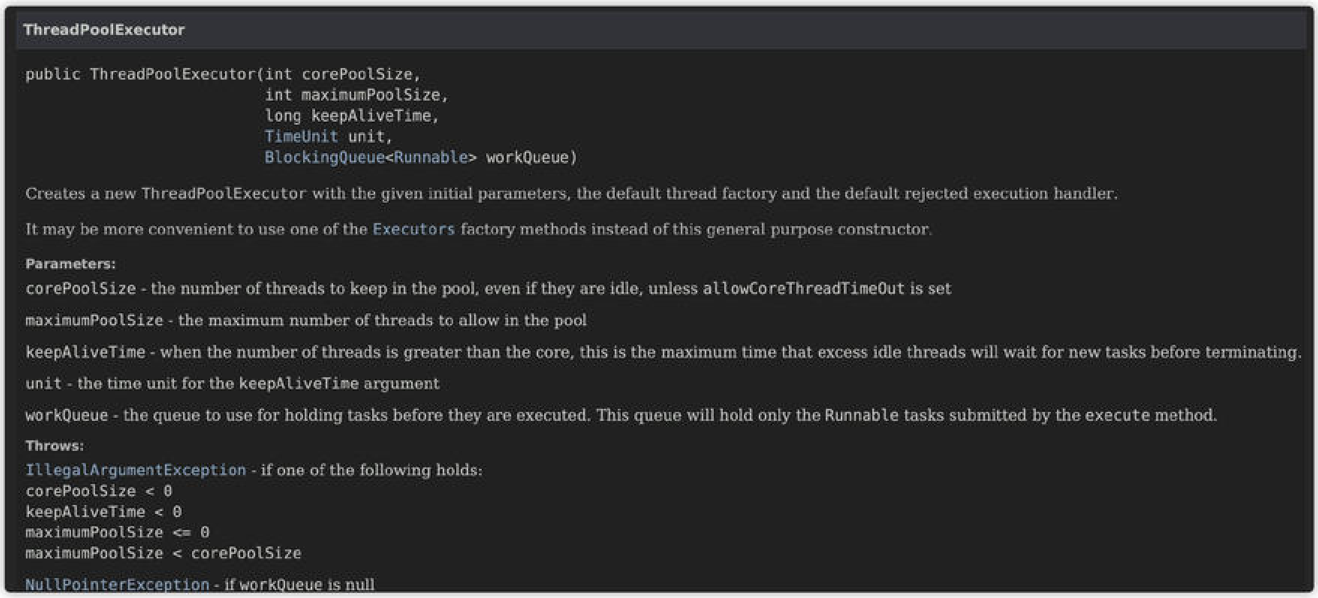
又又又踩坑了

慎用 Executors 组件


ThreadPoolExecutor

```
public ThreadPoolExecutor(int corePoolSize,
                          int maximumPoolSize,
                          long keepAliveTime,
                          TimeUnit unit,
                          BlockingQueue<Runnable> workQueue)
```

Creates a new `ThreadPoolExecutor` with the given initial parameters, the default thread factory and the default rejected execution handler.

It may be more convenient to use one of the `Executors` factory methods instead of this general purpose constructor.

Parameters:

`corePoolSize` - the number of threads to keep in the pool, even if they are idle, unless `allowCoreThreadTimeOut` is set

`maximumPoolSize` - the maximum number of threads to allow in the pool

`keepAliveTime` - when the number of threads is greater than the core, this is the maximum time that excess idle threads will wait for new tasks before terminating.

`unit` - the time unit for the `keepAliveTime` argument

`workQueue` - the queue to use for holding tasks before they are executed. This queue will hold only the `Runnable` tasks submitted by the `execute` method.

Throws:

`IllegalArgumentException` - if one of the following holds:

```
corePoolSize < 0
keepAliveTime < 0
maximumPoolSize <= 0
maximumPoolSize < corePoolSize
```

