第7章 工具类及常用算法

部分内容摘自

《Java面向对象编程》，孙卫琴

《Java 程序设计》，唐大仕

7.1.1 Java基础类库

- java.lang包是Java语言的核心类库
- java.io包是Java语言的标准输入/输出类库
- java.util包包括了Java语言中的一些低级的实用工具
- java.awt包是Java语言用来构建图形用户界面(GUI)的类库
- java.applet包是用来实现运行于Internet浏览器中的Java Applet的工具类库
- java.net包是Java语言用来实现网络功能的类库
- 其他包

JDK API文档

- JDK API文档可以从java.sun.com网站下载，安装后，打开index.html即可
- 相关软件\docs.rar
- 或者相关软件\ jdk1.8帮助文件.rar

Java中一些常用类

java文档：

http://tool.oschina.net/apidocs/apidoc?api=jdk_7u4

java最权威的教材：

http://docs.oracle.com/javase/tutorial/

7.1.2 java.lang.Object类

- Object类是Java程序中**所有类**的直接或间接父类

(1) equals( )

- 用来比较两个对象是否相同，如果相同，则返回**true**，否则返回**false**，它比较的是两个对象状态和功能上的相同，而不是引用上的相同。
- **Objet. equals默认实现：**

```
public boolean equals(Object obj) {
    return (this == obj);
}
```

- **Integer. equals实现：**

```
public boolean equals(Object obj) {
    if (obj instanceof Integer) {
        return value == ((Integer)obj).intValue();
    }
    return false;
}
```

equals( )

- **String. equals()**

例子：
ch07.eq.TestEqualsString
ch07. eq.TestEqualsObject

```
public boolean equals(Object anObject) {
    if (this == anObject) {
        return true;
    }
    if (anObject instanceof String) {
        String anotherString = (String)anObject;
        int n = value.length;
        if (n == anotherString.value.length) {
            char v1[] = value;
            char v2[] = anotherString.value;
            int i = 0;
            while (n-- != 0) {
                if (v1[i] != v2[i])
                    return false;
                i++;
            }
            return true;
        }
    }
    return false;
}
```

(1) equals( )

- 用来比较两个对象是否相同，如果相同，则返回**true**，否则返回**false**，它比较的是两个对象状态和功能上的相同，而不是引用上的相同。

```
Integer one = new Integer (1);
Integer anotherOne = new Integer (1);
if (one.equals (anotherOne))
System.out.println (“objects are equal”);
```

- 例中，**equals( )**方法返回**true**，因为对象**One**和**anotherOne** 包含相同的整数值**1.**

(2) getClass ( )

- **getClass ( )**方法是**final**方法，它不能被重载。它返回一个对象在运行时所对应的类的表示，从而可以得到相应的信息。下面的方法得到并显示对象的类名：

```
void PrintClassName( Object obj )
{
   System.out.println(“ The object’s class is “ +
obj.getClass( ).getName( ) );
}
```

(3) toString( )

- **toString( )**方法用来返回对象的字符串表示，可以用来显示一个对象。例如：
- **System.out.println (Thread.currentThread ( ).toString ( ));//**显示当前的线程。
- 通过重载**toString ( )**方法可以适当地显示对象的信息以进行调试。

(4) finalize( )

- 用于在垃圾收集前清除对象，前面已经讲述。

7.1.3 基本数据类型的包装类

- Java的基本数据类型用于定义简单的变量和属性将十分方便，但为了与面向对象的环境一致，Java中提供了基本数据类型的包装类（wrapper），它们是这些基本类型的面向对象的代表。与8种基本数据类型相对应，基本数据类型的包装类也有8种，分别是：Character，Byte，Short，Integer，Long，Float，Double，Boolean。
- 这几个类有以下共同特点。

（1）这些类都提供了一些常数，以方便使用，如Integer.MAX_VALUE（整数最大值），Double.NaN(非数字)，Double. POSITIVE_INFINITY（正无穷）等。

（2）提供了valueOf(String)，toString()，用于从字符串转换及或转换成字符串。

（3）通过xxxxValue()方法可以得到所包装的值，Integer对象的intValue()方法。

（4）对象中所包装的值是不可改变的（immutable）。要改变对象中的值只有重新生成新的对象。

（5）toString(), equals()等方法进行了覆盖。

- 除了以上特点外，有的类还提供了一些实用的方法以方便操作。例如，Double类就提供了更多的方法来与字符串进行转换。
- 例子：ch07. DoubleAndString

```
//double转成string的几种方法
                d=3.14159；
                s = "" + d；
                s = Double.toString( d )；
                s = new Double(d).toString()；
                s = String.valueOf( d )；
// String转成double的几种方法
                s = "3.14159"；
                try{
                        d = Double.parseDouble( s )；
                        d = new Double(s).doubleValue()；
                        d = Double.valueOf( s ).doubleValue()；
                }
                catch(NumberFormatException e )
                {
                        e.printStackTrace()；
                }
```

7.1.4 Math类

- Math类用来完成一些常用的数学运算
- public final static double E；// 数学常量e
- public final static double PI；// 圆周率常量
- public static double abs(double a)；// 绝对值
- public static double exp(double a)；// 参数次幂
- public static double floor(double a)；// 不大于参数的最大整数
- public static double IEEEremainder(double f1，double f2)；// 求余
- public static double log(double a)；// 自然对数
- public static double max(double a，double b)；// 最大值
- public static float min(float a，float b)；// 最小值

7.1.4 Math类

- public static double pow(double a，double b)；// 乘方
- public static double random()；// 产生0和1(不含1)之间的伪随机数
- public static double rint(double a)；// 四舍五入
- public static double sqrt(double a)；// 平方根
- public static double sin(double a)；// 正弦
- public static double cos(double a)；// 余弦
- public static double tan(double a)；// 正切
- public static double asin(double a)；// 反正弦
- public static double acon(double a)；// 反余弦
- public static double atan(double a)；// 反正切

System类 ch07. SystemDemo

public static long currentTimeMillis() 取得当前毫秒

public static Properties getProperties() 取得当前全部环境属性

public static String getProperty(String key) 取得当前属性

public static void gc() 强制垃圾回收

public static void arraycopy(Object src, int srcPos, Object dest, int destPos, int length) 数组复制

public static void exit(int status) 系统退出

System.out.println();

