Оглавление

[Java 8 2](#_Toc74776033)

[java.io 2](#_Toc74776034)

[interface Serializable 2](#_Toc74776035)

[java.lang 3](#_Toc74776036)

[interface CharSequence 3](#_Toc74776037)

[interface Comparable<T> 6](#_Toc74776038)

[class Object 7](#_Toc74776039)

[class String 9](#_Toc74776040)

[Многопоточность 10](#_Toc74776041)

[Многопоточность в Java: основы 10](#_Toc74776042)

[Многопоточность в Java: средства стандартной библиотеки 31](#_Toc74776043)

[Новая тема 40](#_Toc74776044)

# Java 8

## java.io

### interface Serializable

package java.io;

public interface Serializable

## java.lang

### interface CharSequence

package java.lang;

import

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7. | java.util.NoSuchElementException;  java.util.PrimitiveIterator;  java.util.Spliterator;  java.util.Spliterators;  java.util.function.IntConsumer;  java.util.stream.IntStream;  java.util.stream.StreamSupport; |

public interface CharSequence

методы

|  |  |
| --- | --- |
| 1. | int length(); |
| 1. | char charAt(int index); |
| 1. | CharSequence subSequence(int start, int end); |
| 1. | String toString(); |

методы дефолтные

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14.  15.  16.  17.  18.  19.  20.  21.  22.  23.  24.  25.  26.  27.  28.  29.  30.  31.  32.  33.  34.  35. | public default IntStream chars() {  class CharIterator  implements PrimitiveIterator.OfInt {  int cur = 0;  public boolean hasNext() {  return cur < length();  }  public int nextInt() {  if (hasNext()) {  return charAt(cur++);  } else {  throw new NoSuchElementException();  }  }  @Override  public void forEachRemaining(  IntConsumer block) {  for (; cur < length(); cur++) {  block.accept(charAt(cur));  }  }  }  return StreamSupport.intStream(() ->  Spliterators.spliterator(  new CharIterator(),  length(),  Spliterator.ORDERED),  Spliterator.SUBSIZED | Spliterator.SIZED  | Spliterator.ORDERED,  false);  } |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14.  15.  16.  17.  18.  19.  20.  21.  22.  23.  24.  25.  26.  27.  28.  29.  30.  31.  32.  33.  34.  35.  36.  37.  38.  39.  40.  41.  42.  43.  44.  45.  46.  47.  48.  49.  50.  51.  52.  53.  54.  55.  56.  57.  58.  59.  60.  61.  62. | public default IntStream codePoints() {  class CodePointIterator  implements PrimitiveIterator.OfInt {  int cur = 0;  @Override  public void forEachRemaining(  IntConsumer block) {  final int length = length();  int i = cur;  try {  while (i < length) {  char c1 = charAt(i++);  if (!Character.isHighSurrogate(c1)  || i >= length) {  block.accept(c1);  } else {  char c2 = charAt(i);  if (Character.isLowSurrogate(c2)) {  i++;  block.accept(  Character.toCodePoint(c1, c2));  } else {  block.accept(c1);  }  }  }  } finally {  cur = i;  }  }  public boolean hasNext() {  return cur < length();  }  public int nextInt() {  final int length = length();  if (cur >= length) {  throw new NoSuchElementException();  }  char c1 = charAt(cur++);  if (Character.isHighSurrogate(c1) && cur  < length) {  char c2 = charAt(cur);  if (Character.isLowSurrogate(c2)) {  cur++;  return Character.toCodePoint(c1, c2);  }  }  return c1;  }  }  return StreamSupport.intStream(() ->  Spliterators.spliteratorUnknownSize(  new CodePointIterator(),  Spliterator.ORDERED),  Spliterator.ORDERED,  false);  } |

