Java SynchronizedSet 线程不安全之坑

Java进阶...

一、前言

一般而言，想要构造出线程安全的 Set，我们会使用 `Collections.synchronizedSet` 方法，如下所示。

```
Set<User> set = Collections.synchronizedSet(new HashSet<>());
```

但这并不意味着，你可以安全的使用该集合的任何方法，如果没有仔细的了解过其实现的话，一不小心就会踩进坑中。

最近我在使用该集合的 `stream` 方法时发现了线程不安全问题，都是血的教训啊，下面写个Case来复现下吧。

二、问题引出

2.1 辅助类

本 Case 牵扯到的所有辅助类如下：

```
public class ThreadPoolUtils {
    private static final long KEEP_ALIVE_TIME = 60L;

    private static Logger log = LogManager.getLogger(ThreadPoolUtils.class);

    public static ThreadPoolExecutor poolExecutor(int core, int max, Object... name) {
        ThreadFactory factory = Objects.nonNull(name) ?
                new ThreadFactoryBuilder().setNameFormat(Joiner.on(" ").join(name)).bu
                new ThreadFactoryBuilder().build();

        return new ThreadPoolExecutor(core, max, KEEP_ALIVE_TIME, TimeUnit.SECONDS, ne
                new ThreadPoolExecutor.AbortPolicy());
    }

    public static void sleep(long timeout, TimeUnit unit) {
        try {
            unit.sleep(timeout);
        } catch (InterruptedException e) {
            log.info("ThreadPoolUtils#sleep error, timeout: {}", timeout, e);
        }
    }
}
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class User {
    private long userId;

    private long timestamp;
```

```
            User user = (User)object;
            return user.getUserId() == ((User) object).getUserId();
        }
        return false;
    }

    @Override
    public int hashCode() {
        int result = 17;
        result = 31 * result + (int) (userId ^ (userId >>> 32));
        result = 31 * result + (int) (timestamp ^ (timestamp >>> 32));
        return result;
    }
}
```

2.2 测试类

Case 想要达到如下效果：

1. 线程 A 不停地往 Set 中添加元素。
2. 线程 B 不停地对 Set 做 Stream 操作。
3. 线程 B 在 Stream 的执行过程中间，线程 A 必须要进行添加操作。

为了达到这个效果，线程 B 在 Stream 过程中，增加了 Filter 并在其中 Sleep 10ms，确保在这段 Sleep 过程中，线程 A 会进行添加操作。

```
public class SynchronizedSetTest {
    private Set<User> set = Collections.synchronizedSet(new HashSet<>());
    private static Logger log = LogManager.getLogger(SynchronizedSetTest.class);

    public static void main(String[] args) {
        new SynchronizedSetTest().testStream();
    }

    public void testStream() {
        ThreadPoolExecutor executor = ThreadPoolUtils.poolExecutor(2, 2, "synchronized
        executor.execute(this::add);
        executor.execute(this::stream);
    }

    public void add() {
        while (true) {
            int size = RandomUtils.nextInt(1, 10);
            IntStream.range(0, size).forEach(e -> {
                set.add(random());
                log.info("SynchronizedSetTest#add size: {}", set.size());
                ThreadPoolUtils.sleep(10, TimeUnit.MILLISECONDS);
            });
        }
    }

    public void stream() {
        while (true) {
            List<User> userList = set.stream()
                    .filter(e -> {
                        ThreadPoolUtils.sleep(10, TimeUnit.MILLISECONDS);
eck...");
tamp() > 30L;
```

```
        }
    }

    private User random() {
        return User.builder().userId(RandomUtils.nextLong(1, 100000)).timestamp(System
    }
}
```

