

目录

第1章 基础

01 开篇词：为什么学习本专栏

02 String、Long 源码解析和面试题

03 Java 常用关键字理解

04 Arrays、Collections、Objects 常用方法源码解析

第2章 集合

05 ArrayList 源码解析和设计思路

06 LinkedList 源码解析

07 List 源码会问哪些面试题

08 HashMap 源码解析

09 TreeMap 和 LinkedHashMap 核心源码解析

10 Map源码会问哪些面试题

11 HashSet、TreeSet 源码解析

12 彰显细节：看集合源码对我们实际工作的帮助和应用

13 差异对比：集合在 Java 7 和 8 有何不同和改进

14 简化工作：Guava Lists Maps 实际工作运用和源码

第3章 并发集合类

15 CopyOnWriteArrayList 源码解析和设计思路

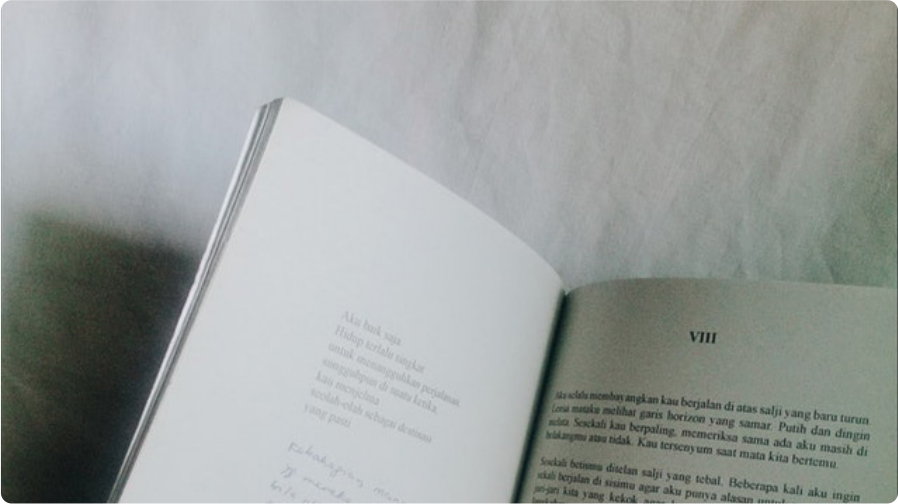
16 ConcurrentHashMap 源码解析和设计思路

17 并发 List、Map源码面试题

18 场景集合：并发 List、Map的应用

37 ThreadPoolExecutor 源码解析

更新时间：2019-11-19 09:53:47



“当你做成功一件事，千万不要等待享受荣誉，应该再做那些需要的事。”
—— 巴斯德

引导语

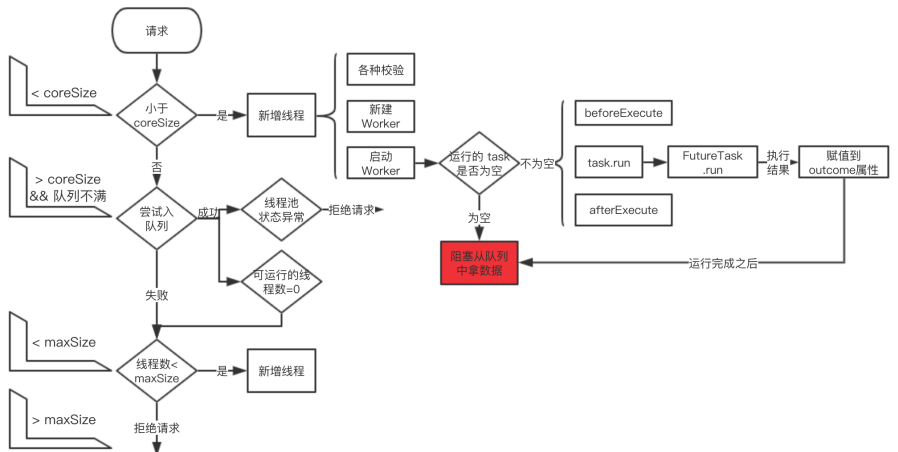
线程池我们在工作中经常会用到。在请求量大时，使用线程池，可以充分利用机器资源，增加请求的处理速度，本章节我们就和大家一起来学习线程池。

