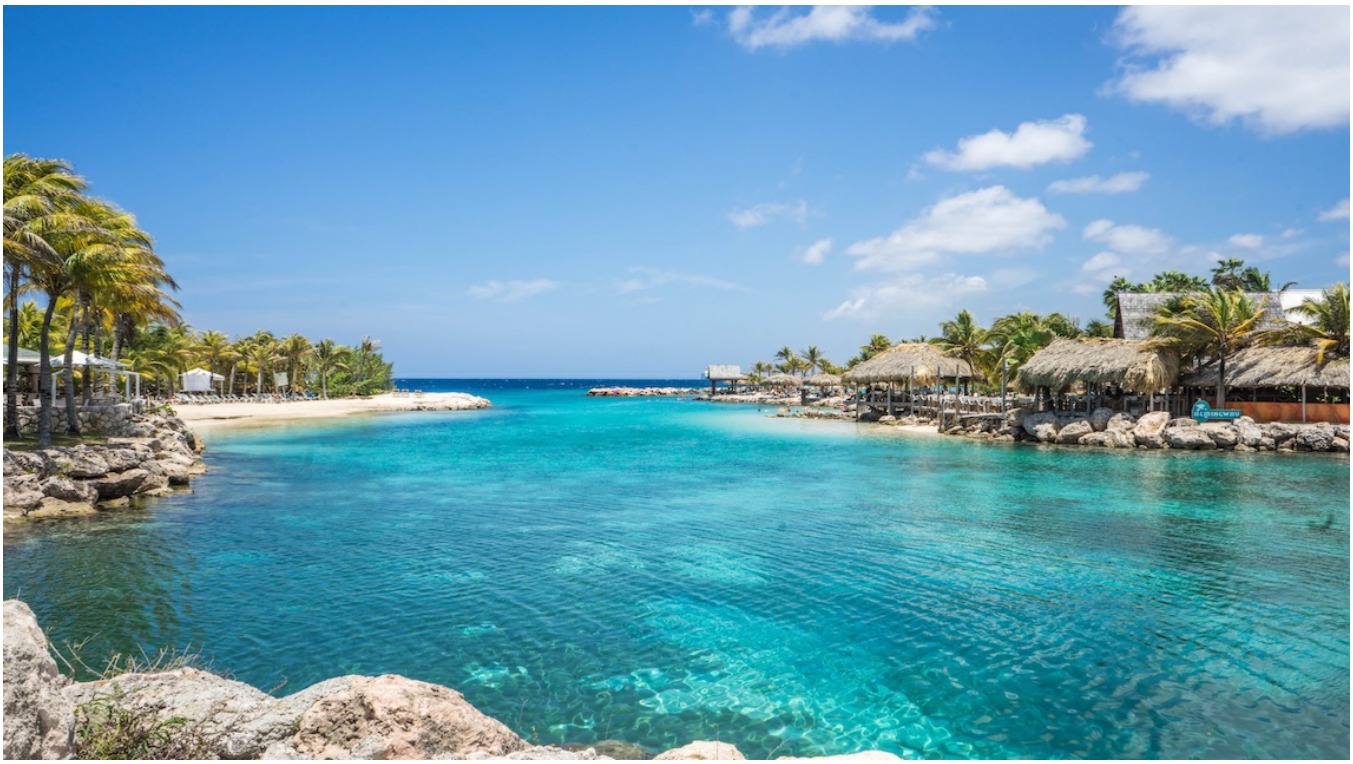
22 | Executor与线程池：如何创建正确的线程池？

2019-04-18 王宝令



虽然在Java语言中创建线程看上去就像创建一个对象一样简单，只需要new Thread()就可以了，但实际上创建线程远不是创建一个对象那么简单。创建对象，仅仅是在JVM的堆里分配一块内存而已；而创建一个线程，却需要调用操作系统内核的API，然后操作系统要为线程分配一系列的资源，这个成本就很高了，所以**线程是一个重量级的对象，应该避免频繁创建和销毁**。

