Java线程池ThreadPoolExecutor

线程池的好处

1. 降低资源的消耗

通过重复利用已创建的线程降低线程创建和销毁所造成的消耗

2. 提高响应速度

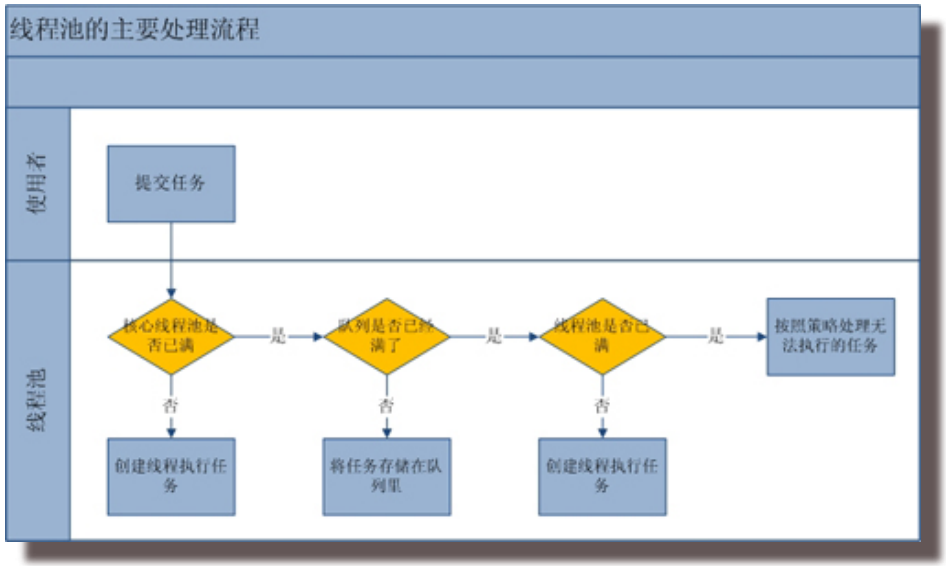
当任务到达时，任务可以不需要等到线程创建就能立即执行

3. 提高线程的可管理型

线程是稀缺资源，如果无限制地创建，不仅会消耗系统资源，还会降低系统的稳定性，使用线程池可以进行统一分配、调优和监控。

实现原理

当提交一个新任务到线程池时，线程池的处理流程为：



1). 线程池判断核心线程池里的线程是否都在执行任务。

如果不是，则创建一个新的工作线程来执行任务。如果核心线程池里的线程都在执行任务，则进入下个流程。

2). 线程池判断工作队列是否已经满。

日 一 二 三 四 五 六
26 27 28 29 30 31 1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 1 2 3 4 5 6

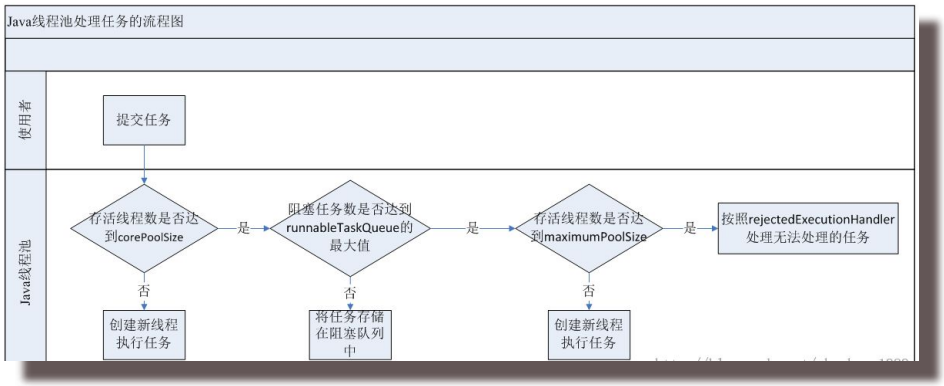
标签
9)
1)
0)
(8)
程(6)
eclipse(4)
Java集合(4)
0 0

如果工作队列没有满，则将新提交的任务存储在这个工作队列里。如果工作队列满了，则进入下个流程。

3). 线程池判断线程池的线程是否都处于工作状态。

如果没有，则创建一个新的工作线程来执行任务。如果已经满了，则交给饱和策略来处理这个任务。

ThreadPoolExecutor执行execute()方法的示意图：



1). 如果当前运行的线程少于corePoolSize，则创建新线程来执行任务（注意，执行这一步骤需要获取全局锁）

2). 如果运行的线程等于或多于corePoolSize，则将任务加入BlockingQueue。

3). 如果无法将任务加入BlockingQueue（队列已满），则创建新的线程来处理任务（注意，执行这一步骤需要获取全局锁）。

更多

积分与排名

积分 - 82167

排名 - 4771

最新评论

生哈希(has

_bin仔

改。...