本章的基础是第四章队列和第五章线程，没有看过这两章的同学可以先看一看。

本章的顺序，先说源码，弄懂原理，接着看一看面试题，最后看看实际工作中是如何运用线程池的。

1 整体架构图

我们画了线程池的整体图，如下：



目录

第1章 基础

01 开篇词：为什么学习本专栏

02 String、Long 源码解析和面试题

03 Java 常用关键字理解

04 Arrays、Collections、Objects 常用方法源码解析

第2章 集合

05 ArrayList 源码解析和设计思路

06 LinkedList 源码解析

07 List 源码会问哪些面试题

08 HashMap 源码解析

09 TreeMap 和 LinkedHashMap 核心源码解析

10 Map源码会问哪些面试题

11 HashSet、TreeSet 源码解析

12 彰显细节：看集合源码对我们实际工作的帮助和应用

13 差异对比：集合在 Java 7 和 8 有何不同和改进

14 简化工作：Guava Lists Maps 实际工作运用和源码

第3章 并发集合类

15 CopyOnWriteArrayList 源码解析和设计思路

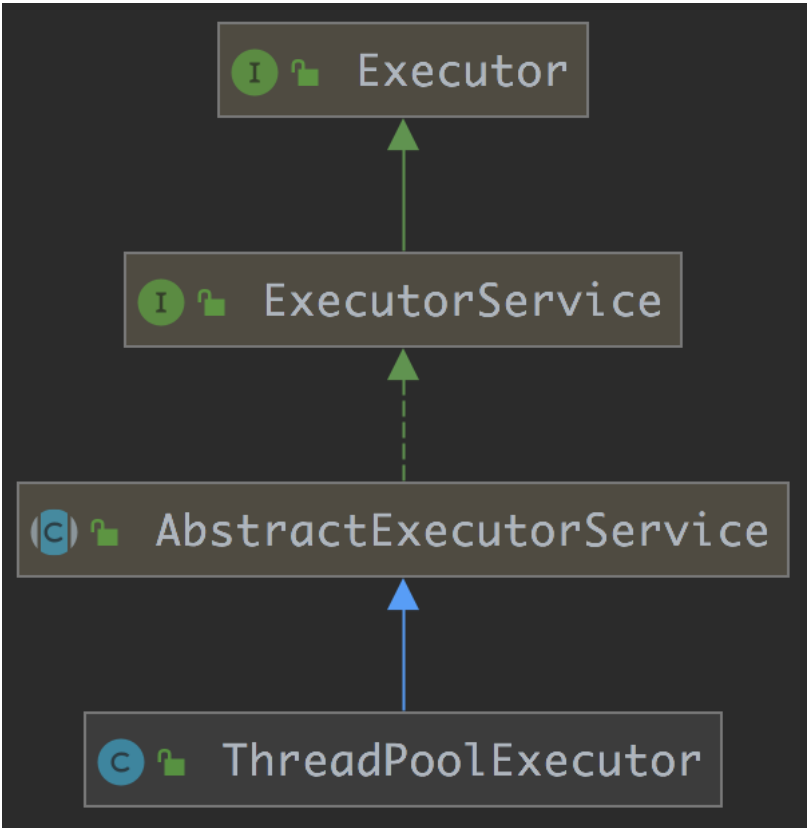
16 ConcurrentHashMap 源码解析和设计思路

17 并发 List、Map源码面试题

18 场景集合：并发 List、Map的应用

1.1 类结构

首先我们来看一下 ThreadPoolExecutor 的类结构，如下图：



从上图中，我们从命名上来看，都有 Executor 的共同命名，Executor 的中文意思为执行的意思，表示对提供的任务进行执行，我们在第五章线程中学习到了几种任务：Runnable、Callable、FutureTask，之前我们都是使用 Thread 来执行这些任务的，除了 Thread，这些 Executor 命名的类和接口也是可以执行这几种任务的，接下来我们大概的看下这几个类的大概含义：

1. Executor：定义 execute 方法来执行任务，入参是 Runnable，无出参：



2. ExecutorService：Executor 的功能太弱，ExecutorService 丰富了对任务的执行和管理的功能，主要代码如下：

```
// 关闭，不会接受新的任务，也不会等待未完成任务
// 如果需要等待未完成任务，可以使用 awaitTermination 方法
void shutdown();
// executor 是否已经关闭了，返回值 true 表示已关闭
boolean isShutdown();
// 所有的任务是否都已经终止，是的话，返回 true
```

<div>← 慕课专栏</div> <div>≡ 面试官系统精讲Java源码及大厂真题 / 37 ThreadPoolExecutor 源码解析</div>	
目录	
第1章 基础	<pre>throws InterruptedException; // 提交有返回值的任务，使用 get 方法可以阻塞等待任务的执行结果返回 <T> Future<T> submit(Callable<T> task); // 提交没有返回值的任务，如果使用 get 方法的话，任务执行完之后得到的是 null 值 Future<?> submit(Runnable task); // 给定任务集合，返回已经执行完成的 Future 集合，每个返回的 Future 都是 isDone = true 的状态 <T> List<Future<T>> invokeAll(Collection<? extends Callable<T>> tasks) throws InterruptedException; // 给定任务中有一个执行成功就返回，如果抛异常，其余未完成的任务将被取消 <T> T invokeAny(Collection<? extends Callable<T>> tasks) throws InterruptedException, ExecutionException;</pre>
01 开篇词：为什么学习本专栏	
02 String、Long 源码解析和面试题	
03 Java 常用关键字理解	
04 Arrays、Collections、Objects 常用方法源码解析	3. AbstractExecutorService 是一个抽象类，封装了 Executor 的很多通用功能，比如：
第2章 集合	<pre>// 把 Runnable 转化成 RunnableFuture // RunnableFuture 是一个接口，实现了 Runnable 和 Future // FutureTask 是 RunnableFuture 的实现类，主要是对任务进行各种管理 // Runnable + Future => RunnableFuture => FutureTask protected <T> RunnableFuture<T> newTaskFor(Runnable runnable, T value) { return new FutureTask<T>(runnable, value); } protected <T> RunnableFuture<T> newTaskFor(Callable<T> callable) { return new FutureTask<T>(callable); } // 提交无返回值的任务 public Future<?> submit(Runnable task) { if (task == null) throw new NullPointerException(); // ftask 其实是 FutureTask RunnableFuture<Void> ftask = newTaskFor(task, null); execute(ftask); return ftask; } // 提交有返回值的任务 public <T> Future<T> submit(Callable<T> task) { if (task == null) throw new NullPointerException(); // ftask 其实是 FutureTask RunnableFuture<T> ftask = newTaskFor(task); execute(ftask); return ftask; }</pre>
05 ArrayList 源码解析和设计思路	
06 LinkedList 源码解析	
07 List 源码会问哪些面试题	
08 HashMap 源码解析	
09 TreeMap 和 LinkedHashMap 核心源码解析	
10 Map源码会问哪些面试题	
11 HashSet、TreeSet 源码解析	
12 彰显细节：看集合源码对我们实际工作的帮助和应用	
13 差异对比：集合在 Java 7 和 8 有何不同和改进	
14 简化工作：Guava Lists Maps 实际工作运用和源码	
第3章 并发集合类	
15 CopyOnWriteArrayList 源码解析和设计思路	
16 ConcurrentHashMap 源码解析和设计思路	
17 并发 List、Map源码面试题	
18 场景集合：并发 List、Map的应用	

