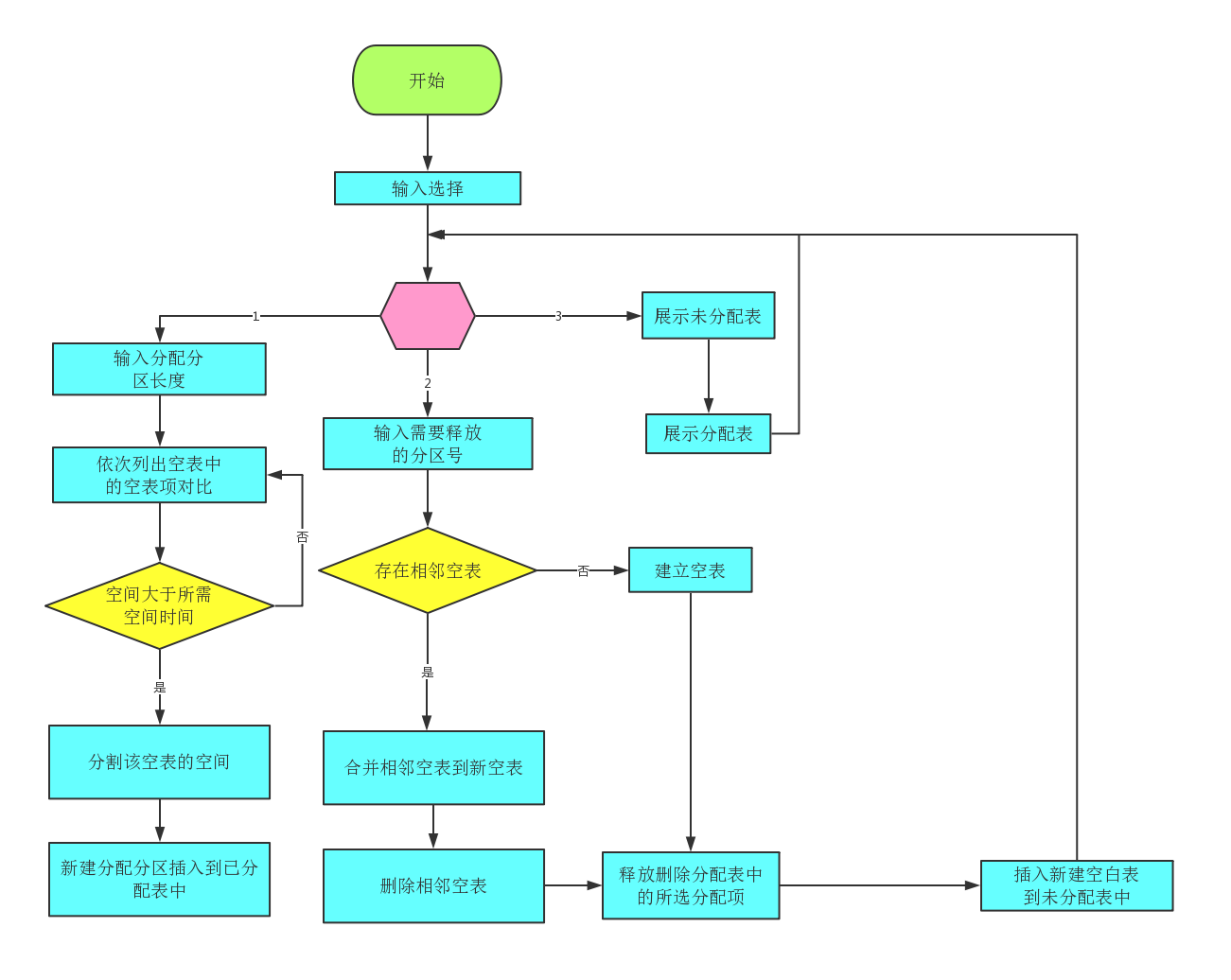
动态分区与回收实验报告

一、首次适应算法（FF）：

一、流程图



二、源程序

import java.util.Scanner;  
  
/\*\*  
 \* @author tanHongxia  
 \*/  
public class FirstFit {  
 public static void main(String[] args) {  
 Scanner input = new Scanner(System.in);  
 Memory memory = new Memory();  
 /\*\*  
 \* 初始化菜单  
 \* \*/  
 while (true){  
 menu();  
 int choose = input.nextInt();  
 switch (choose){  
 case 1:  
 System.out.print("请输入分区长度: ");  
 int size = input.nextInt();  
 memory.firstFit(size);  
 break;  
 case 2:  
 System.out.print("请输入分区ID: ");  
 int recoverId = input.nextInt();  
 memory.collection(recoverId);  
 break;  
 case 3:  
 System.exit(0);  
 break;  
 default:  
 System.out.println("请重新选择操作！！！");  
 }  
 }  
 }  
  
