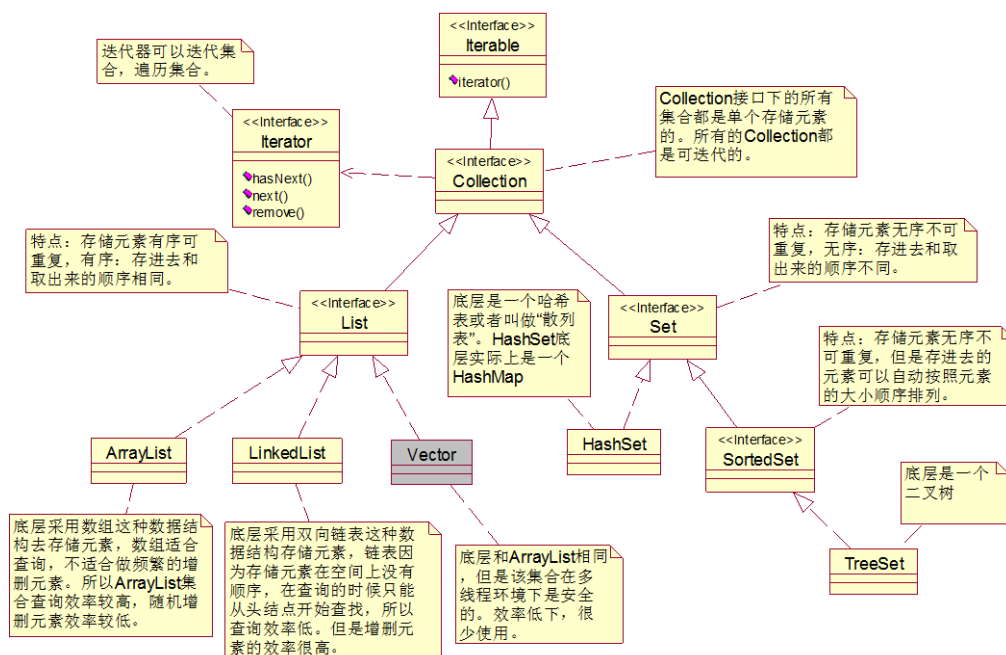


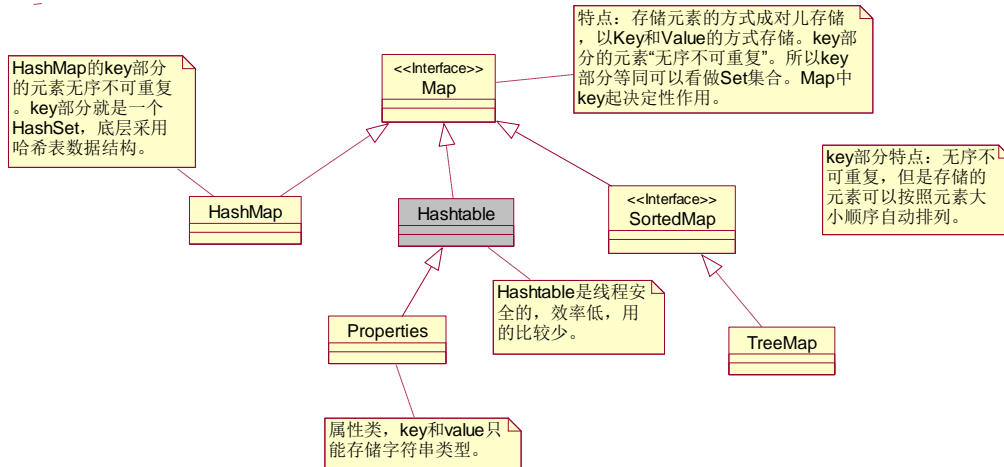
1. 纲要

- 主要集合概述
- Collection 和 Iterator
- List
- Set
- Map
- Collections 工具类
- Comparable 与 Comparator

