# java多线程学习

## 1.1传统线程

**package** cn.zz.test1;

**public** **class** TraditionalThread {

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

//继承重写run()

Thread thread1 = **new** Thread() {

@Override

**public** **void** run() {

**while**(**true**) {

System.out.println(Thread.currentThread().getName());

System.out.println(**this**.getName());

}

}

};

//thread1.start();

//实现Runnable接口

Thread thread2 = **new** Thread(**new** Runnable() {

@Override

**public** **void** run() {

**while**(**true**) {

//Thread.currentThread()得到当前的线程对象

System.out.println(Thread.currentThread().getName());

//this不代表线程

//System.out.println(this.getName());

}

}

});

//thread2.start();

//第二种更能体现面向对象，线程所运行代码在对象里面

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

@Override

**public** **void** run() {

System.out.println("Runnable");

}

}) {

@Override

**public** **void** run() {

System.out.println("Thread");

}

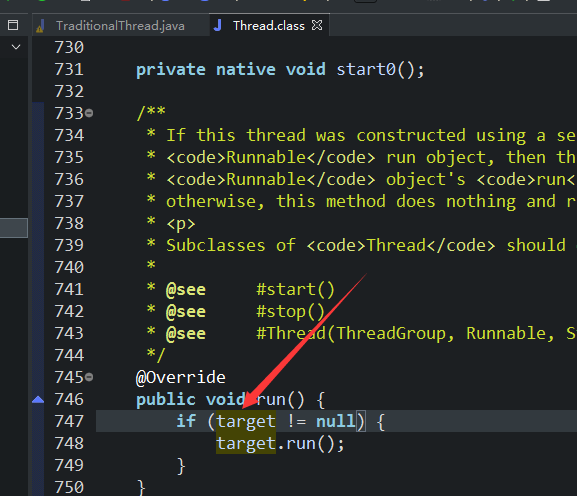
}.start();

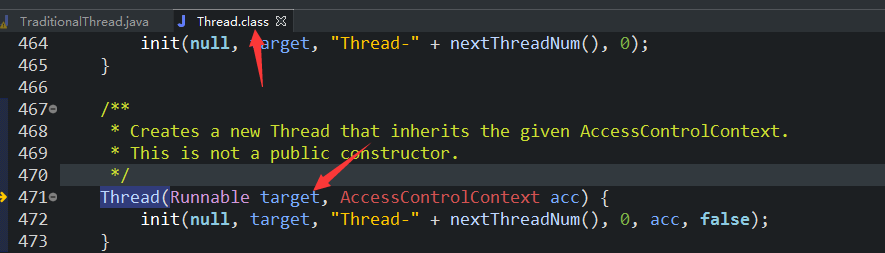
//输出Thread

}

}

原理，Thread源码说明如果target为空，就直接使用线程重写的run方法





## 1.2传统定时器

**package** cn.zz.test1;

**import** java.util.Date;

**import** java.util.Timer;

**import** java.util.TimerTask;

**public** **class** TraditionalTimer {

**private** **static** **int** count = 0;

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

/\*

\* new Timer().schedule(new TimerTask() {

\*

\* @Override public void run() { System.out.println("boom");

\*

\* } }, 10000,3000);

\*/

/\*

\* new Timer().schedule(new TimerTask() {

\*

\* @Override public void run() { System.out.println("1 boom"); new

\* Timer().schedule(new TimerTask() {

\*

\* @Override public void run() { System.out.println("2 boom");

\*

\*

\* } }, 2000);

\*

\* } }, 2000);

\*/

**class** MyTimerTask **extends** **TimerTask** {

@Override

**public** **void** run() {

count = (count+1)%2;

System.out.println("boom");

**new** Timer().schedule(**new** MyTimerTask(), 2000+2000\*count);

}

}

**new** Timer().schedule(**new** MyTimerTask(), 2000);

**while** (**true**) {

System.out.println(**new** Date().~~getSeconds~~());

**try** {

Thread.sleep(1000);

} **catch** (InterruptedException e) {

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

e.printStackTrace();

}

}

}

}

## 1.3线程的同步互斥与通信

### 互斥：同一资源，同一对象，只能独自访问

#### 代码块互斥

**package** cn.zz.test1;

**public** **class** **TraditionalThreadSyn** {

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

**new** TraditionalThreadSyn().init();

}

**public** **void** **init**() {

**Outer** **outer** = **new** Outer();

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

***@Override***

**public** **void** **run**() {

**while** (**true**) {

**/\*try {**

**Thread.sleep(10);**

**} catch (InterruptedException e) {**

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

**e.printStackTrace();**

**}\*/**

outer.log("nike");