### interface Comparable<T>

package java.lang;

import

|  |  |
| --- | --- |
| 1. | import java.util.\*; |

public interface Comparable<T>

методы

|  |  |
| --- | --- |
| 1. | public int compareTo(T o); |

### class Object

package java.lang;

public class Object

конструкторы по умолчанию

методы

|  |  |
| --- | --- |
| 1. | public final native Class<?> getClass(); |
| 1. | public native int hashCode(); |
| 1.  2.  3. | public boolean equals(Object obj) {  return (this == obj);  } |
| 1.  2. | protected native Object clone()  throws CloneNotSupportedException; |
| 1.  2.  3.  4. | public String toString() {  return getClass().getName() + "@"  + Integer.toHexString(hashCode());  } |
| 1. | public final native void notify(); |
| 1. | public final native void notifyAll(); |
| 1.  2. | public final native void wait(long timeout)  throws InterruptedException; |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14.  15.  16.  17.  18. | public final void wait(long timeout, int nanos)  throws InterruptedException {  if (timeout < 0) {  throw new IllegalArgumentException(  "timeout value is negative");  }  if (nanos < 0 || nanos > 999999) {  throw new IllegalArgumentException(  "nanosecond timeout value out of range");  }  if (nanos > 0) {  timeout++;  }  wait(timeout);  } |
| 1.  2.  3.  4. | public final void wait()  throws InterruptedException {  wait(0);  } |
| 1. | protected void finalize() throws Throwable { } |

методы приватные

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5. | private static native void registerNatives();  static {  registerNatives();  } |

### class String

xv

sad

# Многопоточность

## Многопоточность в Java: основы

Мотивация

* Одновременное выполнение нескольких действий (например, отрисовка пользовательского интерфейса и передача файлов по сети);
* Ускорение вычислений (при наличии нескольких вычислительных ядер).

Закон Амдала

S — ускорение;

P — доля вычислений, которые возможно распараллелить;

N — количество вычислительных ядер.

Параллелизм в Java

Запуск нескольких JVM на одной или на разных компьютерах:

* нет общей памяти;
* взаимодействие через файловую систему или сетевое соединение.

Запуск нескольких потоков внутри JVM:

* есть общая память;
* обширная поддержка в языке и стандартной библиотеке.

Проблемы параллельных программ

* Гонка (race condition) — проблемы при использовании общей памяти, решение — эксклюзивный доступ к данным (примитивы синхронизации);
* Взаимная блокировка (deadlock).

java.lang.Thread

Потоки представлены экземплярами класса java.lang.Thread.

Основные методы:

* String getName();
* long getId();
* boolean isDeamon();
* StackTraceElement[] getStackTrace();
* ThreadGroup getThreadGroup().

Thread dump

Список всех потоков с их состояниями и stack trace’ами (кнопка в IDE (значок фотоаппарата в меню run)).

Создание потока

1‑й способ

|  |  |
| --- | --- |
| 1.  2.  3.  4. | public static void main() {  Thread thread1 = new NewThread();  thread1.start();  } |
| 1.  2.  3.  4.  5.  6.  7. | public class NewThread extends Thread {  @Override  public void run() {  ...  }  } |

или

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10. | public static void main() {  Thread thread2 = new Thread() {  @Override  public void run() {  ...  }  };  thread2.start();  } |

2‑й способ

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5. | public static void main() {  Runnable runnable = () -> ...;  Thread thread3 = new Thread(runnable);  thread3.start();  } |

Жизненный цикл потока

* создание объекта Thread;
* запуск (thread.start());
* работа (выполняется метод run(), thread.isAlive() == true);
* завершение (метод run() закончился или бросил исключение);
* завершенный поток нельзя перезапустить.

