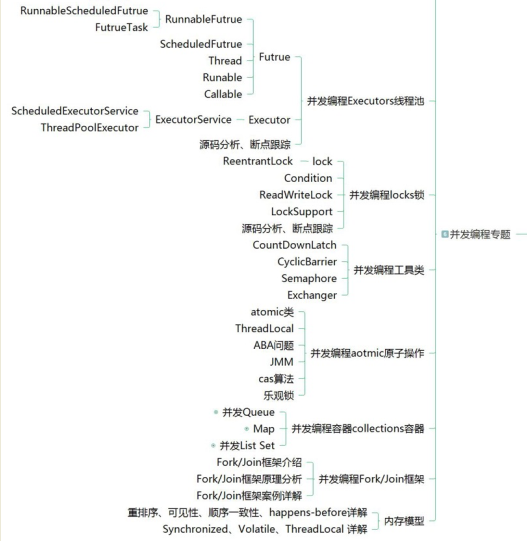
Java并发编程（一）

一.项目摘要：

**1.知识图谱**



**2.项目摘要图：**

二.项目具体实施：

**A.** **【并发编程Executors线程池】**

**RunnableFutrue**

**RunnableScheduledFutrue**

**FutrueTask**

**ScheduledFutrue**

**Thread**

**Runable**

**Callable**

**Executor**

**ExecutorService**

**ScheduledExecutorService（ThreadPoolExecutor）**

**源码分析、断点跟踪**

**B.** **【并发编程locks锁】**

**lock**

**ReentrantLock**

**Condition**

**ReadWriteLock**

**LockSupport**

**源码分析、断点跟踪**

**C.** **【并发类编程工具】**

**CountDownLatch**

**CyclicBarrier**

**Semaphore**

**Exchange**

**D.** **【并发编程容器collections】**

**并发Queue：BlockingQueue**

**Map：ConcurrentHashMap、HashMap、HashTable**

**并发List Set：CopyOnWriteArrayList、CopyOnWriteArraySet、**

**ArrayList、 LinkedList**

**E.** **【并发编程aotmic原子操作】**

**atomic类**

**ThreadLocal**

**ABA问题**

**JMM**

**cas算法**

**乐观锁**

**F.** **【内存模型】**

**重排序、可见性、顺序一致性**

**happens-before详解**

**Synchronized详解**

**Volatile详解**

**ThreadLocal详解**

**G.【一些框架】**

**11.future**

**12.MasterWorker**

**14.Disruptor框架**

三.具体代码：

**1.Thread：**

**(1)ThreadDemo.java**

**package com.thread;**

**import java.util.Queue;**

**/\*\***

**\***

**\*1. extend Thead class**

**\*2. override run method**

**\*/**

**public class ThreadDemo extends Thread{**

**private String threadName="";**

**private Queue<Integer> tickets=null;**

**public ThreadDemo(String threadName,Queue<Integer> tickets) {**

**this.tickets=tickets;**

**this.threadName=threadName;**

**}**

**@Override**

**public void run() {**

**Integer ticket=null;**

**while ((ticket=tickets.poll())!=null) {**

**System.out.println(threadName+":"+ticket);**

**}**

**}**

**}**

**(2)测试类：ThreadDemoTest.java**

**package com.thread.test;**

**import java.util.LinkedList;**

**import java.util.Queue;**

**import com.thread.ThreadDemo;**

**public class ThreadDemoTest {**

**private static Queue<Integer> tickets = new LinkedList<Integer>();**

**public static void main(String[] args) {**

**for (int i =0; i <= 100; i++) {**

**tickets.add(i);**

**}**

**new ThreadDemo("thread1",tickets).start();**

**new ThreadDemo("thread2",tickets).start();**

**}**

**}**

**(3) RunnableDemo.java**

**package com.runnable;**

**import java.util.Queue;**

**/\***

**\* implement Runnable interface**

**\* override run method**

**\*/**

**public class RunnableDemo implements Runnable {**

**private Queue<Integer> tickets=null;**

**public RunnableDemo(Queue<Integer> tickets) {**

**this.tickets=tickets;**

**}**

**@Override**

**public void run() {**

**// TODO Auto-generated method stub**

**Integer ticket=null;**

**while ((ticket=tickets.poll())!=null) {**

**System.out.println(Thread.currentThread().getName()+":"+ticket);**

**}**

**}**

**}**

**(4) RunnableDemoTest.java**

**package com.runnable.test;**

**import java.util.LinkedList;**

**import java.util.Queue;**

**import com.runnable.RunnableDemo;**

**public class RunnableDemoTest {**

**private static Queue<Integer> tickets = new LinkedList<Integer>();**

**public static void main(String[] args) {**

**Runnable runnable=new RunnableDemo(tickets);**

**for (int i =0; i <= 100; i++) {**

**tickets.add(i);**

**}**

**new Thread(runnable,"thread1").start();**

**new Thread(runnable,"thread2").start();**

**}**

**}**

**(5) SynchronizedDemo.java**

**package com.synchronizedDemo;**

**public class SynchronizedDemo {**

**synchronized public void methodA() {**

**}**

**synchronized static void methodB() {**

**}**

**public void methodC() {**

**synchronized (this) {**

**}**

**}**

**public void methodD() {**

**synchronized (SynchronizedDemo.class) {**

**}**

**}**

**}**

**(6) DirtyreadDemo.java**

**package com.dirtyreadDemo;**

**public class DirtyreadDemo {**

**private String username="oldUsername";**

**private String password="oldPassword";**

**public synchronized void updateUser(String username,String