}

}

}).start();

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

***@Override***

**public** **void** **run**() {

**while** (**true**) {

**/\*try {**

**Thread.sleep(10);**

**} catch (InterruptedException e) {**

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

**e.printStackTrace();**

**}\*/**

outer.log("lining");

}

}

}).start();

}

**//互斥**

**class** **Outer** {

**String** syn = "";

**public** **void** **log**(**String** log) {

**//同一个对象**

**synchronized** (syn) {

**for** (**int** **i** = 0; i < log.length(); i++) {

**System**.***out***.print(log.charAt(i));

}

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

}

**//用this也可以**

**synchronized** (**this**) {

**for** (**int** **i** = 0; i < log.length(); i++) {

**System**.***out***.print(log.charAt(i));

}

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

}

}

}

}

#### 方法互斥

**package** cn.zz.test1;

**public** **class** **TraditionalThreadSyn2** {

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

**new** TraditionalThreadSyn2().init();

}

**public** **void** **init**() {

**Outer** **outer** = **new** Outer();

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

***@Override***

**public** **void** **run**() {

**while** (**true**) {

**/\***

**\* try { Thread.sleep(10); } catch (InterruptedException e)**

**\* { // TODO Auto-generated catch block e.printStackTrace();**

**\* }**

**\*/**

outer.log("nike");

}

}

}).start();

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

***@Override***

**public** **void** **run**() {

**while** (**true**) {

**/\***

**\* try { Thread.sleep(10); } catch (InterruptedException e)**

**\* { // TODO Auto-generated catch block e.printStackTrace();**

**\* }**

**\*/**

outer.log("lining");

}

}

}).start();

}

**// 互斥 同一个对象**

**class** **Outer** {

**//直接在方法上加互斥也可以**

**public** **synchronized** **void** **log**(**String** log) {

**for** (**int** **i** = 0; i < log.length(); i++) {

**System**.***out***.print(log.charAt(i));

}

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

}

}

}

#### 静态方法互斥

**package** cn.zz.test1;

**public** **class** **TraditionalThreadSyn3class** {

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

**new** TraditionalThreadSyn3class().init();

}

**public** **void** **init**() {

**Outer** **outer** = **new** Outer();

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

***@Override***

**public** **void** **run**() {

**while** (**true**) {

**/\***

**\* try { Thread.sleep(10); } catch (InterruptedException e)**

**\* { // TODO Auto-generated catch block e.printStackTrace();**

**\* }**

**\*/**

outer.log("nike");

}

}

}).start();

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

***@Override***

**public** **void** **run**() {

**while** (**true**) {

**/\***

**\* try { Thread.sleep(10); } catch (InterruptedException e)**

**\* { // TODO Auto-generated catch block e.printStackTrace();**

**\* }**

**\*/**

outer.*log2*("lining");

}

}

}).start();

}

**// 互斥 同一个对象 类**

**static** **class** **Outer** {

**//直接在方法上加互斥也可以**

**public** **void** **log**(**String** log) {

**synchronized** (**Outer**.**class**) {

**for** (**int** **i** = 0; i < log.length(); i++) {

**System**.***out***.print(log.charAt(i));

}

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

}

}

**public** **static** **synchronized** **void** **log2**(**String** log) {

**for** (**int** **i** = 0; i < log.length(); i++) {

**System**.***out***.print(log.charAt(i));

}

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

}

}

}

### 总结：

要用到共同数据（包括同步锁）的若干的方法应该归在同一个类身上，这种设计正好体现了类聚和程序的健壮性。

两个线程执行的代码要实现同步互斥的效果，他们必须用同一个lock对象。锁是上在代表要操作的资源的类的内部方法中，而不是线程代码中。

例子：实现一个主线程执行10次，子线程执行5次，主次切换，循环50次；

思路：先写一个类，分别执行主线程，子线程，做互斥

**package** cn.zz.test1;

**public** **class** **TraditionalThreadCommicute** {

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

**Outer** **outer** = **new** Outer();

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

***@Override***

**public** **void** **run**() {

**for** (**int** **i** = 0; i < 50; i++) {

outer.sub();

}

}

}).start();

**for** (**int** **i** = 0; i < 50; i++) {

outer.main();

}

}

}

**class** **Outer** {

**private** **Boolean** runMain = **true**;

**public** **synchronized** **void** **main**() {

**while** (!runMain) {

**try** {

**this**.wait();

} **catch** (**InterruptedException** **e**) {

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

e.printStackTrace();

}

}

**for** (**int** **i** = 0; i < 10; i++) {

**System**.***out***.println("main " + i);

}

runMain = **false**;