`NullPointerException` - if `workQueue` is null

非常贴心

OOM

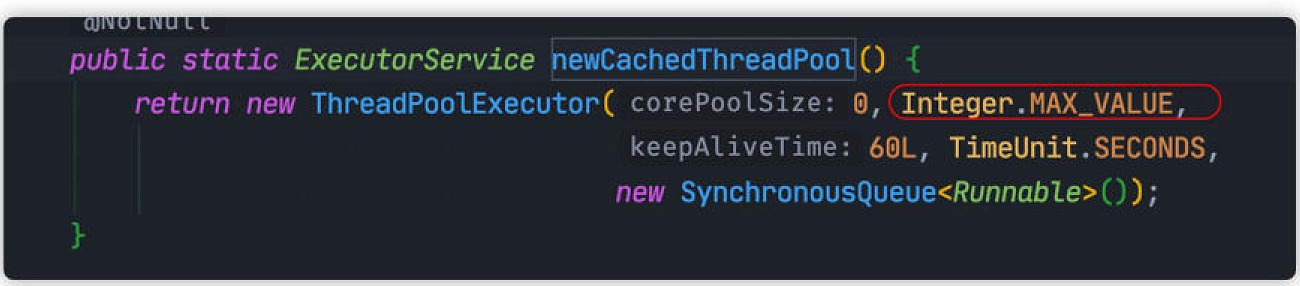
```
Exception: java.lang.OutOfMemoryError thrown from the UncaughtExceptionHandler in thread "http-nio-8081-ClientPoller"
```

OOM

OOM

```
java.lang.OutOfMemoryError: unable to create native thread: possibly out of memory or process/resource limits reached
    at java.base/java.lang.Thread.start0(Native Method) [na:na]
    at java.base/java.lang.Thread.start(Thread.java:813) [na:na]
```

OOM

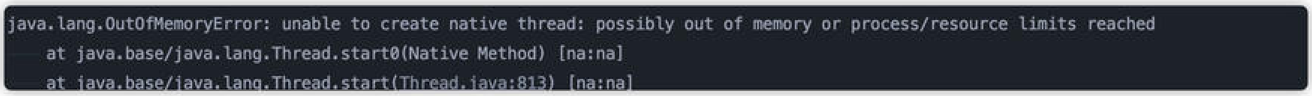


```java
@NotNull
public static ExecutorService newCachedThreadPool() {
    return new ThreadPoolExecutor( corePoolSize: 0, Integer.MAX_VALUE,
                                   keepAliveTime: 60L, TimeUnit.SECONDS,
                                  new SynchronousQueue<Runnable>());
}
```

OOM

复用线程池

WRK OOM



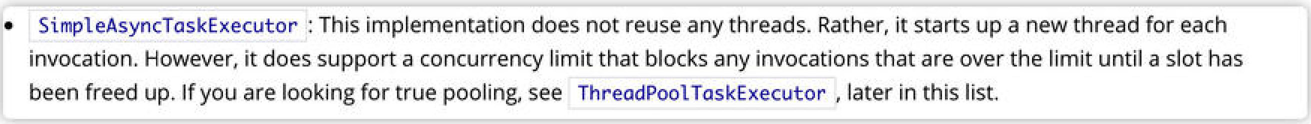
```
java.lang.OutOfMemoryError: unable to create native thread: possibly out of memory or process/resource limits reached
	at java.base/java.lang.Thread.start0(Native Method) [na:na]
	at java.base/java.lang.Thread.start(Thread.java:813) [na:na]
```

