实战Java高并发程序设计第10周

DATAGURU专业数据分析社区

法律声明

【声明】本视频和幻灯片为炼数成金网络课程的教学资料，所有资料只能在课程内使用，不得在课程以外范围散播，违者将可能被追究法律和经济责任。

课程详情访问炼数成金培训网站

http://edu.dataguru.cn

炼数成金逆向收费式网络课程

- Dataguru（炼数成金）是专业数据分析网站，提供教育，媒体，内容，社区，出版，数据分析业务等服务。我们的课程采用新兴的互联网教育形式，独创地发展了逆向收费式网络培训课程模式。既继承传统教育重学习氛围，重竞争压力的特点，同时又发挥互联网的威力打破时空限制，把天南地北志同道合的朋友组织在一起交流学习，使到原先孤立的学习个体组合成有组织的探索力量。并且把原先动辄成千上万的学习成本，直线下降至百元范围，造福大众。我们的目标是：低成本传播高价值知识，构架中国第一的网上知识流转阵地。
- 关于逆向收费式网络的详情，请看我们的培训网站 http://edu.dataguru.cn

内容提要

- 多线程调试的方法
- 线程dump及分析
- JDK8对并发的新支持
  - LongAdder
  - CompletableFuture
  - StampedLock

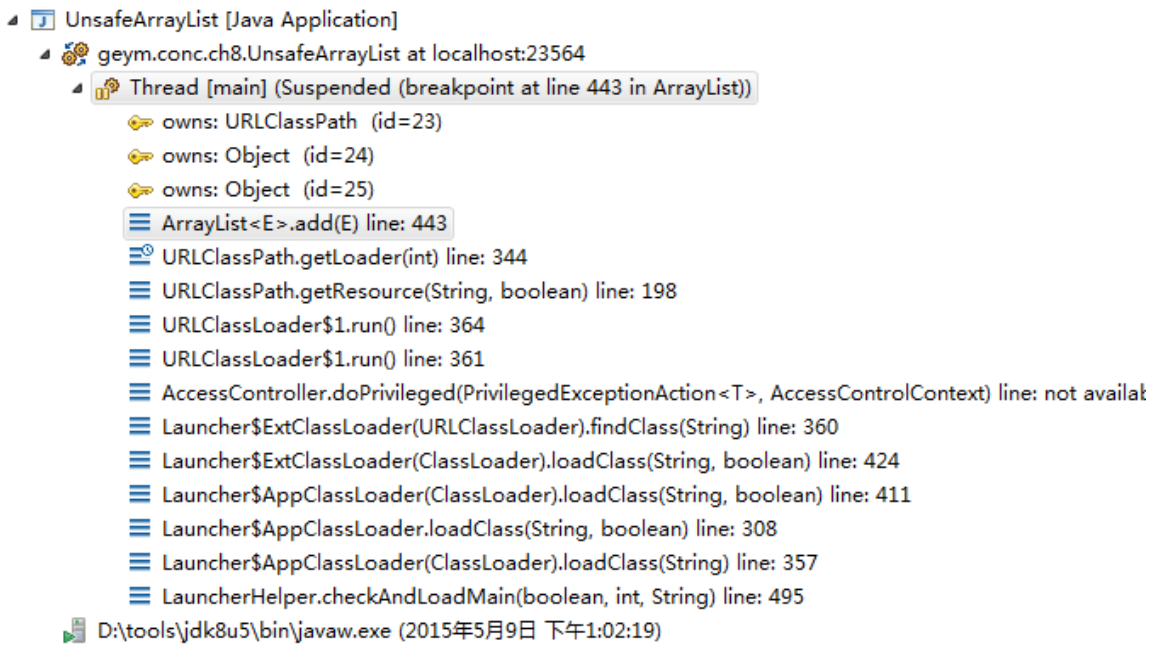
多线程调试的方法

- 使用Eclipse进行多线程调试

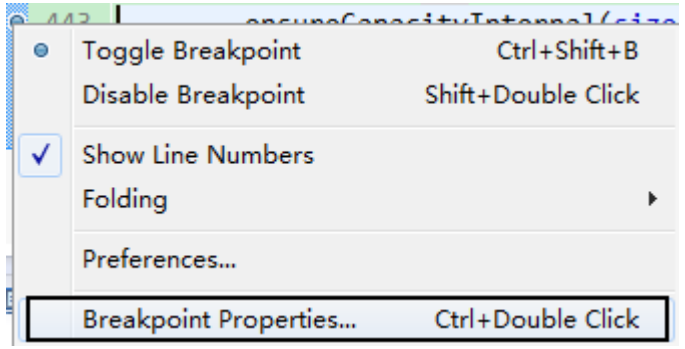
```
01 public class UnsafeArrayList {
02     static ArrayList al=new ArrayList();
03     static class AddTask implements Runnable{
04         @Override
05         public void run() {
06                 try {
07                                                         Thread.sleep(100);
08                                                 } catch (InterruptedException e) {}
09             for(int i=0;i<1000000;i++)
10                 al.add(new Object());
11         }
12     }
