BITTIGER

JAVA MULTITHREADING_2

Outline

- Implement BlockingQueue
- Read & Write Lock
- Thread Pool
- Callable
- Fork/Join Framework

一点点小小的黑科技.....

Intellij IDEA

jconsole

jstat

jmap

Runnable VS Thread

多个Thread执行同一个run()

多个Thread执行多个独立run()

```
/Library/Java/JavaVirtualMachines/jdk1
0
1
2
3
4
5
6
7
8
9

0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9

Process finished with exit code 0
```

```java
public class Runner {
    public class MyRunnable implements Runnable{
        int i = 0;
        @Override
        public void run(){
            while (i < 10){
                System.out.println(i++);
            }
        }
    }
    public class MyThread extends Thread{
        int i = 0;
        @Override
        public void run(){
            while (i < 10){
                System.out.println(i++);
            }
        }
    }
    public static void main(String[] args){
        Runner r = new Runner();
        MyRunnable myRunnable = r.new MyRunnable();

        Thread t1 = new Thread(myRunnable);
        Thread t2 = new Thread(myRunnable);

        t1.start();
        t2.start();
        System.out.println();
        Thread t3 = r.new MyThread();
        Thread t4 = r.new MyThread();

        t3.start();
        t4.start();
    }
}
```

Implement BlockingQueque

Put 和 Take 不能同时发生！

Lock

多个Thread 同时Put 和 Take

Lock

Queue Full ——-> Block Put

Condition

Queue Empty ——> Block Take

Take <—相互通知—-> Put

Condition

Condition

```
2  public class MyBlockingQueue {
3      private Lock lock = new ReentrantLock();
4      private Condition isFull = lock.newCondition();
5      private Condition isEmpty = lock.newCondition();
6
7      private Deque<String> que;
8      private int size;
9
10     public MyBlockingQueue(int size){
11         this.size = size;
12         que = new ArrayDeque<>();
13     }
```

1个Lock 和 2 个 Condition

注意 Size

每个Lock 可以绑定多个Condition.

```
14      public void put(String val){
15          lock.lock();
16          try{
17              while(que.size() == this.size){
18                  isFull.await();
19              }
20              Thread.sleep(100);
21              this.que.addLast(val);
22              isEmpty.signalAll();
23          }catch (InterruptedException e){
24              e.printStackTrace();
25          }finally {
26              lock.unlock();
27          }
28
29      }
30      public void take(){
31          lock.lock();
32          try{
33              while(que.isEmpty()){
34                  isEmpty.await();
35              }
36              Thread.sleep(100);
37              this.que.removeFirst(val);
38              isFull.signalAll();
39          }catch(InterruptedException e){
40              e.printStackTrace();
41          }finally {
42              lock.unlock();
43          }
44      }
```

Lock（同一时间点有且仅有一个thread 执行put 和 take）

signalAll() 唤醒所有waiting Thread. signal() 唤醒 priority 最高的那个或者随机唤醒

挂起 执行 Take 的线程

唤醒 执行 Put 的线程

unlock() in finally block

ReentrantReadWriteLock

ReentrantReadWriteLock.readLock()

ReentrantReadWriteLock.writeLock()

Read 和 Write 互斥 & 可同时Read,不能同时Write

Read 加 readLock. Write 加 writeLock.

```java
public ReentrantReadWriteLock  readWriteLock = new ReentrantReadWriteLock();
public Lock readLock = readWriteLock.readLock();
public Lock writeLock = readWriteLock.writeLock();

private List<Integer> list = new ArrayList<>();

public void addElement(int element){
    writeLock.lock();

    try{
        list.add(element);
        System.out.println(Thread.currentThread().getName() + "  Adding... " + element
        + " Current Running Thread Number: " + Thread.activeCount());
        Thread.sleep(100);
    }catch (Exception e){
        e.printStackTrace();
    }finally {
        writeLock.unlock();
    }
}

public void getElement(int index){
    readLock.lock();

    try{
        int element = list.get(index);
        System.out.println(Thread.currentThread().getName() + " Getting... " + element
                + " Current Running Thread Number: " + Thread.activeCount());
        Thread.sleep(100);
    }catch (Exception e){
        e.printStackTrace();
    }finally {
        readLock.unlock();
    }
```