<div>← 慕课专栏</div> <div>面试官系统精讲Java源码及大厂真题 / 37 ThreadPoolExecutor 源码解析</div>	
目录	
第1章 基础	
01 开篇词：为什么学习本专栏	
02 String、Long 源码解析和面试题	
03 Java 常用关键字理解	
04 Arrays、Collections、Objects 常用方法源码解析	
第2章 集合	
05 ArrayList 源码解析和设计思路	
06 LinkedList 源码解析	
07 List 源码会问哪些面试题	
08 HashMap 源码解析	
09 TreeMap 和 LinkedHashMap 核心源码解析	
10 Map源码会问哪些面试题	
11 HashSet、TreeSet 源码解析	
12 彰显细节：看集合源码对我们实际工作的帮助和应用	
13 差异对比：集合在 Java 7 和 8 有何不同和改进	
14 简化工作：Guava Lists Maps 实际工作运用和源码	
第3章 并发集合类	
15 CopyOnWriteArrayList 源码解析和设计思路	
16 ConcurrentHashMap 源码解析和设计思路	
17 并发 List、Map源码面试题	
18 场景集合：并发 List、Map的应用	
	<div>2. 线程池解决两个问题：1：通过减少任务间的调度开销 (主要是通过线程池中的线程被重复使用的方式)，来提高大量任务时的执行性能；2：提供了一种方式来管理线程和消费，维护基本数据统计等工作，比如统计已完成的任务数；</div> <div>3. Executors 为常用的场景设定了可直接初始化线程池的方法，比如 Executors#newCachedThreadPool 无界的线程池，并且可以自动回收； Executors#newFixedThreadPool 固定大小线程池； Executors#newSingleThreadExecutor 单个线程的线程池；</div> <div>4. 为了在各种上下文中使用线程池，线程池提供可供扩展的参数设置：1：coreSize：当新任务提交时，发现运行的线程数小于 coreSize，一个新的线程将被创建，即使这时候其它工作线程是空闲的，可以通过 getCorePoolSize 方法获得 coreSize；2：maxSize：当任务提交时，coreSize < 运行线程数 <= maxSize，但队列没有满时，任务提交到队列中，如果队列满了，在 maxSize 允许的范围内新建线程；</div> <div>5. 一般来说，coreSize 和 maxSize 在线程池初始化时就已经设定了，但我们也可以通过 setCorePoolSize、setMaximumPoolSize 方法动态的修改这两个值；</div> <div>6. 默认的，core threads 需要到任务提交后才创建的，但我们可以分别使用 prestartCoreThread、prestartAllCoreThreads 两个方法来提前创建一个、所有的 core threads；</div> <div>7. 新的线程被默认 ThreadFactory 创建时，优先级会被限制成 NORM_PRIORITY，默认会被设置成非守护线程，这个和新建线程的继承是不同的；</div> <div>8. Keep-alive times 参数的作用：1：如果当前线程池中有超过 coreSize 的线程；2：并且线程空闲的时间超过 keepAliveTime，当前线程就会被回收，这样可以避免线程没有被使用时的资源浪费；</div> <div>9. 通过 setKeepAliveTime 方法可以动态的设置 keepAliveTime 的值；</div> <div>10. 如果设置 allowCoreThreadTimeOut 为 true 的话，core thread 空闲时间超过 keepAliveTime 的话，也会被回收；</div> <div>11. 线程池新建时，有多种队列可供选择，比如：1：SynchronousQueue，为了避免任务被拒绝，要求线程池的 maxSize 无界，缺点是当任务提交的速度超过消费的速度时，可能出现无限制的线程增长；2：LinkedBlockingQueue，无界队列，未消费的任务可以在队列中等待；3：ArrayBlockingQueue，有界队列，可以防止资源被耗尽；</div> <div>12. 队列的维护：提供了 getQueue () 方法方便我们进行监控和调试，严禁用于其他目的，remove 和 purge 两个方法可以对队列中的元素进行操作；</div> <div>13. 在 Executor 已经关闭或对最大线程和最大队列都使用饱和时，可以使用 RejectedExecutionHandler 类进行异常捕捉，有如下四种处理策略： ThreadPoolExecutor.AbortPolicy、ThreadPoolExecutor.DiscardPolicy、 ThreadPoolExecutor.CallerRunsPolicy、ThreadPoolExecutor.DiscardOldestPolicy；</div> <div>14. 线程池提供了很多可供扩展的钩子函数，比如有：1：提供在每个任务执行之前 beforeExecute 和执行之后 afterExecute 的钩子方法，主要用于操作执行环境，比如初始化 ThreadLocals、收集统计数据、添加日志条目等；2: 如果在执行器执行完成之后想干一些事情，可以实现 terminated 方法，如果钩子方法执行时发生异常，工作线程可能会失败并立即终止。</div>
	<p>可以看到 ThreadPoolExecutor 的注释是非常多的，也是非常重要的，我们很多面试的题目，在注释上都能找到答案。</p>
	<h3>1.3 ThreadPoolExecutor 重要属性</h3>
	<p>接下来我们来看一看 ThreadPoolExecutor 都有哪些重要属性，如下：</p>

