# JAVA编程进阶上机报告

****

**学 院 智能与计算学部**

**专 业 软件工程**

**班 级 软件工程2班**

**学 号 3619058914**

**姓 名 周陈铮**

1. **实验要求**

第四次实验是使用多线程编程技术，编写矩阵乘法。

具体要求如下：

* 编写矩阵随机生成类 MatrixGenerator 类，随机生成任意大小的矩阵，矩阵单元使用 double 存储。
* 使用串行方式实现矩阵乘法。
* 使用多线程方式实现矩阵乘法。
* 比较串行和并行两种方式使用的时间，利用第三次使用中使用过的 jvm状态查看命令，分析产生时间差异的原因是什么。

**二、类的设计**

矩阵类（Matrix）与产生矩阵类（MatrixGenerator）,这两个类都在matrix包中

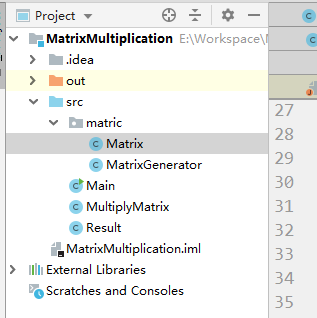
MulptiplyMatrix实现不同方式的矩阵乘法

Result类用于返回结果

Main类完成类的组装

**三、源代码**

**类的组织**



**Matrix类**

**package** matric;  
  
**public class** Matrix {  
 **private int row**;  
 **private int col**;  
 **private double**[][] **myMatrix**;  
  
 **protected** Matrix(**int** row, **int** col){  
 **this**.**row** = row;  
 **this**.**col** = col;  
 **myMatrix** = **new double**[row][col];  
 }  
  
 **public int** getRow() {  
 **return row**;  
 }  
  
 **public void** setRow(**int** row) {  
 **this**.**row** = row;  
 }  
  
 **public int** getCol() {  
 **return col**;  
 }  
  
 **public void** setCol(**int** col) {  
 **this**.**col** = col;  
 }  
  
 **public double** getMatrix(**int** i, **int** j) {  
 **return myMatrix**[i][j];  
 }  
  
 **public void** setMatrix(**int** i, **int** j, **int** value) {  
 **this**.**myMatrix**[i][j] = value;  
 }  
}

**MatrixGenerator类**

**package** matric;  
  
**public class** MatrixGenerator {  
 **public static** Matrix initalMatrix(**int** row, **int** col){  
 Matrix matrix1 = **new** Matrix(row, col);  
 **for**(**int** i = 0; i < row; i++){  
 **for**(**int** j = 0; j < col; j++){  
 **int** value = (**int**)(Math.*random*()\*10);  
 matrix1.setMatrix(i, j, value);  
 }  
 }  
 **return** matrix1;  
 }  
}

**MultiplyMatrix类**

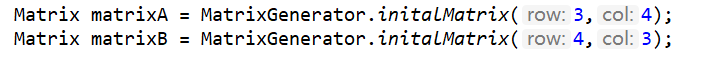
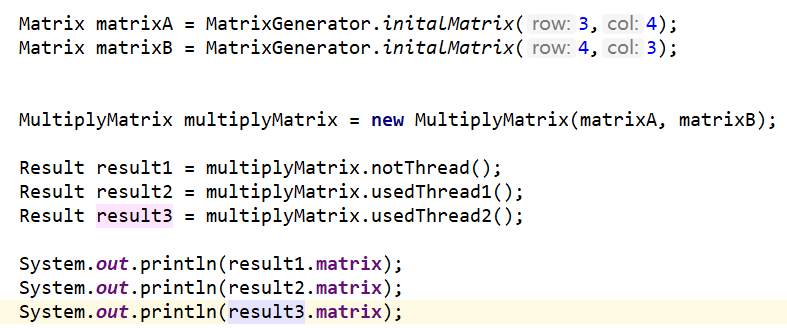
**import** matric.Matrix;  
**import** matric.MatrixGenerator;  
  