Примеры

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13. | public static void main() {  for (int i = 0; i < 10; i++) {  new HelloThread().start();  }  System.out.println("main поток");  }  private static class HelloThread extends Thread {  @Override  public void run() {  System.out.println(getName());  }  } |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11. | Thread-0  Thread-2  Thread-1  Thread-4  Thread-3  Thread-5  Thread-6  main поток  Thread-8  Thread-7  Thread-9 |

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14.  15. | public static void main() {  for (int i = 0; i < 10; i++) {  new Thread(new HelloRunnable()).start();  }  System.out.println("main поток");  }  private static class HelloRunnable  implements Runnable {  @Override  public void run() {  System.out.println(  Thread.currentThread().getName());  }  } |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11. | Thread-1  main поток  Thread-7  Thread-5  Thread-3  Thread-0  Thread-2  Thread-9  Thread-8  Thread-6  Thread-4 |

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8. | public static void main() {  for (int i = 0; i < 10; i++) {  new Thread(() -> System.out.println(  Thread.currentThread().getName()))  .start();  }  System.out.println("main поток");  } |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11. | Thread-0  Thread-5  Thread-1  main поток  Thread-3  Thread-6  Thread-4  Thread-7  Thread-8  Thread-9  Thread-2 |

С таким вариантом нужно быть осторожным. В данном случае нет никакого состояния, поэтому так делать можно.

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14.  15.  16.  17. | public static void main() {  HelloRunnable helloRunnable =  new HelloRunnable();  for (int i = 0; i < 10; i++) {  new Thread(helloRunnable).start();  }  System.out.println("main поток");  }  private static class HelloRunnable  implements Runnable {  @Override  public void run() {  System.out.println(  Thread.currentThread().getName());  }  } |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11. | Thread-0  Thread-4  Thread-5  Thread-3  Thread-6  main поток  Thread-2  Thread-1  Thread-8  Thread-7  Thread-9 |

Прерывание потока

* thread.interrapt();
* если поток находится в ожидании (sleep, join wait), то ожидание прерывается исключением InterruptedException;
* Иначе у потока просто устанавливается флаг interrupted: флаг проверяется методами interrupted() и isInterrupted(), проверять флаг и завершать поток надо самостоятельно;
* thread.join().

Примеры

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14.  15.  16.  17.  18.  19.  20.  21.  22.  23.  24.  25.  26.  27.  28.  29.  30.  31.  32.  33.  34.  35.  36.  37.  38.  39.  40.  41.  42.  43.  44.  45.  46.  47.  48.  49.  50. | public static void main()  throws InterruptedException {  Thread worker = new WorkerThread();  Thread sleeper = new SleeperThread();  System.out.println("Starting threads");  worker.start();  sleeper.start();  Thread.sleep(100L);  System.out.println("Interrupted threads");  worker.interrupt();  sleeper.interrupt();  System.out.println("Joining threads");  worker.join();  sleeper.join();  System.out.println("All done");  }  private static class WorkerThread  extends Thread {  @Override  public void run() {  long sum = 0;  for (int i = 0; i < 1\_000\_000\_000; i++) {  sum += i;  if (i % 100 == 0 && isInterrupted()) {  System.out.println("i = " + i);  break;  }  }  }  }  private static class SleeperThread  extends Thread {  @Override  public void run() {  try {  Thread.sleep(10\_000L);  } catch (InterruptedException e) {  System.out.println("Sleep interrupted");  }  }  } |
| 1.  2.  3.  4.  5.  6. | Starting threads  Interrupted threads  Joining threads  Sleep interrupted  i = 92633100  All done |

Программа зависает и ждет выполнение потоков.