← 慕课专栏	面试官系统精讲Java源码及大厂真题 / 37 ThreadPoolExecutor 源码解析
目录	<div>//2:runState 线程池的状态，提供了生命周期的控制，源码中有很多大于状态的值，状态仅举例</div> <div>//RUNNING (-536870912)：接受新任务或者处理队列里的任务。</div> <div>//SHUTDOWN (0)：不接受新任务，但仍在处理已经在队列里面的任务。</div> <div>//STOP (-536870912)：不接受新任务，也不处理队列中的任务，对正在执行的任务进行中断。</div> <div>//TIDYING (1073741824)：所以任务都被中断，workerCount 是 0，整理状态</div> <div>//TERMINATED (1610612736)：terminated() 已经完成的时候</div> <div> </div> <div>//runState 之间的转变过程：</div> <div>//RUNNING -> SHUTDOWN：调用 shutdown(),finalize()</div> <div>//(RUNNING or SHUTDOWN) -> STOP：调用shutdownNow()</div> <div>//SHUTDOWN -> TIDYING -> workerCount ==0</div> <div>//STOP -> TIDYING -> workerCount ==0</div> <div>//TIDYING -> TERMINATED -> terminated() 执行完成之后</div> <div>private final AtomicInteger ctl = new AtomicInteger(ctlOf(RUNNING, 0));</div> <div>private static final int COUNT_BITS = Integer.SIZE - 3;// 29</div> <div>private static final int CAPACITY = (1 << COUNT_BITS) - 1;// =(2^29)-1=536870911</div> <div> </div> <div>// Packing and unpacking ctl</div> <div>private static int ctlOf(int rs, int wc) { return rs wc; }</div> <div>private static int workerCountOf(int c) { return c & CAPACITY; }</div> <div>private static int runStateOf(int c) { return c & ~CAPACITY; }</div> <div> </div> <div>// runState is stored in the high-order bits</div> <div>private static final int RUNNING = -1 << COUNT_BITS;//-536870912</div> <div>private static final int SHUTDOWN = 0 << COUNT_BITS;//0</div> <div>private static final int STOP = 1 << COUNT_BITS;//-536870912</div> <div>private static final int TIDYING = 2 << COUNT_BITS;//1073741824</div> <div>private static final int TERMINATED = 3 << COUNT_BITS;//1610612736</div> <div> </div> <div>// 已完成任务的计数</div> <div>volatile long completedTasks;</div> <div>// 线程池最大容量</div> <div>private int largestPoolSize;</div> <div>// 已经完成的任务数</div> <div>private long completedTaskCount;</div> <div>// 用户可控制的参数都是 volatile 修饰的</div> <div>// 可以使用 threadFactory 创建 thread</div> <div>// 创建失败一般不抛出异常，只有在 OutOfMemoryError 时候才会</div> <div>private volatile ThreadFactory threadFactory;</div> <div>// 饱和或者运行中拒绝任务的 handler 处理类</div> <div>private volatile RejectedExecutionHandler handler;</div> <div>// 线程存活时间设置</div> <div>private volatile long keepAliveTime;</div> <div>// 设置 true 的话，核心线程空闲 keepAliveTime 时间后，也会被回收</div> <div>private volatile boolean allowCoreThreadTimeOut;</div> <div>// coreSize</div> <div>private volatile int corePoolSize;</div> <div>// maxSize 最大限制 (2^29)-1</div> <div>private volatile int maximumPoolSize;</div> <div>// 默认的拒绝策略</div> <div>private static final RejectedExecutionHandler defaultHandler = new AbortPolicy();</div> <div> </div> <div>// 队列会 hold 住任务，并且利用队列的阻塞的特性，来保持线程的存活周期</div> <div>private final BlockingQueue<Runnable> workQueue;</div> <div> </div> <div>// 大多数情况下是控制对 workers 的访问权限</div> <div>private final ReentrantLock mainLock = new ReentrantLock();</div> <div>private final Condition termination = mainLock.newCondition();</div> <div> </div> <div>// 包含线程池中所有的工作线程</div> <div>private final HashSet<Worker> workers = new HashSet<Worker>();</div>
第1章 基础	
01 开篇词：为什么学习本专栏	
02 String、Long 源码解析和面试题	
03 Java 常用关键字理解	
04 Arrays、Collections、Objects 常用方法源码解析	
第2章 集合	
05 ArrayList 源码解析和设计思路	
06 LinkedList 源码解析	
07 List 源码会问哪些面试题	
08 HashMap 源码解析	
09 TreeMap 和 LinkedHashMap 核心源码解析	
10 Map源码会问哪些面试题	
11 HashSet、TreeSet 源码解析	
12 彰显细节：看集合源码对我们实际工作的帮助和应用	
13 差异对比：集合在 Java 7 和 8 有何不同和改进	
14 简化工作：Guava Lists Maps 实际工作运用和源码	
第3章 并发集合类	
15 CopyOnWriteArrayList 源码解析和设计思路	
16 ConcurrentHashMap 源码解析和设计思路	
17 并发 List、Map源码面试题	
18 场景集合：并发 List、Map的应用	

目录

第1章 基础

01 开篇词：为什么学习本专栏

02 String、Long 源码解析和面试题

03 Java 常用关键字理解

04 Arrays、Collections、Objects 常用方法源码解析

第2章 集合

05 ArrayList 源码解析和设计思路

06 LinkedList 源码解析

07 List 源码会问哪些面试题

08 HashMap 源码解析

09 TreeMap 和 LinkedHashMap 核心源码解析

10 Map源码会问哪些面试题

11 HashSet、TreeSet 源码解析

12 彰显细节：看集合源码对我们实际工作的帮助和应用

13 差异对比：集合在 Java 7 和 8 有何不同和改进

14 简化工作：Guava Lists Maps 实际工作运用和源码

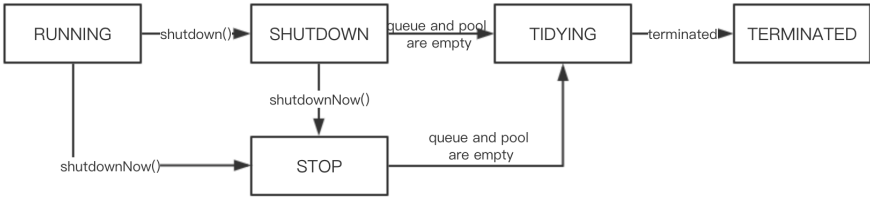
第3章 并发集合类

15 CopyOnWriteArrayList 源码解析和设计思路

16 ConcurrentHashMap 源码解析和设计思路

17 并发 List、Map源码面试题

18 场景集合：并发 List、Map的应用



Worker 我们可以理解成线程池中任务运行的最小单元，Worker 的大致结构如下：

```
// 线程池中任务执行的最小单元
// Worker 继承 AQS，具有锁功能
// Worker 实现 Runnable，本身是一个可执行的任务
private final class Worker
    extends AbstractQueuedSynchronizer
    implements Runnable
{
    // 任务运行的线程
    final Thread thread;

    // 需要执行的任务
    Runnable firstTask;

    // 非常巧妙的设计,Worker本身是个 Runnable,把自己作为任务传递给 thread
    // 内部有个属性又设置了 Runnable
    Worker(Runnable firstTask) {
        setState(-1); // inhibit interrupts until runWorker
        this.firstTask = firstTask;
        // 把 Worker 自己作为 thread 运行的任务
        this.thread = getThreadFactory().newThread(this);
    }

    /** Worker 本身是 Runnable，run 方法是 Worker 执行的入口，runWorker 是外部的方法 */
    public void run() {
        runWorker(this);
    }

    private static final long serialVersionUID = 6138294804551838833L;

    // Lock methods
    // 0 代表没有锁住，1 代表锁住
    protected boolean isHeldExclusively() {
        return getState() != 0;
    }
    // 尝试加锁，CAS 赋值为 1，表示锁住
    protected boolean tryAcquire(int unused) {
        if (compareAndSetState(0, 1)) {
            setExclusiveOwnerThread(Thread.currentThread());
            return true;
        }
        return false;
    }
    // 尝试释放锁，释放锁没有 CAS 校验，可以任意的释放锁
    protected boolean tryRelease(int unused) {
        setExclusiveOwnerThread(null);
        setState(0);
        return true;
    }

    public void lock()    { acquire(1); }
```