System.out.print ();

7.2 字符串

- 程序中需要用到的字符串可以分为两大类，一类是创建之后不会再做修改和变动的字符串常量；另一类是创建之后允许再做更改和变化的字符串。前者是String类，后者是StringBuffer/StringBuilder类。

Java API提供了四个处理字符数据的类：

- Character：这个类的实例可以容纳单一的字符数值。该类还定义了一些简洁的方法来操作或者检查单一字符数据。
- String：这个类用于处理由多个字符组成的不可变数据。
- StringBuffer：这个类用于存储和操作由多个字符组成的可变数据。
- StringBuilder:类似StringBuffer

字符类——构造器和方法

- Character(char)—Character类唯一的构造器，它创建一个字符对象，其中包含由参数提供的值，一旦创建了Character对象，它包含的值就不能改变。
- compareTo(Character)—这个实例方法比较两个字符对象包含的值，这个方法返回一个整数值，表示当前对象中的值是大于、等于还是小于参数所包含的值

字符类——构造器和方法

- equals(Object)—这个实例方法比较当前对象包含的值与参数对象包含的值，如果两个对象包含的值相等，那么这个方法返回true
- toString()—这个实例方法将此对象转换为字符串
- charValue()—这个实例方法以原始char值的形式返回此字符对象包含的值
- isUpperCase()—这个实例方法判断一个原始char值是否是大写字母

Character常用方法

Method	Description
`boolean isUpperCase(char)` `boolean isLowerCase(char)`	Determines whether the specified primitive `char` value is upper- or lowercase, respectively.
`char toUpperCase(char)` `char toLowerCase(char)`	Returns the upper- or lowercase form of the specified primitive `char` value.

`boolean isLetter(char)`
`boolean isDigit(char)`
`boolean isLetterOrDigit(char)`
	Determines whether the specified primitive `char` value is a letter, a digit, or a letter or a digit, respectively.

`boolean isWhitespace(char)`[a]	Determines whether the specified primitive `char` value is white space according to the Java platform.
`boolean isSpaceChar(char)`[b]	Determines whether the specified primitive `char` value is a white-space character according to the Unicode specification.
`boolean isJavaIdentifierStart(char)`[c] `boolean isJavaIdentifierPart(char)`[d]	Determines whether the specified primitive `char` value can be the first character in a legal identifier or be a part of a legal identifier, respectively.

String –不变类

- 使用字符串常量时，需要创建String对象，和其它对象不同，String对象可以通过简单赋值语句创建：
  - String name = “Petter”;
- 此外，也可根据String类的构造函数创建String对象：
  - String name = new String(“Petter”);
- 对于程序任何位置出现的双引号标记的字符串，系统都会自动创建一个String对象。
- 可通过String对象的方法对字符串进行操作

String构造函数

Constructor	Description
`String()`	Creates an empty string.

`String(byte[])`
`String(byte[], int, int)`
`String(byte[], int, int, String)`
`String(byte[], String)`
	Creates a string whose value is set from the contents of an array of bytes. The two integer arguments, when present, set the offset and the length, respectively, of the subarray from which to take the initial values. The `String` argument, when present, specifies the character encoding to use to convert bytes to characters.

`String(char[])` `String(char[], int, int)`	Creates a string whose value is set from the contents of an array of characters. The two integer arguments, when present, set the offset and the length, respectively, of the subarray from which to take the initial values.
`String(String)`	Creates a string whose value is set from another string. Using this constructor with a literal string argument is not recommended, because it creates two identical strings.
`String(StringBuffer)`	Creates a string whose value is set from a string buffer.

String

1、length() 字符串的长度

例：char chars[]={'a','b'.'c'};

String s=new String(chars);

int len=s.length();

2、charAt() 截取一个字符

例：char ch;

ch="abc".charAt(1); 返回'b‘

4、getBytes()

字符存储在字节数组中。

5、toCharArray()

6、equals()和equalsIgnoreCase() 比较两个字符串

7、regionMatches() 用于比较一个字符串中特定区域与另一特定区域，它有一个重载的形式允许在比较中忽略大小写。

boolean regionMatches(int startIndex,String str2,int str2StartIndex,int numChars)

boolean regionMatches(boolean ignoreCase,int startIndex,String str2,int str2StartIndex,int numChars)

8、startsWith()和endsWith()　　startsWith()方法决定是否以特定字符串开始，endWith()方法决定是否以特定字符串结束

9、equals()和==

equals()方法比较字符串对象中的字符，==运算符比较两个对象是否引用同一实例。

例：String s1="Hello";

String s2=new String(s1);

s1.equals(s2); //true

s1==s2;//false

11、indexOf()和lastIndexOf()

indexOf() 查找字符或者子串第一次出现的地方。

lastIndexOf() 查找字符或者子串是后一次出现的地方。

12、substring()　它有两种形式，第一种是：String substring(int startIndex)

第二种是：String substring(int startIndex,int endIndex)

13、concat() 连接两个字符串

14 、replace(char original,char replacement) 替换

它有两种形式，第一种形式用一个字符在调用字符串中所有出现某个字符的地方进行替换，形式如下：

String replace(char original,char replacement)

例如：String s="Hello".replace('l','w');

15、trim() 去掉起始和结尾的空格

16、valueOf() 转换为字符串

17、toLowerCase() 转换为小写

18、toUpperCase() 转换为大写

19、 replaceAll(String regex,String replacement) 替换

20、split(String regex)拆分

21、match (String regex)符合特定正则表达式

正则表达式简介

如何保证用户输入的是有效的邮政编码

如何保证用户输入的是有效车牌号

...

百度百科 正则表达式

正则表达式例子

只能输入1个数字： ^\d$

只能输入8个数字 :^\d{8}$

只能输入大写字母 :^[A-Z]+$

匹配QQ号码 :[0-9]{5,10}

StringBuffer 构造函数

Constructor	Description
`StringBuffer()`	Creates an empty string buffer whose initial capacity is 16 characters.
`StringBuffer(int)`	Creates an empty string buffer with the specified initial capacity.
`StringBuffer(String)`	Creates a string buffer whose value is initialized by the specified `String`. The capacity of the string buffer is the length of the original string plus 16.