获取readLock 和 writeLock
(static inner class)

Write 加 writeLock

当前running的线程数

Read 加 readLock

```java
    public static void main(String[] args) throws InterruptedException {
        ReadWriteLock readWriteLock = new ReadWriteLock();

        Adder adder = readWriteLock.new Adder(readWriteLock);
        Setter setter = readWriteLock.new Setter(readWriteLock);

        List<Thread> writeThread = new ArrayList<>();
        for(int i = 0; i < 3; i++){
            writeThread.add(new Thread(adder));
        }

        List<Thread> readThread = new ArrayList<>();
        for(int i = 0; i < 3; i++){
            readThread.add(new Thread(setter));
        }

        for(int i = 0; i < 3; i++){
            writeThread.get(i).start();
        }
        Thread.sleep(10);
        for(int i = 0; i < 3; i++){
            readThread.get(i).start();
        }
    }
```

```java
public class Adder implements Runnable{
        ReadWriteLock readWriteLock;
        public Adder(ReadWriteLock readWriteLock){
            this.readWriteLock = readWriteLock;
        }
        @Override
        public void run(){
            readWriteLock.addElement(new Random().nextInt(10));
        }
    }

    public class Setter implements Runnable{
        ReadWriteLock readWriteLock;
        int size;
        public Setter(ReadWriteLock readWriteLock){
            this.readWriteLock = readWriteLock;
            this.size = readWriteLock.list.size();
        }
        @Override
        public void run(){
            readWriteLock.getElement(0);
        }
    }
```

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/ja
Thread-0  Adding... 5 Current Running Thread Number: 5
Thread-2  Adding... 9 Current Running Thread Number: 7
Thread-1  Adding... 1 Current Running Thread Number: 6
Thread-3 Getting... 5 Current Running Thread Number: 5
Thread-4 Getting... 5 Current Running Thread Number: 5
Thread-5 Getting... 5 Current Running Thread Number: 5

Process finished with exit code 130 (interrupted by signal 2: SIGINT)
```

来点高端的。。。

线程池 Why？

Minimize system resources(重复新建消亡线程消耗系统资源)

Improve performance(线程新建消亡消耗时间)

Manageable (统一分配，调度和监控线程)

ExecutorService (线程池Interface)

Executors (线程池Factory Class)

- newCachedThreadPool()
  - 无固定线程数，短生存期异步任务。
- newFixedThreadPool(int nThreads)
  - 固定线程数，稳定并发任务。
- newSingleThreadExecutor()
  - 单线程线程池，代替new Thread()
- newScheduledThreadPool()

CPU 密集型 (Ncpu)
I/O 密集型(k*Ncpu)
资源密集型

```
3  Runtime.getRuntime().availableProcessors()
4  Runtime.getRuntime().freeMemory()
5  Runtime.getRuntime().totalMemory()
```

```
3  ExecutorService fixedThreadPool = Executors.newFixedThreadPool(4);
```

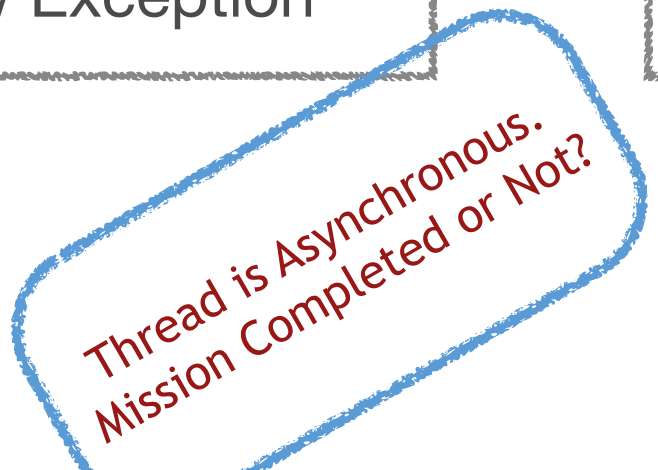
Callable VS Runnable