<div><div>← 慕课专栏</div><div>面试官系统精讲Java源码及大厂真题 / 37 ThreadPoolExecutor 源码解析</div></div>	
目录	
第1章 基础	
01 开篇词：为什么学习本专栏	
02 String、Long 源码解析和面试题	
03 Java 常用关键字理解	
04 Arrays、Collections、Objects 常用方法源码解析	
第2章 集合	
05 ArrayList 源码解析和设计思路	
06 LinkedList 源码解析	
07 List 源码会问哪些面试题	
08 HashMap 源码解析	
09 TreeMap 和 LinkedHashMap 核心源码解析	
10 Map源码会问哪些面试题	
11 HashSet、TreeSet 源码解析	
12 彰显细节：看集合源码对我们实际工作的帮助和应用	
13 差异对比：集合在 Java 7 和 8 有何不同和改进	
14 简化工作：Guava Lists Maps 实际工作运用和源码	
第3章 并发集合类	
15 CopyOnWriteArrayList 源码解析和设计思路	
16 ConcurrentHashMap 源码解析和设计思路	
17 并发 List、Map源码面试题	
18 场景集合：并发 List、Map的应用	

```
void interruptIfStarted() {
    Thread t;
    if (getState() >= 0 && (t = thread) != null && !t.isInterrupted()) {
        try {
            t.interrupt();
        } catch (SecurityException ignore) {
        }
    }
}
```

理解 Worker 非常关键，主要有以下几点：

- Worker 很像是任务的代理，在线程池中，最小的执行单位就是 Worker，所以 Worker 实现了 Runnable 接口，实现了 run 方法；
- 在 Worker 初始化时 this.thread = getThreadFactory ().newThread (this) 这行代码比较关键，它把当前 Worker 作为线程的构造器入参，我们在后续的实现中会发现这样的代码：Thread t = w.thread;t.start (), 此时的 w 是 Worker 的引用申明，此处 t.start 实际上执行的就是 Worker 的 run 方法；
- Worker 本身也实现了 AQS，所以其本身也是一个锁，其在执行任务的时候，会锁住自己，任务执行完成之后，会释放自己。

2 线程池的任务提交

线程池的任务提交从 submit 方法说起，submit 方法是 AbstractExecutorService 抽象类定义的，主要做了两件事情：

- 把 Runnable 和 Callable 都转化成 FutureTask，这个我们之前看过源码了；
- 使用 execute 方法执行 FutureTask。

execute 方法是 ThreadPoolExecutor 中的方法，源码如下：

```
public void execute(Runnable command) {
    if (command == null)
        throw new NullPointerException();
    int c = ctl.get();
    // 工作的线程小于核心线程数，创建新的线程，成功返回，失败不抛异常
    if (workerCountOf(c) < corePoolSize) {
        if (addWorker(command, true))
            return;
        // 线程池状态可能发生变化
        c = ctl.get();
    }
    // 工作的线程大于等于核心线程数，或者新建线程失败
    // 线程池状态正常，并且可以入队的话，尝试入队列
    if (isRunning(c) && workQueue.offer(command)) {
        int recheck = ctl.get();
        // 如果线程池状态异常 尝试从队列中移除任务，可以移除的话就拒绝掉任务
        if (!isRunning(recheck) && remove(command))
            reject(command);
        // 发现可运行的线程数是 0，就初始化一个线程，这里是个极限情况，入队的时候，突然发现
        // 可用线程都被回收了
        else if (workerCountOf(recheck) == 0)
            // Runnable是空的，不会影响新增线程，但是线程在 start 的时候不会运行
            // Thread.run() 里面有判断
```

面试官系统精讲Java源码及大厂真题 / 37 ThreadPoolExecutor 源码解析

目录

第1章 基础

01 开篇词：为什么学习本专栏

02 String、Long 源码解析和面试题

03 Java 常用关键字理解

04 Arrays、Collections、Objects 常用方法源码解析

第2章 集合

05 ArrayList 源码解析和设计思路

06 LinkedList 源码解析

07 List 源码会问哪些面试题

08 HashMap 源码解析

09 TreeMap 和 LinkedHashMap 核心源码解析

10 Map源码会问哪些面试题

11 HashSet、TreeSet 源码解析

12 彰显细节：看集合源码对我们实际工作的帮助和应用

13 差异对比：集合在 Java 7 和 8 有何不同和改进

14 简化工作：Guava Lists Maps 实际工作运用和源码

第3章 并发集合类

15 CopyOnWriteArrayList 源码解析和设计思路

16 ConcurrentHashMap 源码解析和设计思路

17 并发 List、Map源码面试题

18 场景集合：并发 List、Map的应用

```
else if (!addWorker(command, false))
    reject(command);
}
```