password) {**

**this.username=username;**

**try {**

**Thread.sleep(4000);**

**} catch (InterruptedException e) {**

**// TODO Auto-generated catch block**

**e.printStackTrace();**

**}**

**this.password=password;**

**System.out.println("update: username: "+username+",password:"+password);**

**}**

**public void queryUser() {**

**System.out.println("query: username: "+username+",password:"+password);**

**}**

**}**

**(7) DirtyreadDemo2.java**

**package com.dirtyreadDemo;**

**public class DirtyreadDemo2 {**

**private String username="oldUsername";**

**private String password="oldPassword";**

**public synchronized void updateUser(String username,String password) {**

**this.username=username;**

**try {**

**Thread.sleep(4000);**

**} catch (InterruptedException e) {**

**// TODO Auto-generated catch block**

**e.printStackTrace();**

**}**

**this.password=password;**

**System.out.println("update: username: "+username+",password:"+password);**

**}**

**public synchronized void queryUser() {**

**System.out.println("query: username: "+username+",password:"+password);**

**}**

**}**

**(8) DirtyreadDemoTest.java**

**package com.dirtyreadDemo.test;**

**import org.junit.Test;**

**import com.dirtyreadDemo.DirtyreadDemo;**

**import com.dirtyreadDemo.DirtyreadDemo2;**

**public class DirtyreadDemoTest {**

**@Test**

**public void dirtyreadDemoTest() {**

**String username="newUsername";**

**String password="newPassword";**

**final DirtyreadDemo dirtyreadDemo= new DirtyreadDemo();**

**new Thread(new Runnable() {**

**@Override**

**public void run() {**

**// TODO Auto-generated method stub**

**dirtyreadDemo.updateUser(username, password);**

**}**

**}).start();**

**try {**

**Thread.sleep(1000);**

**} catch (InterruptedException e) {**

**// TODO Auto-generated catch block**

**e.printStackTrace();**

**}**

**dirtyreadDemo.queryUser();**

**}**

**@Test**

**public void dirtyreadDemo2Test() {**

**String username="newUsername";**

**String password="newPassword";**

**final DirtyreadDemo2 dirtyreadDemo= new DirtyreadDemo2();**

**new Thread(new Runnable() {**

**@Override**

**public void run() {**

**// TODO Auto-generated method stub**

**dirtyreadDemo.updateUser(username, password);**

**}**

**}).start();**

**try {**

**Thread.sleep(1000);**

**} catch (InterruptedException e) {**

**// TODO Auto-generated catch block**

**e.printStackTrace();**

**}**

**dirtyreadDemo.queryUser();**

**}**

**}**

**(9) VolatileDemo.java**

**package com.volatileDemo;**

**/\***

**\* volatile does not guarantee atomicity**

**\*/**

**public class VolatileDemo {**

**public volatile int inc=0;**

**public void increase() {**

**inc++;**

**}**

**}**

**(10) VolatileDemo1.java**

**package com.volatileDemo;**

**public class VolatileDemo1 {**

**public int inc=0;**

**public synchronized void increase() {**

**inc++;**

**}**

**}**

**(11) VolatileDemo2.java**

**package com.volatileDemo;**

**import java.util.concurrent.locks.Lock;**

**import java.util.concurrent.locks.ReentrantLock;**

**public class VolatileDemo2 {**

**public volatile int inc=0;**

**Lock lock=new ReentrantLock();**

**public void increase() {**

**lock.lock();**

**try {**

**inc++;**

**}finally {**

**lock.unlock();**

**}**

**}**

**}**

**(12) VolatileDemo3.java**

**package com.volatileDemo;**

**import java.util.concurrent.atomic.AtomicInteger;**

**public class VolatileDemo3 {**

**public AtomicInteger inc=new AtomicInteger();**

**public void increase() {**

**inc.getAndIncrement();**

**}**

**}**

**(13) VolatileDemo4.java**

**package com.volatileDemo;**

**public class VolatileDemo4 {**

**volatile boolean flag=false;**

**public void method01() {**

**while (!flag) {**

**doSomething();**

**}**

**}**

**volatile boolean inited=false;**

**public void method02() {**

**init();**

**inited=true;**

**while (!inited) {**

