11-Java线程（下）：为什么局部变量是线程安全的？

我们一遍一遍重复再重复地讲到，多个线程同时访问共享变量的时候，会导致并发问题。那在Java语言里，是不是所有变量都是共享变量呢？工作中我发现不少同学会给方法里面的局部变量设置同步，显然这些同学并没有把共享变量搞清楚。那Java方法里面的局部变量是否存在并发问题呢？下面我们就先结合一个例子剖析下这个问题。

比如，下面代码里的 fibonacci() 这个方法，会根据传入的参数 n ，返回 1 到 n 的斐波那契数列，斐波那契数列类似这样： 1、1、2、3、5、8、13、21、34……第1项和第2项是1，从第3项开始，每一项都等于前两项之和。在这个方法里面，有个局部变量：数组 r 用来保存数列的结果，每次计算完一项，都会更新数组 r 对应位置中的值。你可以思考这样一个问题，当多个线程调用 fibonacci() 这个方法的时候，数组 r 是否存在数据竞争（Data Race）呢？

```
// 返回斐波那契数列
int[] fibonacci(int n) {
  // 创建结果数组
  int[] r = new int[n];
  // 初始化第一、第二个数
  r[0] = r[1] = 1;  // ①
  // 计算2..n
  for(int i = 2; i < n; i++) {
      r[i] = r[i-2] + r[i-1];
  }
  return r;
}
```

你自己可以在大脑里模拟一下多个线程调用 fibonacci() 方法的情景，假设多个线程执行到 ① 处，多个线程都要对数组r的第1项和第2项赋值，这里看上去感觉是存在数据竞争的，不过感觉再次欺骗了你。

其实很多人也是知道局部变量不存在数据竞争的，但是至于原因嘛，就说不清楚了。

那它背后的原因到底是怎样的呢？要弄清楚这个，你需要一点编译原理的知识。你知道在CPU层面，是没有方法概念的，CPU的眼里，只有一条条的指令。编译程序，负责把高级语言里的方法转换成一条条的指令。所以你可以站在编译器实现者的角度来思考“怎么完成方法到指令的转换”。

