         maximumPoolSize < corePoolSize ||
11.         keepAliveTime < 0)
12.         throw new IllegalArgumentException();
13.     if (workQueue == null || threadFactory == null || handler == null)
14.         throw new NullPointerException();
15.     this.corePoolSize = corePoolSize;
16.     this.maximumPoolSize = maximumPoolSize;
17.     this.workQueue = workQueue;
18.     this.keepAliveTime = unit.toNanos(keepAliveTime);
19.     this.threadFactory = threadFactory;
20.     this.handler = handler;
21. }
```

构造方法参数讲解

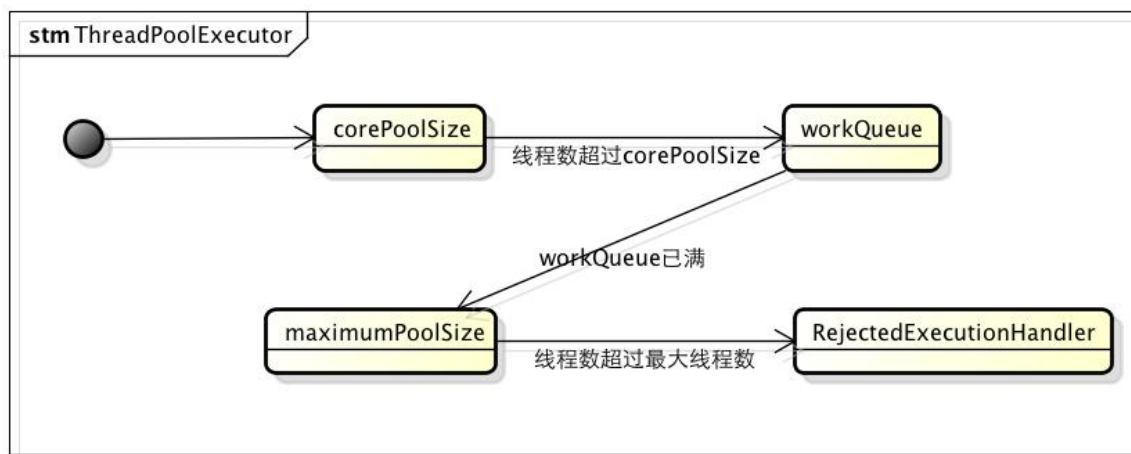
参数名	作用
-----	----

corePoolSize	核心线程池大小
maximumPoolSize	最大线程池大小
keepAliveTime	线程池中超过corePoolSize数目的空闲线程最大存活时间；可以allowCoreThreadTimeOut(true)使得核心线程有效时间
TimeUnit	keepAliveTime时间单位
workQueue	阻塞任务队列
threadFactory	新建线程工厂
RejectedExecutionHandler	当提交任务数超过maximumPoolSize+workQueue之和时，任务会交给RejectedExecutionHandler来处理

重点讲解：

其中比较容易让人误解的是：corePoolSize，maximumPoolSize，workQueue之间关系。

- 1.当线程池小于corePoolSize时，新提交任务将创建一个新线程执行任务，即使此时线程池中存在空闲线程。
- 2.当线程池达到corePoolSize时，新提交任务将被放入workQueue中，等待线程池中任务调度执行
- 3.当workQueue已满，且maximumPoolSize>corePoolSize时，新提交任务会创建新线程执行任务
- 4.当提交任务数超过maximumPoolSize时，新提交任务由RejectedExecutionHandler处理
- 5.当线程池中超过corePoolSize线程，空闲时间达到keepAliveTime时，关闭空闲线程
- 6.当设置allowCoreThreadTimeOut(true)时，线程池中corePoolSize线程空闲时间达到keepAliveTime也将关闭

线程管理机制图示：**三、Executors提供的线程池配置方案**

1、构造一个固定线程数目的线程池，配置的corePoolSize与maximumPoolSize大小相同，同时使用了一个无界LinkedBlockingQueue存放阻塞任务，因此多余的任务将存在再阻塞队列，不会由RejectedExecutionHandler处理

Java代码 ☆

```

1. public static ExecutorService newFixedThreadPool(int nThreads) {
2.     return new ThreadPoolExecutor(nThreads, nThreads,
3.         0L, TimeUnit.MILLISECONDS,
4.         new LinkedBlockingQueue<Runnable>());
5. }
  
```

2、构造一个缓冲功能的线程池，配置corePoolSize=0，maximumPoolSize=Integer.MAX_VALUE，keepAliveTime=60s,以及一个无容量的阻塞队列 SynchronousQueue，因此任务提交之后，将会创建新的线程执行；线程空闲超过60s将会销毁

Java代码 ☆