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14.  15.  16.  17.  18.  19.  20.  21.  22.  23.  24.  25.  26.  27.  28.  29.  30.  31.  32.  33.  34.  35.  36.  37.  38.  39.  40.  41.  42.  43.  44.  45.  46.  47.  48.  49.  50. | public static void main()  throws InterruptedException {  Thread worker = new WorkerThread();  Thread sleeper = new SleeperThread();  System.out.println("Starting threads");  worker.start();  sleeper.start();  Thread.sleep(100L);  // System.out.println("Interrupted threads");  // worker.interrupt();  // sleeper.interrupt();  //  // System.out.println("Joining threads");  // worker.join();  // sleeper.join();  System.out.println("All done");  }  private static class WorkerThread  extends Thread {  @Override  public void run() {  long sum = 0;  for (int i = 0; i < 1\_000\_000\_000; i++) {  sum += i;  if (i % 100 == 0 && isInterrupted()) {  System.out.println("i = " + i);  break;  }  }  }  }  private static class SleeperThread  extends Thread {  @Override  public void run() {  try {  Thread.sleep(10\_000L);  } catch (InterruptedException e) {  System.out.println("Sleep interrupted");  }  }  } |
| 1.  2. | Starting threads  All done |

Программа больше не зависает, т. к. потоки стали демонами.

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14.  15.  16.  17.  18.  19.  20.  21.  22.  23.  24.  25.  26.  27.  28.  29.  30.  31.  32.  33.  34.  35.  36.  37.  38.  39.  40.  41.  42.  43.  44. | public static void main()  throws InterruptedException {  Thread worker = new WorkerThread();  worker.setDaemon(true);  Thread sleeper = new SleeperThread();  sleeper.setDaemon(true);  System.out.println("Starting threads");  worker.start();  sleeper.start();  Thread.sleep(100L);  System.out.println("All done");  }  private static class WorkerThread  extends Thread {  @Override  public void run() {  long sum = 0;  for (int i = 0; i < 1\_000\_000\_000; i++) {  sum += i;  if (i % 100 == 0 && isInterrupted()) {  System.out.println("i = " + i);  break;  }  }  }  }  private static class SleeperThread  extends Thread {  @Override  public void run() {  try {  Thread.sleep(10\_000L);  } catch (InterruptedException e) {  System.out.println("Sleep interrupted");  }  }  } |

Возможности встроенной синхронизации

* взаимное исключение (пока один поток что‑то делает, другие не могут ему помешать);
* ожидание и уведомление (поток ожидает уведомление от других потоков).

Ключевое слово synchronized

Синхронизированный метод (статический или нестатический)

|  |  |
| --- | --- |
| 1.  2.  3. | public synchronized void doSomething() {  ...  } |

Синхронизированный блок внутри нестатического метода

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5. | public void doSomething() {  synchronized (this) {  ...  }  } |

Синхронизированный блок внутри статического метода

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7. | public class C {  public static void doSomething() {  synchronized (C.class) {  ...  }  }  } |

Синхронизация блоков — по монитору указанного объекта.

Синхронизация методов — по монитору текущего объекта (this).

Синхронизация статических методов — по монитору класса.

Один момент: два разных синхронизованных метода одного класса не могут выполнять несколькими потоками параллельно, т. к. экземпляр объекта один и тот же, а значит и монитор один и тот же. А если один из них статический, то могут исполняться в параллельных потоках.

Ожидания и уведомления

Допустимо только внутри synchronized.

Приостанавливает текущий поток (поток засыпает и освобождает блокировку текущего объекта): void wait, void wait(long millis), void wait(long millis, int nanos).

Будит поток: void notify(), void notifyAll() (будет все потоки).

