# JAVA实例代码

信1703 姚鋆 20173417

package AAA;

import java.awt.Container;

import java.awt.GridLayout;

import java.awt.event.\*;

import javax.swing.\*;

public class mainWindow extends JFrame implements ActionListener{

/\*\* \* \*/

private static final long serialVersionUID = 1L;

JFrame bankerAlgoritJFrame;//主界面

JLabel[] l; //输入资源种类数，输入启动的进程数，资源请求，标签

JTextField[] t; //对应于标签的文本框

JButton[] b; //对应去标签的按钮

JPanel[] p;

int m=0;//添加的进程数

int n=0;//添加的资源种类数

int[] available;

int[][] max;

int[][] allocation;

int[][] need;

int[] request;

String requestP;

String[] processName;

String[] processName\_Safety;//保存一个安全推进序列

String[] resourceName;

JPanel[] l1;

JTextField[] t1;

public mainWindow(String name){

super(name);

setSize(320, 250);

setLocation(500, 100);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

Container container = getContentPane();

container.setLayout(null);

l = new JLabel[3];

t = new JTextField[2];

b = new JButton[3];

p = new JPanel[3];

p[0] = new JPanel();

p[1] = new JPanel();

p[2] = new JPanel();

p[0].setBounds(5, 20, 300, 30);

p[1].setBounds(5, 60, 300, 30);

p[2].setBounds(5, 100, 300, 30);

l[0] = new JLabel("输入资源的种类数：");

l[1] = new JLabel("输入启动的进程数：");

l[2] = new JLabel("资源请求：");

t[0] = new JTextField(3);

t[1] = new JTextField(3);

b[0] = new JButton("添加资源");

b[0].addActionListener(this);

b[1] = new JButton("启动进程");

b[1].addActionListener(this);

b[2] = new JButton("请求资源");

b[2].addActionListener(this);

p[0].add(l[0]);

p[0].add(t[0]);

p[0].add(b[0]);

p[1].add(l[1]);

p[1].add(t[1]);

p[1].add(b[1]);

p[2].add(l[2]);

p[2].add(b[2]);

container.add(p[0]);

container.add(p[1]);

container.add(p[2]);

setVisible(true);

}

/\* \* 定义添加资源窗口类 \* \*/

class AddResourceWindow extends JFrame implements ActionListener{

private static final long serialVersionUID = 1L;

JLabel ResourceNameL = new JLabel("资源名 ");

JLabel AvailableL = new JLabel(" available");

JButton addResourceOK = new JButton("确定");

JLabel[] resourceNameJLabel = new JLabel[n];

JTextField[] AvailableTextField = new JTextField[n];

JPanel[] addResourceJPanels = new JPanel[n+2];

public AddResourceWindow(){

super("添加资源");

setSize(320, 400);

setLocation(500, 150);

Container container = getContentPane();

container.setLayout(null);

addResourceJPanels[n+1] = new JPanel();

addResourceJPanels[n+1].setBounds(5, 20, 300, 30);

addResourceJPanels[n+1].add(ResourceNameL);

addResourceJPanels[n+1].add(AvailableL);

container.add(addResourceJPanels[n+1]);

/\* \* 动态的添加组件 \* \*/

int y = 60;

char a = 'A';

for(int i=0; i<n; i++, y=y+40){

addResourceJPanels[i] = new JPanel();

resourceNameJLabel[i] = new JLabel(String.valueOf(a)+":");

resourceName[i] = String.valueOf(a);

a = (char)(a+1);

AvailableTextField[i] = new JTextField(4);

AvailableTextField[i].setHorizontalAlignment(JTextField.RIGHT);

addResourceJPanels[i].setBounds(5, y, 300, 30);

addResourceJPanels[i].add(resourceNameJLabel[i]);

addResourceJPanels[i].add(AvailableTextField[i]);

container.add(addResourceJPanels[i]);

}

addResourceJPanels[n] = new JPanel();

addResourceJPanels[n].setBounds(5, y, 300, 30);

addResourceJPanels[n].add(addResourceOK);

addResourceOK.addActionListener(this);

container.add(addResourceJPanels[n]);

setVisible(true);

}

/\* \* 将输入的资源名和数量存入available数组,若不填资源名默认为A，B，C。。。。。 \* \*/

void getResource(){

for(int i=0; i<n; i++){

try {

available[i] = Integer.parseInt(AvailableTextField[i].getText());