```

1. public static ExecutorService newCachedThreadPool() {
2.     return new ThreadPoolExecutor(0, Integer.MAX_VALUE,
3.         60L, TimeUnit.SECONDS,
4.         new SynchronousQueue<Runnable>());
  
```

5. }

3、构造一个只支持一个线程的线程池，配置corePoolSize=maximumPoolSize=1，无界阻塞队列LinkedBlockingQueue；保证任务由一个线程串行执行

Java代码 ☆

```
1. public static ExecutorService newSingleThreadExecutor() {
2.     return new FinalizableDelegatedExecutorService
3.         (new ThreadPoolExecutor(1, 1,
4.                                 0L, TimeUnit.MILLISECONDS,
5.                                 new LinkedBlockingQueue<Runnable>()));
6. }
```

4、构造有定时功能的线程池，配置corePoolSize，无界延迟阻塞队列DelayedWorkQueue；有意思的是：maximumPoolSize=Integer.MAX_VALUE，由于DelayedWorkQueue是无界队列，所以这个值是没有意义的

Java代码 ☆

```
1. public static ScheduledExecutorService newScheduledThreadPool(int corePoolSize) {
2.     return new ScheduledThreadPoolExecutor(corePoolSize);
3. }
4.
5. public static ScheduledExecutorService newScheduledThreadPool(
6.     int corePoolSize, ThreadFactory threadFactory) {
7.     return new ScheduledThreadPoolExecutor(corePoolSize, threadFactory);
8. }
9.
10. public ScheduledThreadPoolExecutor(int corePoolSize,
11.     ThreadFactory threadFactory) {
12.     super(corePoolSize, Integer.MAX_VALUE, 0, TimeUnit.NANOSECONDS,
13.         new DelayedWorkQueue(), threadFactory);
14. }
```

四、定制属于自己的非阻塞线程池

Java代码 ☆

```
1. import java.util.concurrent.ArrayBlockingQueue;
2. import java.util.concurrent.ExecutorService;
3. import java.util.concurrent.RejectedExecutionHandler;
4. import java.util.concurrent.ThreadFactory;
5. import java.util.concurrent.ThreadPoolExecutor;
6. import java.util.concurrent.TimeUnit;
7. import java.util.concurrent.atomic.AtomicInteger;
8.
9.
10. public class CustomThreadPoolExecutor {
11.
12.
13.     private ThreadPoolExecutor pool = null;
14.
15.
16.     /**
17.      * 线程池初始化方法
18.      *
19.      * corePoolSize 核心线程池大小----10
20.      * maximumPoolSize 最大线程池大小----30
21.      * keepAliveTime 线程池中超过corePoolSize数目的空闲线程最大存活时间----30+单位TimeUnit
22.      * TimeUnit keepAliveTime时间单位----TimeUnit.MINUTES
```

```
23. * workQueue 阻塞队列----new ArrayBlockingQueue<Runnable>(10)===10容量的阻塞队列
24. * threadFactory 新建线程工厂----new CustomThreadFactory()===定制的线程工厂
25. * rejectedExecutionHandler 当提交任务数超过maxmumPoolSize+workQueue之和时,
26. *           即当提交第41个任务时(前面线程都没有执行完,此测试方法中用sleep(100)),
27. *           任务会交给RejectedExecutionHandler来处理
28. */
29. public void init() {
30.     pool = new ThreadPoolExecutor(
31.         10,
32.         30,
33.         30,
34.         TimeUnit.MINUTES,
35.         new ArrayBlockingQueue<Runnable>(10),
36.         new CustomThreadFactory(),
37.         new CustomRejectedExecutionHandler());
38. }
39.
40.
41. public void destory() {
42.     if(pool != null) {
43.         pool.shutdownNow();
44.     }
45. }
46.
47.
48. public ExecutorService getCustomThreadPoolExecutor() {
49.     return this.pool;
50. }
51.
52. private class CustomThreadFactory implements ThreadFactory {
53.
54.     private AtomicInteger count = new AtomicInteger(0);
55.
56.     @Override
57.     public Thread newThread(Runnable r) {
58.         Thread t = new Thread(r);
59.         String threadName = CustomThreadPoolExecutor.class.getSimpleName() + count.addAndGet(1);
60.         System.out.println(threadName);
61.         t.setName(threadName);
62.         return t;
63.     }
64. }
65.
66.
67. private class CustomRejectedExecutionHandler implements RejectedExecutionHandler {
68.
69.     @Override
70.     public void rejectedExecution(Runnable r, ThreadPoolExecutor executor) {
71.         // 记录异常
72.         // 报警处理等
73.         System.out.println("error.....");
74.     }
75. }
76.
77.
78.
79. // 测试构造的线程池
80. public static void main(String[] args) {
81.     CustomThreadPoolExecutor exec = new CustomThreadPoolExecutor();
```