构造区别 StringBuffer必须用new的方式构造

```
String s = "Dot saw I was Tod";
StringBuffer dest = new StringBuffer(s);
```

StringBuffer

append( ),表示将括号里的某种数据插入原Buffer中

charAt( ),返回此序列中指定索引处的 char 值

toString( ),返回此序列中数据的字符串表示形式。

subString( ), 返回一个新的 String，它包含此序列当前所包含的字符子序列。

delete( ),移除此序列的子字符串中的字符。

deletecharAt ()，移除此序列指定位置的 char。

insert( ),表示将括号里的某种数据类型的变量插入StringBuffer 中

StringBuilder

基本同StringBuffer

String和StringBuffer/StringBuilder区别

1.速度方面：StringBuilder>StringBuffer**>>**String

2.StringBuffer与StringBuilder是**可变对象**，String是**不变对象**。

3. StringBuilder：线程非安全的

StringBuffer：线程安全的

不理解线程安全、非安全没关系

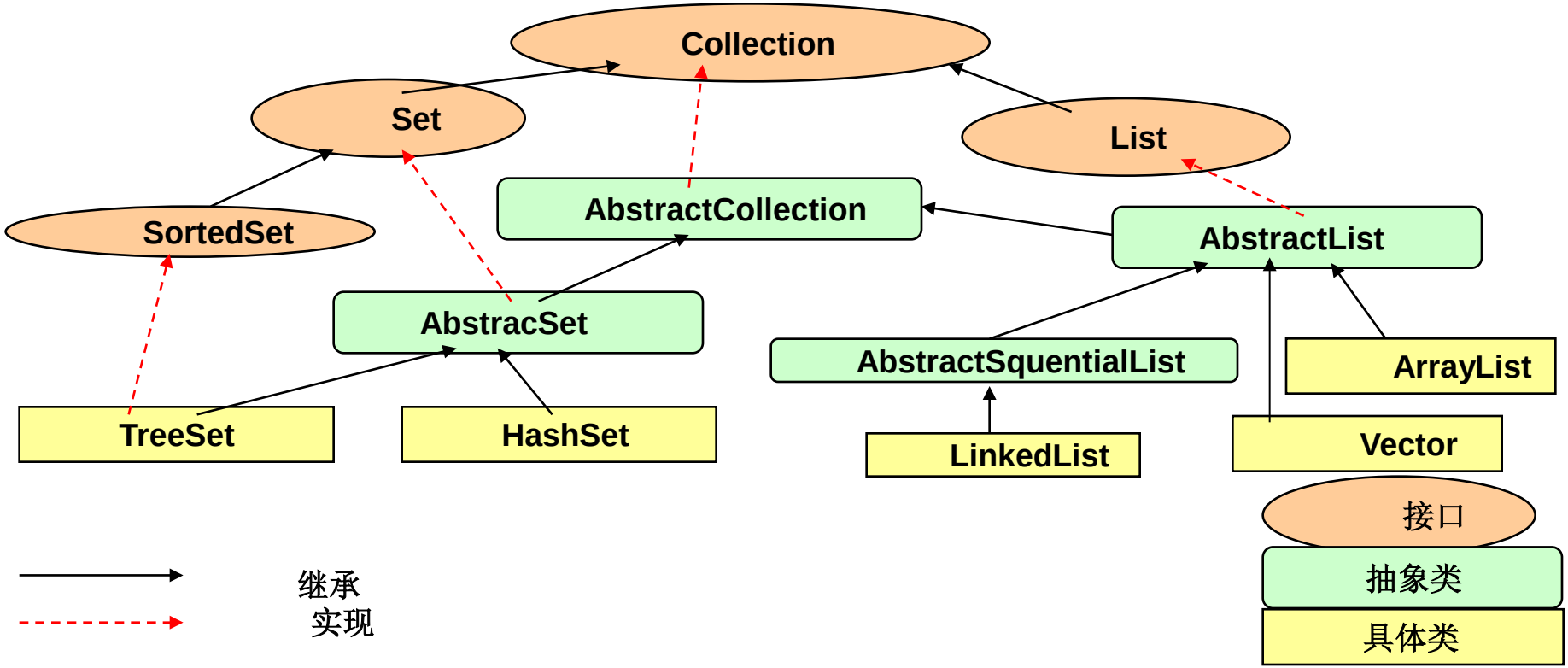
例子：ch07.StrBufStrBuilder

7.2.3 StringTokenizer类

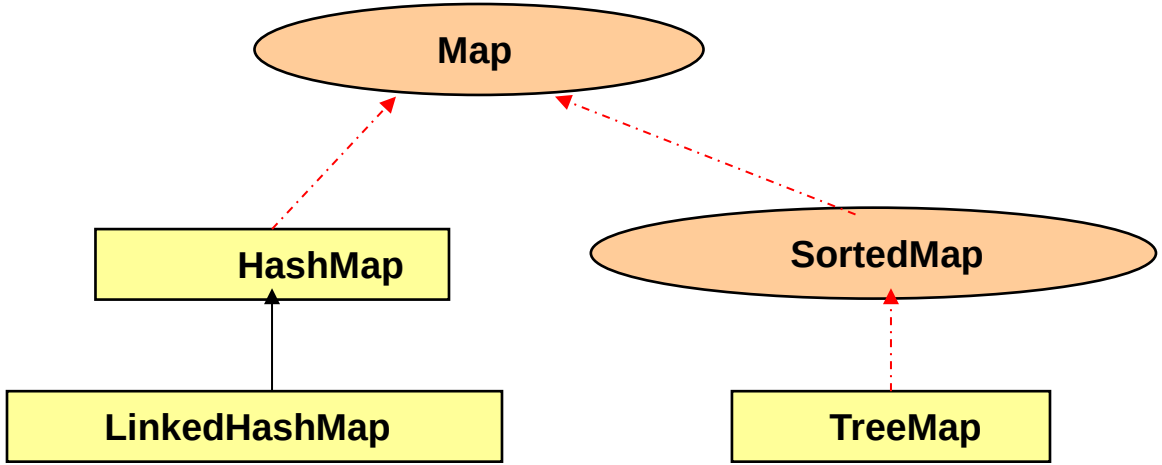
- java.util.StringToken类提供了对字符串进行解析和分割的功能。比如，要对一条语句进行单词的区分，就可以用到该类。
- StringTokenizer的构造方法有：
- StringTokenizer(String str);
- StringTokenizer(String str, String delim);
- StringTokenizer(String str, String delim, boolean returnDelims);
- 其中，str是要解析的字符串，delim是含有分隔符的字符串，returnDelims表示是否将分隔符也作为一个分割串。
- 该类的重要方法有：
- ↘ public int countTokens()；// 分割串的个数
- ↘ public boolean hasMoreTokens()；// 是否还有分割串
- ↘ public String nextToken()；// 得到下一分割串
- 例子:ch07. TestStringTokenizer