那如何避免呢？应对方案估计你已经知道了，那就是线程池。

线程池的需求是如此普遍，所以Java SDK并发包自然也少不了它。但是很多人在初次接触并发包里线程池相关的工具类时，多少会都有点蒙，不知道该从哪里入手，我觉得根本原因在于线程池和一般意义上的池化资源是不同的。一般意义上的池化资源，都是下面这样，当你需要资源的时候就调用acquire()方法来申请资源，用完之后就调用release()释放资源。若你带着这个固有模型来看并发包里线程池相关的工具类时，会很遗憾地发现它们完全匹配不上，Java提供的线程池里面压根就没有申请线程和释放线程的方法。

```
class XXXPool{
  // 获取池化资源
  XXX acquire() {
  }
  // 释放池化资源
  void release(XXX x){
  }
}
```

线程池是一种生产者-消费者模式

为什么线程池没有采用一般意义上池化资源的设计方法呢？如果线程池采用一般意义上池化资源的设计方法，应该是下面示例代码这样。你可以来思考一下，假设我们获取到一个空闲线程T1，然后该如何使用T1呢？你期望的可能是这样：通过调用T1的execute()方法，传入一个Runnable对象来执行具体业务逻辑，就像通过构造函数Thread(Runnable target)创建线程一样。可惜的是，你翻遍Thread对象的所有方法，都不存在类似execute(Runnable target)这样的公共方法。

```
//采用一般意义上池化资源的设计方法
class ThreadPool{
  // 获取空闲线程
  Thread acquire() {
  }
  // 释放线程
  void release(Thread t){
  }
}
//期望的使用
ThreadPool pool;
Thread T1=pool.acquire();
//传入Runnable对象
T1.execute(()->{
  //具体业务逻辑
  ......
});
```

所以，线程池的设计，没有办法直接采用一般意义上池化资源的设计方法。那线程池该如何设计

呢？目前业界线程池的设计，普遍采用的都是**生产者-消费者模式**。线程池的使用方是生产者，线程池本身是消费者。在下面的示例代码中，我们创建了一个非常简单的线程池 MyThreadPool，你可以通过它来理解线程池的工作原理。

```
//简化的线程池，仅用来说明工作原理
class MyThreadPool{
  //利用阻塞队列实现生产者-消费者模式
  BlockingQueue<Runnable> workQueue;
  //保存内部工作线程
  List<WorkerThread> threads
    = new ArrayList<>();
  // 构造方法
  MyThreadPool(int poolSize,
    BlockingQueue<Runnable> workQueue){
    this.workQueue = workQueue;
    // 创建工作线程
    for(int idx=0; idx<poolSize; idx++){
      WorkerThread work = new WorkerThread();
      work.start();
      threads.add(work);
    }
  }
  // 提交任务
  void execute(Runnable command){
    workQueue.put(command);
  }
  // 工作线程负责消费任务，并执行任务
  class WorkerThread extends Thread{
    public void run() {
      //循环取任务并执行
      while(true){ ①
        Runnable task = workQueue.take();
        task.run();
      }
    }
  }
}
```

```
/** 下面是使用示例 **/
// 创建有界阻塞队列
BlockingQueue<Runnable> workQueue =
  new LinkedBlockingQueue<>(2);
// 创建线程池
MyThreadPool pool = new MyThreadPool(
  10, workQueue);
// 提交任务
pool.execute(()->{
    System.out.println("hello");
});
```

在MyThreadPool的内部，我们维护了一个阻塞队列workQueue和一组工作线程，工作线程的个数由构造函数中的poolSize来指定。用户通过调用execute()方法来提交Runnable任务，execute()方法的内部实现仅仅是将任务加入到workQueue中。MyThreadPool内部维护的工作线程会消费workQueue中的任务并执行任务，相关的代码就是代码①处的while循环。线程池主要的工作原理就这些，是不是还挺简单的？

如何使用Java中的线程池

Java并发包里提供的线程池，远比我们上面的示例代码强大得多，当然也复杂得多。Java提供的线程池相关的工具类中，最核心的是**ThreadPoolExecutor**，通过名字你也能看出来，它强调的是Executor，而不是一般意义上的池化资源。

ThreadPoolExecutor的构造函数非常复杂，如下面代码所示，这个最完备的构造函数有7个参数。

```
ThreadPoolExecutor(
  int corePoolSize,
  int maximumPoolSize,
  long keepAliveTime,
  TimeUnit unit,
  BlockingQueue<Runnable> workQueue,
  ThreadFactory threadFactory,
  RejectedExecutionHandler handler)
```

