java.util.Date

new Date() getYear() + 1900 才是正常年份

getMonth() 从0开始

getTime() 获取这个时间对象对应的毫秒值

setYear(新年份)

java.util.Calendar

Calendar.getInstance() 创建子类对象

.get(Calendar.YEAR) 获取时间的某一部分

.set(Calendar.YEAR, 新年份) 修改时间某一部分

.add(Calendar.YEAR, +-) 增减时间某一部分

.getTime() 返回Date对象

java.math.BigDecimal

new BigDecimal("数字")

.add()

.substract()

.multiply()

.divide(除数， 保留位数，舍去方式)

java.lang.String

"字面量" 属于常量池, 常量池中的字符串没有重复的

new String("字符串") 属于堆

"字面量" + "字面量"

字符串变量 + 字符串变量 =》 替换为

String a = "aa";

String b = "bb";

String c = a + b; // 替换为下面的代码

String c = new StringBuilder(a).append(b).toString();

.length()

.split(正则)

.substring(begin, end)

.indexOf('子串')

.matchs(正则)

如何匹配数字、单词字符、邮件地址

[abc] [a-g] [^] {m,n} \* + ?

1. 不同进制下数字类型的表示

int a = 10; // 以10进制的方式进行赋值

jdk 6 以前

int b = 010; // 以八进制的方式进行赋值 （0打头的数字）,

实际上是10进制的8

int c = 0xa; // 以十六进制的方式赋值（0x打头的数字）， 值是10进制的10

jbk 7，8 里

int e = 0b1010;

System.out.println(e);

// 0b 打头的数字， 表示10进制的10

1. final

class Student{

String name;

public Student(String name) {

this.name = name;

}

}

public class TestStudent {

public static void main(String[] args) {

final Student stu = new Student("张三");

// stu 的值不能比修改, stu 的地址不能修改了

// stu = new Student("啊啊"); // 错误的

stu.name = "李四"; // stu中name并没有加final ,可以修改

// 对象中的内容能否被修改，跟变量的final没有关系，是由对象自身内部来决定的

}

}

1. String, StringBuilder, StringBuffer

String中内容是不可变的

StringBuilder, StringBuffer 他们都可以通过append之类的方法，修改对象中的内容

StringBuilder 线程不安全

StringBuffer 线程安全

public static void main(String[] args) {

// String s1 = "abc";

// test1(s1);

// System.out.println(s1);

StringBuffer sb1 = new StringBuffer("abc");

test2(sb1);

System.out.println(sb1);

}

public static void test1(String str) {

str = "def";

}

public static void test2(StringBuffer sb) {

sb.append("def");

}

# java 中的集合类

List 列表

Set 集合

Queue 队列

Map 映射

## List 列表集合类

java.util.List 接口

java.util.ArrayList 是List接口的一个重要实现

// 数组的缺点:

/\*

public static void main(String[] args) {

int[] array = { 1, 2, 3, 4, 5 };

// 需求是 3 删除，剩下的内容生成新数组

int[] newArray = new int[array.length - 1];

int j = 0;

// 循环效率随着数组长度增大会越来越低

for (int i = 0; i < array.length; i++) {

if (i == 2) {

continue;

}

newArray[j++] = array[i];

}

System.out.println(Arrays.toString(newArray));

}

\*/

// java 中提供了针对数组元素拷贝特别高效的方法

// System.arraycopy(旧数组, 旧起始下标, 新数组, 新起始下标, 元素个数)

// 演示System.arraycopy的基本用法

/\*

public static void main(String[] args) {

int[] array1 = { 1, 2, 3, 4, 5 };

int[] array2 = new int[5];

System.out.println("拷贝之前:" + Arrays.toString(array2));

System.arraycopy(array1, 2, array2, 3, 2);

System.out.println("拷贝之后:" + Arrays.toString(array2));

}

\*/

// 使用arraycopy来改进数组复制的效率

public static void main(String[] args) {

int[] array = { 1, 2, 3, 4, 5 };

// 需求是 3 删除，剩下的内容生成新数组

int[] newArray = new int[array.length - 1];

System.arraycopy(array, 0, newArray, 0, 2);

System.arraycopy(array, 3, newArray, 2, 2);

System.out.println(Arrays.toString(newArray));

}

### ArrayList （基于数组实现）

创建方法：

ArrayList list = new ArrayList();// 初始容量 10

刚开始List内部的数组大小是10，如果在放入第11个元素时，数组容量就会翻倍为20

ArrayList list = new ArrayList(初始容量);

list中还有一个size (表示是数组元素的实际大小)

0 0 0 0 0 0 0 0 0 0 size 0

1 0 0 0 0 0 0 0 0 0 size 1

添加元素

.add(元素) 会把元素添加至List集合的尾部

list.add(1)

list.add(5)

list.add(7)