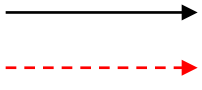
7.3 集合类

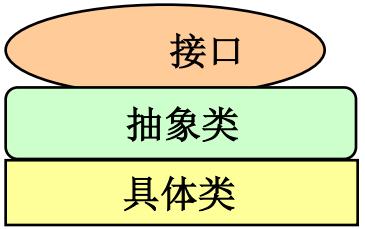
Java集合类框图


Collection
Set
List
SortedSet
AbstractCollection
AbstractList
AbstracSet
AbstractSquentialList
ArrayList
TreeSet
HashSet
LinkedList
Vector
接口
抽象类
具体类
继承
实现

Java集合类框图


Map
HashMap
SortedMap
LinkedHashMap
TreeMap


继承
实现


接口
抽象类
具体类

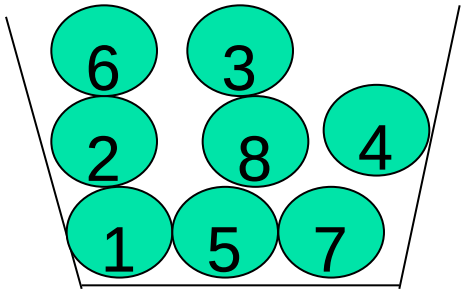
Set 无序 唯一

List 有序 不唯一

Map key->value

Set(集)

- Set是最简单的集合，集合中的对象不按照特定的方式排序，并且没有重复的对象。Set接口主要有两个实现类：HashSet和TreeSet。



Set(集)

Set的一般用法

- Set集合中存放的是对象，并且没有重复对象。
- Java中实现Set接口的类很多，例如：

```
Set     set=new HashSet();
```

- 这样就创建了一个集合对象，我们把它当作抽象的集合接口看待。
- 使用接口的好处在于，实现类将来可以被替换，而程序不用做很多改动。

Set的一般用法 ch07.set.SetDemo

```
public static void main(String[] args) {
    // TODO Auto-generated method stub

    Set set = new HashSet();
    String str = "我是天津大学一名学生";
    char[] chars = str.toCharArray();
    for (int i = 0; i<chars.length; i++) {
        set.add(chars[i]);
    }
    System.out.println("不同字符数量: " + set.size());
}
```

Set的一般用法 ch07.set.SetDemo2

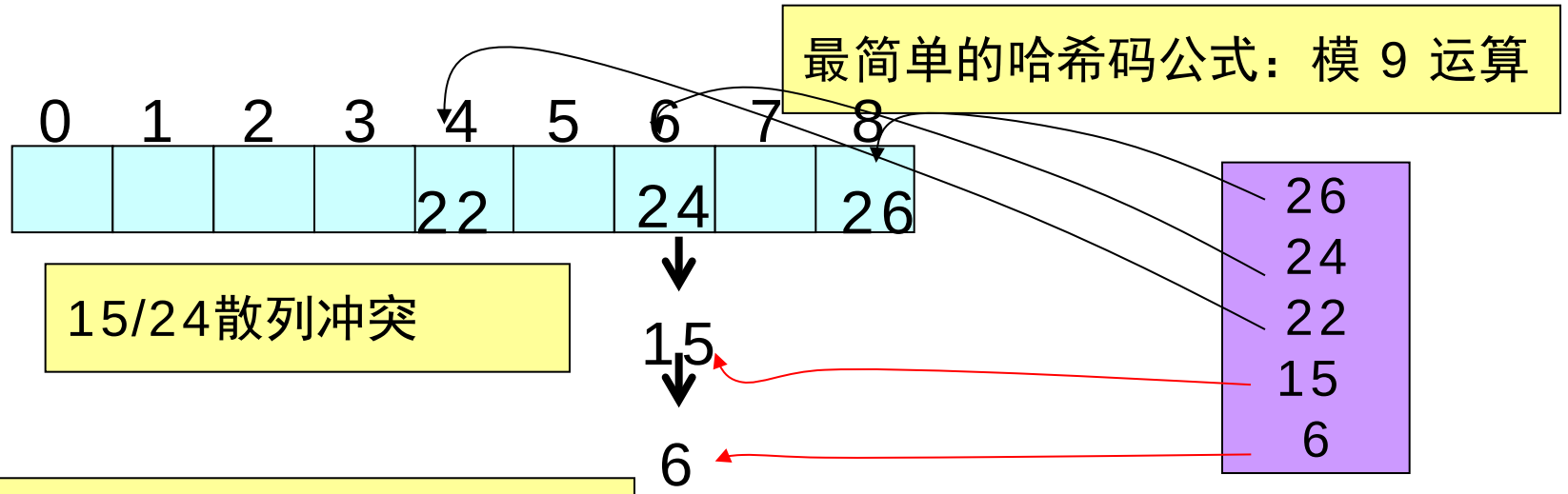
```
public static void main(String[] args) {
    // TODO Auto-generated method stub

    Set set = new HashSet();
    String str = "China will strengthen internation
    String[] strs=str.split(" ");
    for(int i=0;i<strs.length;i++) {
        set.add(strs[i]);
        System.out.println(strs[i]);
    }

    System.out.println("不同单词数量： " + set.size());


}
```

HashSet类

- HashSet类按照哈希算法来存取集合中的对象，具有很好的存取和查找性能。
- 当向集合中加入一个对象时，HashSet会调用对象的hashCode()方法来获得哈希码，然后根据哈希码进一步计算出对象在集合中的位置。