OOM

Spring 异步任务

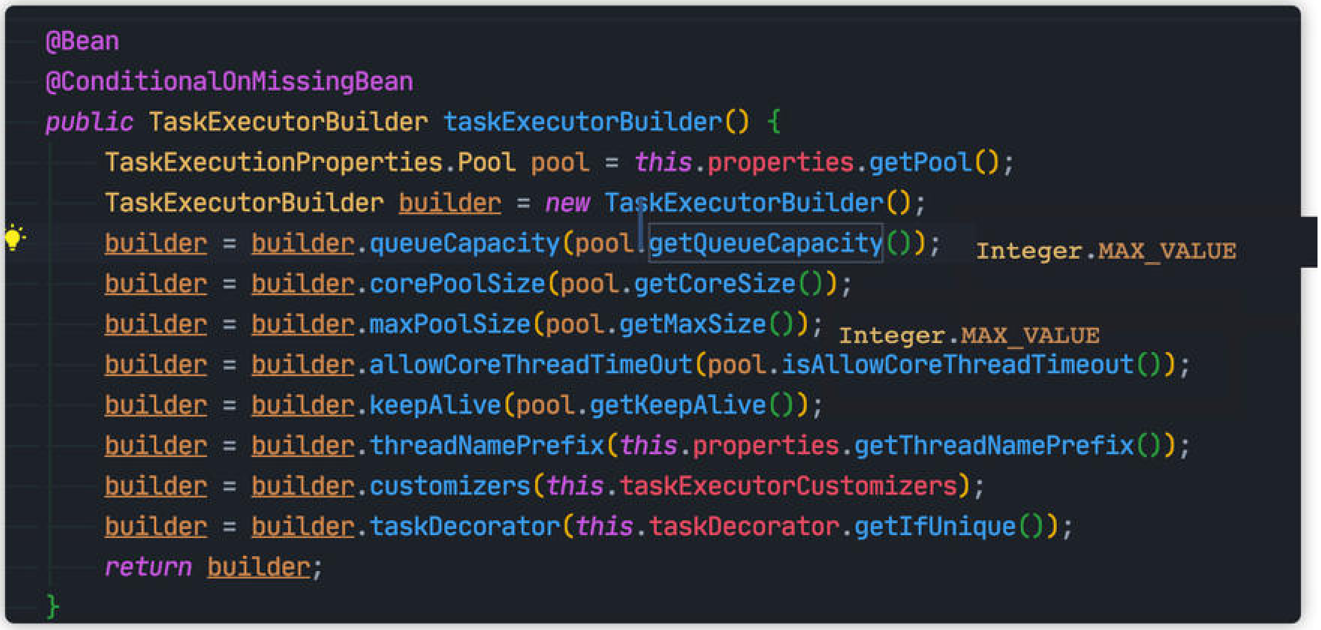
@Async

OOM



- `SimpleAsyncTaskExecutor`: This implementation does not reuse any threads. Rather, it starts up a new thread for each invocation. However, it does support a concurrency limit that blocks any invocations that are over the limit until a slot has been freed up. If you are looking for true pooling, see `ThreadPoolTaskExecutor`, later in this list.

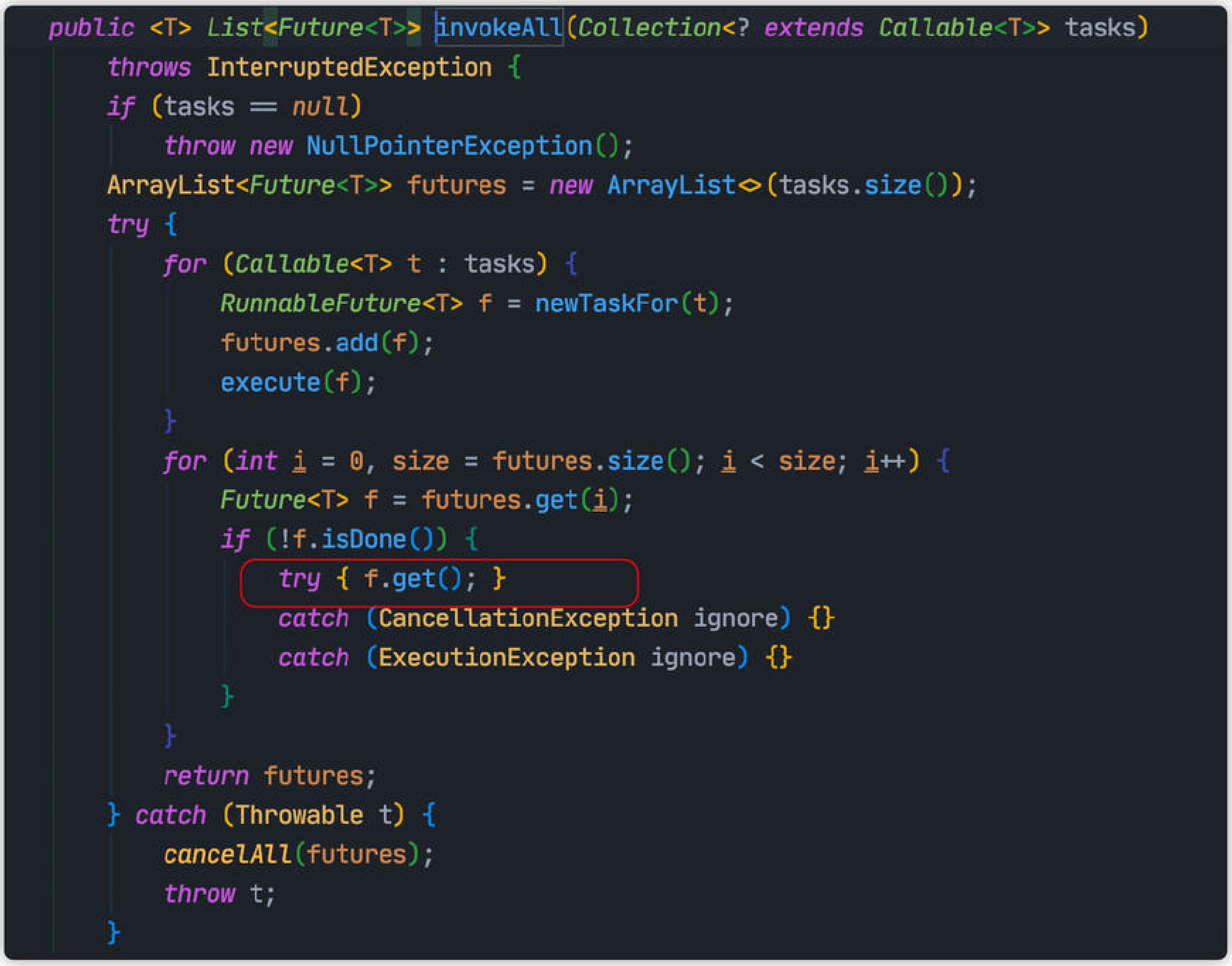
复用线程



```
@Bean
@ConditionalOnMissingBean
public TaskExecutorBuilder taskExecutorBuilder() {
    TaskExecutionProperties.Pool pool = this.properties.getPool();
    TaskExecutorBuilder builder = new TaskExecutorBuilder();
    builder = builder.queueCapacity(pool.getQueueCapacity());  Integer.MAX_VALUE
    builder = builder.corePoolSize(pool.getCoreSize());
    builder = builder.maxPoolSize(pool.getMaxSize()); Integer.MAX_VALUE
    builder = builder.allowCoreThreadTimeOut(pool.isAllowCoreThreadTimeout());
    builder = builder.keepAlive(pool.getKeepAlive());
    builder = builder.threadNamePrefix(this.properties.getThreadNamePrefix());
    builder = builder.customizers(this.taskExecutorCustomizers);
    builder = builder.taskDecorator(this.taskDecorator.getIfUnique());
    return builder;
}
```

