Java高级类型查询速度调研

1. 调研目的

了解，并掌握java中高级类的用法；通过调研了解各种高级类的查找速度，从而再实际情况中选择更好的类型来完成工作。

1. 调研过程

代码如下：

package homework;

import java.util.ArrayList;//引入ArrayList类型包

import java.util.LinkedList;//引入LinkedList类型包

import java.util.HashSet;//引入HashSet类型包

import java.util.List;//引入List类型包

import java.io.\*;//引入文件输入输出包

import java.util.Set;//引入set类型包

import java.util.TreeSet;//引入TreeSet类型包

import java.util.Date;

/\*定义一个抽象类 mytest

\* 因为下面四个子类中 有很多相同的方法，可以用继承减少代码量，其中又有方法虽然方法名相同，功能相同，但用到的方法各不相同，则可以定义为抽象方法

\* 在子类中一一实现

\* 两个Long类型的属性 用于子类方法中记录方法使用前和使用后的时间，相减得到方法的执行时间；

\* \*/

abstract class mytest{

long time1;

long time2;

/\*

\* write 方法

\* @param String 类型的a 用于接受写入的第一个数据，String 类型的b 用于接受写入的第二个数据，String类型的c 用于接受文件写入的路径；

\* @return void

\* 数据写入的是csv格式的Excelb表格模式，在这个模式下 “，”可以分割俩个表格 一行写入两个数据后 用“\n”换行；

\* 注:write方法只能写入String类型数据 如果写入的是整数 则要强制转换类型 Integer.toString()来转换;

\* 写入完成后 用close()方法关闭写入流；

\* \*/

public void write(String a,int b,String c) throws IOException {

FileWriter w = new FileWriter (c,true);

w.write(a);

w.write(",");

w.write(Integer.toString(b));

w.write("\n");

w.close();

}

/\*

\* rondom 生成随机数方法

\* @param int 类型的n 用于接受界定生出数的范围大小的参数n；

\*

\* \*/

public int rondom(int n) {

int a=1+(int)(Math.random()\*(n));

return a;

}

/\*抽象方法 structuer

\* @param int 类型的n 用于接受 范围

\* 该方法的作用是在测试的四个高级类型中存放相应的数据；

\* \*/

public abstract void structuer(int n);

/\*

\* 抽象方法 test

\* 该方法的作用是从1000开始 在1000范围内 随机选出20个数据 让该类中定义的高级类型进行查找，再用time1和time2之差再除以20计算出平均时间

\* 并将范围大小和平均时间写入所选路径的excle表格中

\* 利用while循环 循环增加范围大小 进行反复计算写入，每次增加1000 只到范围变为1000000

\* \*/

public abstract void test() throws IOException;

}

/\*子类ArrayList

\* 这个类是测试高积类ArrayLIst 类型的查找速度

\* \*/

class Arraylist extends mytest{

List<String>list=new ArrayList<String>();//生成ArrayList实例

/\*实现抽象方法\*/

public void structuer(int n) {

for(int i=1;i<=n;i++) {

list.add(Integer.toString(i));//list的方法 add()可以添加进入数据

}

}

/\*实现抽象类

\* \*/

public void test() throws IOException {

int i=1000;

this.structuer(1000000);

Long time3=System.nanoTime();

while(i<=1000000) {

Long time4=System.nanoTime();

System.out.println(time4-time3);

int j=0;

time1=System.nanoTime();

//获取当前毫秒数

for(;j<20;j++ ) {

int a=this.rondom(i);

int b=list.indexOf(Integer.toString(a));//list的indexOf()是查找方法;

}

time2=System.nanoTime();//获取方法结束后的毫秒数

int time =((int) ((time2-time1))/20);//得到平均查找时间

this.write(Integer.toString((i)),time,"E:\\szjava\\MyJava\\JavaWork\\Arraylist.csv");

//this.list.removeAll(list);//清空当前的list表中数据 为下一次循环写入数据做准备;

i=i+1000;

System.out.println(i);

}

System.out.println("生成完毕");

}

}

/\*Linkedlist子类 这个子类用于测试LinkedList高级类型的查找速度\*/

class Linkedlist extends mytest{

LinkedList<String> link= new LinkedList<String>();//实例化LinkedList对象

/\*实现抽像方法\*/

public void structuer(int n) {

for(int i=1;i<=n;i++) {

link.add(Integer.toString(i));//ListedList的添加数据方法也是add();

}

}

/\*实现抽象方法\*/

public void test() throws IOException {

int i=1000;

this.structuer(1000000);

while(i<=1000000) {

int j=0;

time1=System.nanoTime();

for(;j<20;j++ ) {

int a=this.rondom(i);

int b=link.indexOf(Integer.toString(a));

}

time2=System.nanoTime();

int time =((int) ((time2-time1))/20);

this.write(Integer.toString((i)),time,"E:\\szjava\\MyJava\\JavaWork\\Linkedlist.csv");

i=i+1000;

}

System.out.println("生成完毕");

}

}

/\*Hashset子类 用于测试HashSet高级方法的查找方法\*/

class Hashset extends mytest{

Set set=new HashSet();//实例化HashSet对象

/\*实现抽象方法\*/

public void structuer(int n) {

for(int i=1;i<=n;i++) {

set.add(Integer.toString(i));

}

}

/\*实现抽象方法\*/

public void test() throws IOException {

int i=1000;

this.structuer(1000000);

while(i<=1000000) {

int j=0;

for(;j<20;j++ ) {

int a=this.rondom(i);

time1=System.nanoTime();

boolean b=set.contains(Integer.toString(a));//HashSet中的查找方法为contains()返回值为Boolean