集合继承结构图_Collection 部分



集合继承结构图_Map 部分

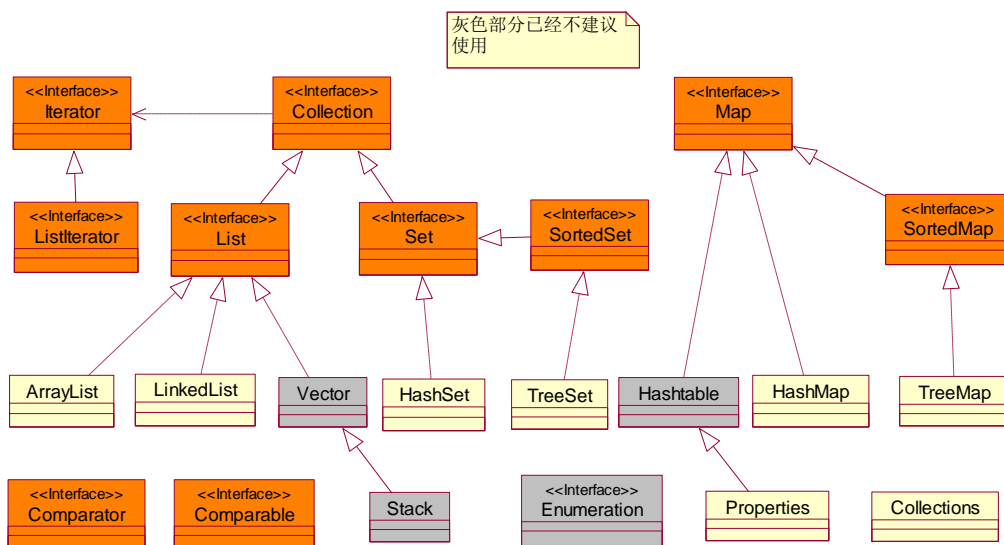


2. 内容

1.1 主要集合概述

Java 集合主要有 3 种重要的类型：

- List: 是一个有序集合，可以放重复的数据
- Set: 是一个无序集合，不允许放重复的数据
- Map: 是一个无序集合，集合中包含一个键对象，一个值对象，键对象不允许重复，值对象可以重复（身份证号—姓名）



1.2 Collection 和 Iterator

Collection 是 List 和 Set 的父接口，在 Collection 中定义了一些主要方法

boolean	<u>add</u> (<u>E</u> o)	确保此 collection 包含指定的元素（可选操作）。
boolean	<u>addAll</u> (<u>Collection</u> <? extends <u>E</u> > c)	将指定 collection 中的所有元素都添加到此 collection 中（可选操作）。
void	<u>clear</u> ()	移除此 collection 中的所有元素（可选操作）。
boolean	<u>contains</u> (<u>Object</u> o)	如果此 collection 包含指定的元素，则返回 true。
boolean	<u>containsAll</u> (<u>Collection</u> <?> c)	如果此 collection 包含指定 collection 中的所有元素，则返回 true。
boolean	<u>equals</u> (<u>Object</u> o)	比较此 collection 与指定对象是否相等。
int	<u>hashCode</u> ()	返回此 collection 的哈希码值。
boolean	<u>isEmpty</u> ()	如果此 collection 不包含元素，则返回 true。
<u>Iterator</u> < <u>E</u> >	<u>iterator</u> ()	返回在此 collection 的元素上进行迭代的迭代器。
boolean	<u>remove</u> (<u>Object</u> o)	从此 collection 中移除指定元素的单个实例，如果存在的话（可选操作）。
boolean	<u>removeAll</u> (<u>Collection</u> <?> c)	移除此 collection 中那些也包含在指定 collection

	中的所有元素（可选操作）。
boolean	retainAll (Collection <?> c) 仅保留此 collection 中那些也包含在指定 collection 的元素（可选操作）。
int	size () 返回此 collection 中的元素数。
Object []	toArray () 返回包含此 collection 中所有元素的数组。
<T> T[]	toArray (T[] a) 返回包含此 collection 中所有元素的数组；返回数组的运行时类型与指定数组的运行时类型相同。

关于 `Iterator` 接口说明，`Iterator` 称为迭代接口，通过此接口可以遍历集合中的数据，此接口主要方法为：

boolean	hasNext () 如果仍有元素可以迭代，则返回 true。
E next ()	返回迭代的下一个元素。

1.3 List 接口

1.3.1 List 接口概述

List 接口下面主要有两个实现 `ArrayList` 和 `LinkedList`，他们都是有顺序的，也就是放进去是什么顺序，取出来还是什么顺序，也就是基于线性存储，可以看作是一个可变数组