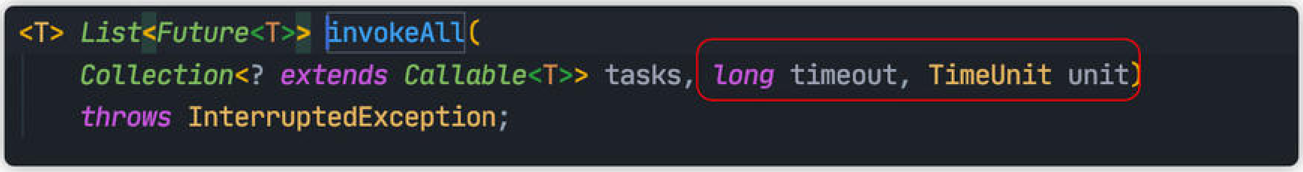
OOM

线程池方法使用不当



```java
public <T> List<Future<T>> invokeAll(Collection<? extends Callable<T>> tasks)
    throws InterruptedException {
    if (tasks == null)
        throw new NullPointerException();
    ArrayList<Future<T>> futures = new ArrayList<>(tasks.size());
    try {
        for (Callable<T> t : tasks) {
            RunnableFuture<T> f = newTaskFor(t);
            futures.add(f);
            execute(f);
        }
        for (int i = 0, size = futures.size(); i < size; i++) {
            Future<T> f = futures.get(i);
            if (!f.isDone()) {
                try { f.get(); }
                catch (CancellationException ignore) {}
                catch (ExecutionException ignore) {}
            }
        }
        return futures;
    } catch (Throwable t) {
        cancelAll(futures);
        throw t;
    }
```

Socket



```java
<T> List<Future<T>> invokeAll(
    Collection<? extends Callable<T>> tasks, long timeout, TimeUnit unit)
    throws InterruptedException;
```

总结

OOM

片面理解

最后最后（点个在看呗）

▯ ▯

所以，所以，可以来个在看吗~

所以，所以，可以来个在看吗~

所以，所以，可以来个在看吗~