} catch (Exception e) {

// TODO: handle exception

JOptionPane.showMessageDialog(null,"available为正整数");

return;

}

}

// for(int i=0; i<n; i++)

// System.out.println(resourceName[i]);

this.dispose();

}

@Override

public void actionPerformed(ActionEvent e) {

// TODO Auto-generated method stub

Object ob = e.getSource();

if(ob == addResourceOK){

getResource();

}

}

}

/\* \* 定义启动进程的窗口类，用来输入进程名，max数组和allocation数组 \* \*/

class StartProcess extends JFrame implements ActionListener{

private static final long serialVersionUID = 1L;

JLabel resourceNameJLabel = new JLabel(" 进程名");

JLabel maxJLabel = new JLabel(" max ");

JLabel allocationJLabel = new JLabel("allocation ");

JButton startProcessOK = new JButton("确定");

JPanel[] startProcessJPanel = new JPanel[m+2];

JLabel[] processNameJLabel = new JLabel[m];

JTextField[][] maxJTextField = new JTextField[m][n];

JTextField[][] allocationJTextField = new JTextField[m][n];

public StartProcess(){

super("启动进程");

setSize(320, 400);

setLocation(500, 150);

Container container = getContentPane();

container.setLayout(null);

/\* \*添加表格标题 \* \*/

startProcessJPanel[m+1] = new JPanel();

startProcessJPanel[m+1].setLayout(new GridLayout(1,3));

startProcessJPanel[m+1].setBounds(5, 20, 300, 30);

startProcessJPanel[m+1].add(resourceNameJLabel);

startProcessJPanel[m+1].add(maxJLabel);

startProcessJPanel[m+1].add(allocationJLabel);

container.add(startProcessJPanel[m+1]);

/\* \* 添加表格（textfield组）进程名默认为p1，p2，p3。。。。 \* \*/

int y=60;

for(int i=0; i<m; i++, y=y+40){

startProcessJPanel[i] = new JPanel();

processNameJLabel[i] = new JLabel("p"+i+":");

startProcessJPanel[i].setBounds(5, y, 300, 30);

processName[i] = "p"+i;

startProcessJPanel[i].add(processNameJLabel[i]);

for(int j=0; j<n; j++){

maxJTextField[i][j] = new JTextField(2);

maxJTextField[i][j].setHorizontalAlignment(JTextField.RIGHT);

startProcessJPanel[i].add(maxJTextField[i][j]);

}

startProcessJPanel[i].add(new JLabel("\*"));

for(int j=0; j<n; j++){

allocationJTextField[i][j] = new JTextField(2);

allocationJTextField[i][j].setHorizontalAlignment(JTextField.RIGHT);

startProcessJPanel[i].add(allocationJTextField[i][j]);

}

container.add(startProcessJPanel[i]);

}

/\* \* 添加确定按钮 \* \*/

startProcessJPanel[m] = new JPanel();

startProcessJPanel[m].setBounds(5, y, 300, 30);

startProcessJPanel[m].add(startProcessOK);

startProcessOK.addActionListener(this);

container.add(startProcessJPanel[m]);

setVisible(true);

}

/\* \* 从textfield中获取进程信息 \* \*/

void getProcessMassage(){

try{

for(int i=0; i<m; i++){

for(int j=0; j<n; j++){

max[i][j] = Integer.parseInt(maxJTextField[i][j].getText());

allocation[i][j] = Integer.parseInt(allocationJTextField[i][j].getText());

need[i][j] = max[i][j] - allocation[i][j];

}

}

}catch(Exception e){

JOptionPane.showMessageDialog(null, "输入信息必须为正整数");

return;

}

if(isSafety()==false){

JOptionPane.showMessageDialog(null, "输入的进程信息使进程处在不安全状态!");

return;

}

// for(int i=0; i<m; i++){

// for(int j=0; j<n; j++){

// System.out.print(max[i][j]);

// }

// System.out.println();

// }

this.dispose();

}

@Override

public void actionPerformed(ActionEvent e) {

// TODO Auto-generated method stub

Object ob = e.getSource();

if(ob == startProcessOK){

getProcessMassage();