13     public static void main(String[] args) throws InterruptedException {
14         Thread t1=new Thread(new AddTask(),"t1");
15         Thread t2=new Thread(new AddTask(),"t2");
16         t1.start();
17         t2.start();
18         Thread t3=new Thread(new Runnable(){
19                                                 @Override
20                                                 public void run() {
21                                                         while(true){
22                                                                 try {
23                                                                         Thread.sleep(1000);
24                                                                 } catch (InterruptedException e) {}
25                                                         }
26                                                 }
27         },"t3");
28         t3.start();
29     }
30 }
```

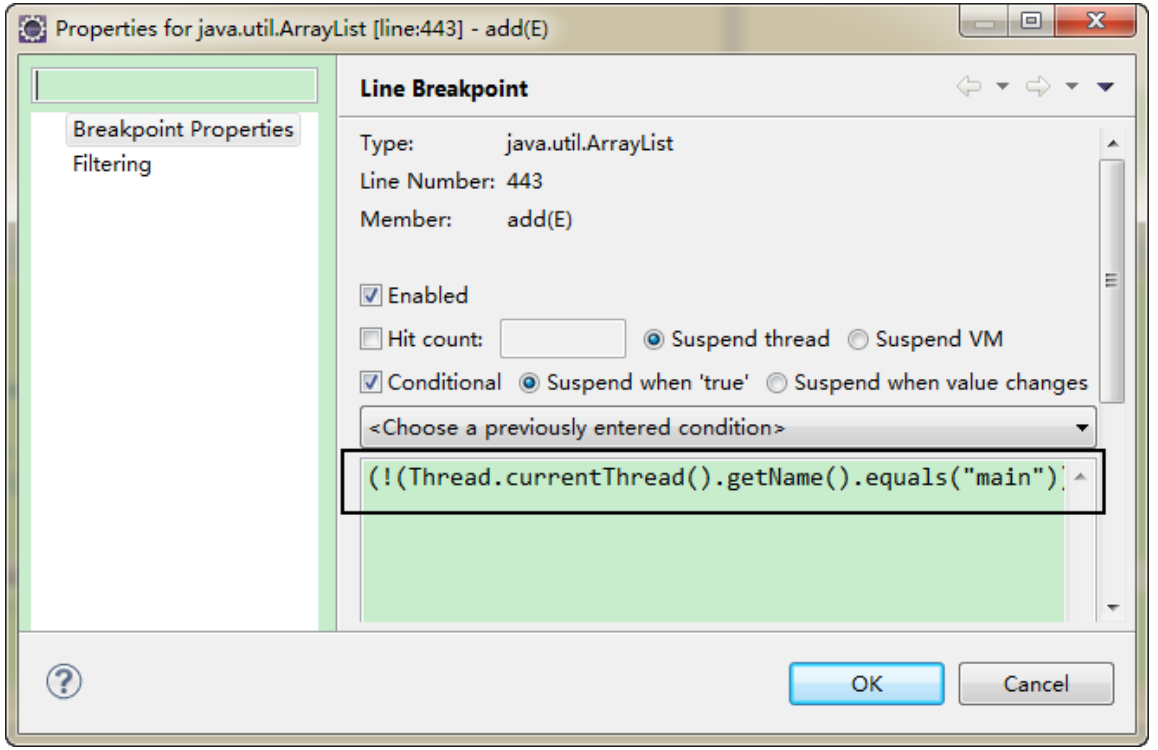
多线程调试的方法

```
442     public boolean add(E e) {
443         ensureCapacityInternal(size + 1);  // Increments modCount!!
444         elementData[size++] = e;
445         return true;
446     }
```