HashSet类-散列技术的原理

- 把对象的哈希码直接用一个固定的公式计算，得出存储位置的方法。
- 优点是：可以快速命中搜索的目标。



HashSet类

- HashSet正常工作的前提：
- 两个你认为相同的对象其hashCode()必须相同且两个对象用equals()方法比较的结果为true时。
- 例子：ch07.set. SetHashCodeEqDemo

TreeSet类

- TreeSet采用树结构来存储数据，数据打印出来是有序的。

TreeSet类

- 当向集合中加入一个对象时，会把它插入到有序的对象集合中。
- TreeSet支持来两种排序方式：自然排序和自定义排序。默认情况下采用自然排序。

自然排序

- 要求被排序的对象实现了Comparable接口的compareTo(Object o)方法
- x.compareTo(y)

  x==y 返回0

  x>y　返回正数

  x<y　返回负数

自然排序

- JDK类库中实现了Comparable接口的一些类。

类	排序
Byte、Double、Float、Integer、Long、Short	按数字大小排序
Character	按字符Unicode值的大小排序
String	按字符串中字符的Unicode值的大小排序

TreeSet的自然排序 ch07.set.TreeSetDemo

```
public static void main (String[] args) {
    // TODO Auto-generated method stub
    Set set = new TreeSet();
    set.add(0);
    set.add(5);
    set.add(2);
    set.add(01);
    Integer[] ds = (Integer[]) set.toArray(new Integer[0]);
    for (int i = 0; i < ds.length; i++) {
        System.out.println(ds[i].toString());

    }

}
```

TreeSet的自然排序 ch07.set.TreeSetDemo2

```
public static void main (String[] args) {
    // TODO Auto-generated method stub
    Set set = new TreeSet();
    set.add(new Student("01", "zhang"));
    set.add(new Student("03", "zhang"));
    set.add(new Student("02", "zhang"));
    Object[] ds =   set.toArray( );
    for (int i = 0; i < ds.length; i++) {
        System.out.println(ds[i].toString());
    }

}
```

Set自定义排序 Comparator 泛型以后讲解

遍历Set中数据的方法 1

```
Set set=new TreeSet();
set.add(new Integer(0));.....
int sum=0;
Integer[] list=(Integer[])set.toArray(new Integer[0]);
for(int i=0;i<list.length;i++){
   sum=sum+list[i];
}
```

遍历Set中数据的方法 2

```
Set set=new TreeSet();
set.add(new Integer(0));.....
int sum=0;
Object[] list=set.toArray();
for(int i=0;i<list.length;i++){
   sum=sum+(Integer)list[i];
}
```

Set用在什么地方？

- 保存/统计不重复的对象

  判断一个文本有多少不同字符

  判断一个文本有多少不同字符单词、汉字

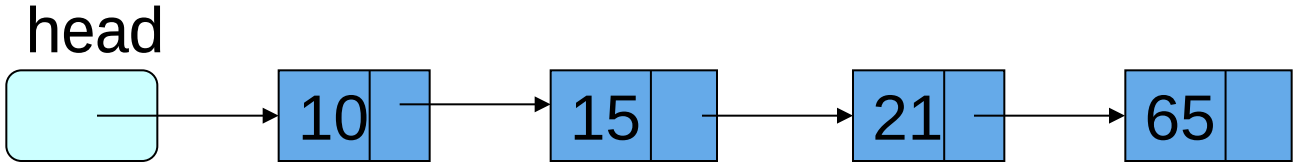
  判断一个对象是否已经存在了

  ...
- 使用TreeSet还可以快速排序

  例子：ch07.set.TreeSetUsage

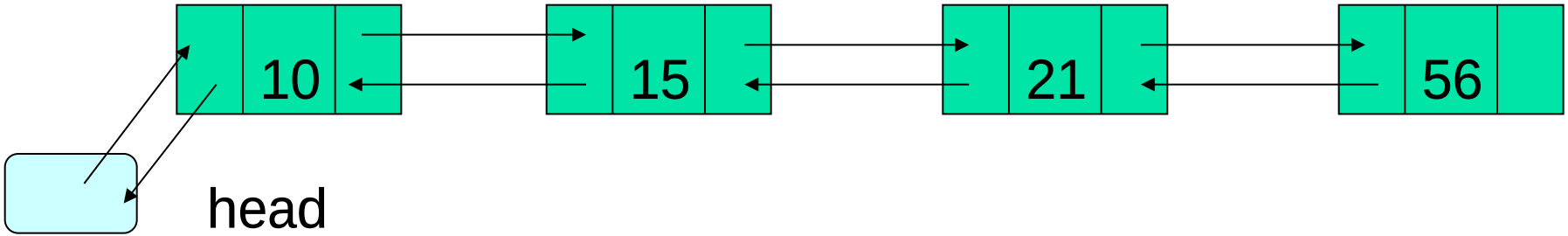
List(列表)

- List的主要特征是按照元素以线性方式存储，允许集合中存放重复对象。List接口的主要实现类包括：ArrayList和LinkedList。
- ArrayList：代表长度可以变化的数组。允许对元素进行快速的随机访问，但是向ArrayList中插入与删除元素速度较慢。



List(列表)

- LinkedList：双向链表。向LinkedList中插入和删除元素的速度较快，随机访问的速度较慢。它单独具有addFirst() 、addLast()、getFirst()、getLast()、removeFirst()、removeLast()方法。



访问List中的元素

List中的对象按照索引位置排序，程序可以按照对象在集合中的索引位置来检索对象。

```
List   list= new   ArrayList();
list.add(new Integer(3));
list.add(new Integer(4));
list.add(new Integer(3));
list.add(new Integer(2));
for(int i=0;i<list.size();i++){
      System.out.print(list.get(i)+” ”);
}
```

输出：3 4 3 2

List特性

- List保持原来加入时候的顺序
- List中可以有重复元素

List本身不带有排序功能

- List本身不带有排序的功能。
- Collections类是对Java集合类库中的辅助类，它提供操纵集合的各种静态方法。
- Collections.sort(List list):对List中对象进行自然排序。

为List排序1 ch07.list. SortList

```
public static void main(String[] args) {
    // TODO Auto-generated method stub
    List  list=new ArrayList ();
    list.add(12);
    list.add(34);
    list.add(11);
    Collections.sort(list);
    for(int i=0;i<list.size();i++) {
        System.out.println(list.get(i));
    }
}
```