-明志健致远

AVA程序员

享

啊

！！！

0

4). 如果创建新线程将使当前运行的线程超出maximumPoolSize，任务将被拒绝，并调用RejectedExecutionHandler.rejectedExecution()方法。

ThreadPoolExecutor采取上述步骤的总体设计思路，是为了在执行execute()方法时，尽可能地避免获取全局锁（那将会是一个严重的可伸缩瓶颈）。在ThreadPoolExecutor完成预热之后（当前运行的线程数大于等于corePoolSize），几乎所有的execute()方法调用都是执行步骤2，二步骤2不需要获取全局锁。

源码分析。上面的流程分析让我们很直观的了解的线程池的工作原理，让我们再通过源代码来看看是如何实现的。线程池执行任务的方法如下：

```
1 public void execute(Runnable command) {
2         if (command == null)
3             throw new NullPointerException();
4         // 如果线程数小于基本线程数，则创建线程并执行当前任务
5         if (poolSize >= corePoolSize || !addIfUnderCorePoolSize(command)) {
6             // 如果线程数大于等于基本线程数或线程创建失败，则将
作队列中。
7             if (runState == RUNNING && workQueue.offer
8                 if (runState != RUNNING || poolSize ==
9                     ensureQueuedTaskHandled(command);
10                }
11                // 如果线程池不处于运行中或任务无法放入队列，
量小于最大允许的线程数量，则创建一个线程执行任务
12            } else if (!addIfUnderCorePoolSize(commanc
13                // 抛出 RejectedExecutionException异常
14                reject(command);// is shutdown or satu
15
```

--吃肉不长肉的小灏哥

3. Re:一致性哈希(hash)算法

博主有地方写错了：
原文：...一个int占8个字节，一个
位...
t 4个字
字节8

-humor_bin

非行榜

新消息主...
In

3. 分布式系统
4. Concurren...

0 0

```
16.  16              }
17.  17          }
18.  18      }
```

工作线程。线程池创建线程时，会将线程封装成工作线程Worker，Worker在执行完任务后，还会无限循环获取工作队列里的任务来执行。我们可以从Worker的run方法里看到这点：

```
1.  1  public void run() {
2.  2
3.  3      try {
4.  4
5.  5          Runnable task = firstTask;
6.  6
7.  7          firstTask = null;
8.  8
9.  9           while (task != null || (task = getTask())
10. 10
11. 11                  runTask(task);
12. 12
13. 13                  task = null;
14. 14
15. 15           }
16. 16
17. 17      } finally {
18. 18
19. 19           workerDone(this);
20. 20
21. 21      }
22. 22
23. 23 }
```


0 0

线程池的创建

我们可以通过ThreadPoolExecutor来创建一个线程池。

```
new ThreadPoolExecutor(corePoolSize, maximumPoolSize,
keepAliveTime, milliseconds,runnableTaskQueue, threadFactory,handler);
```

创建一个线程池需要输入几个参数：

- **corePoolSize**（线程池的基本大小）：当提交一个任务到线程池时，线程池会创建一个线程来执行任务，即使其他空闲的基本线程能够执行新任务也会创建线程，等到需大于线程池基本大小时就不再创建。如果调用了线程池的prestartAllCoreThr会提前创建并启动所有基本线程。
- **runnableTaskQueue**（任务队列）：用于保存等待执行的任务的阻塞队列个阻塞队列。