}

time2=System.nanoTime();

int time =((int) ((time2-time1))/20);

this.write(Integer.toString((i)),time,"E:\\szjava\\MyJava\\JavaWork\\Hashset.csv");

i=i+1000;

}

System.out.println("生成完毕");

}

}

/\*Treeset子类用于测试 TreeSet高级类的查找速度\*/

class Treeset extends mytest{

TreeSet <String>tree=new TreeSet<String>();//实例化TreeSet对象

/\*实现抽象方法\*/

public void structuer(int n) {

for(int i=1;i<=n;i++) {

tree.add(Integer.toString(i));

}

}

/\*实现抽象方法\*/

public void test() throws IOException {

int i=1000;

this.structuer(1000000);

while(i<=1000000) {

int j=0;

time1=System.nanoTime();

for(;j<20;j++ ) {

int a=this.rondom(i);

boolean b=tree.contains(Integer.toString(a));//TreeSet 高级类型中的查找方法为contains()返回值为Boolean类型

}

time2=System.nanoTime();

int time =((int) ((time2-time1))/20);

this.write(Integer.toString((i)),time,"E:\\szjava\\MyJava\\JavaWork\\Treeset.csv");

i=i+1000;

}

System.out.println("生成完毕");

}

}

public class test3 {

public static void main(String[]args) throws IOException {

fun(new Arraylist());

//fun(new Linkedlist());

//fun(new Hashset());

//fun(new Treeset());

}

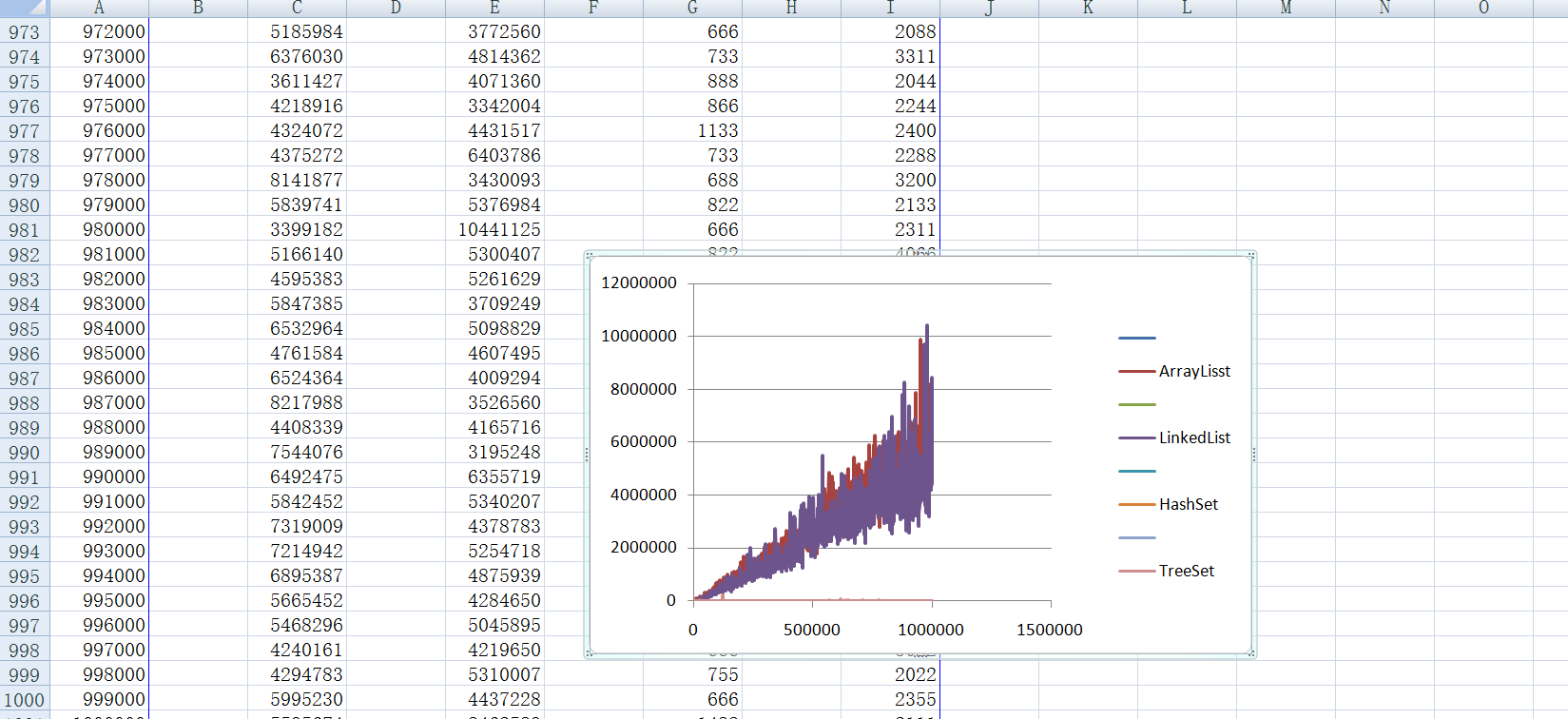
public static void fun(mytest a) throws IOException {

a.test();

}

}

1. 调研结果

生成的数据写入Excel表中 由表格生成统计图有：

由图可以观察得到 在1000-1000000这些容量中，每次取20个随机数进行查询，从1000-1000000之间 一共获取了1000个平均时间点，绘制成这个统计图，在图纸 HashSet和TreeSet这两个都是呈现直线，因为在与另外两个类型相比，他们的数据量太小 以至于排到统计表的最底层，呈现直线，而ArrayList 和LinkedList 在随着容量的变大，查询时间越来越大,所以再以后的工程中，要在大量数据中进行查找时，尽量选择HashSet 和TreeSet。