> 作者：JavaGieGie
>
> 微信公众号：Java开发零到壹

前言

前面两期我们介绍了多线程的基础知识点，都是一些面试高频问题，没有看和忘记的小伙伴可以回顾一下。

本章主要是分析一下大家非常**面熟**的Java内存模型，用代码的方式介绍重排序、可见性以及线程之间通信等原理，大家看完本篇必定有更加清楚的认识和理解。

> 狗剩子：花GieGie~,节日快乐啊！这么早就来蹲坑。
>
> 我：哟，狗剩子你今天又来加班了，365天无休啊你。
>
> 狗剩子：这不今天过节，没有什么好东西送给各位看官，只能肝出来一些干货送给老铁们么。
>
> 我：接招吧，狗儿。

正文

> **我：书接上文，狗剩子你给大伙讲讲什么是volatile?**

上来就搞这么刺激的吗，你让咱家想想...

> **我：ok，小辣鸡，那我换个问题，你了解过Java内存模型吗?**

这个不是三伏天喝冰水，正中下怀么。

Java内存模型（Java Memory Model）简称JMM，首先要知道它是一组规范，是一组**多线程访问Java内存**的规范。

我们都知道市面上Java虚拟机种类有很多，比如HotSpot VM、J9 VM以及各种实现（Oracle / Sun JDK、OpenJDK），而每一种虚拟机在解释Java代码、并进行重排序时都有自己的一套流程，如果没有JMM规范，那很有可能相同代码在不同JVM解释后，得到的运行结果也是不一致的，这是我们不希望看到的。

> **我：有点意思，但这种说法还是有点模糊，你再具体说说它都有哪些规范?**

讨厌，就知道你会这么问，小伙们提到Java内存模型我们第一时间要想到3个部分，`重排序`、`可见性`、`原子性`。

- **重排序**

  先看一段代码，给你几分钟时间，看看这段代码输出有几种结果

```
private static int x = 0, y = 0;
private static int a = 0, b = 0;

Thread one = new Thread(new Runnable() {
    @Override
    public void run() {
        a = 1;
        x = b;
    }
});
```

```
Thread two = new Thread(new Runnable() {
    @Override
    public void run() {
        b = 1;
        y = a;
    }
});
two.start();
one.start();
one.join();
two.join();
System.out.println("x = "+x+", y = "+y);
```