```java
@FunctionalInterface
public interface Callable<V> {
    /**
     * Computes a result, or throws an exception if unable to do so.
     *
     * @return computed result
     * @throws Exception if unable to compute a result
     */
    V call() throws Exception;
}
```

```java
@FunctionalInterface
public interface Runnable {
    /**
     * When an object implementing interface <code>Runnable</code> is used
     * to create a thread, starting the thread causes the object's
     * <code>run</code> method to be called in that separately executing
     * thread.
     * <p>
     * The general contract of the method <code>run</code> is that it may
     * take any action whatsoever.
     *
     * @see     java.lang.Thread#run()
     */
    public abstract void run();
}
```

有返回值，能Throw Exception

无返回值，不能Throw Exception



Interface Future<V>

Class FutureTask<V> (implements Future<V>, Runnable)

```
3   public interface Future<V> {
4       boolean cancel(boolean mayInterruptIfRunning);
5       boolean isCancelled();
6       boolean isDone();
7       V get() throws InterruptedException, ExecutionException;
8       V get(long timeout, TimeUnit unit)
9           throws InterruptedException, ExecutionException, TimeoutException;
10  }
```

get() 返回执行结果，会产生阻塞，一直等到任务执行完毕才返回

```
3  public interface ExecutorService extends Executor {
4      void shutdown();
5      <T> Future<T> submit(Callable<T> task);
6      Future<?> submit(Runnable task);
7      <T> List<Future<T>> invokeAll(Collection<? extends Callable<T>> tasks)
8          throws InterruptedException;
9  }
10
11 public class Executor {
12   void execute(Runnable command);
13 }
```

完成submitted的任务后，关闭线程池

执行Callable 任务并返回Future类

执行Runnable 任务

```java
public static class MyRunnable implements Runnable{
        @Override
        public void run(){
            int res = 0;
            for(int i = 0; i < 10; i++){
                res += i;
            }
            System.out.println("Runnable:  " + res);
        }
    }
    public static class MyCallable implements Callable<Integer> {
        @Override
        public Integer call(){
            int res = 0;
            for(int i = 0; i < 10; i++){
                res += i;
            }
            return res;
        }
    }
    public static void main(String[] args) throws ExecutionException, InterruptedException {
        ExecutorService singleThreadPool = Executors.newSingleThreadExecutor();

        Future<?> furture1 = singleThreadPool.submit(new MyRunnable());
        Future<Integer> furture2 = singleThreadPool.submit(new MyCallable());
        System.out.println("Runnable Future:  " + furture1.get());
        System.out.println("Callable Future:  " + furture2.get());
        singleThreadPool.execute(new MyRunnable());

        singleThreadPool.shutdown();

    }
```

0 +.. 9 计算完成打印结果

0 +.. 9 计算完成返回结果

```
/Library/Java/JavaVirtualMachines/j
Runnable:  45
Runnable Future:  null
Callable Future:  45
Runnable:  45

