SpringBoot + 虚拟线程，鸟枪换大炮！

“虚拟”线程，望文生义，它是“假”的，它不直接调度操作系统的线程，而是由JVM再提供一层线程的接口抽象，由普通线程调度，即一个普通的操作系统线程可以调度成千上万个虚拟线程。

虚拟线程比普通线程的消耗要小得多得多，在内存足够的情况下，我们甚至可以创建上百万的虚拟线程，这在之前(Java19以前)是不可能的。

>
>
>
> 其实如果有用过akka的朋友们会发现，其实两者很相似，只不过使用akka是应用程序来处理，而虚拟线程是JVM来处理，使用上更简洁且方便。
>
>
>

下面我们会在SpringBoot中使用虚拟线程，将默认的异步线程池和http处理线程池替换为虚拟线程，然后对比虚拟线程和普通线程的性能差异，你会发现差别就像马车换高铁，不是一个时代的东西。

配置

首先我们使用的Java版本是java-20.0.2-oracle，SpringBoot版本是3.1.2。

要在SpringBoot中使用虚拟线程很简单，增加如下配置即可：

```
/**
 * 配置是用于稍后测试，spring.virtual-thread=true是使用虚拟线程，false时还是使用默认的普通线程
 */
@Configuration
```

```
@ConditionalOnProperty(prefix = "spring", name = "virtual-
thread", havingValue = "true")
public class ThreadConfig {

    @Bean
    public AsyncTaskExecutor applicationTaskExecutor() {
        return new TaskExecutorAdapter(Executors.newVirtualThreadPerTa
skExecutor());
    }

    @Bean
    public TomcatProtocolHandlerCustomizer<?> protocolHandlerCustomi
zer() {
        return protocolHandler -> {
            protocolHandler.setExecutor(Executors.newVirtualThreadPerTas
kExecutor());
        };
    }
}
```

@Async性能对比

我们写一个异步service，里面睡眠50ms，模拟MySQL或Redis等IO操作：

```
@Service
public class AsyncService {

    /**
     *
     * @param countDownLatch 用于测试
     */
    @Async
    public void doSomething(CountDownLatch countDownLatch) throws I
nterruptedException {
        Thread.sleep(50);
        countDownLatch.countDown();
    }
}
```

最后测试类，很简单，就是循环调用这个方法10万次，计算所有方法执行完成的消耗的时间：

```
@Test
public void testAsync() throws InterruptedException {
    long start = System.currentTimeMillis();
    int n = 100000;
    CountDownLatch countDownLatch = new CountDownLatch(n);
    for (int i = 0; i < n; i++) {
        asyncService.doSomething(countDownLatch);
    }
    countDownLatch.await();
    long end = System.currentTimeMillis();
    System.out.println("耗时：" + (end - start) + "ms");
}
```

普通线程耗时：678秒左右，超过10分钟了

图片

虚拟线程耗时：3.9秒!!

图片

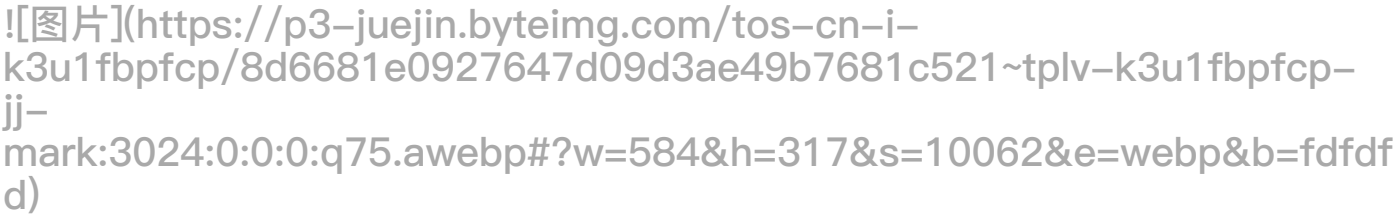
朋友们，接近200倍的性能差距！！

HTTP请求性能对比

让我们再看看http请求的对比，简单写个get请求，里面什么也不做，一样睡50ms，模拟IO操作：

```
@RequestMapping(”/get”)
public Object get() throws Exception {
    Thread.sleep(50);
    return ”ok”;
}
```

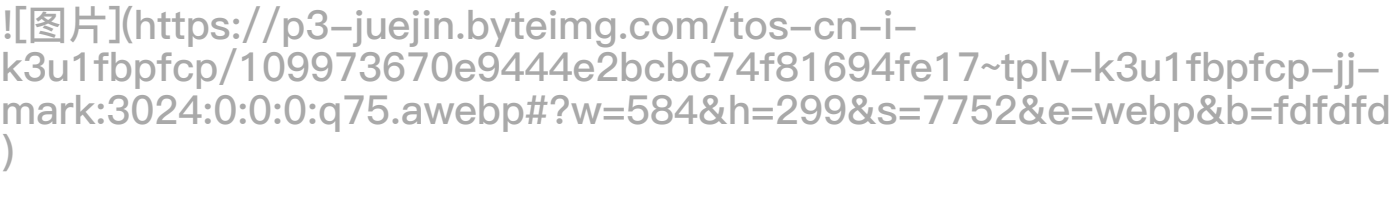
然后我们使用jmeter请求接口，500个并发线程，运行1万次，看看效果如何：



图片

可以看到最小用时50ms，这个没毛病，接口里面睡眠了50ms，但是不管是中位数还是90/95/99线都大于150ms了，这是因为系统线程是一个很昂贵的资源，SpringBoot中tomcat默认的最大连接数应该是200，在连接池的线程被耗尽后，这200个线程在那干等50ms结束，而剩下的请求也只能等待，无法进行其它的操作。下面再看下虚拟线程的表现：

「虚拟线程耗时：」



图片

可以看到即使是最大耗时，也保持在100ms以下，即线程等待时间显著的减少，虚拟线程更好的利用了系统资源。

总结

从上面的性能对比来看，虚拟线程在性能方面有明显的优势，但是要注意的是，我们上面的测试都是让线程等待了50ms，这是模拟什么场景?

没错，是IO密集型场景，即线程大部分时间是在等待IO，这样虚拟线程才可以发挥出它的优势，如果是CPU密集型场景，那么可能效果并不大。

不过我们目前大部分的应用都是IO密集型应用较多，比如典型的WEB应用，大量的时间在等待网络IO（DB、缓存、HTTP等等），使用虚拟线程的效果还是非常明显的。

最后说一句(求!别白嫖！)

如果这篇文章对您有所帮助，或者有所启发的话，求一键三连：点赞、转发、在看。

公众号：**woniuxgg**，在公众号中回复：笔记 就可以获得蜗牛为你精心准备的java实战语雀笔记，回复面试、开发手册、有超赞的粉丝福利!

原文链接: https://juejin.cn/post/7382892803314286603