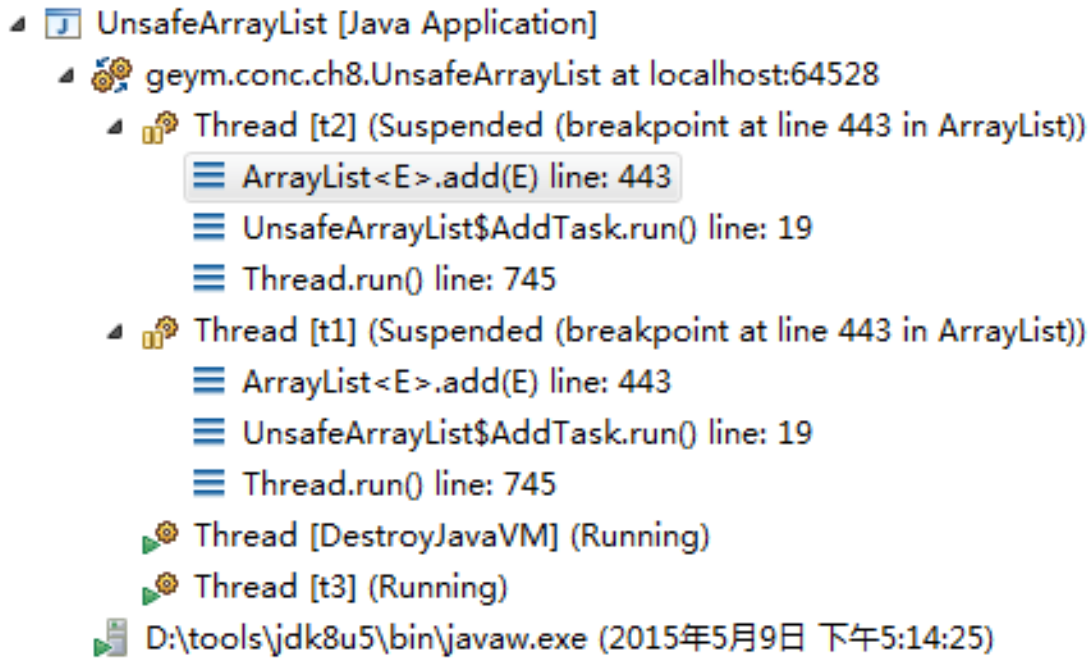


多线程调试的方法

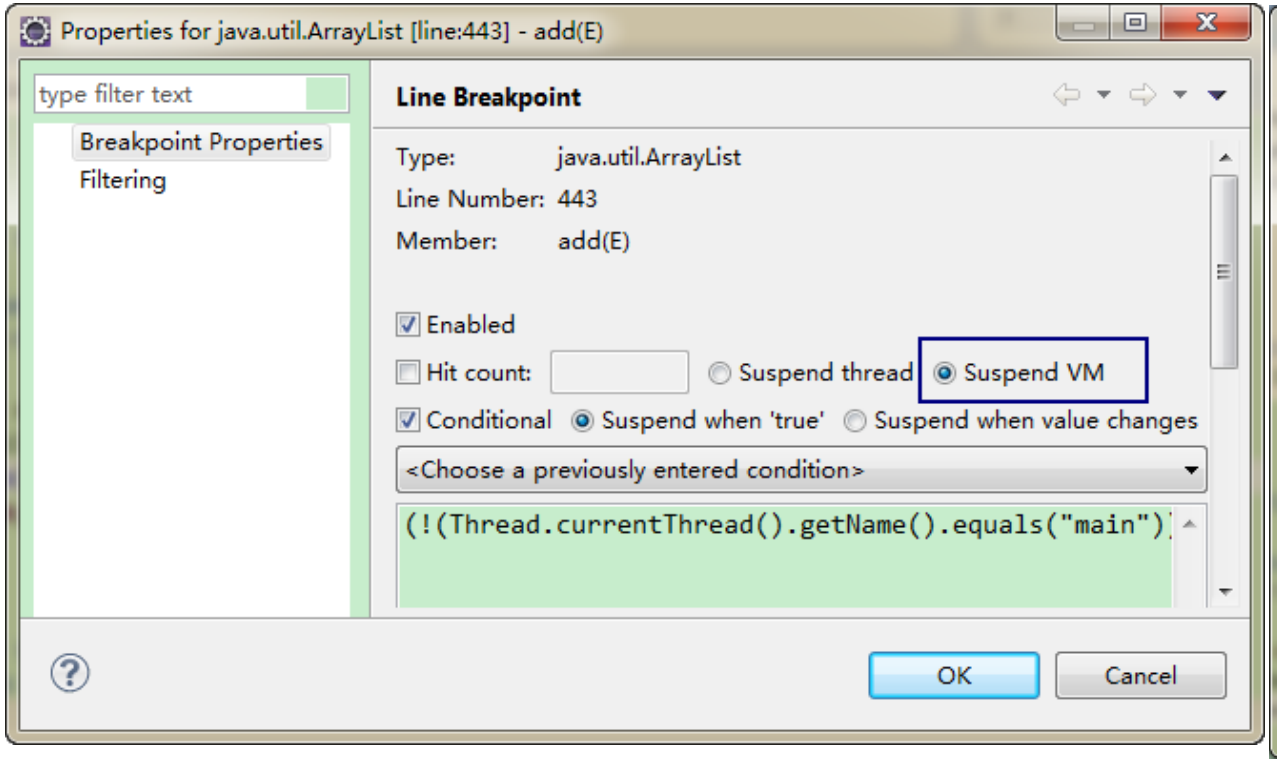

Toggle Breakpoint Ctrl+Shift+B
Disable Breakpoint Shift+Double Click
Show Line Numbers
Folding
Preferences...
Breakpoint Properties... Ctrl+Double Click

