# 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");

}

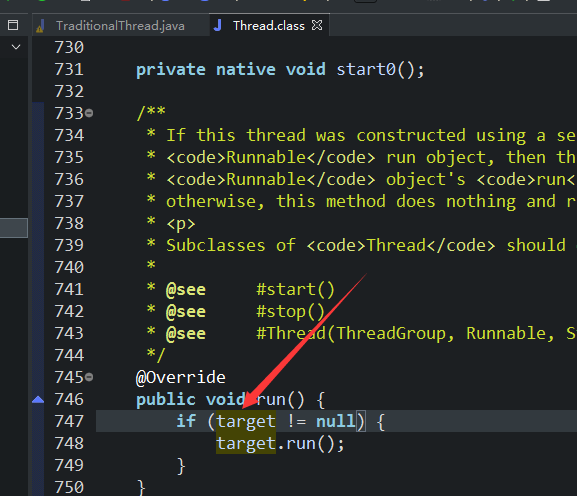
}.start();

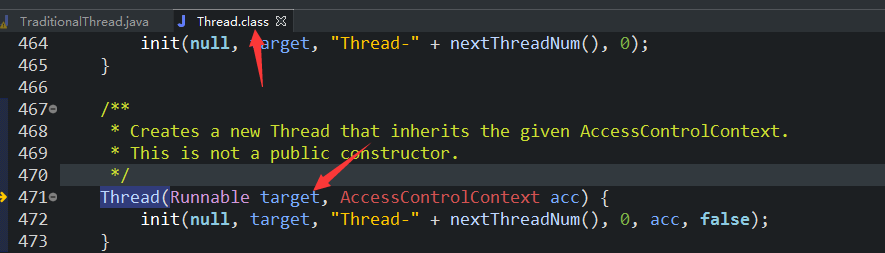
//输出Thread

}

}

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