 /\*\*  
 \* 菜单  
 \* \*/  
 public static void menu(){  
 System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*首次适应算法\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 System.out.println(" 1---分配");  
 System.out.println(" 2---回收");  
 System.out.println(" 3---退出");  
 System.out.print("请选择操作:");  
 }  
}

*import* com.sun.xml.internal.bind.v2.model.core.ID;  
  
*import* java.awt.\*;  
*import* java.sql.SQLOutput;  
*import* java.util.LinkedList;  
*import* java.util.Scanner;  
  
*/\*\*  
 \* 内存  
 \* \*/  
public class* Memory{  
 */\*\*  
 \* 内存大小  
 \* \*/  
 public int* size;  
 */\*\*  
 \* 最小剩余分区大小  
 \* \*/  
 public static final int* MIN\_SIZE = 5;  
 */\*\*  
 \* 内存分区  
 \* \*/  
 public* LinkedList<Partition> partition;  
 */\*\*  
 \* 上次分配的空闲分区位置  
 \* \*/  
 public int* pointer;  
  
 */\*\*  
 \* 分区类  
 \* \*/  
 class* Partition{  
 */\*\*  
 \* 分区起始地址  
 \* \*/  
 public int* head;  
 */\*\*  
 \* 分区长度  
 \* \*/  
 public int* size;  
 */\*\*  
 \* 分区状态  
 \* \*/  
 public boolean* state;  
  
 *public* Partition(*int* head, *int* size) {  
 *this*.head = head;  
 *this*.size = size;  
 *this*.state = *true*;  
 }  
 }  
  
 */\*\*  
 \* 默认内存大小为100KB  
 \* \*/  
 public* Memory(){  
 *this*.size = 100;  
 *this*.pointer = 0;  
 *this*.partition = *new* LinkedList<>();  
 partition.add(*new* Partition(0, size));  
 }  
 */\*\*  
 \* 首次适应算法  
 \* @param* ***size*** *指定需要分配的大小  
 \*/  
 public void* firstFit(*int* size){  
 *//遍历分区链表  
 for* (pointer = 0; pointer < partition.size(); pointer++){  
 Partition temp = partition.get(pointer);  
 *//找到可用分区（空闲且大小足够）  
 if* (temp.state && (temp.size > size)){  
 doAllocation(size, pointer, temp);  
 showAllocatedTable();  
 showFreeTable();  
 *return*;  
 }  
 }  
 *//遍历结束后未找到可用分区, 则内存分配失败* System.out.println("无可用内存空间!");  
 }

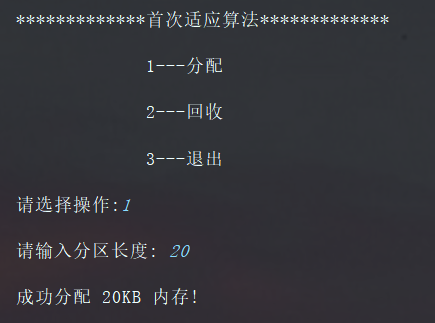
*/\*\*  
 \* 执行分配  
 \* @param* ***size*** *申请大小  
 \* @param* ***location*** *当前可用分区位置  
 \* @param* ***temp*** *可用空闲区  
 \*/  
 public void* doAllocation(*int* size, *int* location, Partition temp) {  
 *//如果分割后分区剩余大小过小（MIN\_SIZE）则将分区全部分配，否则分割为两个分区  
 if* (temp.size - size <= MIN\_SIZE){  
 temp.state = *false*;  
 } *else* {  
 Partition split = *new* Partition(temp.head + size, temp.size - size);  
 partition.add(location + 1, split);  
 temp.size = size;  
 temp.state = *false*;  
 }  
 System.out.println("成功分配 " + size + "KB 内存!");  
 }  
  