**sleep();**

**}**

**}**

**public void init() {**

**}**

**public void sleep() {**

**}**

**public void doSomething() {**

**}**

**}**

**//Double check**

**class Singleton{**

**private volatile static Singleton instance=null;**

**private Singleton() {}**

**public static Singleton getInstance() {**

**if(instance==null) {**

**synchronized (Singleton.class) {**

**if (instance==null) {**

**instance=new Singleton();**

**}**

**}**

**}**

**return instance;**

**}**

**}**

**14-1：CommonCook.java**

**package com.futrueDemo;**

**public class CommonCook {**

**public static void main(String[] args) throws InterruptedException {**

**long startTime=System.currentTimeMillis();**

**OnlineShopping thread=new OnlineShopping();**

**thread.start();**

**thread.join();**

**Thread.sleep(2000);**

**Shicai shicai=new Shicai();**

**System.out.println("第二步：食材到位");**

**System.out.println("第三步：开始展现厨艺");**

**cook(thread.chuju, shicai);**

**System.out.println("总共用时"+(System.currentTimeMillis()-startTime)+"ms");**

**}**

**static class OnlineShopping extends Thread{**

**private Chuju chuju;**

**@Override**

**public void run() {**

**System.out.println("第一步:下单");**

**System.out.println("第二步：等待送达");**

**try {**

**Thread.sleep(5000);**

**} catch (InterruptedException e) {**

**// TODO Auto-generated catch block**

**e.printStackTrace();**

**}**

**System.out.println("第一步：快递送到");**

**chuju =new Chuju();**

**}**

**}**

**static void cook(Chuju chuju,Shicai shicai) {}**

**static class Chuju {}**

**static class Shicai {}**

**}**

**14-2: FutureCook.java**

**package com.futrueDemo;**

**import java.util.concurrent.Callable;**

**import java.util.concurrent.ExecutionException;**

**import java.util.concurrent.FutureTask;**

**public class FutureCook {**

**public static void main(String[] args) throws InterruptedException, ExecutionException {**

**long startTime =System.currentTimeMillis();**

**Callable<Chuju> onlineShopping=new Callable<Chuju>() {**

**@Override**

**public Chuju call() throws Exception {**

**// TODO Auto-generated method stub**

**System.out.println("第一步，下单");**

**System.out.println("第一步：等待送货");**

**Thread.sleep(5000);**

**System.out.println("第一步:快递送到");**

**return new Chuju();**

**}**

**};**

**FutureTask<Chuju> task=new FutureTask<>(onlineShopping);**

**new Thread(task).start();**

**//第二步 : 去超市购买食材**

**Thread.sleep(2000);**

**Shicai shicai=new Shicai();**

**System.out.println("第三步：食材到位");**

**//第三步： 用厨具烹饪食材**

**if (!task.isDone()) { //联系快递员，询问是否到货**

**System.out.println("第三步：厨具还没到，心情好就等着（心情不好就条用cancel方法取消订单）");**

**}**

**Chuju chuju=task.get();**

**System.out.println("第三步:厨具到位，开始展现厨艺");**

**cook(chuju, shicai);**

**System.out.println("总共用时："+(System.currentTimeMillis()-startTime)+"ms");**

**}**

**static void cook(Chuju chuju,Shicai shicai) {}**

**static class Chuju {}**

**static class Shicai {}**

**}**

**15-1. ConnectionPool.java**

**package com.futrueDemo;**

**import java.util.concurrent.ConcurrentHashMap;**

**public class ConnectionPool {**

**private ConcurrentHashMap<String, Connection> pool=new ConcurrentHashMap<String,Connection>();**

**public Connection getConnect(String key) {**

**Connection conn=null;**

**if(pool.containsKey(key)) {**

**conn=pool.get(key);**

**}else {**

**conn=createConnection();**

**}**

**return conn;**

**}**

**public Connection createConnection() {**

**return new Connection();**

**}**

**class Connection{};**

**}**

**15-2：FutureConnectionPool.java**

**package com.futrueDemo;**

**import java.util.concurrent.Callable;**

**import java.util.concurrent.ConcurrentHashMap;**

**import java.util.concurrent.ExecutionException;**

**import java.util.concurrent.FutureTask;**

**public class FutureConnectionPool {**

**private ConcurrentHashMap<String,FutureTask<Connection>> pool=new ConcurrentHashMap<String,FutureTask<Connection>>();**