Process finished with exit code 0
```

向线程池submit 任务

线程池execute 任务

来点实际的。。。

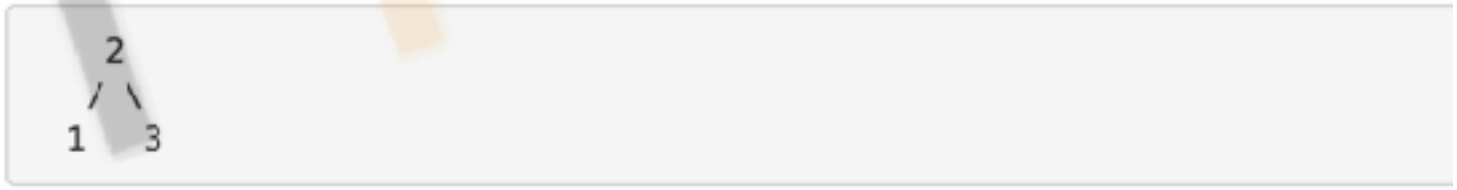
98. Validate Binary Search Tree

Given a binary tree, determine if it is a valid binary search tree (BST).

Assume a BST is defined as follows:

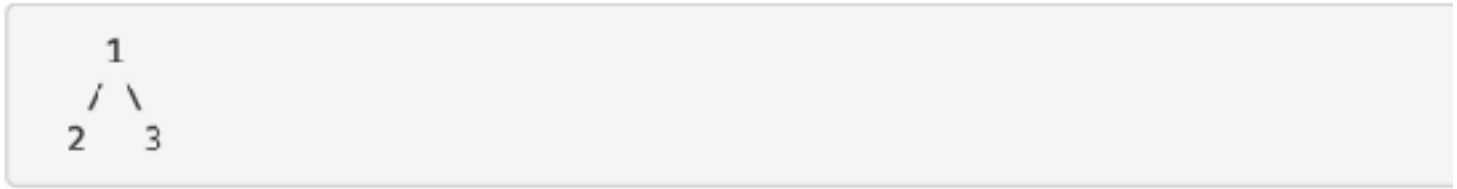
- The left subtree of a node contains only nodes with keys **less than** the node's key.