Vector

- Vector类可以实现动态的对象数组。几乎与ArrayList相同。
- 由于Vector在各个方面都没有突出的性能，所以现在已经不提倡使用。

List的用途

- 按顺序存放元素，比数组更方便

关于Iterator

Iterator是神马？

```
//不考虑泛型的Iterator定义
public interface Iterator  {
    boolean hasNext();
    Object next();
    void remove();
}
public interface Collection extends Iterable;
public interface Iterable {
    Iterator iterator();
}
```

Collection和Iterator接口

- 在Collection接口中声明了适用于Java集合(只包括Set和List)的通用方法。因此Set和List对象可以调用以上方法，Map对象不可以。
- Iterator接口隐藏了底层集合的数据结构，向客户程序提供了遍历各种数据集合的统一接口。
- 如果集合中的元素没有排序，Iterator遍历集合中元素的顺序也是无序的。

遍历Set中数据的方法 3

```
Set set=new TreeSet();
set.add(new Integer(0));.....
int sum=0;
Iterator  it=set.iterator();
 while(it.hasNext()){
      sum=sum+(Integer) it.next();
}
```

使用Iterator遍历Set/List

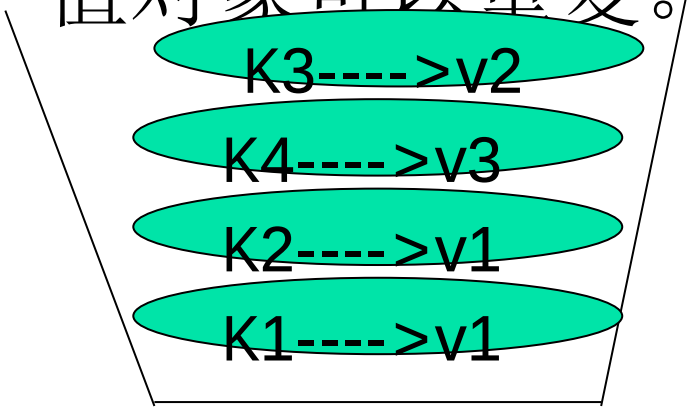
- 例子：ch07.it. IteratorDemo

遍历Set三种方法对比

```
Integer[] list=(Integer[])set.toArray(new Integer[0]);
for(int i=0;i<list.length;i++){
  sum=sum+list[i]; } //方法1
-----------------------------------------------
Object[] list=set.toArray();
for(int i=0;i<list.length;i++){
  sum=sum+(Integer)list[i]; } //方法2
-----------------------------------------------
Iterator  it=set.iterator();
while(it.hasNext()){
      sum=sum+(Integer) it.next(); }// 方法3
```

Map(映射)

- Map(映射)：集合中的每一个元素包含一对键对象和值对象，集合中没有重复的键对象，值对象可以重复。



Map(映射)

- 向Map集合中加入元素时，必须提供一对键对象和值对象。
- Map的两个主要实现类：HashMap和TreeMap。

Map(映射)

- Map最基本的用法，就是提供类似字典的能力。
- 在Map中检索元素时，只要给出键对象，就会返回值对象。

```
Map  map=new HashMap();
map.put("1","Monday");
map.put("one","Monday");
map.put("2","Tuesday");
map.put("3","Wensday");

System.out.println(map.get(“1"));
System.out.println(map.get("one"));
```

Map的遍历：

- 通过Set.keySet() 返回键的集合来遍历。
- 例如：ch07.map.MapIterator
- 通过Set.entrySet() 返回“键值对”的集合来遍历。
- Map.Entry的对象代表一个“词条”，就是一个键值对。可以从中取值getValue ()或键getKey()。
- 例如：ch07.map.MapIterator2

Map两个最重要实现

- HashMap按照哈希算法来存取键值对象。
- TreeMap按照排序规则对keySet进行排序。

Map的遍历不保证顺序，但是TreeMap遍历使用Key排序

例如：ch07.map.HashMapTreeMap

```
public static void main(String[] args) {
    testMap(new HashMap());
    System.out.println("======================");
    testMap(new TreeMap());
}
public static void testMap(Map map) {
    map.put("zhang", 10);
    map.put("li", 20);
    map.put("wan", 30);
    map.put("an", 40);
    Iterator it = map.keySet().iterator();
    while (it.hasNext()) {
        Object key = it.next();
        Object value = map.get(key);
        System.out.println(key + " " + value);
    }
}
```

泛型 Generic

- 回顾Set的遍历

```
set.add("1");
Integer[] list=(Integer[])set.toArray(new Integer[0]);

Object[] list=set.toArray();
sum=sum+(Integer)list[i];
Iterator  it=set.iterator();
sum=sum+(Integer) it.next();.....
map.put("1",1);
Iterator  it=map.keyset().iterator();
while(it.hasNext()){
String  key=(String)it.next();
String  value=(String)map.get(key);
```

这些错误在编译的时候是无法检查出来的

运行时异常

```
Set set=new HashSet();
  set.add(new StringBuffer("Mike"));
  return set;
}
public void printSet(Set set){
Iterator it=set.iterator();
while(it.hasNext()){
  String str=(String)it.next();
  //编译时不报错，运行会抛出ClassCastException
  System.out.println(str);
}
```

- 对于软件来说，错误发现的越早，越能降低软件开发和维护的成本。从而提高软件的健壮性。
- 为了做到这一点，在JDK版本升级过程中，致力于把一些运行时异常转变为编译时错误。
- 在JDK1.5版本中，引入泛型的概念，它把ClassCastException运行时异常转变为编译时类型不匹配错误。

泛型的例子

```
//集合中元素必须为String类型
Set<String>  set=new HashSet<String>();
set.add("Tom");                          //合法
//编译出错，类型不匹配
set.add(new StringBuffer("Mike"));
```

```
//元素必须为String
List<String>  list=new ArrayList<String>();
list.add("Tom");                          //合法
String  name=list.get(0);        //合法，无需强制类型转换
Object  obj=list.get(0);         //合法，允许向上转型
//编译出错，无法类型转换
StringBuffer  sb=(StringBuffer)list.get(0);
```