**public Connection getConnection(String key) throws InterruptedException, ExecutionException {**

**FutureTask<Connection> connectionTask=pool.get(key);**

**if(connectionTask!=null) {**

**return connectionTask.get();**

**}else {**

**Callable<Connection> callable=new Callable<Connection>() {**

**@Override**

**public Connection call() throws Exception {**

**// TODO Auto-generated method stub**

**return createConnection();**

**}**

**};**

**FutureTask<Connection> newTask=new FutureTask<Connection>(callable);**

**connectionTask=pool.putIfAbsent(key,newTask);**

**if(connectionTask==null) {**

**connectionTask=newTask;**

**connectionTask.run();**

**}**

**}**

**return connectionTask.get();**

**}**

**public Connection createConnection() {**

**return new Connection();**

**}**

**class Connection{}**

**}**

**16-1: ThreadLocalVar.java**

**package com.threadLocal;**

**import java.util.HashMap;**

**import java.util.Map;**

**public class ThreadLocalVar<T> {**

**Map<Long,T> threadVarMap=new HashMap<Long,T>();**

**public T get() {**

**return threadVarMap.get(Thread.currentThread().getId());**

**}**

**public void set(T value) {**

**threadVarMap.put(Thread.currentThread().getId(), value);**

**}**

**}**

**16-2: MyTest.java**

**package com.threadLocal;**

**public class MyTest {**

**ThreadLocalVar<Long> longLocal=new ThreadLocalVar<Long>();**

**ThreadLocalVar<String> stringLocal=new ThreadLocalVar<String>();**

**public void set() {**

**longLocal.set(Thread.currentThread().getId());**

**stringLocal.set(Thread.currentThread().getName());**

**}**

**public long getLong() {**

**return longLocal.get();**

**}**

**public String getString() {**

**return stringLocal.get();**

**}**

**public static void main(String[] args) throws InterruptedException {**

**final MyTest test=new MyTest();**

**test.set();**

**System.out.println(test.getLong());**

**System.out.println(test.getString());**

**for(int i=0;i<3;i++) {**

**Thread thread=new Thread() {**

**public void run() {**

**test.set();**

**System.out.println(test.getLong());**

**System.out.println(test.getString());**

**};**

**};**

**thread.start();**

**thread.join();**

**}**

**System.out.println(test.getLong());**

**System.out.println(test.getString());**

**}**

**}**

**Disruptor实例**

**17-1: LongEvent.java**

**package com.disruptorDemo01;**

**public class LongEvent {**

**private long value;**

**public long getValue() {**

**return value;**

**}**

**public void setValue(long value) {**

**this.value=value;**

**}**

**}**

**17-2: LongEventFactory.java**

**package com.disruptorDemo01;**

**import com.lmax.disruptor.EventFactory;**

**//2.创建事件**

**public class LongEventFactory implements EventFactory{**

**@Override**

**public Object newInstance() {**

**// TODO Auto-generated method stub**

**return new LongEvent();**

**}**

**}**

**17-3: LongEventHandler.java**

**package com.disruptorDemo01;**

**import com.lmax.disruptor.EventHandler;**

**//消费者(事件处理器)**

**public class LongEventHandler implements EventHandler<LongEvent>{**

**@Override**

**public void onEvent(LongEvent longEvent, long l, boolean b) throws Exception {**

**// TODO Auto-generated method stub**

**System.out.println(longEvent.getValue());**