下面我们一一介绍这些参数的意义，你可以**把线程池类比为一个项目组，而线程就是项目组**

的成员。

- **corePoolSize**：表示线程池保有的最小线程数。有些项目很闲，但是也不能把人都撤了，至少要留corePoolSize个人坚守阵地。
- **maximumPoolSize**：表示线程池创建的最大线程数。当项目很忙时，就需要加人，但是也不能无限制地加，最多就加到maximumPoolSize个人。当项目闲下来时，就要撤人了，最多能撤到corePoolSize个人。
- **keepAliveTime & unit**：上面提到项目根据忙闲来增减人员，那在编程世界里，如何定义忙和闲呢？很简单，一个线程如果在一段时间内，都没有执行任务，说明很闲，keepAliveTime 和 unit 就是用来定义这个“一段时间”的参数。也就是说，如果一个线程空闲了keepAliveTime & unit这么久，而且线程池的线程数大于 corePoolSize ，那么这个空闲的线程就要被回收了。
- **workQueue**：工作队列，和上面示例代码的工作队列同义。
- **threadFactory**：通过这个参数你可以自定义如何创建线程，例如你可以给线程指定一个有意义的名字。
- **handler**：通过这个参数你可以自定义任务的拒绝策略。如果线程池中所有的线程都在忙碌，并且工作队列也满了（前提是工作队列是有界队列），那么此时提交任务，线程池就会拒绝接收。至于拒绝的策略，你可以通过handler这个参数来指定。ThreadPoolExecutor已经提供了以下4种策略。
  - CallerRunsPolicy：提交任务的线程自己去执行该任务。
  - AbortPolicy：默认的拒绝策略，会throws RejectedExecutionException。
  - DiscardPolicy：直接丢弃任务，没有任何异常抛出。
  - DiscardOldestPolicy：丢弃最老的任务，其实就是把最早进入工作队列的任务丢弃，然后把新任务加入到工作队列。

Java在1.6版本还增加了 allowCoreThreadTimeOut(boolean value) 方法，它可以让所有线程都支持超时，这意味着如果项目很闲，就会将项目组的成员都撤走。

使用线程池要注意些什么

考虑到ThreadPoolExecutor的构造函数实在是有些复杂，所以Java并发包里提供了一个线程池的静态工厂类Executors，利用Executors你可以快速创建线程池。不过目前大厂的编码规范中基本上都不建议使用Executors了，所以这里我就不再花篇幅介绍了。

不建议使用Executors的最重要的原因是：Executors提供的很多方法默认使用的都是无界的LinkedBlockingQueue，高负载情境下，无界队列很容易导致OOM，而OOM会导致所有请求都无法处理，这是致命问题。所以**强烈建议使用有界队列**。

使用有界队列，当任务过多时，线程池会触发执行拒绝策略，线程池默认的拒绝策略会throw RejectedExecutionException 这是个运行时异常，对于运行时异常编译器并不强制catch它，所以开发人员很容易忽略。因此**默认拒绝策略要慎重使用**。如果线程池处理的任务非常重要，建

议自定义自己的拒绝策略；并且在实际工作中，自定义的拒绝策略往往和降级策略配合使用。

使用线程池，还要注意异常处理的问题，例如通过ThreadPoolExecutor对象的execute()方法提交任务时，如果任务在执行的过程中出现运行时异常，会导致执行任务的线程终止；不过，最致命的是任务虽然异常了，但是你却获取不到任何通知，这会让你误以为任务都执行得很正常。虽然线程池提供了很多用于异常处理的方法，但是最稳妥和简单的方案还是捕获所有异常并按需处理，你可以参考下面的示例代码。

```
try {
  //业务逻辑
} catch (RuntimeException x) {
  //按需处理
} catch (Throwable x) {
  //按需处理
}
```

总结

线程池在Java并发编程领域非常重要，很多大厂的编码规范都要求必须通过线程池来管理线程。线程池和普通的池化资源有很大不同，线程池实际上是生产者-消费者模式的一种实现，理解生产者-消费者模式是理解线程池的关键所在。

创建线程池设置合适的线程数非常重要，这部分内容，你可以参考《10 | Java线程（中）：创建多少线程才是合适的？》的内容。另外《Java并发编程实战》的第7章《取消与关闭》的7.3节“处理非正常的线程终止”详细介绍了异常处理的方案，第8章《线程池的使用》对线程池的使用也有更深入的介绍，如果你感兴趣或有需要的话，建议你仔细阅读。