泛型的例子

```
Map<String,String> map=new HashMap<String,String>();
Iterator<String> it=map.keySet().iterator();
while(it.hasNext()){
 String key=it.next(); String value=map.get(key);
}
Set<String> set=new HashSet<String> ();
Iterator<String> it=set.iterator();
while(it.hasNext()){
 String key=it.next();
}
```

如何定义一个支持泛型的类

```
public class MySet<T> {
        T t;
        public MySet() {
        }
        public void set(T t){
                this.t=t;
        }
        public T get(){
                return t;
        }
}
```

支持泛型的类

```
public class MyMap<K,V> {
private K k;
private V v;
public MyMap() {}
public void put(K k,V v){
this.k=k; this.v=v;
}
public K getK(){return k;}
public V getV(){return v;}
}
```

泛型的使用

```
Map<String,Integer> map=new HashMap<String, Integer>();
Iterator<String> it=map.keySet().iterator();
map.put(“a”,1);
while(it.hasNext()){
 String key=it.next(); Integer value=map.get(key);
}
Set<String> set=new HashSet<String> ();
Iterator<String> it=set.iterator();
while(it.hasNext()){
 String key=it.next();
}
```

泛型的优点

取消了强制类型转换;

在编译时候发现问题;

```
List list=new ArrayList();
list.put(5);//编译不报错
String s=(String)list.get(0);//运行报错

List<String> list=new ArrayList<String>();
list.put(5); //编译报错
String s=list.get(0);
```

java中使用泛型的例子

```
public interface Iterator<E> {
    E next();.....
public interface Iterable<T> {
    Iterator<T> iterator();.....
public interface Collection<E> extends Iterable<E> {
    boolean add(E e);.....
public interface List<E> extends Collection<E>
    boolean add(E e);.....
public class ArrayList<E> extends AbstractList<E>
        implements List<E>, RandomAccess, Cloneable, java.io.Serializable{
    public boolean add(E e) {...}...
```

泛型结束后，重新考虑自定义排序问题
关于Comparator

- java.util.Comparator<Type>接口提供具体的排序方式，<Type>指定被比较的对象的类型，Comparator有个compare(Type x,Type y)方法，用于比较两个对象的大小。
- compare(x, y)的返回值为0，则表示x和y相等，如果返回值大于0，则表示x大于y，如果返回值小于0，则表示x小于y。

```
public interface Comparator<T> {
    int compare(T o1, T o2);
}
```

使用Comparator排序

- **ch07.** comparator**.** SortListDemoComparator
- ch07. comparator. TreeSetDemoComparator

排序的实现方法汇总

方法1 对象实现comparable接口

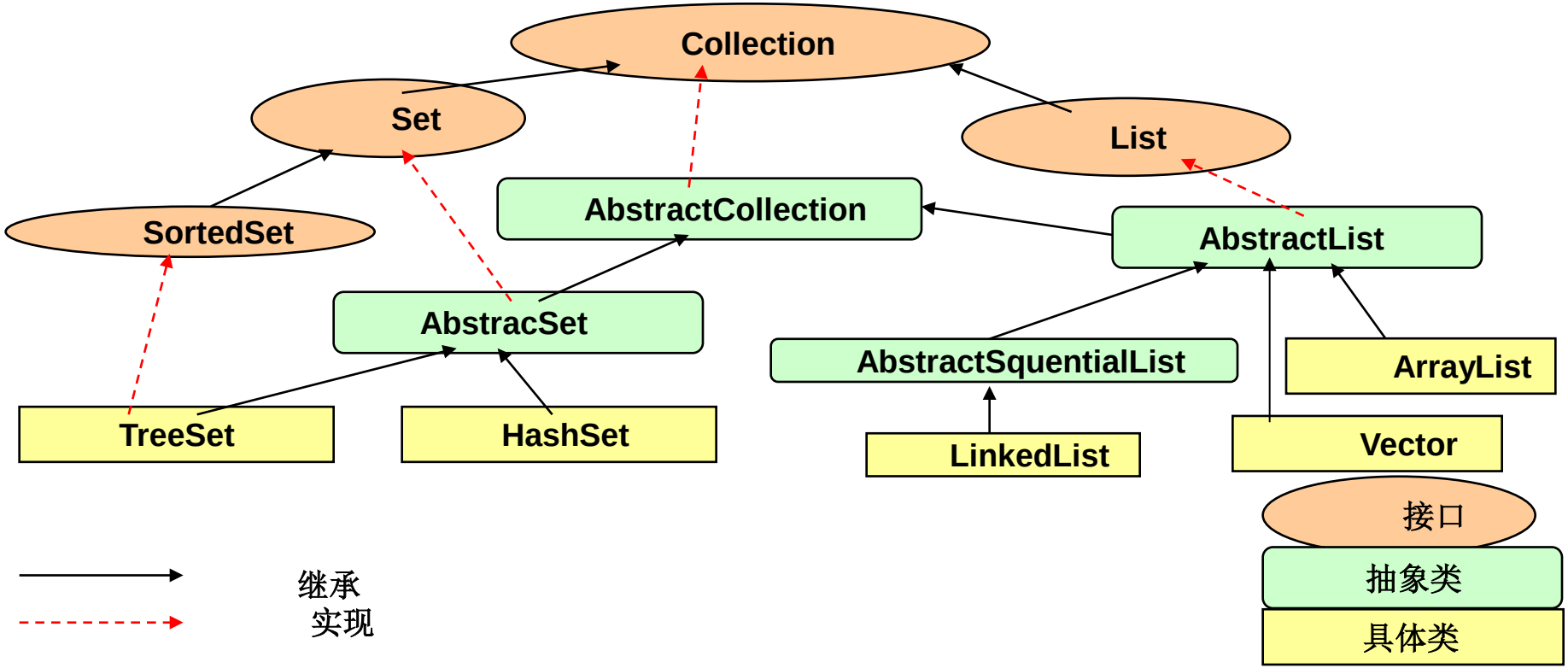
```
public interface Comparable<T> {
public int compareTo(T o);
}
```