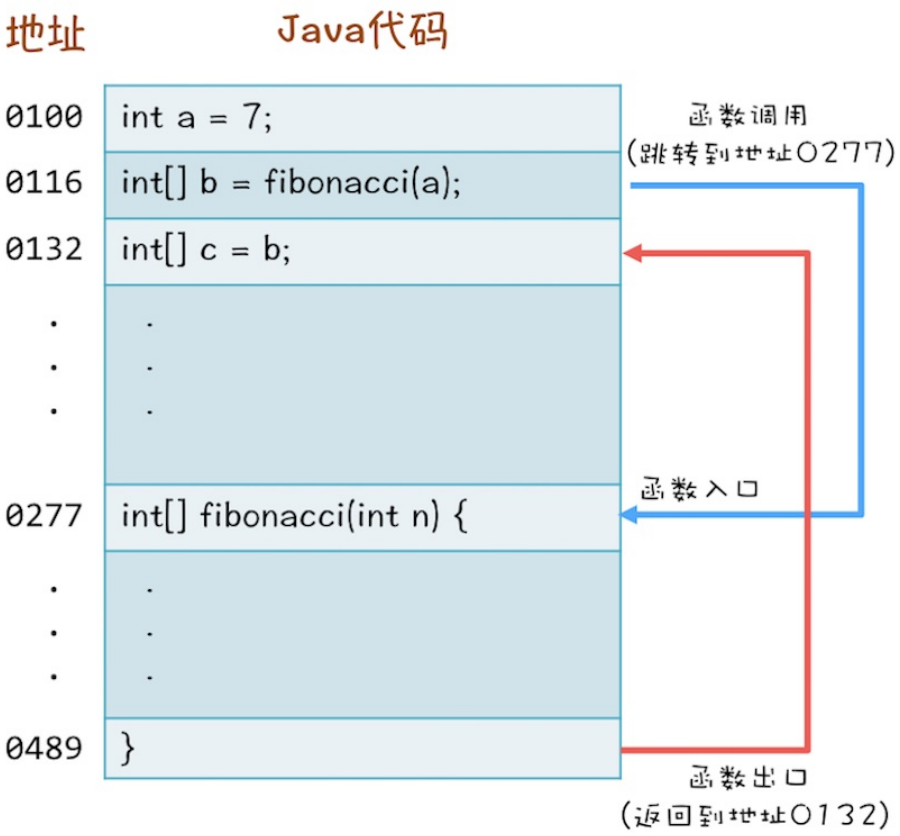
方法是如何被执行的

高级语言里的普通语句，例如上面的`r[i] = r[i-2] + r[i-1];`翻译成CPU的指令相对简单，可方法的调用就比较复杂了。例如下面这三行代码：第1行，声明一个int变量a；第2行，调用方法 fibonacci(a)；第3行，将b赋值给c。

```
int a = 7;
int[] b = fibonacci(a);
int[] c = b;
```

当你调用fibonacci(a)的时候，CPU要先找到方法 fibonacci() 的地址，然后跳转到这个地址去执行代码，最

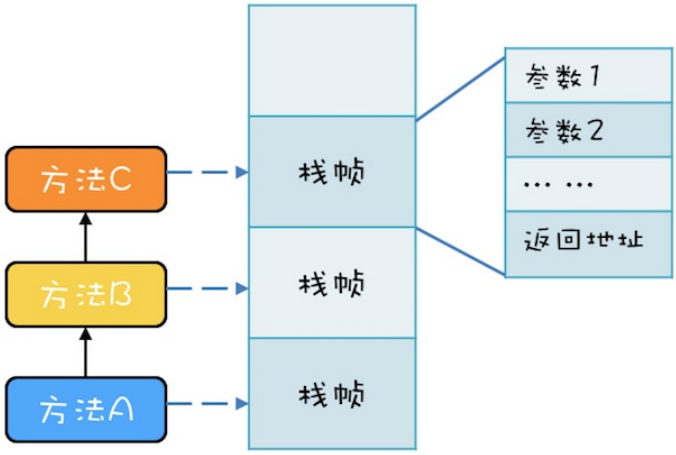
后CPU执行完方法 fibonacci() 之后，要能够返回。首先找到调用方法的下一条语句的地址：也就是`int[] c=b;`的地址，再跳转到这个地址去执行。 你可以参考下面这个图再加深一下理解。



方法的调用过程

到这里，方法调用的过程想必你已经清楚了，但是还有一个很重要的问题，“CPU去哪里找到调用方法的参数和返回地址？”如果你熟悉CPU的工作原理，你应该会立刻想到：**通过CPU的堆栈寄存器**。CPU支持一种栈结构，栈你一定很熟悉了，就像手枪的弹夹，先入后出。因为这个栈是和方法调用相关的，因此经常被称为**调用栈**。

例如，有三个方法A、B、C，他们的调用关系是A->B->C（A调用B，B调用C），在运行时，会构建出下面这样的调用栈。每个方法在调用栈里都有自己的独立空间，称为**栈帧**，每个栈帧里都有对应方法需要的参数和返回地址。当调用方法时，会创建新的栈帧，并压入调用栈；当方法返回时，对应的栈帧就会被自动弹出。也就是说，**栈帧和方法是同生共死的**。