0 1 2 3 4 5 6 7 8 9

1 5 7

// List

List list = new ArrayList();

// 添加元素至末尾

list.add(1); //int => Integer

System.out.println(list);

list.add(3);

System.out.println(list);

list.add(5);

System.out.println(list);

// 添加元素到指定位置

list.add(0, 7);

System.out.println(list);

// 获取元素

System.out.println("获取下标为2的元素:" + list.get(2));

// 下标不能超过实际的size属性值

System.out.println("获取size: "+ list.size());

// 遍历集合:

// 方法1：

for(int i = 0; i < list.size(); i++) {

System.out.println("下标为" + i + "的元素:" + list.get(i));

}

System.out.println("===============================");

// 方法2： for each 遍历

for(Object e : list){ // e代表每次遍历出的元素

System.out.println("e:" + e + "实际类型：" + e.getClass());

}

// 方法3：Iterator(迭代器) 遍历 （了解）

System.out.println("===============================");

Iterator iter = list.iterator(); // 得到迭代器

// iter.hasNext() 有没有下一个元素 ，有返回true， 没有则返回false

// iter.next() // 让指针移动到下一个元素,并返回

while(iter.hasNext()) {

System.out.println(iter.next());

}

// 删除下标为2的元素

list.remove(2);

System.out.println(list);

// 清空集合

list.clear();

System.out.println(list);

List list2 = new ArrayList();

list2.add(5);list2.add(6);list2.add(7);

// 添加所有元素至list

list.addAll(list2);

System.out.println(list);

// 将list转换为数组

Object[] array = list.toArray();

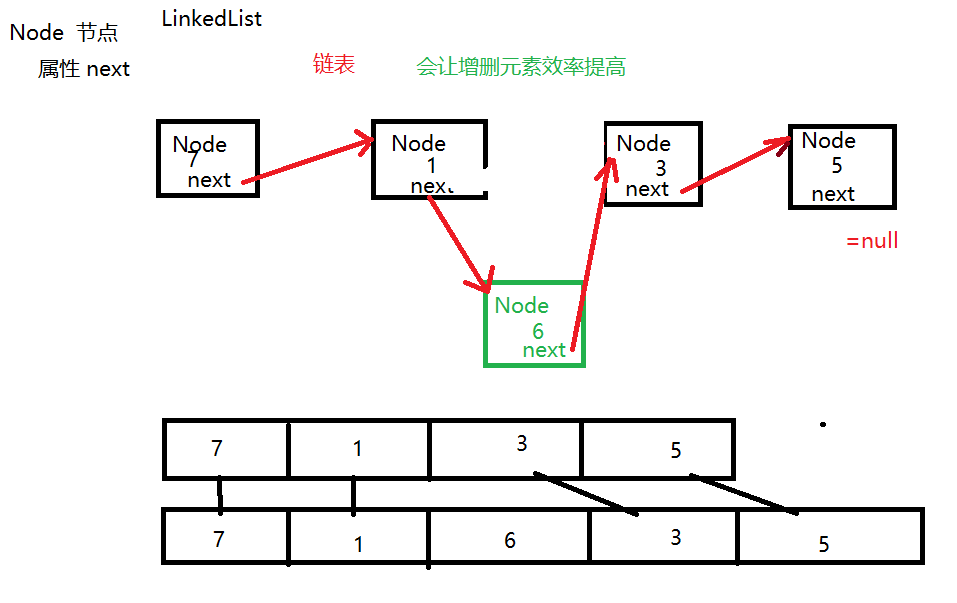
System.out.println(Arrays.toString(array));

总结：内部基于数组实现，从性能上讲，查询快

增加和删除慢（内部需要移动数组的元素）

### LinkedList (基于链表实现)

从性能上讲，增加和删除块，查询(随机访问)慢



List list = new LinkedList();

list.add(1); // 添加元素

list.get(0); // 获取元素

list.remove(0); // 删除元素

1. 下标 +size
2. for(Object o : list)
3. list.iterator() .hasNext() .next()

## Set 集合

与list集合的区别：

1. list 代表的是有序的一组数据
2. set 不关心数据的存储顺序（有的实现考虑顺序，有的实现不考虑顺序）
3. set 集合中的元素不能重复(值相等的元素，在set只存储一份)

主要实现类： HashSet ， LinkedHashSet

LinkedHashSet记录了元素添加时的顺序，在遍历时仍然是使用添加时的顺序进行遍历

HashSet 不会记录添加顺序，遍历时是无序的

### HashSet 哈希（散列）Set

人名 ： 数量在千以上, 如何快速的找到其中某一个人名

Mary Tom Jerry ... ... 一千个 Mike

数组

0 1 2 12 .... 25

A B M

Entry(Mary, next)