```

82.    // 1.初始化
83.    exec.init();
84.
85.    ExecutorService pool = exec.getCustomThreadPoolExecutor();
86.    for(int i=1; i<100; i++) {
87.        System.out.println("提交第" + i + "个任务!");
88.        pool.execute(new Runnable() {
89.            @Override
90.            public void run() {
91.                try {
92.                    Thread.sleep(3000);
93.                } catch (InterruptedException e) {
94.                    e.printStackTrace();
95.                }
96.                System.out.println("running====");
97.            }
98.        });
99.    }
100.
101.
102.
103.    // 2.销毁----此处不能销毁,因为任务没有提交执行完,如果销毁线程池,任务也就无法执行了
104.    // exec.destory();
105.
106.    try {
107.        Thread.sleep(10000);
108.    } catch (InterruptedException e) {
109.        e.printStackTrace();
110.    }
111. }
112. }

```

方法中建立一个核心线程数为30个，缓冲队列有10个的线程池。每个线程任务，执行时会先睡眠3秒，保证提交10任务时，线程数目被占用完，再提交30任务时，阻塞队列被占用完，，这样提交第41个任务是，会交给CustomRejectedExecutionHandler 异常处理类来处理。

提交任务的代码如下：

Java代码 ☆

```

1. public void execute(Runnable command) {
2.     if (command == null)
3.         throw new NullPointerException();
4.     /*
5.      * Proceed in 3 steps:
6.      *
7.      * 1. If fewer than corePoolSize threads are running, try to
8.      * start a new thread with the given command as its first
9.      * task. The call to addWorker atomically checks runState and
10.     * workerCount, and so prevents false alarms that would add
11.     * threads when it shouldn't, by returning false.
12.     *
13.     * 2. If a task can be successfully queued, then we still need
14.     * to double-check whether we should have added a thread
15.     * (because existing ones died since last checking) or that
16.     * the pool shut down since entry into this method. So we
17.     * recheck state and if necessary roll back the enqueueing if
18.     * stopped, or start a new thread if there are none.
19.     *
20.     * 3. If we cannot queue task, then we try to add a new

```

```

21.     * thread. If it fails, we know we are shut down or saturated
22.     * and so reject the task.
23.     */
24.     int c = ctl.get();
25.     if (workerCountOf(c) < corePoolSize) {
26.         if (addWorker(command, true))
27.             return;
28.         c = ctl.get();
29.     }
30.     if (isRunning(c) && workQueue.offer(command)) {
31.         int recheck = ctl.get();
32.         if (!isRunning(recheck) && remove(command))
33.             reject(command);
34.         else if (workerCountOf(recheck) == 0)
35.             addWorker(null, false);
36.     }
37.     else if (!addWorker(command, false))
38.         reject(command);
39. }

```

注意：41以后提交的任务就不能正常处理了，因为，execute中提交到任务队列是用的offer方法，如上面代码，这个方法是非阻塞的，所以就会交给CustomRejectedExecutionHandler 来处理，所以对于大数据量的任务来说，这种线程池，如果不设置队列长度会OOM，设置队列长度，会有任务得不到处理，接下来我们构建一个阻塞的自定义线程池

五、定制属于自己的阻塞线程池

Java代码 ☆