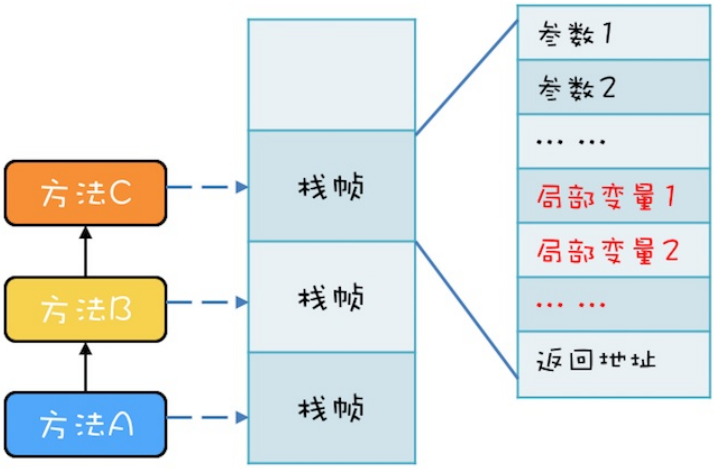
调用栈结构

利用栈结构来支持方法调用这个方案非常普遍，以至于CPU里内置了栈寄存器。虽然各家编程语言定义的方法千奇百怪，但是方法的内部执行原理却是出奇的一致：都是**靠栈结构解决**的。Java语言虽然是靠虚拟机解释执行的，但是方法的调用也是利用栈结构解决的。

局部变量存哪里？

我们已经知道了方法间的调用在CPU眼里是怎么执行的，但还有一个关键问题：方法内的局部变量存哪里？

局部变量的作用域是方法内部，也就是说当方法执行完，局部变量就没用了，局部变量应该和方法同生共死。此时你应该会想到调用栈的栈帧，调用栈的栈帧就是和方法同生共死的，所以局部变量放到调用栈里那儿是相当的合理。事实上，的确是这样的，**局部变量就是放到了调用栈里**。于是调用栈的结构就变成了下图这样。

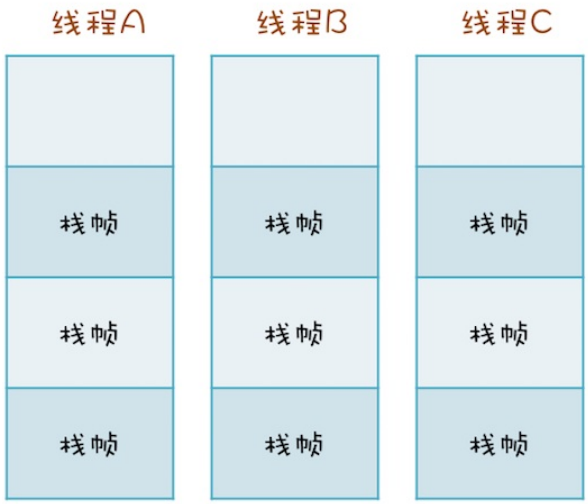


保护局部变量的调用栈结构

这个结论相信很多人都知道，因为学Java语言的时候，基本所有的教材都会告诉你 new 出来的对象是在堆里，局部变量是在栈里，只不过很多人并不清楚堆和栈的区别，以及为什么要区分堆和栈。现在你应该很清楚了，局部变量是和方法同生共死的，一个变量如果想跨越方法的边界，就必须创建在堆里。

调用栈与线程

两个线程可以同时用不同的参数调用相同的方法，那调用栈和线程之间是什么关系呢？答案是：**每个线程都有自己独立的调用栈**。因为如果不是这样，那两个线程就互相干扰了。如下面这幅图所示，线程A、B、C每个线程都有自己独立的调用栈。



线程与调用栈的关系图

现在，让我们回过头来再看篇首的问题：Java方法里面的局部变量是否存在并发问题？现在你应该很清楚了，一点问题都没有。因为每个线程都有自己的调用栈，局部变量保存在线程各自的调用栈里面，不会共享，所以自然也就没有并发问题。再次重申一遍：没有共享，就没有伤害。

线程封闭

方法里的局部变量，因为不会和其他线程共享，所以没有并发问题，这个思路很好，已经成为解决并发问题的一个重要技术，同时还有个响当当的名字叫做**线程封闭**，比较官方的解释是：**仅在单线程内访问数据**。由于不存在共享，所以即便不同步也不会有并发问题，性能杠杠的。

采用线程封闭技术的案例非常多，例如从数据库连接池里获取的连接Connection，在JDBC规范里并没有要求这个Connection必须是线程安全的。数据库连接池通过线程封闭技术，保证一个Connection一旦被一个线程获取之后，在这个线程关闭Connection之前的这段时间里，不会再分配给其他线程，从而保证了Connection不会有并发问题。