**}**

**}**

**17-4: LongEventProducer.java**

**package com.disruptorDemo01;**

**//数据生产者**

**import java.nio.ByteBuffer;**

**import com.lmax.disruptor.RingBuffer;**

**public class LongEventProducer {**

**private final RingBuffer<LongEvent> ringBuffer;**

**public LongEventProducer(RingBuffer<LongEvent> ringBuffer) {**

**// TODO Auto-generated constructor stub**

**this.ringBuffer=ringBuffer;**

**}**

**public void onData(ByteBuffer bb) {**

**long sequence=ringBuffer.next();**

**try {**

**LongEvent event=ringBuffer.get(sequence);**

**event.setValue(bb.getLong(0));**

**} finally {**

**ringBuffer.publish(sequence);**

**}**

**}**

**}**

**17-5: LongEventProducerWithTranslator.java**

**package com.disruptorDemo01;**

**import java.nio.ByteBuffer;**

**import com.lmax.disruptor.EventTranslatorOneArg;**

**import com.lmax.disruptor.RingBuffer;**

**public class LongEventProducerWithTranslator {**

**private static final EventTranslatorOneArg<LongEvent, ByteBuffer> TRANSLATOR=new EventTranslatorOneArg<LongEvent, ByteBuffer>() {**

**@Override**

**public void translateTo(LongEvent longEvent, long l, ByteBuffer bb) {**

**// TODO Auto-generated method stub**

**System.out.println(bb.getLong());**

**}**

**};**

**private final RingBuffer<LongEvent> ringBuffer;**

**public LongEventProducerWithTranslator(RingBuffer<LongEvent> ringBuffer) {**

**this.ringBuffer=ringBuffer;**

**}**

**public void onData(ByteBuffer bb) {**

**ringBuffer.publishEvent(TRANSLATOR,bb);**

**}**

**}**

**17-6: LongEventMain.java**

**package com.disruptorDemo01;**

**import java.nio.ByteBuffer;**

**import java.util.concurrent.Executor;**

**import java.util.concurrent.Executors;**

**import com.lmax.disruptor.RingBuffer;**

**import com.lmax.disruptor.dsl.Disruptor;**

**public class LongEventMain {**

**public static void main(String[] args) throws InterruptedException {**

**Executor executor=Executors.newCachedThreadPool();**

**LongEventFactory factory=new LongEventFactory();**

**int bufferSize=1024;**

**Disruptor<LongEvent> disruptor=new Disruptor<LongEvent>(factory, bufferSize, executor);**

**disruptor.handleEventsWith(new LongEventHandler());**

**disruptor.start();**

**RingBuffer<LongEvent> ringBuffer=disruptor.getRingBuffer();**

**LongEventProducer producer=new LongEventProducer(ringBuffer);**

**ByteBuffer bb=ByteBuffer.allocate(8);**

**for (long l=0;true;l++) {**

**bb.putLong(0,1);**

**producer.onData(bb);**

**Thread.sleep(1000);**

**}**

**}**

**}**

三.源码链接：

[**https://github.com/1945487635/concurrentDemo**](https://github.com/1945487635/concurrentDemo)

五.参考链接：

**1.** [**https://www.cnblogs.com/lwbqqyumidi/p/3804883.html**](https://www.cnblogs.com/lwbqqyumidi/p/3804883.html)

**参考内容：Thread,Runnable**

**2.** [**https://blog.csdn.net/pengdandezhi/article/details/65661995**](https://blog.csdn.net/pengdandezhi/article/details/65661995)

**参考内容：脏读**

**3.** [**https://www.cnblogs.com/dolphin0520/p/3920373.html**](https://www.cnblogs.com/dolphin0520/p/3920373.html)

**参考内容：volatile关键字**

**4.** **futrue：<https://www.cnblogs.com/cz123/p/7693064.html>**

**5.** **ThreadLocal：**[**https://www.cnblogs.com/cz123/p/7469245.html**](https://www.cnblogs.com/cz123/p/7469245.html)

本内容由安康学院“雨季”原创。