- The right subtree of a node contains only nodes with keys **greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.

Example 1:



Binary tree `[2,1,3]`, return true.

Example 2:

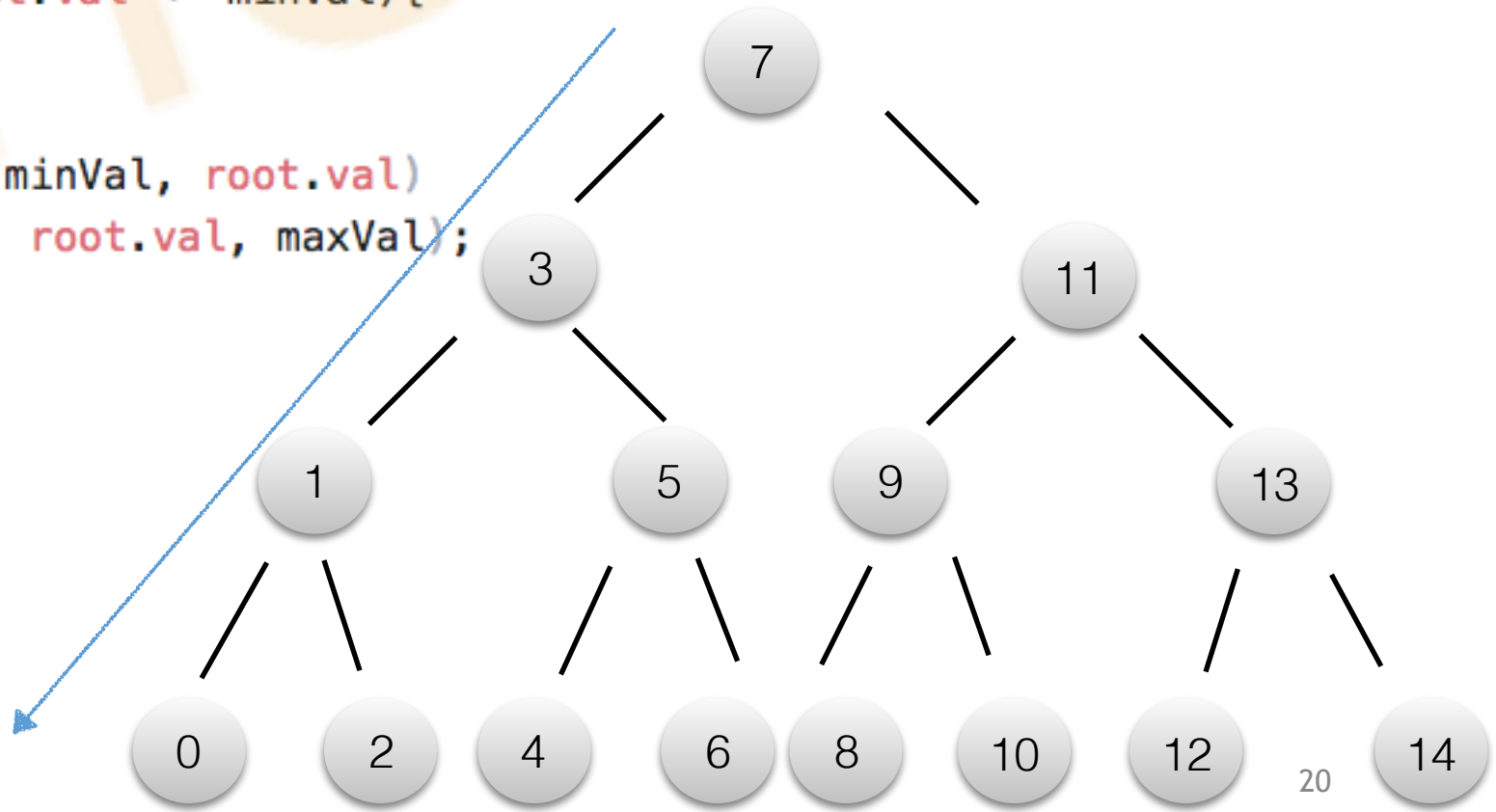


Binary tree `[1,2,3]`, return false.

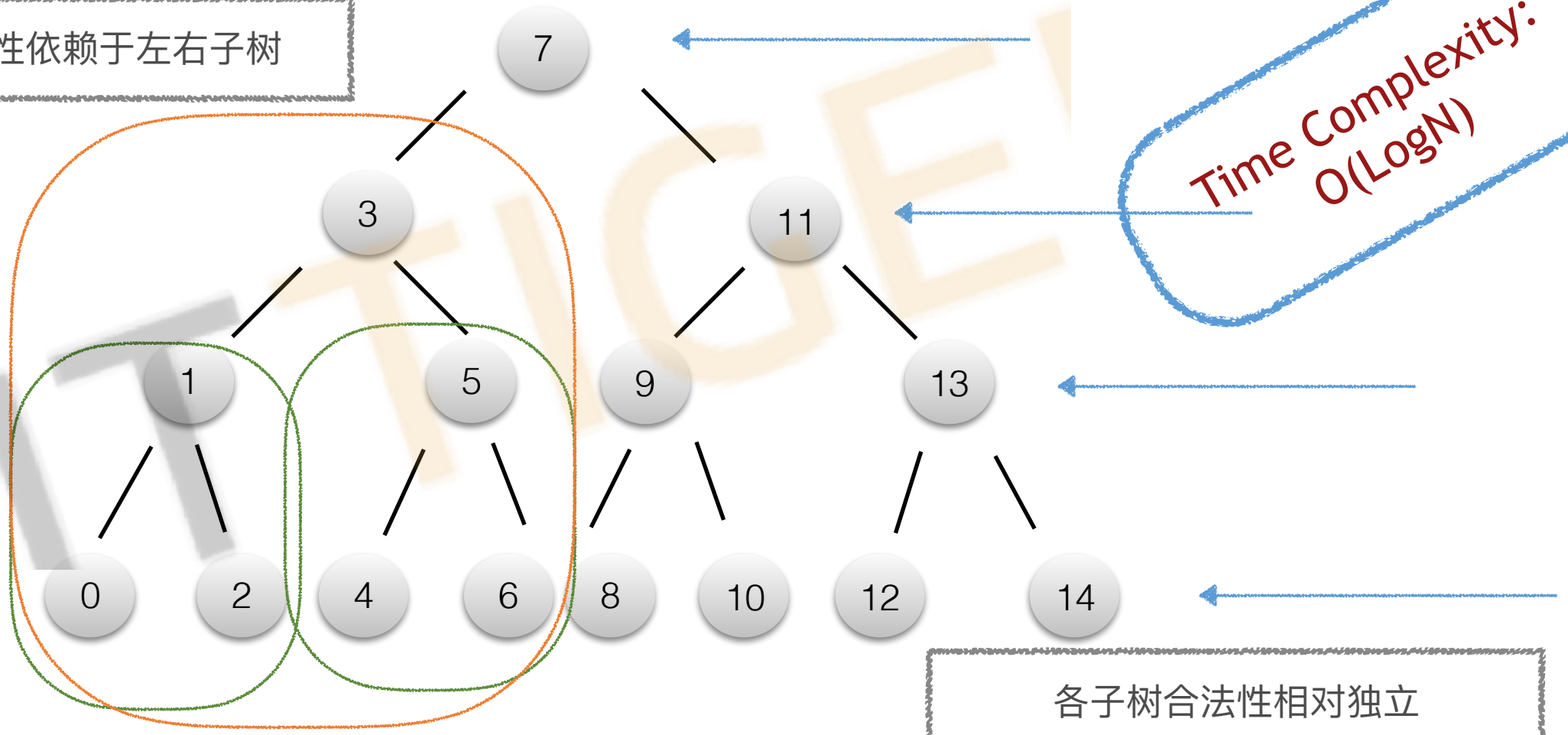
```java
public boolean isValidBST(TreeNode root) {
    return isValidBST(root, Long.MIN_VALUE, Long.MAX_VALUE);
}