Примеры

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14.  15.  16.  17.  18.  19.  20.  21.  22.  23.  24.  25.  26.  27.  28.  29.  30.  31.  32.  33.  34.  35.  36.  37.  38.  39.  40.  41.  42.  43.  44.  45.  46.  47.  48.  49.  50.  51.  52.  53. | public static void main()  throws InterruptedException {  Account account = new Account(100\_000);  System.out.println(  "Begin balance = " + account.getBalance());  Thread withdrawThread =  new WithdrawThread(account);  Thread depositThread =  new DepositThread(account);  withdrawThread.start();  depositThread.start();  withdrawThread.join();  depositThread.join();  System.out.println(  "End balance = " + account.getBalance());  }  private static class WithdrawThread  extends Thread {  private final Account account;  private WithdrawThread(Account account) {  this.account = account;  }  @Override  public void run() {  for (int i = 0; i < 20\_000; i++) {  account.withdraw(1);  }  }  }  private static class DepositThread  extends Thread {  private final Account account;  private DepositThread(Account account) {  this.account = account;  }  @Override  public void run() {  for (int i = 0; i < 20\_000; i++) {  account.deposit(1);  }  }  } |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14.  15.  16.  17.  18.  19.  20.  21.  22.  23.  24.  25.  26.  27.  28.  29.  30.  31.  32.  33.  34.  35.  36.  37.  38. | public class Account {  private long balance;  public Account() {  this(0L);  }  public Account(long balance) {  this.balance = balance;  }  public long getBalance() {  return balance;  }  public void deposit(long amount) {  checkAmountNonNegative(amount);  balance += amount;  }  public void withdraw(long amount) {  checkAmountNonNegative(amount);  if (balance < amount) {  throw new IllegalArgumentException(  "not enough money");  }  balance -= amount;  }  private static void checkAmountNonNegative(  long amount) {  if (amount < 0) {  throw new IllegalArgumentException(  "negative amount");  }  }  } |
| 1.  2. | Begin balance = 100000  End balance = 104013 |

Чтобы это исправить помечаем методы synchronized.

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14.  15.  16.  17.  18.  19.  20.  21.  22.  23.  24.  25.  26.  27.  28.  29.  30.  31.  32.  33.  34.  35.  36.  37.  38. | public class Account {  private long balance;  public Account() {  this(0L);  }  public Account(long balance) {  this.balance = balance;  }  public long getBalance() {  return balance;  }  public synchronized void deposit(long amount) {  checkAmountNonNegative(amount);  balance += amount;  }  public synchronized void withdraw(long amount) {  checkAmountNonNegative(amount);  if (balance < amount) {  throw new IllegalArgumentException(  "not enough money");  }  balance -= amount;  }  private static void checkAmountNonNegative(  long amount) {  if (amount < 0) {  throw new IllegalArgumentException(  "negative amount");  }  }  } |
| 1.  2. | Begin balance = 100000  End balance = 100000 |

Рекомендуется в synchronized помещать только маленький кусочек работы. В данном случае вся полезная работа оказывается в synchronized блоке, но в большинстве реальных ситуаций можно выделить маленький кусочек, требующий синхронизации.

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14.  15.  16.  17. | public void deposit(long amount) {  checkAmountNonNegative(amount);  synchronized (this) {  balance += amount;  }  }  public void withdraw(long amount) {  checkAmountNonNegative(amount);  synchronized (this) {  if (balance < amount) {  throw new IllegalArgumentException(  "not enough money");  }  balance -= amount;  }  } |

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14.  15.  16.  17.  18.  19.  20.  21.  22.  23.  24.  25.  26.  27.  28.  29.  30.  31. | public static void main()  throws InterruptedException {  Account account = new Account(0);  new DepositThread(account).start();  System.out.println(  "Calling waitAndWithdraw() ...");  account.waitAndWithdraw(50\_000\_000);  System.out.println(  "waitAndWithdraw() finished");  }  private static class DepositThread  extends Thread {  private final Account account;  private DepositThread(Account account) {  this.account = account;  }  @Override  public void run() {  for (int i = 0; i < 50\_000\_000; i++) {  account.deposit(1);  }  }  } |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14.  15.  16.  17.  18.  19.  20.  21.  22.  23.  24.  25.  26.  27.  28.  29.  30.  31.  32.  33.  34.  35.  36.  37.  38.  39.  40.  41.  42.  43.  44.  45.  46.  47.  48.  49.  50. | public class Account {  private long balance;  public Account() {  this(0L);  }  public Account(long balance) {  this.balance = balance;  }  public long getBalance() {  return balance;  }  public synchronized void deposit(long amount) {  checkAmountNonNegative(amount);  balance += amount;  notifyAll();  }  public synchronized void withdraw(long amount) {  checkAmountNonNegative(amount);  if (balance < amount) {  throw new IllegalArgumentException(  "not enough money");  }  balance -= amount;  }  public synchronized void waitAndWithdraw(  long amount)  throws InterruptedException {  checkAmountNonNegative(amount);  while (balance < amount) {  wait();  // System.out.println("Wakeup: " + balance);  }  balance -= amount;  }  private static void checkAmountNonNegative(  long amount) {  if (amount < 0) {  throw new IllegalArgumentException(  "negative amount");  }  }  } |
| 1.  2. | Calling waitAndWithdraw() ...  waitAndWithdraw() finished |