课后思考

使用线程池，默认情况下创建的线程名字都类似pool-1-thread-2这样，没有业务含义。而很多情况下为了便于诊断问题，都需要给线程赋予一个有意义的名字，那你知道有哪些办法可以给线程池里的线程指定名字吗？

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

精选留言

undifined

14

思考题：

1. 给线程池设置名称前缀

```
ThreadPoolTaskExecutor threadPoolTaskExecutor = new ThreadPoolTaskExecutor();
threadPoolTaskExecutor.setThreadNamePrefix("CUSTOM_NAME_PREFIX");
```

2. 在ThreadFactory中自定义名称前缀

```
class CustomThreadFactory implements ThreadFactory {
@Override
public Thread newThread(Runnable r) {
Thread thread = new Thread("CUSTOM_NAME_PREFIX");
return thread;
}
}

ThreadPoolExecutor threadPoolExecutor = new ThreadPoolExecutor(10,
100,
120,
TimeUnit.SECONDS,
new LinkedBlockingQueue<>(),
new CustomThreadFactory(),
```

new ThreadPoolExecutor.AbortPolicy()
);

2019-04-18

Seven Blue 9

回答下Lrwin和张天屹同学的问题：当线程池中无可用线程，且阻塞队列已满，那么此时就会触发拒绝策略。对于采用何种策略，具体要看执行的任务重要程度。如果是一些不重要任务，可以选择直接丢弃。但是如果为重要任务，可以采用降级处理，例如将任务信息插入数据库或者消息队列，启用一个专门用作补偿的线程池去进行补偿。所谓降级就是在服务无法正常提供功能的情况下，采取的补救措施。具体采用何种降级手段，这也是要看具体场景。技术的世界里没有一尘不变的方案。另外，看到很多同学都提到让老师多讲讲源码，其实我觉得真没必要，老师目前的思路起到提纲契领的作用，让我们有大的思路，有全局观，具体细节我觉得大家私下去研究更合适。小弟不才，可以加微信（SevenBlue）一起讨论。

2019-04-22

作者回复

我觉得那些源码用的时候看一下就可以了，现在都是用开源项目，天天都得看源码，看源码能局部最优而已

2019-04-22

任大鹏 6

老师的文中已经给出了一个答案：
threadFactory：通过这个参数你可以自定义如何创建线程，例如你可以给线程指定一个有意义的名字。

2019-04-18

西西弗与卡夫卡 3

线程命名常用方法是：线程的构造函数传入名字，或者调用setName设置

2019-04-18

摇山樵客™ 2

guava的ThreadFactoryBuilder.setNameFormat可以指定一个前缀，使用%d表示序号；
或者自己实现ThreadFactory并制定给线程池，在实现的ThreadFactory中设定计数和调用Thread.setName

2019-04-18

作者回复

2019-04-18

magict4 2

老师您好，请问有什么推荐的替代 Executors 的方案吗？

2019-04-18

作者回复

我也没用其他的

2019-04-19

张天屹 2

老师你好，使用有界队列虽然避免了OOM 但是如果请求量太大，我又不想丢弃和异常的情况下一般怎么实践呢。我对降级这一块没经验，我能直观想到的就是存放在缓存，如果缓存内存也不够了就只能持久化了

2019-04-18

作者回复

可以放数据库，放mq,redis，本地文件都可以，具体要看实际需求

2019-04-30

随风 1

老师，有个问题一直不是很明确，①一个项目中如果多个业务需要用到线程池，是定义一个公共的线程池比较好，还是按照业务定义各自不同的线程池？②如果定义一个公共的线程池那里面的线程数的理论值应该是按照老师前面章节讲的去计算吗？还是按照如果有多少个业务就分别去计算他们各自创建线程池线程数的加和?③如果不同的业务各自定义不同的线程池，那线程数的理论值也是按照前面的去计算吗？

2019-04-29

作者回复

建议不同类别的业务用不同的线程池，至于线程池的数量，各自计算各自的，然后去做压测。虽然你的系统有多个线程池，但是并不是所有的线程池里的线程都是忙碌的，你只需要针对有性能瓶颈的业务优化就可以了。

2019-04-29

Zach_ 1

老师，有一个问题想问一下:

如果corePoolSize为10，maxinumPoolSize为20，而此时线程池中有15个线程在运行，过了一段时间后，其中有3个线程处于等待状态的时间超过keepAliveTime指定的时间，则结束这3个线程，此时线程池中则还有12个线程正在运行；若有六个线程处于等待状态的时间超过keepAliveTime指定的时间，则只会结束5个线程，此时线程池中则还有10个线程，即核心线程数。

是这样吗？

2019-04-22

作者回复

是的

2019-04-22

Red Cape 1

请问老师，有界队列的长度怎么确定呢

2019-04-22

作者回复

看场景，拍脑门

2019-04-22

曾轼麟

1

```
public class ReNameThreadFactory implements ThreadFactory {
/**
* 线程池编号（static修饰）(容器里面所有线程池的数量)
*/
private static final AtomicInteger POOLNUMBER = new AtomicInteger(1);

/**
* 线程编号(当前线程池线程的数量)
*/
private final AtomicInteger threadNumber = new AtomicInteger(1);

/**
* 线程组
*/
private final ThreadGroup group;

/**
* 业务名称前缀
*/
private final String namePrefix;


/**
* 重写线程名称（获取线程池编号，线程编号，线程组）
*
* @param prefix 你需要指定的业务名称
*/
public ReNameThreadFactory(@NonNull String prefix) {
SecurityManager s = System.getSecurityManager();
group = (s != null) ? s.getThreadGroup() :
Thread.currentThread().getThreadGroup();
//组装线程前缀
namePrefix = prefix + "-poolNumber:" + POOLNUMBER.getAndIncrement() + "-threadNumber:"
;
}


@Override
public Thread newThread(Runnable r) {
Thread t = new Thread(group, r,
//方便dump的时候排查（重写线程名称）
namePrefix + threadNumber.getAndIncrement(),
```

```
0);
if (t.isDaemon()) {
t.setDaemon(false);
}
if (t.getPriority() != Thread.NORM_PRIORITY) {
t.setPriority(Thread.NORM_PRIORITY);
}
return t;
}
}
```

2019-04-21

作者回复

[]

2019-04-28

密码123456 1

有个问题，不能理解。既然execute使用newfixedthreadpool设置固定的线程池。在实际使用execute执行并发任务，cpu利用率会过高。按照道理说，只有开始的时候，线程会创建消耗资源。在创建之后都不会消耗资源才对啊？

2019-04-19

晓杰 1

希望老师把线程异常处理这块可以再深入讲一讲

2019-04-19

海鸿 1

1.利用guava的ThreadFactoryBuilder
2.自己实现ThreadFactory

2019-04-18

作者回复

[]

2019-04-19

Lrwin 1

如果线程池处理的任务非常重要，建议自定义自己的拒绝策略；并且在实际工作中，自定义的拒绝策略往往和降级策略配合使用。

老师，请问这个怎么理解？能举个例子吗？

2019-04-18

张三 1

打卡！

2019-04-18

郑晨Cc

1

可参照SDK中的 DefaultThreadFactory 自定义DYIThreadFactory

```
static class DIYThreadFactory implements ThreadFactory {
private static final AtomicInteger poolNumber = new AtomicInteger(1);
private final ThreadGroup group;
private final AtomicInteger threadNumber = new AtomicInteger(1);
private final String namePrefix;

DIYThreadFactory(String diyName) {
SecurityManager s = System.getSecurityManager();
group = (s != null) ? s.getThreadGroup() :
Thread.currentThread().getThreadGroup();
namePrefix = diyName +
"-thread-";
}

public Thread newThread(Runnable r) {
Thread t = new Thread(group, r,
namePrefix + threadNumber.getAndIncrement(),
0);
if (t.isDaemon())
t.setDaemon(false);
if (t.getPriority() != Thread.NORM_PRIORITY)
t.setPriority(Thread.NORM_PRIORITY);
return t;
}
}

ExecutorService executor = Executors.newFixedThreadPool(4,new DIYThreadFactory("xxx"));
```

2019-04-18

作者回复

[]

2019-04-30

闫循鸣

0

线程池是 消费者拿到生产者生产的线程对象调用run方法 就是固定的几个父进程开启关闭几个子进程

2019-06-13

魏斌斌

0

老师，线程池里面用到了阻塞队列，当队列满的时候提交任务，不是会挂起生产者线程吗？

2019-06-13

suke

0

老师,能不能介绍下java中的线程和具体操作系统层面的线程的关系

2019-05-27