}

}

}

/\* \* 请求资源窗口类 \* \*/

class RequestResourceWindow extends JFrame implements ActionListener{

/\*\* \* \*/

private static final long serialVersionUID = 1L;

JLabel rProcessNameL = new JLabel("进程名：");

JButton reqResOK = new JButton("确定");

JButton reqResClose = new JButton("关闭");

JTextField rProcessNameT = new JTextField(4);

JLabel[] resNameL = new JLabel[n];

JTextField[] resNameT = new JTextField[n];

JPanel[] resJPanel = new JPanel[n+2];

Container container = getContentPane();

public RequestResourceWindow(){

super("请求资源");

setSize(320, 400);

setLocation(500, 150);

Container container = getContentPane();

container.setLayout(null);

resJPanel[n] = new JPanel();

resJPanel[n].setBounds(5, 20, 300, 30);

resJPanel[n].add(rProcessNameL);

resJPanel[n].add(rProcessNameT);

container.add(resJPanel[n]);

int y = 60;

char a = 'A';

for(int i=0; i<n; i++, y=y+40){

resJPanel[i] = new JPanel();

resNameL[i] = new JLabel(String.valueOf(a)+":");

resourceName[i] = String.valueOf(a);

a = (char)(a+1);

resNameT[i] = new JTextField(2);

resNameT[i].setHorizontalAlignment(JTextField.RIGHT);

resJPanel[i].setBounds(5, y, 300, 30);

resJPanel[i].add(resNameL[i]);

resJPanel[i].add(resNameT[i]);

container.add(resJPanel[i]);

}

resJPanel[n+1] = new JPanel();

resJPanel[n+1].setBounds(5, y, 300, 30);

resJPanel[n+1].add(reqResOK);

resJPanel[n+1].add(reqResClose);

reqResOK.addActionListener(this);

reqResClose.addActionListener(this);

container.add(resJPanel[n+1]);

setVisible(true);

}

/\* \* 获取请求资源的信息,并计算请求资源后是否处于安全状态 \* \*/

private void getRequestRes(){

try{

for(int i=0; i<n; i++){

request[i] = Integer.parseInt(resNameT[i].getText());

}

}catch(Exception e){

JOptionPane.showMessageDialog(null, "输入请求资源数必须为正整数");

return;

}

requestP = rProcessNameT.getText();

/\* \* 判断输入的请求是否合法，完成核心算法的第一第二步 \* \*/

if(P\_id(requestP)==-1){

JOptionPane.showMessageDialog(null, "输入的进程不存在");

return;

}else if(R\_N(P\_id(requestP))==false){

JOptionPane.showMessageDialog(null, "进程所需要的资源数量已超过其宣布的最大量");

return;

}else if(R\_A()==false){

JOptionPane.showMessageDialog(null, "没有足够的资源分配给该进程");

return;

}

String[] s = new String[2];

s[0] = null;

s[1] = "";

if(bankerAlgorithm()==true){

s[0] = "此时系统出于安全状态！一个安全推进序列为：\n";

for(int i=0; i<m; i++){

s[1] = s[1] + processName\_Safety[i] + " ";

}

JOptionPane.showMessageDialog(null, s[0]+s[1]);

return;

}else{

JOptionPane.showMessageDialog(null, "此时系统处于不安全状态！");

return;

}

//

// System.out.println(P\_id(requestP.trim()));

// System.out.println(requestP);

//

// for(int i=0; i<n; i++){

// System.out.print(request[i]+" ");

// }

// System.out.println(requestP);

}

@Override

public void actionPerformed(ActionEvent arg0) {

// TODO Auto-generated method stub

Object ob = arg0.getSource();

if(ob == reqResOK){

getRequestRes();

}else if(ob == reqResClose){

this.dispose();

}

}

}

/\* \* 添加资源按钮功能，初始化资源种类数，弹出对话框初始化资源available \* \*/

void addResource(){

try{

n = Integer.parseInt(t[0].getText());

}catch(Exception e){

JOptionPane.showMessageDialog(null, "资源数为正整数");

return;

}

//System.out.println(n);

available = new int[n];

for(int i=0; i<n; i++){

available[i] = 0;

}

resourceName = new String[n];

new AddResourceWindow();

// for(int i=0; i<n; i++){

// System.out.print(resourceName[i]);

// }

// System.out.println();