运行程序，刚运行就抛错了：

```
00:33:28.179 [synchronizedSet-test-pool] INFO  jit.wxs.SynchronizedSetTest - Synchroni
00:33:28.188 [synchronizedSet-test-pool] INFO  jit.wxs.SynchronizedSetTest - Synchroni
00:33:28.189 [synchronizedSet-test-pool] INFO  jit.wxs.SynchronizedSetTest - Synchroni
00:33:28.191 [synchronizedSet-test-pool] INFO  jit.wxs.SynchronizedSetTest - Synchroni
00:33:28.199 [synchronizedSet-test-pool] INFO  jit.wxs.SynchronizedSetTest - Synchroni
00:33:28.201 [synchronizedSet-test-pool] INFO  jit.wxs.SynchronizedSetTest - Synchroni
00:33:28.209 [synchronizedSet-test-pool] INFO  jit.wxs.SynchronizedSetTest - Synchroni
00:33:28.211 [synchronizedSet-test-pool] INFO  jit.wxs.SynchronizedSetTest - Synchroni
00:33:28.219 [synchronizedSet-test-pool] INFO  jit.wxs.SynchronizedSetTest - Synchroni
00:34:55.316 [synchronizedSet-test-pool] INFO  jit.wxs.SynchronizedSetTest - Synchroni
00:34:55.327 [synchronizedSet-test-pool] INFO  jit.wxs.SynchronizedSetTest - Synchroni
Exception in thread "synchronizedSet-test-pool" java.util.ConcurrentModificationExcept
        at java.util.HashMap$KeySpliterator.forEachRemaining(HashMap.java:1561)
        at java.util.stream.AbstractPipeline.copyInto(AbstractPipeline.java:482)
        at java.util.stream.AbstractPipeline.wrapAndCopyInto(AbstractPipeline.java:472
        at java.util.stream.ReduceOps$ReduceOp.evaluateSequential(ReduceOps.java:708)
        at java.util.stream.AbstractPipeline.evaluate(AbstractPipeline.java:234)
        at java.util.stream.ReferencePipeline.collect(ReferencePipeline.java:499)
        at jit.wxs.SynchronizedSetTest.stream(SynchronizedSetTest.java:54)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:
        at java.lang.Thread.run(Thread.java:748)
Disconnected from the target VM, address: '127.0.0.1:62588', transport: 'socket'
00:34:55.340 [synchronizedSet-test-pool] INFO  jit.wxs.SynchronizedSetTest - Synchroni
```

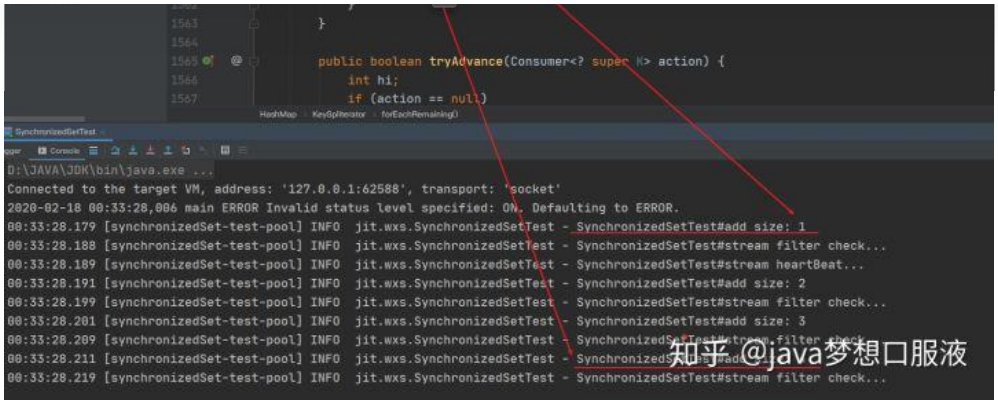
`ConcurrentModificationException` 这个异常如果对集合比较了解的话，是很熟悉的的。当我们对 ArrayList 迭代过程中进行添加/删除操作，就会报这个错误，错误原因就是 Collection 底层的 `modCount` 导致的。

下面描述下两个线程刚刚的执行情况：

1. 线程 A 开始添加元素，添加第一个，此时集合大小为 1。
2. 线程 B 开始 Stream 操作，执行到 Filter 时，被 Sleep 住。
3. 线程 A 在线程 B Sleep 期间，一直添加元素。
4. 线程 B Filter 执行完毕，执行最后 Collect() 操作。
5. 根据上面的异常栈，得知线程 B Collect() 最后会调用 HashMap 的 `forEachRemaining` 方法。