总结

调用栈是一个通用的计算机概念，所有的编程语言都会涉及到，Java调用栈相关的知识，我并没有花费很大的力气去深究，但是靠着那点C语言的知识，稍微思考一下，基本上也就推断出来了。工作了十几年，我发现最近几年和前些年最大的区别是：很多技术的实现原理我都是靠推断，然后看源码验证，而不是像以前一样纯粹靠看源码来总结了。

建议你也多研究原理性的东西、通用的东西，有这些东西之后再学具体的技术就快多了。

课后思考

常听人说，递归调用太深，可能导致栈溢出。你思考一下原因是什么？有哪些解决方案呢？

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。

猜你喜欢

精选留言：

- uyong 2019-03-23 09:54:19
  栈溢出原因：
  因为每调用一个方法就会在栈上创建一个栈帧，方法调用结束后就会弹出该栈帧，而栈的大小不是无限的，所以递归调用次数过多的话就会导致栈溢出。而递归调用的特点是每递归一次，就要创建一个新的栈帧，而且还要保留之前的环境（栈帧），直到遇到结束条件。所以递归调用一定要明确好结束条件，不要出现死循环，而且要避免栈太深。
  解决方法：
  1. 简单粗暴，不要使用递归，使用循环替代。缺点：代码逻辑不够清晰；
  2. 限制递归次数；
  3. 使用尾递归，尾递归是指在方法返回时只调用自己本身，且不能包含表达式。编译器或解释器会把尾递归做优化，使递归方法不论调用多少次，都只占用一个栈帧，所以不会出现栈溢出。然鹅，Java没有尾递归优化。 [25赞]

  作者回复2019-03-23 19:26:09
  很全面了。所有的递归算法都可以用非递归算法实现，大学老师好像还出过这样一道题……

- 西西弗与卡夫卡 2019-03-23 05:36:15
  因为调用方法时局部变量会进线程的栈帧，线程的栈内存是有限的，而递归没控制好容易造成太多层次调用，最终栈溢出。

  解决思路一是开源节流，即减少多余的局部变量或扩大栈内存大小设置，减少调用层次涉及具体业务逻辑，优化空间有限；二是改弦更张，即想办法消除递归，比如说能否改造成尾递归（Java会优化掉尾递归）[12赞]

  作者回复2019-03-24 12:53:21
  没怎么听说Java尾递归优化的事情，不太确定。最好不要依赖这个

- bing 2019-03-23 08:20:27
  当遇到递归时，可能出现栈空间不足，出现栈溢出，再申请资源扩大栈空间，如果空间还是不足会出现内存溢出oom。
  合理的设置栈空间大小；
  写递归方法注意判断层次；
  能用递归的地方大多数能改写成非递归方式。 [3赞]

  作者回复2019-03-23 19:27:08
  全面！

- Xiao 2019-03-27 08:26:42
  如果方法内部又有多线程，那方法内部的局部变量是不是也不是线程安全。 [2赞]

  作者回复2019-03-27 08:35:43
  你看看编译器允不允许这样做吧

- suynan 2019-03-25 22:44:47
  对于这句话：“ new 出来的对象是在堆里，局部变量在栈里”
  我觉得应该是对象在堆里，引用（句柄）在栈里 [2赞]

  作者回复2019-03-26 08:18:27
  是的

- Geek_cc0a3b 2019-03-24 15:09:54
  new 出来的对象是在堆里，局部变量是在栈里，那方法中new出来的对象属于局部变量，是保存在堆里还是在栈里呢？ [2赞]

  作者回复2019-03-24 17:03:01
  对象堆里，但是指针在栈里

- ack 2019-03-23 09:46:25
  如果是jvm栈空间太小了导致的栈溢出，可以通过-Xss增大栈空间大小。并且递归方法一定要有终止的return条件 [2赞]

- QQ怪 2019-03-23 12:28:12
  老师抛砖引玉的学习方法我觉得非常好，十分清楚的弄懂了本文全部内容，而且十分易懂。
  课后问题：递归调用太深会在创建过多的调用栈，也就是会创建过多栈帧，导致栈溢出，但是递归方法写起来简单，但会出现以上问题，解决办法可以改成非递归写法，自己手动维护个栈 [1赞]