public boolean isValidBST(TreeNode root, long minVal, long maxVal) {
    if (root == null){
        return true;
    }
    if (root.val >= maxVal || root.val <= minVal){
        return false;
    }
    return isValidBST(root.left, minVal, root.val)
        && isValidBST(root.right, root.val, maxVal);
}
```

Time Complexity: O(N)



当前树合法性依赖于左右子树



Time Complexity:
O(LogN)

各子树合法性相对独立

Divide & Conquer

程序入口

```java
public boolean isValidBST(TreeNode root) {
    ExecutorService executorService = Executors.newCachedThreadPool();
    return isValidBST(executorService, root);
}

public boolean isValidBST(ExecutorService executorService, TreeNode root) {
    if (root == null){
        return true;
    }

    IsValidBST isValidBSTCallable = new IsValidBST(executorService, root, Integer.MIN_VALUE, Integer.MAX_VALUE);

    try {
        return isValidBSTCallable.isValidTree();
    } catch (ExecutionException e) {
        e.printStackTrace();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    return false;
}
```

Overload Method

Initialize inner class

返回结果

Wrapper Class

```java
public class IsValidBST implements Callable<Boolean> {
    TreeNode root;
    long minVal;
    long maxVal;
    ExecutorService executorService;

    public IsValidBST(ExecutorService executorService, TreeNode root, long minVal, long maxVal) {
        this.executorService = executorService;
        this.root = root;
        this.minVal = minVal;
        this.maxVal = maxVal;
    }

    @Override
    public Boolean call() throws ExecutionException, InterruptedException {
        return isValidTree();
    }

    public Boolean isValidTree() throws ExecutionException, InterruptedException {
        if (root == null){
            return true;
        }
        if(root.val >= maxVal || root.val <= minVal){
            return false;
        }
        Future<Boolean> left = executorService.submit(new IsValidBST(executorService, root.left, minVal, root.val));
        Future<Boolean> right = executorService.submit(new IsValidBST(executorService, root.right, root.val, maxVal));

        return left.get() && right.get();
    }
}
```

class variable

Override Call() Method

向线程池 Submit 任务

get() 得到结果

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/
BFS Traversal:
[100]
[49, 150]
[24, 74, 125, 175]
[11, 36, 61, 87, 112, 137, 162, 188]
[5, 17, 30, 42, 55, 67, 80, 93, 106, 118, 131, 143, 156, 16
[2, 8, 14, 20, 27, 33, 39, 45, 52, 58, 64, 70, 77, 83, 90,
[0, 3, 6, 9, 12, 15, 18, 22, 25, 28, 31, 34, 37, 40, 43, 47
[1, 4, 7, 10, 13, 16, 19, 21, 23, 26, 29, 32, 35, 38, 41, 4

Inorder Traversal:
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22

Height:   8

true
true
true

Normal Recursion:   1239

Runnable Parallel:   149

Callable Parallel:   123
```

Mission Completed or Not?