方法2 自定义comparator

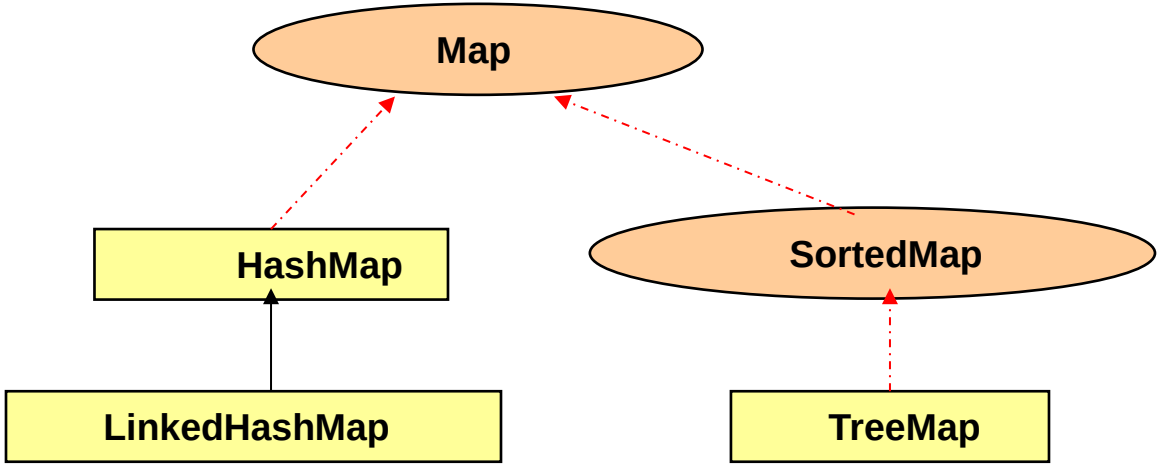
```
public interface Comparator<T> {
int compare(T o1, T o2);
.....
}
```

List/Set/Map总结

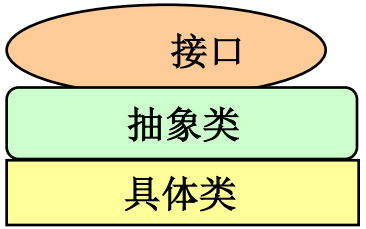
Java集合类框图


Collection
Set
List
SortedSet
AbstractCollection
AbstractList
AbstracSet
AbstractSquentialList
ArrayList
TreeSet
HashSet
LinkedList
Vector
接口
抽象类
具体类
继承
实现

Java集合类框图


Map
HashMap
SortedMap
LinkedHashMap
TreeMap


继承
实现


接口
抽象类
具体类

Collection 方法	描述
public boolean add(E e)	向集合中增加元素
public boolean addAll(Collection<? Extends E>c)	向集合中加入一组数据,泛型指定了操作上限
public void clear()	清空所有的内容
public boolean contains(Object o)	判断是否有指定的内容查找
public boolean containsAll(Collection<?> c)	查找一组数据是否存在
public boolean equals(Object o)	对象比较
public int hashCode()	返回hash码
public boolean isEmpty()	判断集合的内容是否为空
public iterator()<E>interator()	为interrator接口实例化 迭代输出
public boolean remove(Object o)	从集合中删除指定的对象
boolean removeAll(Collection<?> c)	从集合中删除一组对象
public int size()	取得集合的长度
public Object toArray()	取得全部的内容,以数组的形式返回

List 在Collection中增加方法	描述
public void add(int index, E element)	在指定的位置处加入一个元素
public boolean addAll(Collection<? extends E> c)	在指定的位置加入一组元素
public E get(int index)	通过索引位置可以取出每个元素
public ListIterator<E> listIterator()	为ListIterator接口实例化
public E remove(int index)	删除指定位置的内容
public E set(int index, E element)	修改指定位置的内容
public List<E> subList(int fromIndex, int toIndex)	截取自集合

Map方法	描述
V put(K key, V value)	增加内容
V get(Object key)	取得内容 很据**key**取内容
boolean containsKey(Object key)	查找指定的**key**是否存在
boolean containsValue(Object value)	查找指定的**value**是否存在
boolean isEmpty()	判断集合是否为空
Set<K> keySet()	将全部的**key**变为**set**集合
Collection<V> values()	将全部**value**变为**collection**集合
V remove(Object key)	根据**key**删除内容
void putAll(Map<? extends K,? extends V> m)	增加一组数据

总结

- HashSet：如果集合中对象所属的类重新定义了equals()方法，那么这个类也必须重新定义hashCode()方法，并且保证当两个对象用equals()方法比较的结果为true时，这两个对象的hashCode()方法的返回值相等。
- HashMap要求其中的key同样具有上述属性

- TreeSet：如果对集合中的对象进行自然排序，则要求对象所属的类实现Comparable接口，并且保证这个类的compareTo()和equals()方法采用相同的比较规则来比较两个对象是否相等。

- HashMap：如果集合中键对象所属的类重新定义了equals()方法，那么这个类也必须重新定义hashCode()方法，并且保证当两个键对象用equals()方法比较的结果为true时，这两个键对象的hashCode()方法的返回值相等。

- TreeMap:如果对集合中的键对象进行自然排序，则要求键对象所属的类实现Comparable接口，并且保证这个类的compareTo()和equals()方法采用相同的比较规则来比较两个键对象是否相等。

- 为了增强程序的健壮性，在编写Java程序时要养成以下良好习惯：
- 如果Java类重新定义了equals()方法，那么这个类也必须重新定义hashCode()方法，并且保证当两个对象用equals()方法比较的结果为true时，这两个对象的hashCode()方法的返回值相等。

- 如果Java类实现了Comparable接口，那么就应该重新定义compareTo()、equals()方法和hashCode()方法，保证compareTo()和equals()方法采用相同的比较规则来比较两个对象是否相等，并且保证当两个对象用equals()方法比较的结果为true时，这两个对象的hashCode()方法的返回值相等。

- HashMap和HashSet具有较好的性能，是Map和Set的首选实现类。只有在排序的场合，才考虑使用TreeSet和TreeMap。LinkedList和ArrayList各有优点，如果经常对元素进行插入和删除操作，那么可以用LinkedList，如果经常随机访问元素，那么可以用Arraylist。

思考

- 如果定义了一个类，其hashCode()方法统统返回0，这个类在HashSet中或者HashMap的key，会发生什么现象？？
- 如果定义了一个类，其hashCode()方法统统返回0，这个类在HashSet中或者HashMap的key，会发生什么现象？？