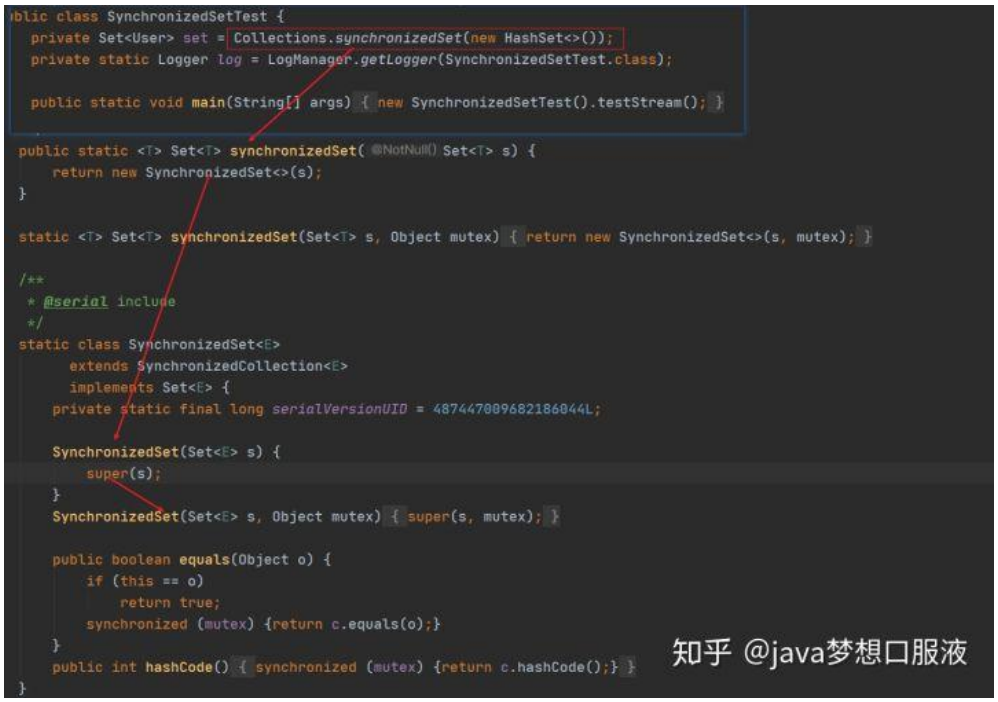
> 之所以调用 HashMap，是因为 HashSet 就是 HashMap 的一种特殊实现。

对 `java.util.HashMap.KeySpliterator#forEachRemaining` 进行 Debug，如下图所示。此时 Set 的 `modCount` 已经更新到了 4（这是没问题的，因为线程 A 一直在添加，添加了 4 次），然而线程 B 的 `mc` 仍然为开始 Stream 时的 1，因此抛出了异常。



三、源码查看

`Collections.synchronizedSet` 创建了 `SynchronizedSet` 对象，构造方法又调用了父类 `SynchronizedCollection` 。



看到 `SynchronizedCollection` 后就一切都明白了。首先把当前对象作为同步对象，因此加了对象锁的方法都是线程安全的，没有加 `synchronized` 修饰的方法就都是非线程安全的，使用过程中必须手动加同步块：

- `itearator()`
- `spliterator()`
- `stream()`
- `parallelStream()`

这边有一个有意思的地方， `forEach()` 是线程安全，而 `itearator()` 不是线程安全，`stream().forEach()` 也不是线程安全的。不同的遍历方式线程安全与否也不一样，不太明白 JDK 是怎么考虑这样设计的。

发布于 2020-11-02 14:01

Class static Java

推荐阅读

你知道 Java 类是如何被加载的吗?

一：前言最近给一个非Java方向的朋友讲了下双亲委派模型，朋友让我写篇文章深度研究下JVM的ClassLoader，我确实也好久没写JVM相关的文章了，有点手痒痒，涂了皮炎平也抑制不住。 我在向...

阿里云云栖... 发表于我是程序员

Java中的构造函数——通过示例学习Java编程（14）

作者：CHAITANYA SINGH 来源：通过示例学习Java编程（14）：Java中的构造函数-方家话题 构造函数是用来初始化新创建的对象的代码块。构造函数类似于java中的实例方法（Instance Method），...

lea

解密，Java中创建对象的5种方式

作为Java开发者，我们每天创建很多对象，但我们通常使用依赖管理系统，比如Spring去创建对象。然而这里有很多创建对象的方法，我们会在这篇文章中学到。 Java中有5种创建对象的方式，下面给...

毛奇志 发表于Java基...

在Java的反射 Class.forNar

老刘