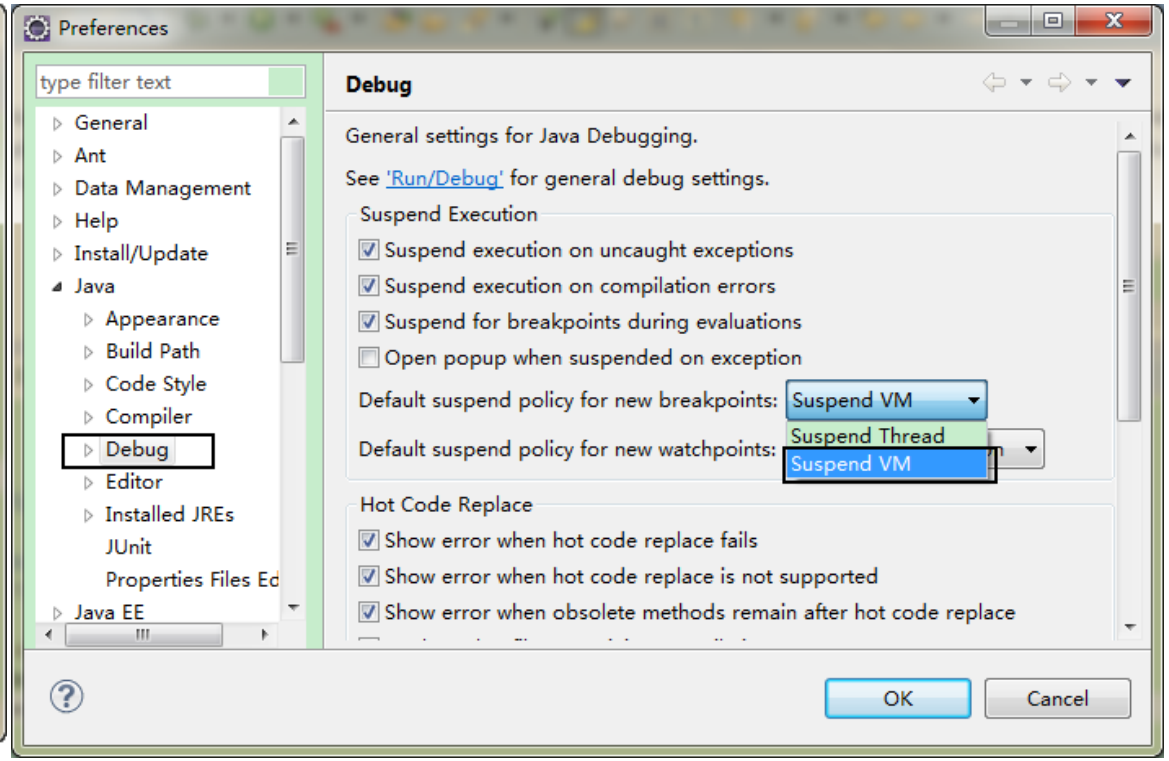

Properties for java.util.ArrayList [line:443] - add(E)
Breakpoint Properties
Filtering
Line Breakpoint
Type: java.util.ArrayList
Line Number: 443
Member: add(E)
Enabled
Hit count:
Suspend thread
Suspend VM
Conditional
Suspend when 'true'
Suspend when value changes
<Choose a previously entered condition>
(!(Thread.currentThread().getName().equals("main"))
OK
Cancel

多线程调试的方法


UnsafeArrayList [Java Application]
geym.conc.ch8.UnsafeArrayList at localhost:64528
Thread [t2] (Suspended (breakpoint at line 443 in ArrayList))
ArrayList<E>.add(E) line: 443
UnsafeArrayList$AddTask.run() line: 19
Thread.run() line: 745
Thread [t1] (Suspended (breakpoint at line 443 in ArrayList))
ArrayList<E>.add(E) line: 443
UnsafeArrayList$AddTask.run() line: 19
Thread.run() line: 745
Thread [DestroyJavaVM] (Running)
Thread [t3] (Running)
D:\tools\jdk8u5\bin\javaw.exe (2015年5月9日 下午5:14:25)

多线程调试的方法


Properties for java.util.ArrayList [line:443] - add(E)
type filter text
Breakpoint Properties
Filtering
Line Breakpoint
Type: java.util.ArrayList
Line Number: 443
Member: add(E)
Enabled
Hit count:
Suspend thread
Suspend VM
Conditional
Suspend when 'true'
Suspend when value changes
<Choose a previously entered condition>
(!(Thread.currentThread().getName().equals("main")
OK
Cancel


Preferences