 */\*\*  
 \* 内存回收  
 \* @param* ***id*** *指定要回收的分区号  
 \*/  
 public void* collection(*int* id){  
 *if* (id >= partition.size()){  
 System.out.println("无此分区编号!");  
 *return*;  
 }  
 Partition temp = partition.get(id);  
 *int* size = temp.size;  
 *if* (temp.state) {  
 System.out.println("指定分区未被分配, 无需回收");  
 *return*;  
 }  
 *//如果回收分区不是尾分区且后一个分区为空闲, 则与后一个分区合并  
 if* (id < partition.size() - 1 && partition.get(id + 1).state){  
 Partition next = partition.get(id + 1);  
 temp.size += next.size;  
 partition.remove(next);  
 }  
 *//如果回收分区不是首分区且前一个分区为空闲, 则与前一个分区合并  
 if* (id > 0 && partition.get(id - 1).state){  
 Partition previous = partition.get(id - 1);  
 previous.size += temp.size;  
 partition.remove(id);  
 id--;  
 }  
 partition.get(id).state = *true*;  
 System.out.println("内存回收成功!, 本次回收了 " + size + "KB 空间!");  
 showAllocatedTable();  
 showFreeTable();  
 }  
  
 */\*\*  
 \* 展示内存分区状况  
 \*/  
 public void* showAllocatedTable(){  
 System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*已分配分区表\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 System.out.println("分区编号\t分区起始地址\t分区长度\t空闲状态\t");  
 System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 *for* (*int* i = 0; i < partition.size(); i++){  
 Partition temp = partition.get(i);  
 *if*(temp.state == *false*){  
 System.out.println(i + "\t\t" + temp.head + "\t\t" +  
 temp.size + " \t" + temp.state);  
 }  
 }  
 System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 System.out.println();  
 }  
 */\*\*  
 \* 展示空闲分区表  
 \* \*/  
 public void* showFreeTable(){  
 System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*空闲分区表\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 System.out.println("分区编号\t分区起始地址\t分区长度\t空闲状态\t");  
 System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 *for* (*int* i = 0; i < partition.size(); i++){  
 Partition temp = partition.get(i);  
 *if*(temp.state == *true*){  
 System.out.println(i + "\t\t" + temp.head + "\t\t" +  
 temp.size + " \t" + temp.state);  
 }  
 }  
 System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 System.out.println();  
 }  
}