- `ArrayList`：查询数据比较快，添加和删除数据比较慢(基于可变数组)
- `LinkedList`：查询数据比较慢，添加和删除数据比较快（基于链表数据结构）
- `Vector`:`Vector` 已经不建议使用，`Vector` 中的方法都是同步的，效率慢，已经被 `ArrayList` 取代
- `Stack` 是继承 `Vector` 实现了一个栈，栈结构是后进先出，目前已经被 `LinkedList` 取代

```
import java.util.*;

public class ArrayListTest01 {

    public static void main(String[] args) {

        //最好不要这样写，这样属于面向具体编程了
        //无法达到灵活互换
        //最好面向接口编程
        ArrayList arrayList = new ArrayList();

        //采用面向接口编程
        //使用 Collection 会更灵活，如果 List 不能满足要求
        //那么可以采用 HashSet，因为 HashSet 也实现了该接口
        Collection c = new ArrayList();

        //面向接口编程
        //采用 list 接口可以使用 Collection 里的方法
        //也可以使用 list 接口扩展的方法
        List l = new ArrayList();

        //自动装箱，适合于 jdk1.5
        l.add(1);
        l.add(3);

        //jdk1.5 以前，必须如下使用
        l.add(new Integer(2));
        l.add(new Integer(4));

        //可以加入重复数据
```

```
l.add(2);

//不能加入字符串
//在强制转换时会出现 ClassCastException 错误
//l.add("sadsfs");

//可以采用 List 接口中的 get()方法依次取得元素
//输出结果为,不会打乱顺序
/*
    1
    3
    2
    4
    2
*/
for (int i=0;i<l.size();i++) {
    //将 Object 强制转换为 Integer
    Integer e = (Integer)l.get(i);
    System.out.println(e);
}
System.out.println("");

//调用 remove 删除集合中的元素
//如果元素重复会 remove 掉第一个匹配的
l.remove(2);

//采用 Iterator 遍历数据 (while 循环)
//Iterator 是一种模式,主要可以统一数据结构的访问方式
//这样在程序中就不用关心各个数据结构的实现了
//使对不同数据结构的遍历更加简单了,更加统一了
```

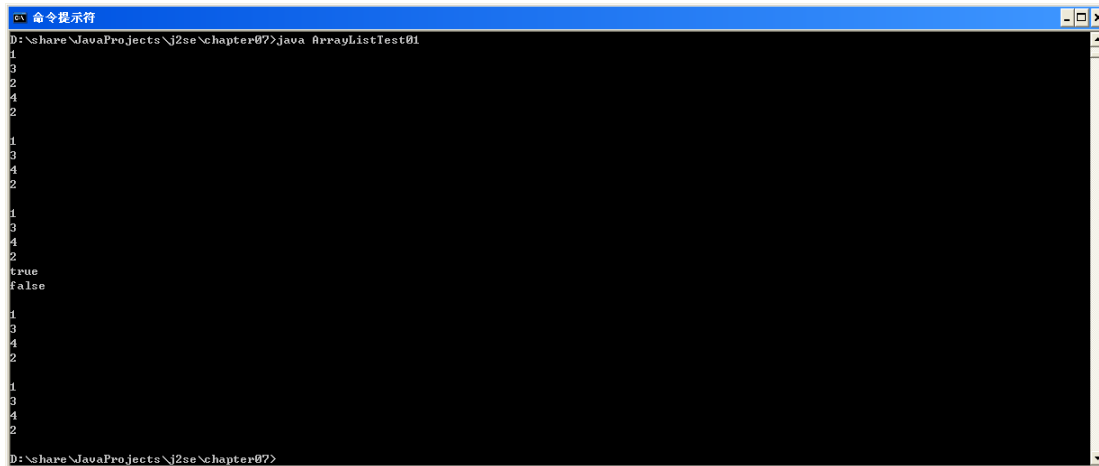
```
Iterator iter = l.iterator();
while (iter.hasNext()) {
    Integer v = (Integer)iter.next();
    System.out.println(v);
}
System.out.println("");
//采用 Iterator 遍历数据 (for 循环)
for (Iterator iter1=l.iterator(); iter1.hasNext();) {
    Integer v = (Integer)iter1.next();
    System.out.println(v);
}

//在集合中是否包含 3，输出为： true
System.out.println(l.contains(3));

//集合是否为空，输出： false
System.out.println(l.isEmpty());

System.out.println("");
//转换成对象数组
Object[] oArray1 = l.toArray();
for (int i=0; i<oArray1.length; i++) {
    Integer v = (Integer)oArray1[i];
    System.out.println(v);
}
System.out.println("");
//运行时自动创建相应类型的数组
Integer[] iArray = new Integer[l.size()];
l.toArray(iArray);
for (int i=0; i<iArray.length; i++) {
```

```
        int v = iArray[i];
        System.out.println(v);
    }
}
```



```
命令提示符
D:\share\JavaProjects\j2se\chapter07>java ArrayListTest01
1
3
2
4
2
1
3
4
2
2
true
false
1
3
4
2
1
3
4
2
D:\share\JavaProjects\j2se\chapter07>
```

1.3.2 LinkedList

用法同 ArrayList

```
import java.util.*;

public class LinkedListTest01 {

    public static void main(String[] args) {

        //最好不要这样写，这样属于面向具体编程了
        //无法达到灵活互换
        //最好面向接口编程
        LinkedList arrayList = new LinkedList();

        //采用面向接口编程
```