**import** java.util.ArrayList;  
**import** java.util.List;  
  
**public class** MultiplyMatrix {  
  
 **private** Matrix **matrixA**;  
 **private** Matrix **matrixB**;  
 **private int row** ;  
 **private int col** ;  
 **private** Matrix **res**;  
 **public** MultiplyMatrix(Matrix matrixA, Matrix matrixB) {  
 **this**.**matrixA** = matrixA;  
 **this**.**matrixB** = matrixB;  
 **this**.**row** = matrixA.getRow();  
 **this**.**col** = matrixB.getCol();  
 **this**.**res** = MatrixGenerator.*initalMatrix*(**row**,**col**);  
 }  
  
 *//串行* **public** Result notThread(){  
 **long** startTime = System.*nanoTime*();  
  
 **for**(**int** i = 0; i < **row**; i++){  
 countEachRow(i);  
 }  
  
 **long** endTime = System.*nanoTime*();  
 **return new** Result(endTime - startTime, **res**);  
 }  
  
 *//并行一* **public** Result usedThread1(){  
  
 Thread thread1 = **new** Thread(() -> {  
 **for**(**int** i = 0; i < **row**; i = i + 2){  
 countEachRow(i);  
 }  
  
 });  
  
 Thread thread2 = **new** Thread(() -> {  
 **for**(**int** i = 1; i < **row**; i = i + 2){  
 countEachRow(i);  
 }  
 });  
  
 **long** startTime = System.*nanoTime*();  
 **try** {  
 thread1.start();  
 thread2.start();  
 thread1.join();  
 thread2.join();  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 **long** endTime = System.*nanoTime*();  
 **return new** Result(endTime - startTime, **res**);  
 }  
  
 *//并行二* **public** Result usedThread2(){  
 List<Thread> list = **new** ArrayList<>();  
 **for**(**int** i = 0; i < **row**; i++){  
 **int** tempRow = i;  
 list.add(**new** Thread(() -> countEachRow(tempRow)));  
 }  
 **long** startTime = System.*nanoTime*();  
 **for**(Thread thread: list){  
 thread.start();  
 }  
 **for**(Thread thread: list){  
 **try** {  
 thread.join();  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 **long** endTime = System.*nanoTime*();  
  
 **return new** Result(endTime - startTime, **res**);  
 }  
  
 **private void** countEachRow(**int** i){  
 **for**(**int** j = 0; j < **col**; j++){  
 **int** temp = 0;  
 **for**(**int** k = 0; k < **matrixA**.getCol(); k++){  
 temp += **matrixA**.getMatrix(i, k) \* **matrixB**.getMatrix(k, j);  
 }  
 **res**.setMatrix(i, j, temp);  
 }  
 }  
  
  
}

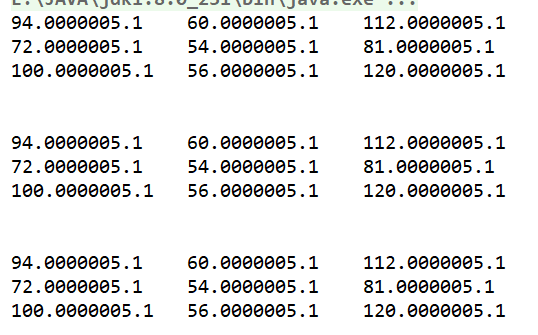
**Result类**

**import** matric.Matrix;  
  
**public class** Result {  
 **public long time**;  
 **public** Matrix **matrix**;  
  
 **public** Result(**long** time, Matrix matrix) {  
 **this**.**time** = time;  
 **this**.**matrix** = matrix;  
 }  
  
  
}

**四、实验结果**

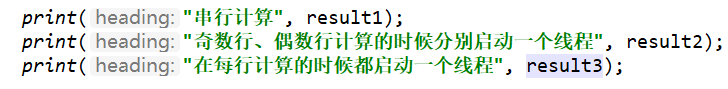
**先从较小矩阵开始**

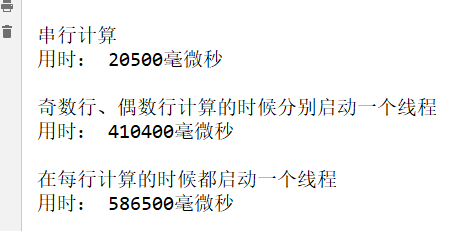




**说明矩阵乘法的结果都是对的**

**此时，分别用时**





**再将矩阵变大，此时就不再打印矩阵**