三、程序运行环境、使用工具说明

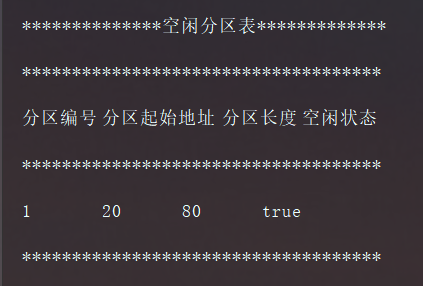
Jdk1.8 ideal

四、程序运行结果

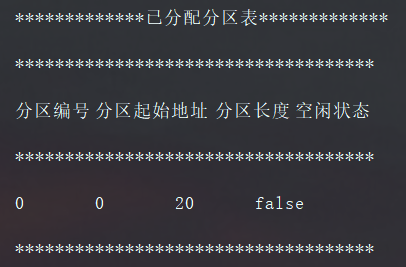
分区分配：



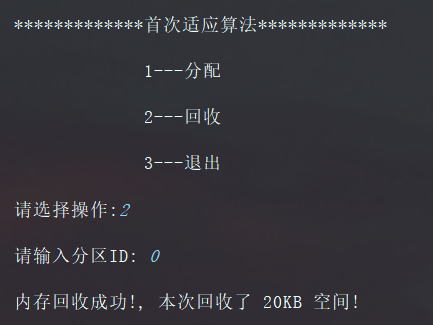
分区分配---空闲分区表：



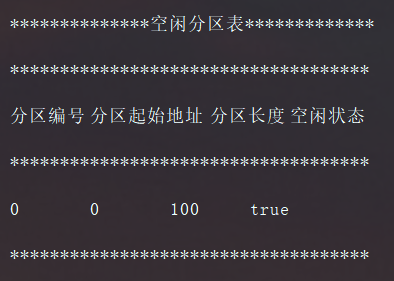
分区分配---已分配分区表：



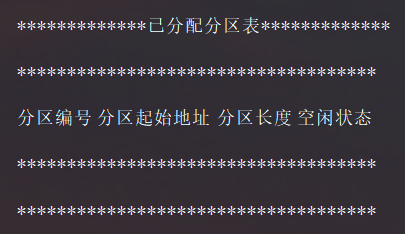
分区回收：



分区回收---空闲分区表：



分区回收---已分配分区表：

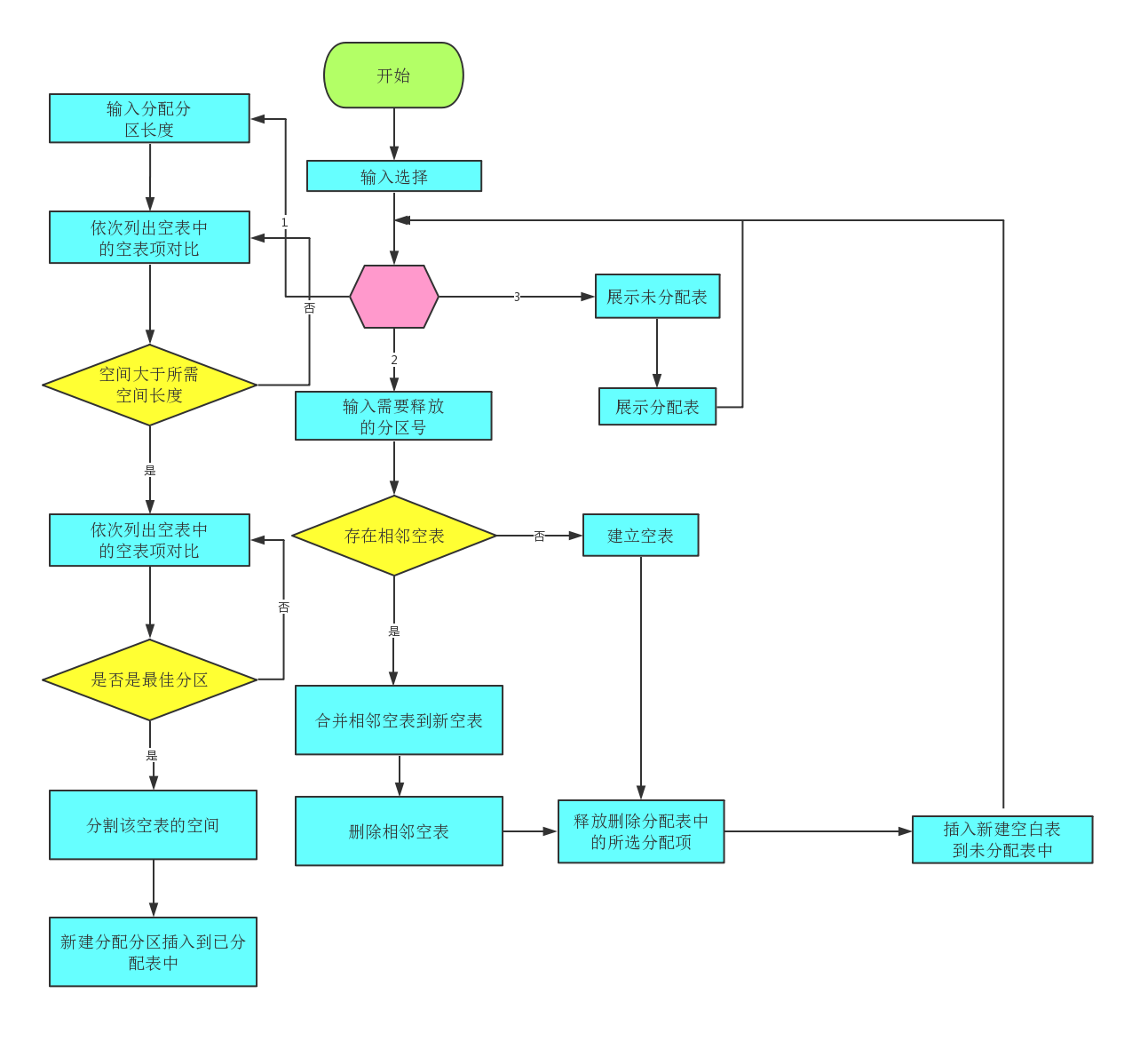


五、结果的简单说明或分析

对于首次适应算法有了更清晰的认识与学习

二、最佳适应算法（BF）：

一、流程图



二、源程序

*import* java.util.Scanner;  
  
*/\*\*  
 \* @author tanHongxia  
 \*/  
public class* BestFit {  
 *public static void* main(String[] args) {  
 Scanner input = *new* Scanner(System.in);  
 Memory memory = *new* Memory();  
 */\*\*  
 \* 初始化菜单  
 \* \*/  
 while* (*true*){  
 *menu*();  
 *int* choose = input.nextInt();  
 *switch* (choose){  
 *case* 1:  
 System.out.print("请输入分区长度: ");  
 *int* size = input.nextInt();  
 memory.bestFit(size);  
 *break*;  
 *case* 2:  
 System.out.print("请输入分区ID: ");  
 *int* recoverId = input.nextInt();  
 memory.collection(recoverId);  
 *break*;  
 *case* 3:  
 System.*exit*(0);  
 *break*;  
 *default*:  
 System.out.println("请重新选择操作！！！");  
 }  
 }  
 }  
  