Fork/Join Framework

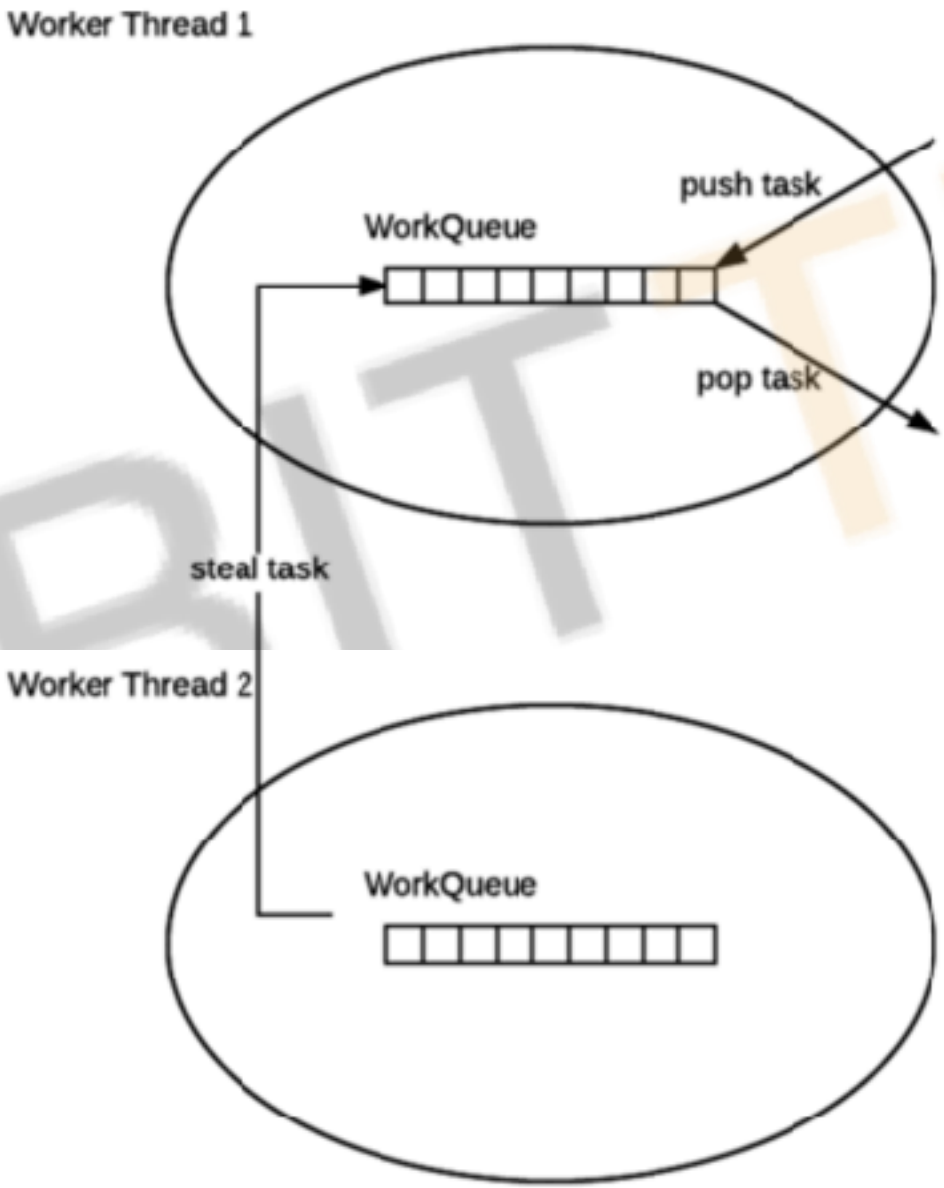
Class ForkJoinPool VS NormalThreadPool

Divide & Conquer

Recursion

CPU 密集型

Work Stealing



有限的线程处理无限的任务

- 每个工作线程都维护着一个工作队列（WorkQueue），这是一个双端队列（Deque），里面存放任务（ForkJoinTask）
- 每个工作线程在运行中产生新的任务时，会放入工作队列的队尾，使用的是 *LIFO* 方式
- 每个工作线程在处理自己的工作队列同时，会尝试窃取一个任务，窃取的任务位于其他线程的工作队列的队首，使用的是 *FIFO* 方式
- 在既没有自己的任务，也没有可以窃取的任务时，进入休眠

```
2  public abstract class RecursiveTask<V> extends ForkJoinTask<V> {
3      protected abstract V compute();
4  }
```

有返回值

```
6  public abstract class RecursiveAction extends ForkJoinTask<Void> {
7      protected abstract void compute();
8  }
```

无返回值

fork() Arranges to asynchronously execute this task in the pool

join()Returns the result of the computation when it is done

```java
public class ForkJoinTraversalTask extends RecursiveTask<Boolean> {
    TreeNode root;
    int height;
    long minVal;
    long maxVal;
    public ForkJoinTraversalTask(TreeNode root, int height, long minVal, long maxVal){
        this.root = root;
        this.height = height;
        this.minVal = minVal;
        this.maxVal = maxVal;
    }
    public Boolean isValidBST(TreeNode root, long minVal, long maxVal) {
        if (root == null){
            return true;
        }
        if (root.val >= maxVal || root.val <= minVal){