你的答案是不是这三种呢

1. a=1;x=b(0);b=1;y=a(1)，最终结果是x=0, y=1

2. b=1;y=a(0);a=1;x=b(1)，最终结果是x=1, y=0

3. b=1;a=1;x=b(1);y=a(1)，最终结果是x=1, y=1

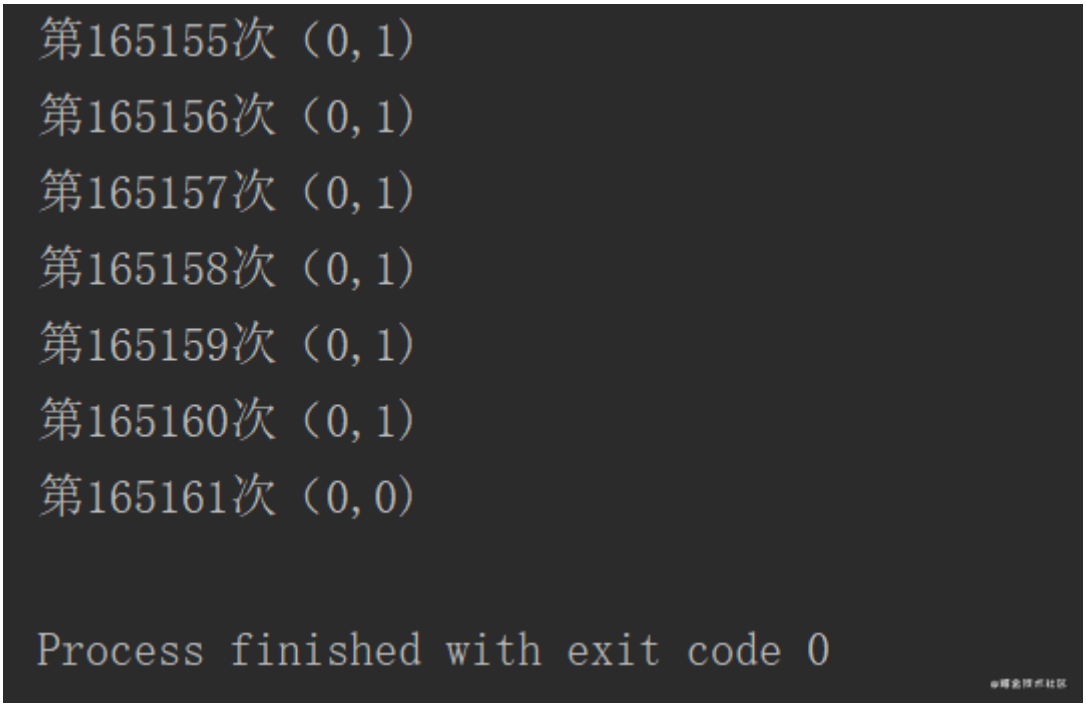
如果是的话，那么恭喜你，可以继续和狗哥我一块继续往下研究**第四种情况**

这里我增加了一个for循环，可以循环打印，直到打印自己想要的结果，小伙伴们自己运行一下。

```
private static int x = 0, y = 0;
private static int a = 0, b = 0;

public static void main(String[] args) throws InterruptedException {
    int i = 0;
    for (; ; ) {
        i++;
        x = 0;
        y = 0;
        a = 0;
        b = 0;

        CountDownLatch latch = new CountDownLatch(3);

        Thread thread1 = new Thread(new Runnable() {
            @Override
```

```
            public void run() {
                try {
                    latch.countDown();
                    latch.await();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                a = 1;
                x = b;
            }
        });
        Thread thread2 = new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    latch.countDown();
                    latch.await();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                b = 1;
                y = a;
            }
        });
        thread2.start();
        thread1.start();
        latch.countDown();
        thread1.join();
        thread2.join();

        String result = "第" + i + "次（" + x + "," + y + ")";
        if (x == 0 && y == 0) {
            System.out.println(result);
            break;
        } else {
            System.out.println(result);
        }
    }
}
```

看看你执行到多少次会出现呢，这里我是执行到将近17万次。



为什么会出现这种情况呢，那是因为这里发生了**重排序**，在重排序后，代码的执行顺序变成了：

- y=2;
- a=1;
- x=b;
- b=1;

这里就可以总结一下`重排序`，通俗的说就是代码的执行顺序和代码在文件中的**顺序不一致**，代码指令并没有严格按照代码语句顺序执行，而是根据自己的规则进行调整了，这就是`重排序`。

> **我：这个例子有点东西，简单明了，我都看懂了？那可见性又怎么理解呢**

既然例子比较直观，那这个问题我继续用例子来解释一波。

- **可见性**

```
public class Visibility {
   int a = 1;
   int b = 2;

   private void change() {
      a = 3;
      b = a;
   }


   private void print() {
      System.out.println("b=" + b + ";a=" + a);
   }

   public static void main(String[] args) {
      while (true) {
         Visibility visibility = new Visibility();
         // 线程1
         new Thread(() -> {
            try {
               Thread.sleep(1);
            } catch (InterruptedException e) {
```

```
                e.printStackTrace();
            }
            visibility.change();
        }).start();
        // 线程2
        new Thread(() -> {
            try {
                Thread.sleep(1);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            visibility.print();
        }).start();
    }
  }
}
```

这里同样建议停留几分钟，你觉得print()打印结果有几种呢，多思考才能理解更深刻。

- a=1，b=2 :线程1未执行到`change()`，此时线程2已执行`print()`
- a=3，b=2:线程1执行到`change()`的a = 3，然后线程2正好执行`print()`
- a=3，b=3：线程1执行完`change()`，然后线程2执行`print()`