execute 方法执行的就是整体架构图的左半边的逻辑，其中多次调用 addWorker 方法，addWorker 方法的作用是新建一个 Worker，我们一起来看下源码：

```
// 结合线程池的情况看是否可以添加新的 worker
// firstTask 不为空可以直接执行，为空执行不了，Thread.run()方法有判断，Runnable为空不执行
// core 为 true 表示线程最大新增个数是 coresize，false 表示最大新增个数是 maxsize
// 返回 true 代表成功，false 失败
// break retry 跳到retry处，且不再进入循环
// continue retry 跳到retry处，且再次进入循环
private boolean addWorker(Runnable firstTask, boolean core) {
    retry:
    // 先是各种状态的校验
    for (;;) {
        int c = ctl.get();
        int rs = runStateOf(c);
        // Check if queue empty only if necessary.
        // rs >= SHUTDOWN 说明线程池状态不正常
        if (rs >= SHUTDOWN &&
            ! (rs == SHUTDOWN &&
                firstTask == null &&
                ! workQueue.isEmpty()))
            return false;

        for (;;) {
            int wc = workerCountOf(c);
            // 工作中的线程数大于等于容量，或者大于等于 coreSize or maxSize
            if (wc >= CAPACITY ||
                wc >= (core ? corePoolSize : maximumPoolSize))
                return false;
            if (compareAndIncrementWorkerCount(c))
                // break 结束 retry 的 for 循环
                break retry;
            c = ctl.get(); // Re-read ctl
            // 线程池状态被更改
            if (runStateOf(c) != rs)
                // 跳转到retry位置
                continue retry;
            // else CAS failed due to workerCount change; retry inner loop
        }
    }

    boolean workerStarted = false;
    boolean workerAdded = false;
    Worker w = null;
    try {
        // 巧妙的设计，Worker 本身是个 Runnable.
        // 在初始化的过程中，会把 worker 丢给 thread 去初始化
        w = new Worker(firstTask);
        final Thread t = w.thread;
        if (t != null) {
            final ReentrantLock mainLock = this.mainLock;
            mainLock.lock();
            try {
                // Recheck while holding lock.
                // Back out on ThreadFactory failure or if
                // shut down before lock acquired.
```


目录	
第1章 基础	
01 开篇词：为什么学习本专栏	
02 String、Long 源码解析和面试题	
03 Java 常用关键字理解	
04 Arrays、Collections、Objects 常用方法源码解析	
第2章 集合	
05 ArrayList 源码解析和设计思路	
06 LinkedList 源码解析	
07 List 源码会问哪些面试题	
08 HashMap 源码解析	
09 TreeMap 和 LinkedHashMap 核心源码解析	
10 Map源码会问哪些面试题	
11 HashSet、TreeSet 源码解析	
12 彰显细节：看集合源码对我们实际工作的帮助和应用	
13 差异对比：集合在 Java 7 和 8 有何不同和改进	
14 简化工作：Guava Lists Maps 实际工作运用和源码	
第3章 并发集合类	
15 CopyOnWriteArrayList 源码解析和设计思路	
16 ConcurrentHashMap 源码解析和设计思路	
17 并发 List、Map源码面试题	
18 场景集合：并发 List、Map的应用	

```
        if (t.isAlive()) // precheck that t is startable
            throw new IllegalStateException();
        workers.add(w);
        int s = workers.size();
        if (s > largestPoolSize)
            largestPoolSize = s;
        workerAdded = true;
    }
} finally {
    mainLock.unlock();
}
if (workerAdded) {
    // 启动线程，实际上去执行 Worker.run 方法
    t.start();
    workerStarted = true;
}
}
} finally {
    if (!workerStarted)
        addWorkerFailed(w);
}
return workerStarted;
}
```

addWorker 方法首先是执行了一堆校验，然后使用 new Worker (firstTask) 新建了 Worker，最后使用 t.start () 执行 Worker，上文我们说了 Worker 在初始化时的关键代码：this.thread = getThreadFactory ().newThread (this)，Worker (this) 是作为新建线程的构造器入参的，所以 t.start () 会执行到 Worker 的 run 方法上，源码如下：

```
public void run() {
    runWorker(this);
}
```

runWorker 方法是非常重要的方法，我们一起看下源码实现：

```
final void runWorker(Worker w) {
    Thread wt = Thread.currentThread();
    Runnable task = w.firstTask;
    //帮助gc回收
    w.firstTask = null;
    w.unlock(); // allow interrupts
    boolean completedAbruptly = true;
    try {
        // task 为空的情况：
        // 1：任务入队列了，极限情况下，发现没有运行的线程，于是新增一个线程；
        // 2：线程执行完任务执行，再次回到 while 循环。
        // 如果 task 为空，会使用 getTask 方法阻塞从队列中拿数据，如果拿不到数据，会阻塞住
        while (task != null || (task = getTask()) != null) {
            //锁住 worker
            w.lock();
            // 线程池 stop 中,但是线程没有到达中断状态，帮助线程中断
            if ((runStateAtLeast(ctl.get(), STOP) ||
                (Thread.interrupted() &&
                 runStateAtLeast(ctl.get(), STOP))) &&
                !wt.isInterrupted())
                wt.interrupt();
            try {
                //执行 before 钩子函数
```