type filter text
General
Ant
Data Management
Help
Install/Update
Java
Appearance
Build Path
Code Style
Compiler
Debug
Editor
Installed JREs
JUnit
Java EE
Debug
General settings for Java Debugging.
See 'Run/Debug' for general debug settings.
Suspend Execution
Suspend execution on uncaught exceptions
Suspend execution on compilation errors
Suspend for breakpoints during evaluations
Open popup when suspended on exception
Default suspend policy for new breakpoints:
Suspend VM
Suspend Thread
Suspend VM
Default suspend policy for new watchpoints:
Hot Code Replace
Show error when hot code replace fails
Show error when hot code replace is not supported
Show error when obsolete methods remain after hot code replace
OK
Cancel

多线程调试的方法


UnsafeArrayList [Java Application]
geym.conc.ch8.UnsafeArrayList at localhost:65475 (Suspended)
Daemon System Thread [Attach Listener] (Suspended)
Daemon System Thread [Signal Dispatcher] (Suspended)
Daemon System Thread [Finalizer] (Suspended)
Daemon System Thread [Reference Handler] (Suspended)
Thread [t1] (Suspended (breakpoint at line 443 in ArrayList))
ArrayList<E>.add(E) line: 443
UnsafeArrayList$AddTask.run() line: 19
Thread.run() line: 745
Thread [t2] (Suspended)
Thread [DestroyJavaVM] (Suspended)
Thread [t3] (Suspended)
D:\tools\jdk8u5\bin\javaw.exe (2015年5月9日 下午5:20:46)