}

/\* \* 判断资源是否添加 \* \*/

private boolean haveResouce(){

for(int i=0;i<n;i++){

if(available[i]!=0){

return true;

}

}

return false;

}

/\* \* 启动进程，添加进程max数组信息，给进程分配allocation \* \*/

void startProcess(){

if(haveResouce()==false || n==0){

JOptionPane.showMessageDialog(null, "请先添加资源:资源数全为零或没有输入资源数");

}else{

try{

m = Integer.parseInt(t[1].getText());

}catch(Exception e){

JOptionPane.showMessageDialog(null, "进程数为正整数");

return;

}

// System.out.print(m);

processName = new String[m];

max = new int[m][n];

allocation = new int[m][n];

need = new int[m][n];

// for(int i=0; i<m; i++)

// for(int j=0; j<n; j++)

// System.out.print(max[i][j]);

new StartProcess();//启动窗口

}

}

/\* \* 打开请求资源对话框，并且计算出是否出于安全状态 \* \*/

void requestResource(){

request = new int[n];

new RequestResourceWindow();

}

/\* \* 核心算法部分 \* \*/

//主函数

public boolean bankerAlgorithm(){

//通过了资源申请的验证，尝试分配资源

int i = P\_id(requestP);

for(int j=0; j<n; j++){

available[j] = available[j] - request[j];

allocation[i][j] += request[j];

need[i][j] -= request[j];

}

if(isSafety()==false){

for(int j=0; j<n; j++){

available[j] = available[j] + request[j];

allocation[i][j] -=request[j];

need[i][j] -= request[j];

}

return false;

}else

return true;

}

/\* \* 根据进程名返回进程ID \* \*/

int P\_id(String PName){

for(int i=0; i<m; i++){

if(processName[i].compareToIgnoreCase(PName) == 0) return i;

}

return -1;

}

/\* \* 判断请求资源数request与所需资源数need之间的关系,如果所有request[j]<=need[i][j]返回true \* \*/

boolean R\_N(int i){

for(int j=0; j<n; j++){

if(request[j]>need[i][j]) return false;

}

return true;

}

/\* \* 判断需求资源数request与可用资源数available之间的关系，如果所有request[j]<=available[j]返回ture \* \*/

boolean R\_A(){

for(int j=0; j<n; j++){

if(request[j]>available[j])return false;

}

return true;

}

/\* \* 判断need[i][j]与work[j]的大小，小于返回true \* \*/

boolean N\_W(int i, int[] work){

for(int j=0; j<n; j++){

if(need[i][j]>work[j]) return false;

}

return true;

}

/\* \* 找到满足need[i][j]<=work[j]且finish[i]==false的进程返回进程id，未找到返回-1 \* \*/

int isNextP(int i, boolean[] finish, int[] work){

for(int j=i; j<m; j++){

if(finish[j]==false && N\_W(j, work)==true)

return j;

}

for(int k=0; k<i; k++){

if(finish[k]==false && N\_W(k, work)==true)

return k;

}

return -1;

}

/\* \* 判断根据finish的值判断是否出于安全状态，全为true则处于安全状态 \* \*/

boolean allTrue(boolean[] finish){

for(int i=0; i<m; i++){

if(finish[i]==false)return false;

}

return true;

}

/\* \* 判断是否安全 \* \*/

boolean isSafety(){

int[] work = new int[n];

boolean[] finish = new boolean[m];

processName\_Safety = new String[m];

for(int j=0; j<n; j++){

work[j] = available[j];

}

for(int j=0; j<m; j++){

finish[j] = false;

}

int i=0;

for(int k=0; k<m; k++){

i = isNextP(i,finish,work);

if(i>=0){

for(int j=0; j<n; j++){

work[j] += allocation[i][j];

}

finish[i] = true;

processName\_Safety[k] = processName[i];

// System.out.println(i);

}else

break;

}

if(allTrue(finish)==true)

return true;

else

return false;

}

/\*\* \* @param args \*/

public static void main(String[] args) {

// TODO Auto-generated method stub

new mainWindow("银行家算法实现");

}

@Override

public void actionPerformed(ActionEvent e) {

// TODO Auto-generated method stub

Object ob = e.getSource();

if(ob == b[0]){

addResource();

}else if(ob == b[1]){

startProcess();

}else if(ob == b[2]){

requestResource();

}

}