Entry(Mari, next)

Entry(Mike, next)

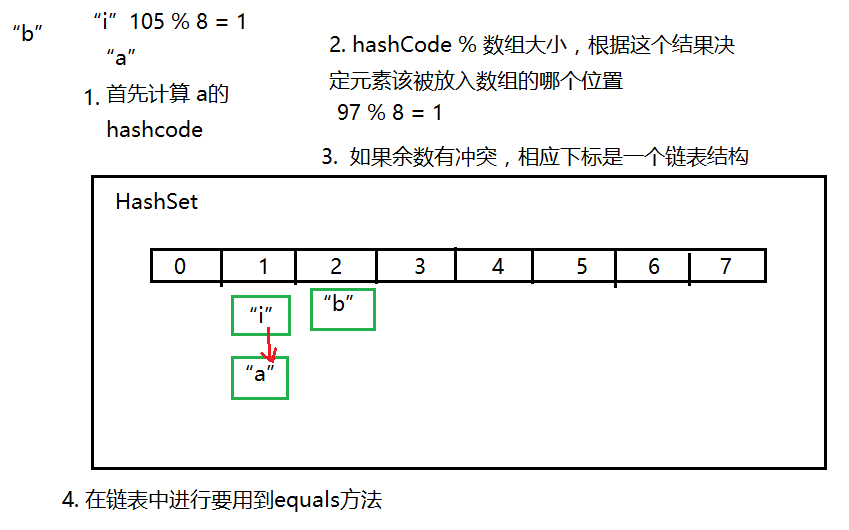
null

为了找到 Mike 这个姓名，先根据首字母，确定数组的下标，只需要跟这个下标的链表中的元素比较一遍就可以了， （一种快速查找的思路）

HashSet 的实现是类似的，把元素的哈希值去找数组下标

哈希值取得好，就可以让元素在集合中的分布更为均匀，eclipse 可以为我们自动生成hashCode方法，jdk中的常见类型都已经实现了hashCode

向HashSet 里添加元素，流程：



// 1. 创建HashSet对象

Set set = new HashSet();

// 2. 添加元素

// 注意：1) 被添加的元素要实现hashCode 和equals方法

// 2) 集合中元素不能重复

set.add("aaa"); // 被加入了集合

set.add("bbb");

set.add("ccc");

set.add("aaa"); // 已经存在，略过

set.add("aaa"); // 已经存在，略过

set.add("aaa"); // 已经存在，略过

System.out.println(set);

// 3. 遍历集合

// 方法1： for-each遍历

System.out.println("==============");

for(Object o : set) {

System.out.println("元素：" + o);

}

System.out.println("集合的大小:" + set.size());

// 方法2： Iterator迭代器

System.out.println("==============");

Iterator iter = set.iterator();

while(iter.hasNext()) {

System.out.println("元素:" + iter.next());

}

// 4. 删除元素

set.remove("bbb");

System.out.println("删除后："+ set);

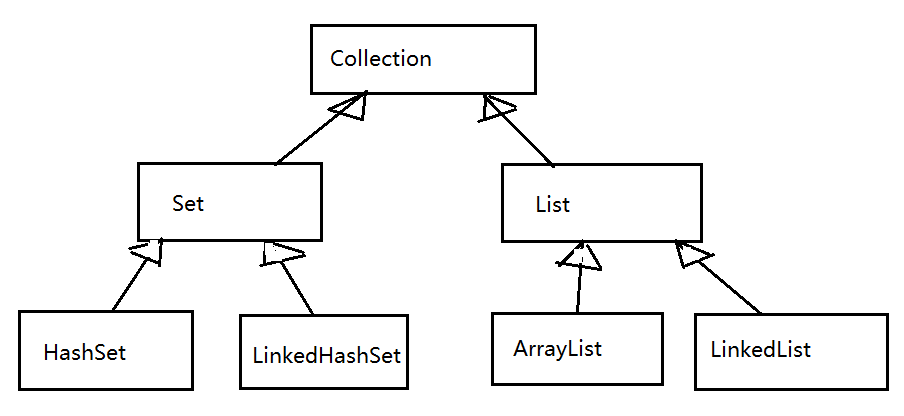
// 5. 清空集合

set.clear();

// 6. 将另一个集合的所有元素，添加到set集合中

// set.addAll(c)

### set和list的关系：



查看源码的几种方法：

// ctrl + 鼠标单击

// 或右键 选 Open Declartion (F3)

// Open Type图标

查看继承关系树：

// 找到顶层的接口或父类, 在父类名上：

右键 选 Open Type Hierarchy (F4)