1. **ArrayBlockingQueue**：是一个基于数组结构的有界阻塞队列，此队列按出）原则对元素进行排序。
2. **LinkedBlockingQueue**：一个基于链表结构的阻塞队列，此队列按FIFO序元素，吞吐量通常要高于ArrayBlockingQueue。静态工厂方法Executors.newFixedThreadPool()使用了这个队列。
3. **SynchronousQueue**：一个不存储元素的阻塞队列。每个插入操作必须等用移除操作，否则插入操作一直处于阻塞状态，吞吐量通常要高于LinkedBl态工厂方法Executors.newCachedThreadPool使用了这个队列。
4. **PriorityBlockingQueue**：一个具有优先级得无限阻塞队列。

评论排行榜

书计划（...
础】Java...
使用简介...
定义样式...
希(hash)...
pring的ao...
加载的一...
HTTPS(3)
与API权...
11. Java虚拟

0 0

- **maximumPoolSize**（线程池最大大小）：线程池允许创建的最大线程数。如果队列满了，并且已创建的线程数小于最大线程数，则线程池会再创建新的线程执行任务。值得注意的是如果使用了无界的任务队列这个参数就没什么效果。
- **ThreadFactory**：用于设置创建线程的工厂，可以通过线程工厂给每个创建出来的线程设置更有意义的名字，Debug和定位问题时非常又帮助。

RejectedExecutionHandler（饱和策略）：当队列和线程池都满了，说明线程池处于饱和状态，那么必须采取一种策略处理提交的新任务。这个策略默认情况下是AbortPolicy，表示无法处理新任务时抛出异常。以下是JDK1.5提供的四种策略。n AbortPolicy：直接抛出异常。

1. **CallerRunsPolicy**：只用调用者所在线程来运行任务。
2. **DiscardOldestPolicy**：丢弃队列里最近的一个任务，并执行当前任务。
3. **DiscardPolicy**：不处理，丢弃掉。
4. 当然也可以根据应用场景需要来实现RejectedExecutionHandler接口自定义
   或持久化不能处理的任务。

- **keepAliveTime**（线程活动保持时间）：线程池的工作线程空闲后，保持有
  如果任务很多，并且每个任务执行的时间比较短，可以调大这个时间，提高
- **TimeUnit**（线程活动保持时间的单位）：可选的单位有天（DAYS），小时
  钟（MINUTES），毫秒(MILLISECONDS)，微秒(MICROSECONDS, 千分之
  (NANOSECONDS, 千分之一微秒)。

向线程池提交任务

我们可以使用execute提交的任务，但是execute方法没有返回值，所以无法判
程池执行成功。通过以下代码可知execute方法输入的任务是一个Runnable类

推荐排行榜
书计划（...
定义样式...
中的几个...
新消息主...
entHash...
HTTPS(6)
则表达式...
础】Java...
使用简介...
多...
11. spring中I...
0 0

```
threadsPool.execute(new Runnable() {
@Override

public void run() {

// TODO Auto-generated method stub

}

});
```

我们也可以使用submit 方法来提交任务，它会返回一个future,那么我们可以通过这个future来判断任务是否执行成功，通过future的get方法来获取返回值，get方法会阻塞住直到任务完成，而使用get(long timeout, TimeUnit unit)方法则会阻塞一段时间后立即退出，此时任务没有执行完。

```
try {

  Object s = future.get();

} catch (InterruptedException e) {

// 处理中断异常

} catch (ExecutionException e) {

// 处理无法执行任务异常

} finally {

// 关闭线程池

executor.shutdown();
```


0 0

```
18.
19.     }
```

线程池的关闭

我们可以通过调用线程池的shutdown或shutdownNow方法来关闭线程池，但是它们的实现原理不同，shutdown的原理是只是将线程池的状态设置成SHUTDOWN状态，然后中断所有没有正在执行任务的线程。shutdownNow的原理是遍历线程池中的工作线程，然后逐个调用线程的interrupt方法来中断线程，所以无法响应中断的任务可能永远无法终止。shutdownNow会首先将线程池的状态设置成STOP，然后尝试停止所有的正在执行或暂停任务的线程，并返回等待执行任务的列表。

只要调用了这两个关闭方法的其中一个，isShutdown方法就会返回true。当
关闭后,才表示线程池关闭成功，这时调用isTerminaed方法会返回true。至于
一种方法来关闭线程池，应该由提交到线程池的任务特性决定，通常调用shu
程池，如果任务不一定要执行完，则可以调用shutdownNow。