```

1. package com.tongbanjie.trade.test.common;
2.
3. import java.util.concurrent.ArrayBlockingQueue;
4. import java.util.concurrent.ExecutorService;
5. import java.util.concurrent.RejectedExecutionHandler;
6. import java.util.concurrent.ThreadFactory;
7. import java.util.concurrent.ThreadPoolExecutor;
8. import java.util.concurrent.TimeUnit;
9. import java.util.concurrent.atomic.AtomicInteger;
10.
11. public class CustomThreadPoolExecutor {
12.
13.
14.     private ThreadPoolExecutor pool = null;
15.
16.
17.     /**
18.      * 线程池初始化方法
19.      *
20.      * corePoolSize 核心线程池大小----1
21.      * maximumPoolSize 最大线程池大小----3
22.      * keepAliveTime 线程池中超过corePoolSize数目的空闲线程最大存活时间----30+单位TimeUnit
23.      * TimeUnit keepAliveTime时间单位----TimeUnit.MINUTES
24.      * workQueue 阻塞队列----new ArrayBlockingQueue<Runnable>(5)===5容量的阻塞队列
25.      * threadFactory 新建线程工厂----new CustomThreadFactory()===定制的线程工厂
26.      * rejectedExecutionHandler 当提交任务数超过maximumPoolSize+workQueue之和时,
27.      * 即当提交第41个任务时(前面线程都没有执行完,此测试方法中用sleep(100)),
28.      * 任务会交给RejectedExecutionHandler来处理
29.      */
30.     public void init() {
31.         pool = new ThreadPoolExecutor(
32.             1,

```

```
33.         3,
34.         30,
35.         TimeUnit.MINUTES,
36.         new ArrayBlockingQueue<Runnable>(5),
37.         new CustomThreadFactory(),
38.         new CustomRejectedExecutionHandler());
39.     }
40.
41.
42.     public void destory() {
43.         if(pool != null) {
44.             pool.shutdownNow();
45.         }
46.     }
47.
48.
49.     public ExecutorService getCustomThreadPoolExecutor() {
50.         return this.pool;
51.     }
52.
53.     private class CustomThreadFactory implements ThreadFactory {
54.
55.         private AtomicInteger count = new AtomicInteger(0);
56.
57.         @Override
58.         public Thread newThread(Runnable r) {
59.             Thread t = new Thread(r);
60.             String threadName = CustomThreadPoolExecutor.class.getSimpleName() + count.addAndGet(1);
61.             System.out.println(threadName);
62.             t.setName(threadName);
63.             return t;
64.         }
65.     }
66.
67.
68.     private class CustomRejectedExecutionHandler implements RejectedExecutionHandler {
69.
70.         @Override
71.         public void rejectedExecution(Runnable r, ThreadPoolExecutor executor) {
72.             try {
73.                 // 核心改造点，由blockingqueue的offer改成put阻塞方法
74.                 executor.getQueue().put(r);
75.             } catch (InterruptedException e) {
76.                 e.printStackTrace();
77.             }
78.         }
79.     }
80.
81.
82.
83.     // 测试构造的线程池
84.     public static void main(String[] args) {
85.
86.         CustomThreadPoolExecutor exec = new CustomThreadPoolExecutor();
87.         // 1.初始化
88.         exec.init();
89.
90.         ExecutorService pool = exec.getCustomThreadPoolExecutor();
91.         for(int i=1; i<100; i++) {
```



```

92.      System.out.println("提交第" + i + "个任务!");
93.      pool.execute(new Runnable() {
94.          @Override
95.          public void run() {
96.              try {
97.                  System.out.println(">>>task is running====");
98.                  TimeUnit.SECONDS.sleep(10);
99.              } catch (InterruptedException e) {
100.                  e.printStackTrace();
101.              }
102.          }
103.      });
104.  }
105.
106.
107.  // 2.销毁----此处不能销毁,因为任务没有提交执行完,如果销毁线程池,任务也就无法执行了
108.  // exec.destory();
109.
110.  try {
111.      Thread.sleep(10000);
112.  } catch (InterruptedException e) {
113.      e.printStackTrace();
114.  }
115. }
116. }

```

解释：当提交任务被拒绝时，进入拒绝机制，我们实现拒绝方法，把任务重新用阻塞提交方法put提交，实现阻塞提交任务功能，防止队列过大，OOM，提交被拒绝方法在下面

Java代码 ☆