 */\*\*  
 \* 菜单  
 \* \*/  
 public static void* menu(){  
 System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*最佳适应算法\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 System.out.println(" 1---分配");  
 System.out.println(" 2---回收");  
 System.out.println(" 3---退出");  
 System.out.print("请选择操作:");  
 }  
}

*import* com.sun.xml.internal.bind.v2.model.core.ID;  
  
*import* java.awt.\*;  
*import* java.sql.SQLOutput;  
*import* java.util.LinkedList;  
*import* java.util.Scanner;  
  
*/\*\*  
 \* 内存  
 \* \*/  
public class* Memory{  
 */\*\*  
 \* 内存大小  
 \* \*/  
 public int* size;  
 */\*\*  
 \* 最小剩余分区大小  
 \* \*/  
 public static final int* MIN\_SIZE = 5;  
 */\*\*  
 \* 内存分区  
 \* \*/  
 public* LinkedList<Partition> partition;  
 */\*\*  
 \* 上次分配的空闲分区位置  
 \* \*/  
 public int* pointer;  
  
 */\*\*  
 \* 分区类  
 \* \*/  
 class* Partition{  
 */\*\*  
 \* 分区起始地址  
 \* \*/  
 public int* head;  
 */\*\*  
 \* 分区长度  
 \* \*/  
 public int* size;  
 */\*\*  
 \* 分区状态  
 \* \*/  
 public boolean* state;  
  
 *public* Partition(*int* head, *int* size) {  
 *this*.head = head;  
 *this*.size = size;  
 *this*.state = *true*;  
 }  
 }  
  
 */\*\*  
 \* 默认内存大小为100KB  
 \* \*/  
 public* Memory(){  
 *this*.size = 100;  
 *this*.pointer = 0;  
 *this*.partition = *new* LinkedList<>();  
 partition.add(*new* Partition(0, size));  
 }

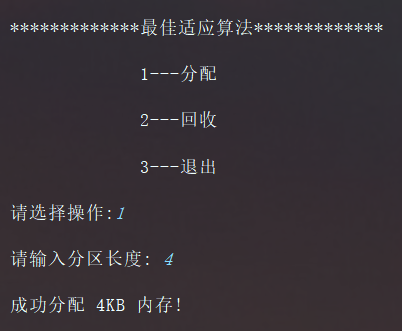
*/\*\*  
 \* 最佳适应算法  
 \* @param* ***size*** *指定需要分配的大小  
 \* \*/  
 public void* bestFit(*int* size){  
 *int* flag = -1;  
 *int* min = *this*.size;  
 *for* (pointer = 0; pointer < partition.size(); pointer++){  
 Partition temp = partition.get(pointer);  
 *if* (temp.state && (temp.size > size)){  
 *if* (min > temp.size - size){  
 min = temp.size - size;  
 flag = pointer;  
 }  
 }  
 }  
 *if* (flag == -1){  
 System.out.println("无可用内存空间!");  
 }*else* {  
 doAllocation(size, flag, partition.get(flag));  
 showAllocatedTable();  
 showFreeTable();  
 }  
 }  
  
 */\*\*  
 \* 执行分配  
 \* @param* ***size*** *申请大小  
 \* @param* ***location*** *当前可用分区位置  
 \* @param* ***temp*** *可用空闲区  
 \*/  
 public void* doAllocation(*int* size, *int* location, Partition temp) {  
 *//如果分割后分区剩余大小过小（MIN\_SIZE）则将分区全部分配，否则分割为两个分区  
 if* (temp.size - size <= MIN\_SIZE){  
 temp.state = *false*;  
 } *else* {  
 Partition split = *new* Partition(temp.head + size, temp.size - size);  
 partition.add(location + 1, split);  
 temp.size = size;  
 temp.state = *false*;  
 }  
 System.out.println("成功分配 " + size + "KB 内存!");  
 }  
  