目录

第1章 基础

01 开篇词：为什么学习本专栏

02 String、Long 源码解析和面试题

03 Java 常用关键字理解

04 Arrays、Collections、Objects 常用方法源码解析

第2章 集合

05 ArrayList 源码解析和设计思路

06 LinkedList 源码解析

07 List 源码会问哪些面试题

08 HashMap 源码解析

09 TreeMap 和 LinkedHashMap 核心源码解析

10 Map源码会问哪些面试题

11 HashSet、TreeSet 源码解析

12 彰显细节：看集合源码对我们实际工作的帮助和应用

13 差异对比：集合在 Java 7 和 8 有何不同和改进

14 简化工作：Guava Lists Maps 实际工作运用和源码

第3章 并发集合类

15 CopyOnWriteArrayList 源码解析和设计思路

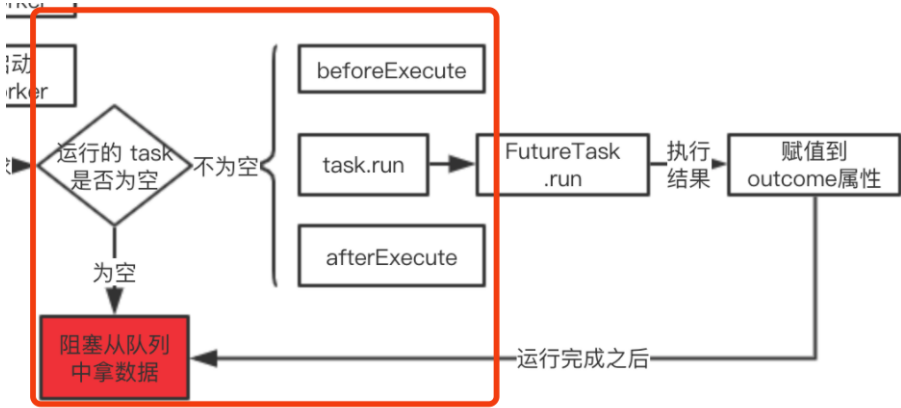
16 ConcurrentHashMap 源码解析和设计思路

17 并发 List、Map源码面试题

18 场景集合：并发 List、Map的应用

```
//同步执行任务
task.run();
} catch (RuntimeException x) {
    thrown = x; throw x;
} catch (Error x) {
    thrown = x; throw x;
} catch (Throwable x) {
    thrown = x; throw new Error(x);
} finally {
    //执行 after 钩子函数,如果这里抛出异常,会覆盖 catch 的异常
    //所以这里异常最好不要抛出来
    afterExecute(task, thrown);
}
} finally {
    //任务执行完成,计算解锁
    task = null;
    w.completedTasks++;
    w.unlock();
}
}
completedAbruptly = false;
} finally {
    //做一些抛出异常的善后工作
    processWorkerExit(w, completedAbruptly);
}
}
```

这个方法执行的逻辑是架构图中的标红部分：



我们聚焦一下这行代码：task.run () 此时的 task 是什么呢？此时的 task 是 FutureTask 类，所以我们继续追索到 FutureTask 类的 run 方法的源码，如下：

```
/**
 * run 方法可以直接被调用
 * 也可以由线程池进行调用
 */
public void run() {
    // 状态不是任务创建, 或者当前任务已经有线程在执行了
    if (state != NEW ||
        !UNSAFE.compareAndSwapObject(this, runnerOffset,
            null, Thread.currentThread()))
        return;
    try {
        Callable<V> c = callable;
        // Callable 不为空, 并且已经初始化完成
        if (c != null && state == NEW) {
```

目录

第1章 基础

01 开篇词：为什么学习本专栏

02 String、Long 源码解析和面试题

03 Java 常用关键字理解

04 Arrays、Collections、Objects 常用方法源码解析

第2章 集合

05 ArrayList 源码解析和设计思路

06 LinkedList 源码解析

07 List 源码会问哪些面试题

08 HashMap 源码解析

09 TreeMap 和 LinkedHashMap 核心源码解析

10 Map源码会问哪些面试题

11 HashSet、TreeSet 源码解析

12 彰显细节：看集合源码对我们实际工作的帮助和应用

13 差异对比：集合在 Java 7 和 8 有何不同和改进

14 简化工作：Guava Lists Maps 实际工作运用和源码

第3章 并发集合类

15 CopyOnWriteArrayList 源码解析和设计思路

16 ConcurrentHashMap 源码解析和设计思路

17 并发 List、Map源码面试题

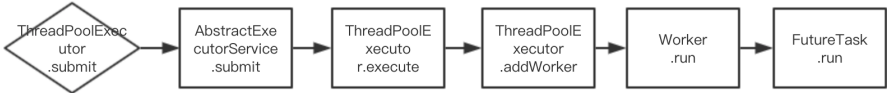
18 场景集合：并发 List、Map的应用

```
// 调用执行
result = c.call();
ran = true;
} catch (Throwable ex) {
    result = null;
    ran = false;
    setException(ex);
}
// 给 outcome 赋值
if (ran)
    set(result);
}
} finally {
    // runner must be non-null until state is settled to
    // prevent concurrent calls to run()
    runner = null;
    // state must be re-read after nulling runner to prevent
    // leaked interrupts
    int s = state;
    if (s >= INTERRUPTING)
        handlePossibleCancellationInterrupt(s);
}
}
```

run 方法中有两行关键代码：

- 1. result = c.call () 这行代码是真正执行业务代码的地方；
- 2. set (result) 这里是给 outCome 赋值，这样 Future.get 方法执行时，就可以从 outCome 中拿值，这个我们在《Future、ExecutorService 源码解析》章节中都有说到。

至此，submit 方法就执行完成了，整体流程比较复杂，我们画一个图释义一下任务提交执行的主流



程：

3 线程执行完任务之后都在干啥

线程执行完任务之后，是消亡还是干什么呢？这是一个值得思考的问题，我们可以从源码中找到答案，从 ThreadPoolExecutor 的 runWorker 方法中，不知道有没有同学注意到一个 while 循环，我们截图释义一下：