- 卡西米 2019-03-23 12:13:41
  请教一个问题JAVA的栈跟cpu的栈有什么关系？ [1赞]

  作者回复2019-03-23 15:27:29
  没关系，只是原理类似

- 悠哉小二儿 2019-04-15 15:30:28
  老师test06被多线程执行。list06是否安全。
  public void test06(Long id) {
  //去计算
  List list06 = test06_1(id);

```
//返回结果。list06是否安全?

}

private List test06_1(Long id) {
List list06_1 = new ArrayList();
//业务执行。。。。
return test06_2(list06_1);
}

private List test06_2(List list06_1) {
List list = new ArrayList();
//业务执行。。。。
list.addAll(list06_1);

return list;
}
```

作者回复2019-04-15 18:30:35
安全，都是局部变量

- 罗杰18380446524 2019-04-09 09:17:05
  老师用的例子每个线程调用该方法时都会new一个数组，所以每个线程都会访问不同的数组并不会共享数组啊。是不是可以换个例子呢🤔

  作者回复2019-04-09 22:37:46
  就是解释为什么它不会共享😁

- 非礼勿言-非礼勿听-非礼勿视 2019-03-30 21:17:15
  递归可能导致栈溢出，基本上都知道原因，这里说下个人对于如何避免的浅见：
  1.递归基本上都可改写成循环方式
  2.限制递归层次
  3.其实和1是一个道理，模拟栈结构

- linqw 2019-03-26 23:54:26
  老师，有些疑惑的点，cpu是通过堆栈寄存器来调用方法，我们以java语言的调用来说，方法调用也是基于堆栈，一个方法在线程中的调用，就会将其方法的栈帧（参数，局部变量，返回地址）入栈，然后这些栈是在java虚拟机中，然后将其栈写到cpu堆栈寄存器调用吗？整体的流程不是很清楚，老师帮忙解答下哦

  作者回复2019-03-27 08:08:14
  java是解释执行的，所以java的栈和cpu的栈不是一回事，他们没有直接关系，仅仅是原理相通

- 灰灰灰 2019-03-26 12:47:00
  还是不大懂怎么定义共享变量。或者说不懂线程怎么争夺一个共享变量。

- 墨雨 2019-03-26 09:25:37
  解决递归调用太深:记录递归调用次数，达到某一深度后直接返回，退出递归。

- alias cd=rm -rf 2019-03-26 09:08:34

思考题
原因是递归方法在遇到递归结果之前，会一直把方法作为栈贞压入线程的调用栈。最终导致栈溢出。
解决方法
增加递归深度的限制。

- WL 2019-03-26 07:28:21
请教一下老师, 在静态方法中的局部变量是线程安全的吗, 静态方法的执行机制能不能也讲一下, 这里不是很理解.

作者回复2019-03-26 11:51:17
局部变量线程安全，静态方法没有隐藏的this参数，执行机制和正常方法没区别

- 悟空 2019-03-25 11:18:41
思考为什么讲这章，分析因为本节中带过了一种解决互斥的办法。不共享去解决互斥。这种也会有个问题。线程多，资源多，需要及时销毁。

作者回复2019-03-25 20:16:05
我是想让大家能推演程序的执行过程😁

- 悟空 2019-03-25 11:14:39
首先，递归是单线程执行，所以引起的空间变化实在单线程的堆栈里发生的。
其次，因为递归会不断调用方法，每次调用方法会重新在当前堆栈开辟新的地址。所以堆栈地址会越来越多。

递归是我一直没想明白的一种思考方式。查看网上说。用数学验证，第一次成功，那么说明第n次成功，如果能验证出n+1次也成功。说明正确。递归也是基于这个原理思考。

那实际哪些问题能抽象成递归，能用递归解决？如何写代码？是我困惑的

- Vickygu 2019-03-25 10:53:35
递归回造成栈溢出，是因为递归在不停的反复调用函数，即创建栈帧，栈帧数量过大就会导致栈溢出。
解决方法：
在设计递归的时候先要估算一个下大约的调用次数，以此来设置栈的大小，并设置递归的一个防溢出阈值来作为最后的手段。