```

1. public void execute(Runnable command) {
2.     if (command == null)
3.         throw new NullPointerException();
4.
5.     int c = ctl.get();
6.     if (workerCountOf(c) < corePoolSize) {
7.         if (addWorker(command, true))
8.             return;
9.         c = ctl.get();
10.    }
11.    if (isRunning(c) && workQueue.offer(command)) {
12.        int recheck = ctl.get();
13.        if (!isRunning(recheck) && remove(command))
14.            reject(command);
15.        else if (workerCountOf(recheck) == 0)
16.            addWorker(null, false);
17.    }
18.    else if (!addWorker(command, false))
19.        // 进入拒绝机制，我们把runnable任务拿出来，重新用阻塞操作put，来实现提交阻塞功能
20.        reject(command);
21. }

```

总结：

- 1、用ThreadPoolExecutor自定义线程池，看线程是用途，如果任务量不大，可以用无界队列，如果任务量非常大，要用有界队列，防止OOM
- 2、如果任务量很大，还要求每个任务都处理成功，要对提交的任务进行阻塞提交，重写拒绝机制，改为阻塞提交。保证不抛弃一个任务
- 3、最大线程数一般设为 $2N+1$ 最好，N是CPU核数
- 4、核心线程数，看应用，如果是任务，一天跑一次，设置为0，合适，因为跑完就停掉了，如果是常用线程池，看任务量，是保留一个核心还是几个核心线程数
- 5、如果要获取任务执行结果，用CompletionService，但是注意，获取任务的结果的要重新开一个线程获取，如果在主线程获取，就要等任务都提交后才获取，就会阻塞大量任务结果，队列过大OOM，所以最好异步开个线程获取结果


- [查看图片附件](#)

13

顶

0

踩

分享到：[开发运维工具组件介绍](#) | [JAVA进阶----主线程等待子线程各种方案比较](#)

- 2015-02-09 17:47
- 浏览 59002
- [评论\(9\)](#)
- 分类: [编程语言](#)
- [查看更多](#)

相关资源推荐

[Java数据结构与算法解析\(一\)——表](#)[关注CSDN程序人生公众号，轻松获得下载积分](#)[操作系统OEM DIY工具](#)[一小时学会搭建网站](#)[黑客基础知识大全（TXT）](#)[java编程语言](#)[Java编程语言第三版](#)[ruby 编程语言。集合 J A V A + python 的优势](#)[Java编程语言. \(第三版 \)](#)[java编程语言](#)[Python Crash Course](#)[微信小程序 VS 原生App](#)[教你怎么免费搭建discuz论坛教程](#)[Wi-Fi 爆重大安全漏洞，Android、iOS、Windows 等...](#)[\[编程语言\]《由浅入深学C++-基础、进阶与必做300题...](#)[由Java说起：编程语言还需要开源吗？](#)[JAVA编程语言及其应用](#)[Java编程语言代码规范.pdf](#)[java2参考大全编程语言](#)[Java编程语言](#)

参考知识库

[Java SE知识库](#) 29130 关注 / 578 收录[Java 知识库](#) 36457 关注 / 3748 收录[Java EE知识库](#) 23490 关注 / 1416 收录[JavaScript知识库](#) 17365 关注 / 1517 收录



[jQuery知识库](#) 10637 关注 / 948 收录



[AngularJS知识库](#) 5525 关注 / 590 收录

评论

9 楼 [无量](#) 2017-10-23

lqi 写道

```
private class CustomRejectedExecutionHandler implements RejectedExecutionHandler {

    @Override
    public void rejectedExecution(Runnable r, ThreadPoolExecutor executor) {
        try {
            // 核心改造点，由blockingqueue的offer改成put阻塞方法
            executor.getQueue().put(r);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

楼主这儿有点没明白：当队列满了 走这个 这个时候队里已经满了 还能put进去吗？没明白这个位置的代码 望楼主赐教

put不进去，阻塞在这里

8 楼 [lqi](#) 2017-08-11

```
private class CustomRejectedExecutionHandler implements RejectedExecutionHandler {

    @Override
    public void rejectedExecution(Runnable r, ThreadPoolExecutor executor) {
        try {
            // 核心改造点，由blockingqueue的offer改成put阻塞方法
            executor.getQueue().put(r);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

楼主这儿有点没明白：当队列满了 走这个 这个时候队里已经满了 还能put进去吗？没明白这个位置的代码 望楼主赐教

7 楼 [无量](#) 2017-04-11

zhangfeiyu2005 写道

博主，有个地方我没看懂，想请教下，谢谢！

文中重写了execute方法，为何要这么做呢？重写的execute和自定义的CustomThreadPoolExecutor之间有何联系？

没有重写execute，只是把execute源码贴了上去，上大家看下，知道为什么阻塞的线程池要自己定制

6 楼 [zhangfeiyu2005](#) 2017-04-09

博主，有个地方我没看懂，想请教下，谢谢！

文中重写了execute方法，为何要这么做呢？重写的execute和自定义的CustomThreadPoolExecutor之间有何联系？

5 楼 [bingyingao](#) 2017-04-07

mark一下

4 楼 [studysoft](#) 2017-03-03

studysoft 写道

在JDK8中看到cachedThreadPool已经变了呢.maximumPoolSize就意义了吧。

Java代码 ☆

1. //...

汗, 忙中出错, 抱歉我看错了.

博主是阿里的吧, 楼主的好多文章我在去阿里面试的时候都遇到了.

面试的时候我这个问题就回答错了,赶紧回来补补课.

3 楼 [studysoft](#) 2017-03-03

在JDK8中看到cachedThreadPool已经变了呢.maximumPoolSize就意义了吧.

Java代码 ☆