合理的配置线程池

要想合理的配置线程池，就必须首先分析任务特性，可以从以下几个角度来

1. 任务的性质：CPU密集型任务，IO密集型任务和混合型任务。
2. 任务的优先级：高，中和低。
3. 任务的执行时间：长，中和短。
4. 任务的依赖性：是否依赖其他系统资源，如数据库连接。

0 0

任务性质不同的任务可以用不同规模的线程池分开处理。CPU密集型任务配置尽可能少的线程数量，如配置$N_{cpu}+1$个线程的线程池。IO密集型任务则由于需要等待IO操作，线程并不是一直在执行任务，则配置尽可能多的线程，如$2*N_{cpu}$。混合型的任务，如果可以拆分，则将其拆分成一个CPU密集型任务和一个IO密集型任务，只要这两个任务执行的时间相差不是太大，那么分解后执行的吞吐率要高于串行执行的吞吐率，如果这两个任务执行时间相差太大，则没必要进行分解。我们可以通过Runtime.getRuntime().availableProcessors()方法获得当前设备的CPU个数。

优先级不同的任务可以使用优先级队列PriorityBlockingQueue来处理。它可以让优先级高的任务先得到执行，需要注意的是如果一直有优先级高的任务提交到队列里，那么优先级低的任务可能永远不能执行。

执行时间不同的任务可以交给不同规模的线程池来处理，或者也可以使用优
行时间短的任务先执行。

依赖数据库连接池的任务，因为线程提交SQL后需要等待数据库返回结果，
越长CPU空闲时间就越长，那么线程数应该设置越大，这样才能更好的利用

建议使用有界队列，有界队列能增加系统的稳定性和预警能力，可以根据需
如几千。有一次我们组使用的后台任务线程池的队列和线程池全满了，不断
的异常，通过排查发现是数据库出现了问题，导致执行SQL变得非常缓慢，
程池里的任务全是需要向数据库查询和插入数据的，所以导致线程池里的工
住，任务积压在线程池里。如果当时我们设置成无界队列，线程池的队列就
可能会撑满内存，导致整个系统不可用，而不只是后台任务出现问题。当然
的任务是用的单独的服务器部署的，而我们使用不同规模的线程池跑不同类
出现这样问题时也会影响到其他任务。

线程池的监控

0 0

通过线程池提供的参数进行监控。线程池里有一些属性在监控线程池的时候可以使用

- taskCount：线程池需要执行的任务数量。
- completedTaskCount：线程池在运行过程中已完成的任务数量。小于或等于taskCount。
- largestPoolSize：线程池曾经创建过的最大线程数量。通过这个数据可以知道线程池是否满过。如等于线程池的最大大小，则表示线程池曾经满了。
- getPoolSize:线程池的线程数量。如果线程池不销毁的话，池里的线程不会自动销毁，所以这个大小只增不减。
- getActiveCount：获取活动的线程数。

通过扩展线程池进行监控。通过继承线程池并重写线程池的beforeExecute，afterExecute和terminated方法，我们可以在任务执行前，执行后和线程池关闭前干一些事情。如监控任务的平均执行时间，最大执行时间和最小执行时间等。这几个方法在线程池里

```
1. <b>protected</b> <b>void</b> beforeExecute(Thread t, Runn
```

参考资料

- <<Java并发编程实战>>。
- JDK1.6源码。

更多内容：http://www.cnblogs.com/study-everyday/

♥ 作者：**明志健致远**
♠ 出处：http://www.cnblogs.com/study-everyday/
♦ 本文版权归作者和博客园共有，欢迎转载，但未 同意必须保留此段声明，且在文章页面明显位置给出原文连接，否则保

0 0

留追究法律责任的权利。

♣ 本博客大多为学习笔记或读书笔记，本文如对您有帮助，还请多推荐下此文，如有错误欢迎指正，相互学习，共同进步。

« 上一篇: 深入分析Volatile的实现原理

» 下一篇: 设计模式六大原则（1）：单一职责原则

posted @ 2017-04-14 11:30 明志健致远 阅读(273) 评论(0) 编辑 收藏

注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。

Copyright ©2018 明志健

0 0