<div>← 慕课专栏</div>	<div>面试官系统精讲Java源码及大厂真题 / 37 ThreadPoolExecutor 源码解析</div>
<div>目录</div>	
<div>第1章 基础</div>	
<div>01 开篇词：为什么学习本专栏</div>	
<div>02 String、Long 源码解析和面试题</div>	<div><div>1090</div><div>1091</div><div>1092</div><div>1093</div><div>1094</div><div>1095</div><div>1096</div><div>1097</div><div>1098</div><div>1099</div><div>1100</div><div>1101</div><div>1102</div><div>1103</div><div>1104</div><div>1105</div><div>1106</div><div>1107</div><div>1108</div><div>1109</div><div>1110</div><div>1111</div><div>1112</div><div>Thread wt = Thread.currentThread(); Runnable task = w.firstTask; //帮助gc回收 w.firstTask = null; w.unlock(); // allow interrupts boolean completedAbruptly = true; try { // task 为空的情况:任务入队了,发现没有运行的线程,允许的话会新增一个线程 // 如果 task 为空,会使用 getTask 方法阻塞从队列中拿数据 while (task != null (task = getTask()) != null) { // 锁住worker w.lock(); // 线程执行完任务后,会重复的执行这个 while 循环 // If pool is stopping, ensure thread is interrupted; // if not, ensure thread is not interrupted. This // requires a recheck in second case to deal with // shutdownNow race while clearing interrupt // 线程池stop中,但是线程没有到达中断状态,帮助线程中断 if ((runStateAtLeast(ctl.get(), STOP) (Thread.interrupted() && runStateAtLeast(ctl.get(), STOP))) && !wt.isInterrupted()) wt.interrupt(); try {</div></div>
<div>03 Java 常用关键字理解</div>	
<div>04 Arrays、Collections、Objects 常用方法源码解析</div>	
<div>第2章 集合</div>	
<div>05 ArrayList 源码解析和设计思路</div>	<div>这个 while 循环有个 getTask 方法，getTask 的主要作用是阻塞从队列中拿任务出来，如果队列中有任务，那么就可以拿出来执行，如果队列中没有任务，这个线程会一直阻塞到有任务为止（或者超时阻塞），下面我们一起来看下 getTask 方法，源码如下：</div>
<div>06 LinkedList 源码解析</div>	
<div>07 List 源码会问哪些面试题</div>	
<div>08 HashMap 源码解析</div>	<pre>// 从阻塞队列中拿任务 private Runnable getTask() { boolean timedOut = false; // Did the last poll() time out? for (;;) { int c = ctl.get(); int rs = runStateOf(c); //线程池关闭 && 队列为空,不需要在运行了,直接放回 if (rs >= SHUTDOWN && (rs >= STOP workQueue.isEmpty())) { decrementWorkerCount(); return null; } int wc = workerCountOf(c); // Are workers subject to culling? // true 运行的线程数大于 coreSize 核心线程也可以被灭亡 boolean timed = allowCoreThreadTimeOut wc > corePoolSize; // 队列以 LinkedBlockingQueue 为例, timedOut 为 true 的话说明下面 poll 方法执行返回的 // 说明在等待 keepAliveTime 时间后,队列中仍然没有数据 // 说明此线程已经空闲了 keepAliveTime 了 // 再加上 wc > 1 workQueue.isEmpty() 的判断 // 所以使用 compareAndDecrementWorkerCount 方法使线程池数量减少 1 // 并且直接 return, return 之后,此空闲的线程会自动被回收 if ((wc > maximumPoolSize (timed && timedOut)) && (wc > 1 workQueue.isEmpty())) { if (compareAndDecrementWorkerCount(c)) return null; continue; } try { // 从队列中阻塞拿 worker Runnable r = timed ? workQueue.poll(keepAliveTime, TimeUnit.NANOSECONDS) : workQueue.take(); if (r != null)</pre>
<div>09 TreeMap 和 LinkedHashMap 核心源码解析</div>	
<div>10 Map源码会问哪些面试题</div>	
<div>11 HashSet、TreeSet 源码解析</div>	
<div>12 彰显细节：看集合源码对我们实际工作的帮助和应用</div>	
<div>13 差异对比：集合在 Java 7 和 8 有何不同和改进</div>	
<div>14 简化工作：Guava Lists Maps 实际工作运用和源码</div>	
<div>第3章 并发集合类</div>	
<div>15 CopyOnWriteArrayList 源码解析和设计思路</div>	
<div>16 ConcurrentHashMap 源码解析和设计思路</div>	
<div>17 并发 List、Map源码面试题</div>	
<div>18 场景集合：并发 List、Map的应用</div>	

<div>← 慕课专栏</div> <div>面试官系统精讲Java源码及大厂真题 / 37 ThreadPoolExecutor 源码解析</div>		
<div>目录</div> <div>第1章 基础</div> <div>01 开篇词：为什么学习本专栏</div> <div>02 String、Long 源码解析和面试题</div> <div>03 Java 常用关键字理解</div> <div>04 Arrays、Collections、Objects 常用方法源码解析</div> <div>第2章 集合</div> <div>05 ArrayList 源码解析和设计思路</div> <div>06 LinkedList 源码解析</div> <div>07 List 源码会问哪些面试题</div> <div>08 HashMap 源码解析</div> <div>09 TreeMap 和 LinkedHashMap 核心源码解析</div> <div>10 Map源码会问哪些面试题</div> <div>11 HashSet、TreeSet 源码解析</div> <div>12 彰显细节：看集合源码对我们实际工作的帮助和应用</div> <div>13 差异对比：集合在 Java 7 和 8 有何不同和改进</div> <div>14 简化工作：Guava Lists Maps 实际工作运用和源码</div> <div>第3章 并发集合类</div> <div>15 CopyOnWriteArrayList 源码解析和设计思路</div> <div>16 ConcurrentHashMap 源码解析和设计思路</div> <div>17 并发 List、Map源码面试题</div> <div>18 场景集合：并发 List、Map的应用</div>	<pre> } catch (InterruptedException retry) { timedOut = false; } } }</pre>	
	代码有两处关键：	
	1. 使用队列的 poll 或 take 方法从队列中拿数据，根据队列的特性，队列中有任务可以返回，队列中无任务会阻塞；	
	2. 方法中的第二个 if 判断，说的是在满足一定条件下（条件看注释），会减少空闲的线程，减少的手段是使可用线程数减一，并且直接 return，直接 return 后，该线程就执行结束了，JVM 会自动回收该线程。	
	4 总结	
	本章节主要以 submit 方法为主线阐述了 ThreadPoolExecutor 的整体架构和底层源码，只要有队列和线程的基础知识的话，理解 ThreadPoolExecutor 并不复杂。ThreadPoolExecutor 还有一些其他的源码，比如说拒绝请求的策略、得到各种属性、设置各种属性等等方法，这些方法都比较简单，感兴趣的同学可以自己去看一看。	
	← 36 从容不迫：重写锁的设计结构和细节	38 线程池源码面试题 →
	精选留言 1	
	欢迎在这里发表留言，作者筛选后可公开显示	
	慕桂英1068294	
	老师，private static final int STOP = 1 << COUNT_BITS; 这个常量计算的值是536870912 不是-536870912把。	
	👍 0 回复	2019-11-28
	敲木鱼的小和尚 回复 慕桂英1068294	
	老师写错了吧，是正数的	
	回复	2019-11-29 01:37:16
	千学不如一看，千看不如一练	