Атомарность

Чтение и запись полей всех типов, кроме long и double, происходит атомарно.

Если поле объявлено с модификатором volatile, то атомарно читаются и пишутся даже long и double.

Видимость

Изменения значений полей, сделанные одним потоком, могут быть не видны в другом потоке.

Изменения, сделанные одним потоком, могут быть видны в другом потоке.

Правила формализованы при помощи отношения happens-before (если в одном поток произошло некоторое событие X, а в другом потоке после этого произошло событие Z, то мы гарантировано знаем, что все, что произошло до X будет видно после Z).

Семантика final.

happens-before

Запись volatile-поля happens-before чтения этого поля.

Освобождение монитора happens-before захват того же монитора.

thread.start() happens-before thread.run().

Завершение thread.run() happens-before выход из thread.join().

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14.  15.  16.  17.  18.  19. | public class Singleton {  private int foo;  private String bar;  private Singleton() {  this.foo = 13;  this.bar = "zap";  }  private static Singleton instance;  public static Singleton getInstance() {  if (instance == null) {  instance = new Singleton();  }  return instance;  }  } |

Для многопоточной программы:

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14.  15.  16.  17.  18.  19.  20. | public class Singleton {  private int foo;  private String bar;  private Singleton() {  this.foo = 13;  this.bar = "zap";  }  private static Singleton instance;  public synchronized static  Singleton getInstance() {  if (instance == null) {  instance = new Singleton();  }  return instance;  }  } |

Ни в коем случае не делать такую штуку (можем получить ссылку на недостроенный объект):

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10. | public static Singleton getInstance() {  if (instance == null) {  synchronized (Singleton.class) {  if (instance == null) {  instance = new Singleton();  }  }  }  return instance;  } |

Но это лечится так:

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14.  15.  16.  17.  18.  19.  20.  21.  22.  23. | public class Singleton {  private int foo;  private String bar;  private Singleton() {  this.foo = 13;  this.bar = "zap";  }  private volatile static Singleton instance;  public static Singleton getInstance() {  if (instance == null) {  synchronized (Singleton.class) {  if (instance == null) {  instance = new Singleton();  }  }  }  return instance;  }  } |

Код

<https://github.com/java-the-best/multithreading-vladykin>

Источники

* <https://www.youtube.com/watch?v=zxZ0BXlTys0&ab_channel=ComputerScienceCenter>

## Многопоточность в Java: средства стандартной библиотеки

java.util.concurrent.atomic

* AtomicBoolean, AtomicInteger, AtomicLong, AtomocReference<V>;
* Операции: V get(), void set(V newValue), boolean compareAndSet(V expect, V update).

compareAndSet

Примитив compareAndSet позволяет реализовать другие операции.