            return false;
        }
        return isValidBST(root.left, minVal, root.val) && isValidBST(root.right, root.val, maxVal);
    }
    @Override
    public Boolean compute(){
        if(height <= 3){
            return isValidBST(root, this.minVal, this.maxVal);
        } else {
            ForkJoinTraversalTask left = new ForkJoinTraversalTask(root.left, height - 1, minVal, root.val);
            ForkJoinTraversalTask right = new ForkJoinTraversalTask(root.right, height - 1, root.val, maxVal);
            left.fork();
            right.fork();

            return left.join() && right.join();
        }
    }
}
```

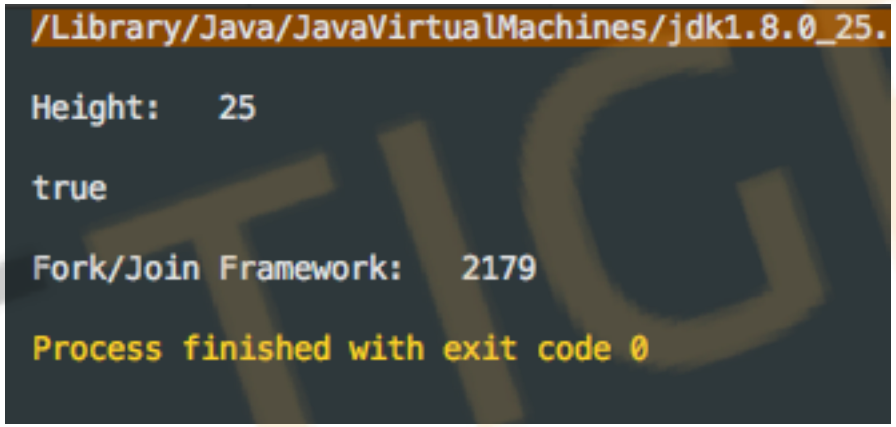
extends RecursionTask

Normal Recursion Method

```java
Long t1 = System.currentTimeMillis();
```

height <= 3 跑 normal recursion method

任务较大，分解执行



Mission Completed !!!

彩蛋

```
3   forkJoinPool.submit(() -> ioMappingUris.parallelStream().forEach(ioMappingUri -> {
4
5       RequestContext.initialize(new HashMap<>(requestContext));
6
7       IoMapping ioMapping = ioMappingServiceGateway.getIoMapping(ioMappingUri);
8
9       synchronized (ioMappingsMap) {
10
11          ioMappingsMap.put(ioMapping.getUri(), ioMapping);
12
13      }
14
15  })).join();
```

Q & A

Thank you