线程Dump分析

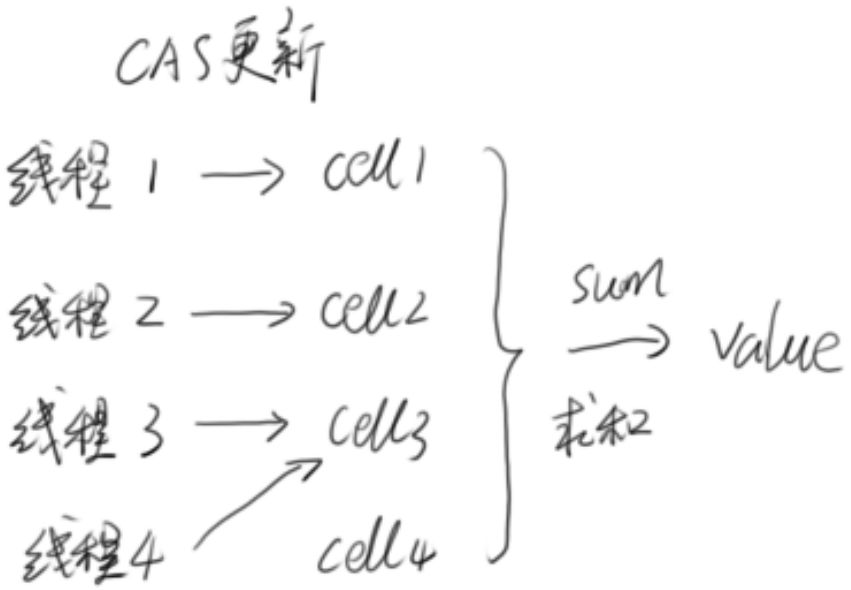
- jstack 3992
- 在%JAVA_HOME%/bin
- 分析死锁案例

JDK8对并发的新支持

- LongAdder
  - 和AtomicInteger类似的使用方式
  - 在AtomicInteger上进行了热点分离
  - public void add(long x)
  - public void increment()
  - public void decrement()
  - public long sum()
  - public long longValue()
  - public int intValue()

JDK8对并发的新支持

- LongAdder



JDK8对并发的新支持

- CompletableFuture
  - 实现CompletionStage接口（40余个方法）
  - Java 8中对Future的增强版
  - 支持流式调用

```
stage.thenApply(x -> square(x)).thenAccept(x -> System.out.print(x)).thenRun(() ->
System.out.println())
```

JDK8对并发的新支持

- CompletableFuture
  - 完成后得到通知

```
01 public static class AskThread implements Runnable {
02         CompletableFuture<Integer> re = null;
03
04         public AskThread(CompletableFuture<Integer> re) {
05                 this.re = re;
06         }
07
08         @Override
09         public void run() {
10                 int myRe = 0;
11                 try {
12                         myRe = re.get() * re.get();
13                 } catch (Exception e) {
14                 }
15                 System.out.println(myRe);
16         }
17 }
18
19 public static void main(String[] args) throws InterruptedException {
20         final CompletableFuture<Integer> future = new CompletableFuture<>();
21         new Thread(new AskThread(future)).start();
22         // 模拟长时间的计算过程
23         Thread.sleep(1000);
24         // 告知完成结果
25         future.complete(60);
26 }
```

JDK8对并发的新支持

- CompletableFuture
  - 异步执行

```
01 public static Integer calc(Integer para) {
02          try {
03                    // 模拟一个长时间的执行
04                    Thread.sleep(1000);
05          } catch (InterruptedException e) {
06          }
07    return para*para;
08 }
09
10 public static void main(String[] args) throws InterruptedException, ExecutionException {
11    final CompletableFuture<Integer> future =
12          CompletableFuture.supplyAsync(() -> calc(50));
13    System.out.println(future.get());
14 }
```

JDK8对并发的新支持

- CompletableFuture
  - 工厂方法

```
static <U> CompletableFuture<U> supplyAsync(Supplier<U> supplier);
static <U> CompletableFuture<U> supplyAsync(Supplier<U> supplier, Executor executor);
static CompletableFuture<Void> runAsync(Runnable runnable);
static CompletableFuture<Void> runAsync(Runnable runnable, Executor executor);
```