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9. | public final int incrementAndGet() {  for (;;) {  int current = get();  int next current + 1;  if (compareAndSet(current, next)) {  return next;  }  }  } |

Примеры

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14.  15.  16.  17.  18.  19.  20.  21.  22.  23.  24.  25.  26.  27.  28.  29.  30. | public class SequenceGeneratorBroken {  private static volatile int counter = 0;  public static int nextInt() {  return counter++;  }  public static void main()  throws InterruptedException {  List<Thread> threads = new ArrayList<>();  for (int i = 0; i < 10; i++) {  Thread thread = new Thread(() -> {  for (int j = 0; j < 1000; j++) {  nextInt();  }  });  thread.start();  threads.add(thread);  }  for (Thread thread : threads) {  thread.join();  }  System.out.println(  "Counter final value: " + counter);  }  } |
| 1. | Counter final value: 9040 |

Правильное решение

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14.  15.  16.  17.  18.  19.  20.  21.  22.  23.  24.  25.  26.  27.  28.  29.  30. | public class SequenceGeneratorGood1 {  private static int counter = 0;  public static synchronized int nextInt() {  return counter++;  }  public static void main()  throws InterruptedException {  List<Thread> threads = new ArrayList<>();  for (int i = 0; i < 10; i++) {  Thread thread = new Thread(() -> {  for (int j = 0; j < 1000; j++) {  nextInt();  }  });  thread.start();  threads.add(thread);  }  for (Thread thread : threads) {  thread.join();  }  System.out.println(  "Counter final value: " + counter);  }  } |
| 1. | Counter final value: 10000 |

Еще одно правильное решение

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14.  15.  16.  17.  18.  19.  20.  21.  22.  23.  24.  25.  26.  27.  28.  29.  30.  31. | public class SequenceGeneratorGood2 {  private static final AtomicInteger counter =  new AtomicInteger();  public static int nextInt() {  return counter.getAndIncrement();  }  public static void main()  throws InterruptedException {  List<Thread> threads = new ArrayList<>();  for (int i = 0; i < 10; i++) {  Thread thread = new Thread(() -> {  for (int j = 0; j < 1000; j++) {  nextInt();  }  });  thread.start();  threads.add(thread);  }  for (Thread thread : threads) {  thread.join();  }  System.out.println(  "Counter final value: " + counter);  }  } |
| 1. | Counter final value: 10000 |

Semaphore

* класс java.util.concurrent.Semaphore;
* ограничивает одновременный доступ к ресурсу;
* в отличии от synchronized-блока, одновременно могут работать несколько потоков (но не более заданного N);
* операции: void acquire(), void release().

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10. | public static void main()  throws InterruptedException {  Semaphore semaphore = new Semaphore(10);  semaphore.acquire();  try {  // ...  } finally {  semaphore.release();  }  } |

Примеры

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14.  15.  16.  17.  18.  19.  20.  21.  22.  23.  24.  25.  26.  27.  28.  29.  30.  31.  32.  33.  34.  35.  36.  37.  38.  39.  40.  41.  42.  43.  44.  45.  46.  47.  48.  49.  50.  51.  52.  53.  54.  55.  56. | public class SemaphoreDemo {  public static void main()  throws InterruptedException {  Semaphore semaphore = new Semaphore(2);  List<Thread> threads = new ArrayList<>();  for (int i = 0; i < 10; i++) {  DemoThread thread =  new DemoThread(semaphore);  threads.add(thread);  thread.start();  }  Thread.sleep(20\_000);  for (Thread thread : threads) {  thread.interrupt();  }  }  private static class DemoThread extends Thread {  private final Semaphore semaphore;  public DemoThread(Semaphore semaphore) {  this.semaphore = semaphore;  }  @Override  public void run() {  try {  runUnsafe();  } catch (InterruptedException e) {  System.out.println(  getName() + " interrupted");  }  }  private void runUnsafe()  throws InterruptedException {  for (;;) {  semaphore.acquire();  try {  System.out.println(  getName() + " acquired semaphore");  Thread.sleep(5\_000L);  } finally {  System.out.println(  getName() + " releasing semaphore");  semaphore.release();  }  }  }  }  } |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14.  15.  16.  17.  18.  19.  20.  21.  22.  23.  24.  25.  26. | Thread-0 acquired semaphore  Thread-1 acquired semaphore  Thread-0 releasing semaphore  Thread-1 releasing semaphore  Thread-1 acquired semaphore  Thread-0 acquired semaphore  Thread-1 releasing semaphore  Thread-0 releasing semaphore  Thread-0 acquired semaphore  Thread-1 acquired semaphore  Thread-1 releasing semaphore  Thread-1 acquired semaphore  Thread-0 releasing semaphore  Thread-0 acquired semaphore  Thread-3 interrupted  Thread-9 interrupted  Thread-2 interrupted  Thread-8 interrupted  Thread-1 releasing semaphore  Thread-1 interrupted  Thread-4 interrupted  Thread-0 releasing semaphore  Thread-6 interrupted  Thread-7 interrupted  Thread-5 interrupted  Thread-0 interrupted |

