```
oundThread = new Thread(() -> {
            int i = 0;
            while (!stopRequested())
                i++;
            System.out.println(i);
        });
        backgroundThread.start();

        TimeUnit.SECONDS.sleep(1);
        requestStop();
    }

    private static synchronized void requestStop() {
        stopRequested = true;
    }

    private static synchronized boolean stopRequested() {
        return stopRequested;
    }
```

又或者使用volatile保证可见性，这样原来那段代码则不会无限循环

```
private static volatile boolean stopRequested;
```

对volatile、synchronized不太熟悉的同学可以查看并发专栏下的文章：

5个案例和流程图让你从0到1搞懂volatile关键字

15000字、6个代码案例、5个原理图让你彻底搞懂Synchronized

并发访问共享可变数据时可以使用volatile或synchronized保证可见性

如果只是读取数据volatile更轻量级，如果需要读写操作则可以使用同步手段synchronized

避免过度同步

同步会带来性能开销，并不是所有操作都是需要同步的

比如Vector、Collections.synchronizedCollection大量使用直接加锁锁住整个方法，性能开销大，过度使用同步

```
    public synchronized E get(int index) {
        if (index >= elementCount)
            throw new ArrayIndexOutOfBoundsException(index);

        return elementData(index);
    }
```

**要避免过度使用同步并且同步区域中要做
不要依赖于线程调度器

线程调度器决定哪个线程能够执行

如果依赖于线程调度器来完成任务，那么程序是不健壮、不可移植的

类似Thread.yield礼让线程 或 调整线程优先级都属于依赖线程调度器，因为无法预估线程的执行顺序

即使在当前操作系统和虚拟机上能完成，在其他环境下也有可能失败，这是无法预估的，因为它们依赖于OS中的线程调度器

总结

对于共享可变数据，如果只读可以使用volatile保证可见性，如果需要写则要使用同步手段

过度使用同步手段会导致开销大，尽量在同步区间少做操作

并发包下的Executor框架将任务与执行分离，使用线程池管理线程，还有并行stream的fork join框架都优于单独使用线程

并发包下的工具使用更简单，了解后尽量使用并发包下的工具

对于可能被并发调用的类需要声明线程安全性文档：绝对线程安全、相对线程安全、线程不安全等

延迟初始化只是把初始化的开销放到第一次使用，大多数情况下还是直接初始化，如果需要可以考虑类加载保证一次初始化或双重检测保证一次初始化

程序的编写不要依赖于线程调度器，这样的程序是不健壮、不可移植的

最后（不要白嫖，一键三连求求拉~）

本篇文章被收入专栏 Effective Java，感兴趣的同学可以持续喔

本篇文章笔记以及案例被收入 [Gitee-CaiCaiJava](http://cxyroad.com/ ”https://gitee.com/tcl192243051/StudyJava/tree/master/EffectiveJava/EffectiveJava/src/main/java/_11%E5%B9%B6%E5%8F%91”)、[Github-CaiCaiJava](http://cxyroad.com/ ”https://github.com/Tc-liang/CaiCaiJava/tree/master/EffectiveJava/EffectiveJava/src/main/java/_11%E5%B9%B6%E5%8F%91”)，除此之外还有更多Java进阶相关知识，感兴趣的同学可以starred持续喔~

有什么问题可以在评论区交流，如果觉得菜菜写的不错，可以点赞、、收藏支持一下~

菜菜，分享更多技术干货，公众号：菜菜的后端私房菜

原文链接: https://juejin.cn/post/7356032959215255603