链滴

Java 并发知识梳理

作者：vcjmhg

原文链接：https://ld246.com/article/1621853820471

来源网站：链滴

<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile
com/bing/20181118.jpg?imageView2/1/w/960/h/540/interlace/1/q/100">
<script async src="https://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js?client=
a-pub-5357405790190342" crossorigin="anonymous"></script> <!-- 黑客派PC帖子内嵌 -->
<ins class="adsbygoogle" style="display:block" data-ad-client="ca-pub-5357405790190342"
data-ad-slot="8316640078" data-ad-format="auto" data-full-width-responsive="true"></in
>
<script>
    (adsbygoogle = window.adsbygoogle || []).push({});
</script>
<h2 id="概述">概述</h2>
随着摩尔定律逐步失效，cpu 单核性能达到瓶颈，并发逐渐逐渐得到广泛应用，因而学习了解以使用并发就显得十分重要。但并发相关的知识比较琐碎，不易系统学习，因而本篇文章参照王宝令老《Java 并发编程》来勾勒出一张“并发全景图”。
<h2 id="是什么-">是什么？</h2>
用学术定义来说就是
<blockquote>
 并发：同一时间段，多个任务都在执行 (单位时间内不一定同时执行)；
</blockquote>
简单来说就是，同一个时间段，让计算机同时做多个事情。
说到 <code>并发</code>，不得不提就是 <code>并行</code>：
<blockquote>
 并行：单位时间内，多个任务同时执行。
</blockquote>
两者大眼一看很像，仔细一想却并不相同，因为 <code>并行</code> 强调某个时点多个任务同时执行，而 <code>并发</code> 强调的是一个时间段内多个任务都在执行。
<h2 id="怎么做-">怎么做？</h2>
大部分并发问题，最终都可以抽象成三类问题分工、同步和互斥。而且针对不同的问题有着不同的方式来解决，具体如下图所示：</
>
<img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https:
/b3logfile.com/siyuan/1568182249394/assets/image-20210523202317-m93bhbt.png?imageV
ew2/2/interlace/1/format/jpg">
<h2 id="分工">分工</h2>
所谓 <code>分工</code>，类似于现实中一个组织完成一个项目，项目经理要拆分任务，安合适的成员去完成。
在并发编程领域，你就是项目经理，线程就是项目组成员。任务分解和分工对于项目成败非常关，不过在并发领域里，分工更重要，它直接决定了并发程序的性能，并且分工非常重要且复杂，因而 Java 并发包中有一系列方法来实现 <code>分工</code>：

 "Executor 与线程池"
 "ForkJoin"
 "Future 的使用"

基于分工思想设计的并发设计模式也有很多：

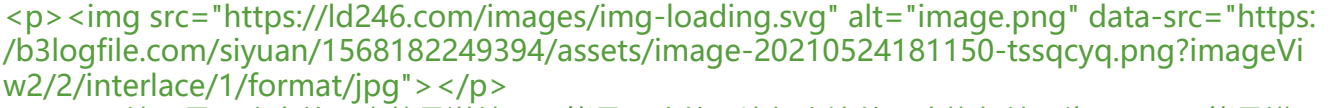
 "Guarded Supension 模式"
 "Balking 模式"
 "Threa-Per-Message 模式"
 "生产者-消费者模式"
 "Work Thread 模式"
 "两阶段终止模式"

<h2 id="同步">同步</h2>
而 <code>同步</code> 更多描述的是一种协同关系，在分完工之后，具体执行时，任务之间有依赖，一个任务之后完成之后，其他依赖它的任务才能开始进行，因而就引入的 <code>同步</coe> 来协同各个任务之间的执行顺序。
针对该类问题，Java 也提供了一系列工具来辅助解决：

"信号量（Semaphore）机制"
"管程（Monitor）"
"CountDownLatch"
"CyclicBarrier"
"Phaser"
"Exchanger"

<h2 id="互斥">互斥</h2>
分工、同步主要为了充分发掘 CPU 的性能来解决问题，但并发问题中，还需要解决正确性问题即保证 <code>线程安全</code>。
当多个线程同时访同一个变量时，最后执行的结果是不确定的，比如下边这段代码：
<pre><code class="language-java highlight-chroma">public cass UnsafeSequence {
    private int value = 0</span
;
    public int getNext() {</span

        <span class=highlight-k">return ++<span class="highlight-n>value;
    }
}
</code></pre>
如果我们有多个线程同时调用 <code>getNext()</code> 时，多个线程之间推进顺序的不同可会有不同的执行结果：
可能会是 2，递推顺序如下：

可能结果是 1，代码执行顺序如下：

因而结果是不确定的，也就是说结果可能是正确的（比如上边的程序执行结果为 2），可能是错的（比如执行结果是 1），执行前是不知道的。而导致不确定的主要源头主要是三个问题可见性问题有序性问题和原子性问题，为了解决这三个问题，Java 引入了内存模型，内存模型提供一系列规则利这些规则，我们可以避免可见性问题、有序性问题，但是还不足以完全解决线程安全问题。解决线程全问题的核心方案还是 <code>互斥</code>。
所谓互斥，指的是同一时刻，只允许一个线程访问共享变量。
实现互斥主要手段是互斥锁主要包含下边这些手段：

 Synchronized
 Lock
 读写锁

除此之外，未来提高速度，也有一些无锁的方案：

 不变模式
 线程本地存储
 CAS
 Copy - on - Write
 原子类

<h2 id="总结">总结</h2>
本文主要从分工、同步和互斥三类问题展开，从解决对应问题角度出发大致梳理了 Java 并发知的学习前景图。后续将分若干部分来讲对应的内容。
<h2 id="引用">引用</h2>

 《深入理解 Java 虚拟机》
 《Java 并发实战》