```
1. public static ExecutorService newCachedThreadPool(ThreadFactory threadFactory) {  
2.     return new ThreadPoolExecutor(0, Integer.MAX_VALUE,  
3.                                   60L, TimeUnit.SECONDS,  
4.                                   new SynchronousQueue<Runnable>(),  
5.                                   threadFactory);  
6. }
```

2 楼 [我才是此去经年](#) 2016-09-05




1 楼 [sjzcmlt](#) 2016-01-26

好牛逼, 谢谢

发表评论

[您还没有登录,请您登录后再发表评论](#)

无量

- 浏览: 562549 次
- 性别: ♂
- 来自: 杭州
-  我现在离线

最近访客

[更多访客>>](#)[大圣可乐](#)[fly to the winds](#)[sioi](#)[ahua186186](#)

文章分类

- [全部博客 \(136\)](#)

- [JAVA基础 \(21\)](#)
- [Spring \(6\)](#)
- [设计模式 \(2\)](#)
- [JDK源码 \(3\)](#)
- [java-功能组件 \(4\)](#)
- [游戏项目 \(2\)](#)
- [linux \(13\)](#)
- [Oracle \(2\)](#)
- [struts \(1\)](#)
- [字符集 \(8\)](#)
- [HTTP协议 \(2\)](#)
- [java-网络通信 \(1\)](#)
- [工具软件推荐 \(2\)](#)
- [tomcat \(1\)](#)
- [java-容器框架 \(2\)](#)
- [java-IO框架 \(2\)](#)
- [java-多线程框架 \(4\)](#)
- [java-NIO框架 \(0\)](#)
- [jquery \(2\)](#)
- [工具使用 \(12\)](#)
- [加密解密 \(1\)](#)
- [redis \(2\)](#)
- [maven \(2\)](#)
- [svn \(1\)](#)
- [eclipse \(1\)](#)
- [mysql \(11\)](#)
- [我的收藏 \(1\)](#)
- [JAVA进阶 \(25\)](#)
- [运维 \(2\)](#)
- [protocol buffer \(1\)](#)
- [优秀博主 \(1\)](#)
- [nginx \(1\)](#)
- [算法 \(1\)](#)
- [故障排查 \(4\)](#)
- [粤语歌曲 \(6\)](#)
- [生活总结 \(5\)](#)
- [高并发 \(4\)](#)
- [语言训练 \(1\)](#)
- [读书笔记 \(5\)](#)
- [诗歌 \(1\)](#)
- [tomcat源码学习 \(1\)](#)
- [软件词汇 \(1\)](#)
- [git \(1\)](#)

社区版块

- [我的资讯 \(0\)](#)
- [我的论坛 \(0\)](#)
- [我的问答 \(0\)](#)

存档分类

- [2017-10 \(1\)](#)
- [2017-05 \(1\)](#)
- [2017-04 \(2\)](#)
- [更多存档...](#)

评论排行榜

- [搭建稳定的开发测试环境](#)

- [一次mysql死锁的排查过程](#)

最新评论

- [无量](#) : lqi 写道 private class CustomRe ...
[JAVA进阶----ThreadPoolExecutor机制](#)
- [lqi](#) : private class CustomRejected ...
[JAVA进阶----ThreadPoolExecutor机制](#)
- [ls8023](#) : 写的不错
[高并发的核心技术-幂等的实现方案](#)
- [无量](#) : linzy410 写道T跟E是两回事请具体讲下
[JAVA基础----java中E,T,?的区别?](#)
- [linzy410](#) : T跟E是两回事
[JAVA基础----java中E,T,?的区别?](#)

声明：ITeye文章版权属于作者，受法律保护。没有作者书面许可不得转载。若作者同意转载，必须以超链接形式标明文章原始出处和作者。

© 2003-2017 ITeye.com. All rights reserved. [京ICP证110151号 京公网安备110105010620]