Организация «честной очереди»

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14.  15.  16.  17.  18.  19.  20.  21.  22.  23.  24.  25.  26.  27.  28.  29.  30.  31.  32.  33.  34.  35.  36.  37.  38.  39.  40.  41.  42.  43.  44.  45.  46.  47.  48.  49.  50.  51.  52.  53.  54.  55.  56. | public class SemaphoreDemo {  public static void main()  throws InterruptedException {  Semaphore semaphore = new Semaphore(2, true);  List<Thread> threads = new ArrayList<>();  for (int i = 0; i < 10; i++) {  DemoThread thread =  new DemoThread(semaphore);  threads.add(thread);  thread.start();  }  Thread.sleep(20\_000);  for (Thread thread : threads) {  thread.interrupt();  }  }  private static class DemoThread extends Thread {  private final Semaphore semaphore;  public DemoThread(Semaphore semaphore) {  this.semaphore = semaphore;  }  @Override  public void run() {  try {  runUnsafe();  } catch (InterruptedException e) {  System.out.println(  getName() + " interrupted");  }  }  private void runUnsafe()  throws InterruptedException {  for (;;) {  semaphore.acquire();  try {  System.out.println(  getName() + " acquired semaphore");  Thread.sleep(5\_000L);  } finally {  System.out.println(  getName() + " releasing semaphore");  semaphore.release();  }  }  }  }  } |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14.  15.  16.  17.  18.  19.  20.  21.  22.  23.  24.  25.  26.  27.  28. | Thread-0 acquired semaphore  Thread-1 acquired semaphore  Thread-0 releasing semaphore  Thread-1 releasing semaphore  Thread-2 acquired semaphore  Thread-4 acquired semaphore  Thread-2 releasing semaphore  Thread-4 releasing semaphore  Thread-3 acquired semaphore  Thread-5 acquired semaphore  Thread-3 releasing semaphore  Thread-5 releasing semaphore  Thread-6 acquired semaphore  Thread-7 acquired semaphore  Thread-7 releasing semaphore  Thread-6 releasing semaphore  Thread-8 acquired semaphore  Thread-9 interrupted  Thread-8 releasing semaphore  Thread-8 interrupted  Thread-1 interrupted  Thread-2 interrupted  Thread-6 interrupted  Thread-4 interrupted  Thread-5 interrupted  Thread-3 interrupted  Thread-7 interrupted  Thread-0 interrupted |

23:00

Sdf

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14.  15.  16.  17.  18.  19.  20.  21.  22.  23.  24.  25.  26.  27.  28.  29.  30.  31.  32.  33.  34.  35.  36.  37.  38.  39.  40.  41.  42.  43.  44.  45.  46.  47.  48.  49.  50. |  |

Код

<https://github.com/java-the-best/multithreading-vladykin>

Источники

* <https://www.youtube.com/watch?v=umTVNoG3760&t=12s>

# Новая тема

Содержание

Код

Ссылка

Источники

* Ссылка с описанием