JDK8对并发的新支持

- CompletableFuture
  - 流式调用

```
01 public static Integer calc(Integer para) {
02          try {
03                    // 模拟一个长时间的执行
04                    Thread.sleep(1000);
05          } catch (InterruptedException e) {
06          }
07    return para*para;
08 }
09
10 public static void main(String[] args) throws InterruptedException, ExecutionException {
11          CompletableFuture<Void> fu=CompletableFuture.supplyAsync(() -> calc(50))
12      .thenApply((i)->Integer.toString(i))
13      .thenApply((str)->"\""+str+"\"")
14      .thenAccept(System.out::println);
15    fu.get();
16 }
```

JDK8对并发的新支持

- CompletableFuture
  - 组合多个CompletableFuture

```
public <U> CompletableFuture<U> thenCompose(Function<? super T, ? extends CompletionStage<U>> fn)
```

```
01 public static Integer calc(Integer para) {
02     return para/2;
03 }
04
05 public static void main(String[] args) throws InterruptedException, ExecutionException {
06     CompletableFuture<Void> fu =
07             CompletableFuture.supplyAsync(() -> calc(50))
08             .thenCompose((i)->CompletableFuture.supplyAsync(() -> calc(i)))
09             .thenApply((str)->"\"" + str + "\"").thenAccept(System.out::println);
10     fu.get();
11 }
```

```
"12"
```

JDK8对并发的新支持

- StampedLock
  - 读写锁的改进
  - 读不阻塞写

```
01 public class Point {
02         private double x, y;
03         private final StampedLock sl = new StampedLock();
04
05         void move(double deltaX, double deltaY) { // an exclusively locked method
06                 long stamp = sl.writeLock();
07                 try {
08                         x += deltaX;
09                         y += deltaY;
10                 } finally {
11                         sl.unlockWrite(stamp);
12                 }
13         }
14
15         double distanceFromOrigin() { // A read-only method
16                 long stamp = sl.tryOptimisticRead();
17                 double currentX = x, currentY = y;
18                 if (!sl.validate(stamp)) {
19                         stamp = sl.readLock();
20                         try {
21                                 currentX = x;
22                                 currentY = y;
23                         } finally {
24                                 sl.unlockRead(stamp);
25                         }
26                 }
27                 return Math.sqrt(currentX * currentX + currentY * currentY);
28         }
29 }
```

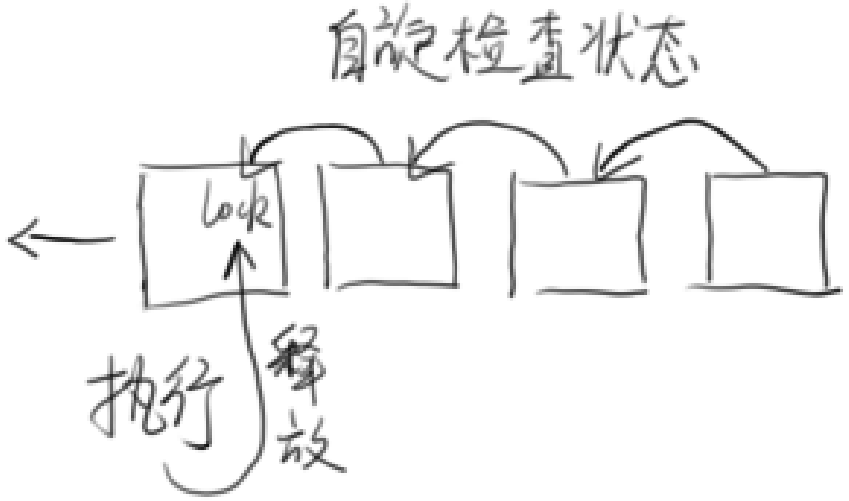
JDK8对并发的新支持

- StampedLock的实现思想
  - CLH自旋锁
  - 锁维护一个等待线程队列，所有申请锁，但是没有成功的线程都记录在这个队列中。每一个节点（一个节点代表一个线程），保存一个标记位（locked），用于判断当前线程是否已经释放锁。
  - 当一个线程试图获得锁时，取得当前等待队列的尾部节点作为其前序节点。并使用类似如下代码判断前序节点是否已经成功释放锁：

```
while (pred.locked) {
}
```

JDK8对并发的新支持

- StampedLock的实现思想
  - 不会进行无休止的自旋，会在在若干次自旋后挂起线程



Thanks

FAQ时间