 */\*\*  
 \* 内存回收  
 \* @param* ***id*** *指定要回收的分区号  
 \*/  
 public void* collection(*int* id){  
 *if* (id >= partition.size()){  
 System.out.println("无此分区编号!");  
 *return*;  
 }  
 Partition temp = partition.get(id);  
 *int* size = temp.size;  
 *if* (temp.state) {  
 System.out.println("指定分区未被分配, 无需回收");  
 *return*;  
 }  
 *//如果回收分区不是尾分区且后一个分区为空闲, 则与后一个分区合并  
 if* (id < partition.size() - 1 && partition.get(id + 1).state){  
 Partition next = partition.get(id + 1);  
 temp.size += next.size;  
 partition.remove(next);  
 }  
 *//如果回收分区不是首分区且前一个分区为空闲, 则与前一个分区合并  
 if* (id > 0 && partition.get(id - 1).state){  
 Partition previous = partition.get(id - 1);  
 previous.size += temp.size;  
 partition.remove(id);  
 id--;  
 }  
 partition.get(id).state = *true*;  
 System.out.println("内存回收成功!, 本次回收了 " + size + "KB 空间!");  
 showAllocatedTable();  
 showFreeTable();  
 }  
  
 */\*\*  
 \* 展示内存分区状况  
 \*/  
 public void* showAllocatedTable(){  
 System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*已分配分区表\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 System.out.println("分区编号\t分区起始地址\t分区长度\t空闲状态\t");  
 System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 *for* (*int* i = 0; i < partition.size(); i++){  
 Partition temp = partition.get(i);  
 *if*(temp.state == *false*){  
 System.out.println(i + "\t\t" + temp.head + "\t\t" +  
 temp.size + " \t" + temp.state);  
 }  
 }  
 System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 System.out.println();  
 }  
 */\*\*  
 \* 展示空闲分区表  
 \* \*/  
 public void* showFreeTable(){  
 System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*空闲分区表\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 System.out.println("分区编号\t分区起始地址\t分区长度\t空闲状态\t");  
 System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 *for* (*int* i = 0; i < partition.size(); i++){  
 Partition temp = partition.get(i);  
 *if*(temp.state == *true*){  
 System.out.println(i + "\t\t" + temp.head + "\t\t" +  
 temp.size + " \t" + temp.state);  
 }  
 }  
 System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 System.out.println();  
 }  
}

三、程序运行环境、使用工具说明

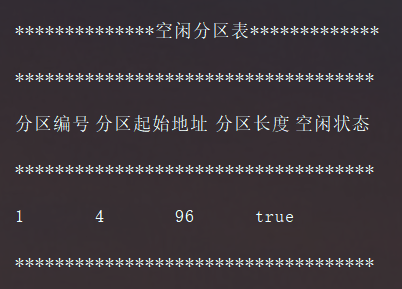
Jdk1.8 ideal

四、程序运行结果

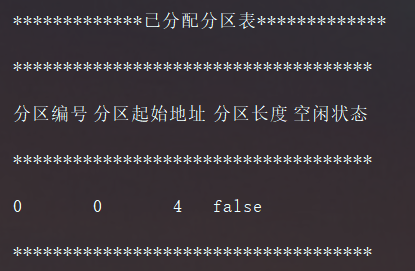
分区分配：



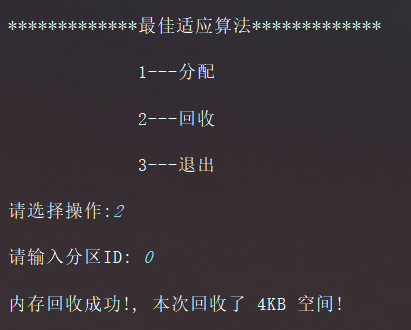
分区分配---空闲分区表：



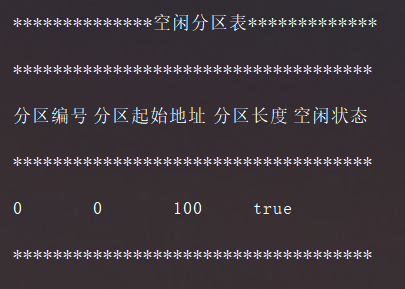
分区分配---已分配分区表：



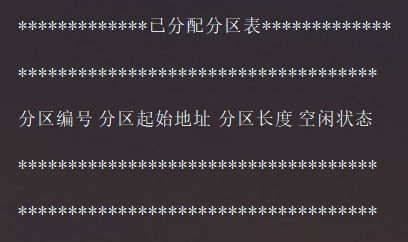
分区回收：



分区回收---空闲分区表：



分区回收---已分配分区表：



五、结果的简单说明或分析

对最佳适应算法了解得更加深入。