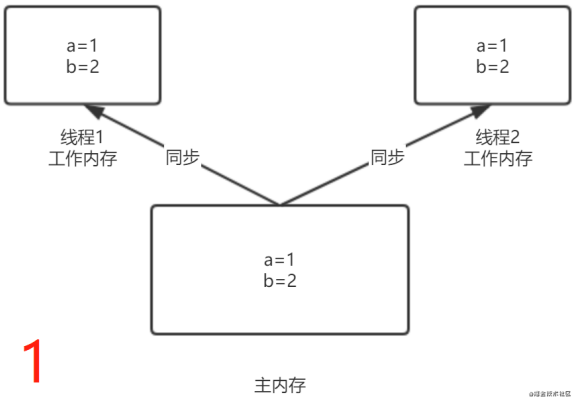
这是大家最容易想到和理解的(**如果没有想到，记得去补习一下花Gie的前两篇基础**)，但是还有一种情况比较特殊：

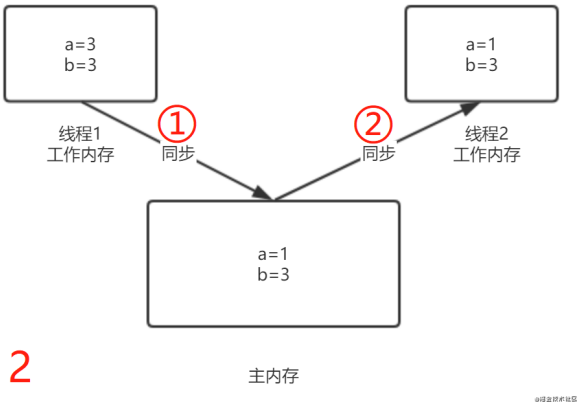
- b=3，a=1

是不是没想到啊(手动得意)，这里我们假如线程1执行完`change()`方法后，此时`a=3且b=3`，但是这时只是`线程1`自己知道这个结果值，对于`线程2`来说，他可能只看到了一部分，出现这种情况的原因，是因为线程之间通信是有延时的，而且多个线程之间不会进行实时同步，所以`线程2`只看到了b的最新值，并没有看到a的改变。

> **我：你这么说的话，我好像有点明白了，但还不是很清晰。**
>
> **你可以再说说这个变量是怎么传递的吗，为什么线程2没有接收到a的变化呢?**

好的呢，我都依你，我直接上个简单的草图吧。





图中我们分析出以下4个步骤。

- 每个线程都会从主内存中获取变量，保存在自己的工作内存（线程私有）中，图1是`线程1`、`线程2`初始化状态；
- 图2是线程1执行完`change()`方法后，先将`b=3`写回主内存（此时`a=3`还尚未写回主内存）
- 线程2从主内存获取最新数据`a = 1，b = 3`，并写到自己的工作线程
- 线程2最终打印出`a=1,b=3`

> **我：这下子我都看明白了，那你给我总结一下为什么会出现可见性原因吧，万一面试官问我我也好回答。**

。。。

造成可见性的原因，主要是因为`CPU有多级缓存`，而每个线程会将自己需要的数据读取到独占缓存中，在数据修改后也是写入到缓存中，然后等待刷回主内存，这就导致了有些线程读写的值是一个过期的值。

> **我：有点6，我给你先点个赞，那还要一个原子性呢?**

原子性我再后面再进行介绍，因为我们先了解`volatile`、`synchronized`之后再了解会更简单（你以为我不会volatile么，斜眼笑）。今天就先到这里吧，写了这么多，大家都懒得看了。

总结

JMM这块只是是非常重要的，熟练掌握以后在排查问题、写需求会更加得心应手，本篇本来想再多介绍一些其他内容，但是再写下去篇幅过长，效果就不是很好，所以先介绍这些，这里花Gie也强烈建议小伙伴们能亲手敲一下，纸上得来终觉浅，动手敲一敲以后写代码才不会虚。

下一章花Gie会继续介绍`happens-before`、`volatile`、`内存结构进阶`等，**希望大家持续关注，明天假期结束了，我们继续肝。**

点关注，防走丢

以上就是本期全部内容，**如有纰漏之处，请留言指教，非常感谢。**我是花GieGie ，有问题大家随时留言讨论 ，我们下期见🐘。

> 文章持续更新，可以微信搜一搜「**Java开发零到壹**」第一时间阅读，后续会持续更新Java面试和各类知识点，有兴趣的小伙伴欢迎关注，一起学习，一起哈🐯🥞。

原创不易，你怎忍心白嫖，如果你觉得这篇文章对你有点用的话，感谢老铁为本文**点个赞、评论或转发一下**，因为这将是我输出更多优质文章的动力，感谢！