**this**.notify();

}

**public** **synchronized** **void** **sub**() {

**while** (runMain) {

**try** {

**this**.wait();

} **catch** (**InterruptedException** **e**) {

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

e.printStackTrace();

}

}

**for** (**int** **i** = 0; i < 5; i++) {

**System**.***out***.println("sub " + i);

}

runMain = **true**;

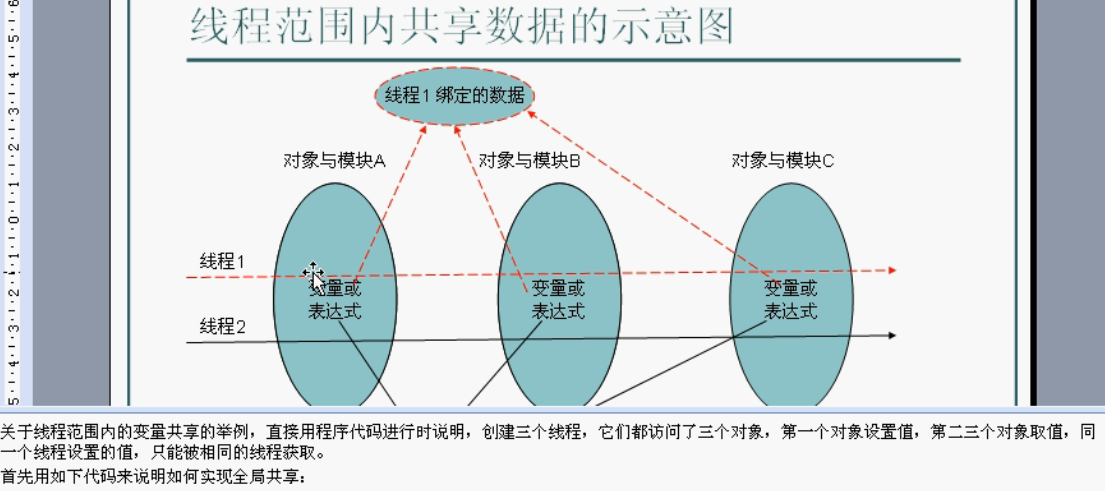
**this**.notify();

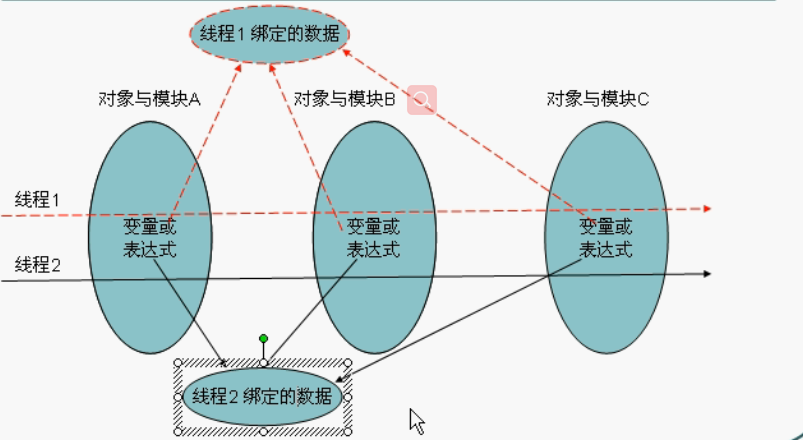
}

}

### 同步

## 2.1ThreadLocal实现线程范围的共享数据





一个ThreadLocal只能放一个变量，n个变量要定义n个ThreadLocal

ThreadLocal是一个map有多少个线程就有多少个ThreadLocal,线程结束时，底层会自动回收该线程的ThreadLocal

**package** cn.zz.test1;

**import** java.util.Random;

**public** **class** **ThreadLocalTest** {

**private** **static** **ThreadLocal**<Integer> *x* = **new** ThreadLocal<Integer>();

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

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

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

***@Override***

**public** **void** **run**() {

**int** **data** = **new** Random().nextInt();

**System**.***out***.println(**Thread**.*currentThread*().getName()+

": "+data);

*x*.set(data);

**new** A().get();

**new** B().get();

}

}).start();

}

}

**static** **class** **A**{

**public** **void** **get**(){

**int** **data** = *x*.get();

**System**.***out***.println("A "+**Thread**.*currentThread*().getName()+

": "+data);

}

}

**static** **class** **B**{

**public** **void** **get**(){

**int** **data** = *x*.get();

**System**.***out***.println("B "+**Thread**.*currentThread*().getName()+

": "+data);

}

}

}