换

```
//使用 Collection 会更灵活，如果 List 不能满足要求
//那么可以采用 HashSet，因为 HashSet 也实现了该接口
Collection c = new LinkedList();

//面向接口编程
//采用 list 接口可以使用 Collection 里的方法
//也可以使用 list 接口扩展的方法
//List l = new ArrayList();
//因为 LinkedList 和 ArrayList 都实现了 List 接口，所以我们可以灵活互

//直接修改为 LinkedList,对我们的程序没有任何影响
List l = new LinkedList();

//自动装箱，适合于 jdk1.5
l.add(1);
l.add(3);

//jdk1.5 以前，必须如下使用
l.add(new Integer(2));
l.add(new Integer(4));

//可以加入重复数据
l.add(2);

for (int i=0;i<l.size();i++) {
    Integer e = (Integer)l.get(i);
    System.out.println(e);
}
System.out.println("");
```

```
l.remove(2);

Iterator iter = l.iterator();
while (iter.hasNext()) {
    Integer v = (Integer)iter.next();
    System.out.println(v);
}
System.out.println("");
for (Iterator iter1=l.iterator(); iter1.hasNext();) {
    Integer v = (Integer)iter1.next();
    System.out.println(v);
}

System.out.println(l.contains(3));

System.out.println(l.isEmpty());

System.out.println("");

Object[] oArray1 = l.toArray();
for (int i=0; i<oArray1.length; i++) {
    Integer v = (Integer)oArray1[i];
    System.out.println(v);
}
System.out.println("");

Integer[] iArray = new Integer[l.size()];
l.toArray(iArray);
for (int i=0; i<iArray.length; i++) {
    int v = iArray[i];
```

```
        System.out.println(v);
    }
}
}
```

修改为 HashSet 实现类，重点了解面向接口编程的好处

```
import java.util.*;

public class LinkedListTest02 {

    public static void main(String[] args) {

        //采用面向接口编程
        //使用 Collection 会更灵活，如果 List 不能满足要求
        //那么可以采用 HashSet，因为 HashSet 也实现了该接口
        //Collection c = new LinkedList();
        //可以修改为 HashSet
        Collection c = new HashSet();

        //不能改为 HashSet，因为 HashSet 不是 List 产品
        //List l = new HashSet();

        //自动装箱，适合于 jdk1.5
        c.add(1);
        c.add(3);

        //jdk1.5 以前，必须如下使用
        c.add(new Integer(2));
        c.add(new Integer(4));
    }
}
```

```
//可以加入重复数据
c.add(2);

/*
for (int i=0;i<c.size();i++) {
    //不能采用 get， 因为 get 是 List 接口扩展的
    //父类不能看到子类扩展的功能
    //反过来子类可以看到父类的功能， 因为子类继承了父类
    Integer e = (Integer)c.get(i);
    System.out.println(e);
}
*/
System.out.println("");

Iterator iter = c.iterator();
while (iter.hasNext()) {
    Integer v = (Integer)iter.next();
    System.out.println(v);
}
System.out.println("");
for (Iterator iter1=c.iterator(); iter1.hasNext();) {
    Integer v = (Integer)iter1.next();
    System.out.println(v);
}

System.out.println(c.contains(3));

System.out.println(c.isEmpty());

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

```
Object[] oArray1 = c.toArray();
for (int i=0; i<oArray1.length; i++) {
    Integer v = (Integer)oArray1[i];
    System.out.println(v);
}
System.out.println("");

Integer[] iArray = new Integer[c.size()];
c.toArray(iArray);
for (int i=0; i<iArray.length; i++) {
    int v = iArray[i];
    System.out.println(v);
}
}
```

1.4 Set 接口

1.4.1 哈希表

哈希表是一种数据结构，哈希表能够提供快速存取操作。哈希表是基于数组的，所以也存在缺点，数组一旦创建将不能扩展。

正常的数组，如果需要查询某个值，需要对数组进行遍历，只是一种线性查找，查找的速度比较慢。如果数组中的元素值和下标能够存在明确的对应关系，那么通过数组元素的值就可以换算出数据元素的下标，通过下标就可以快速定位数组元素，这样的数组就是哈希表。一张哈希表：

元素值	10	11	12	13	14	15	16	17	18
元素下标	0	1	2	3	4	5	6	7	8

以上我们的示例元素值和下标的关系为：

元素下标=元素值-10，此时的示例 hashCode 就是和数组下标一致了,取得 hashCode 方法如下：

```
//取得 hashCode  
public int hashCode(int value) {  
    return value - 10;  
}
```

有了 hashCode 后，我们就可以快速的定位相应的元素，查找到相应的信息

1.4.2 HashSet

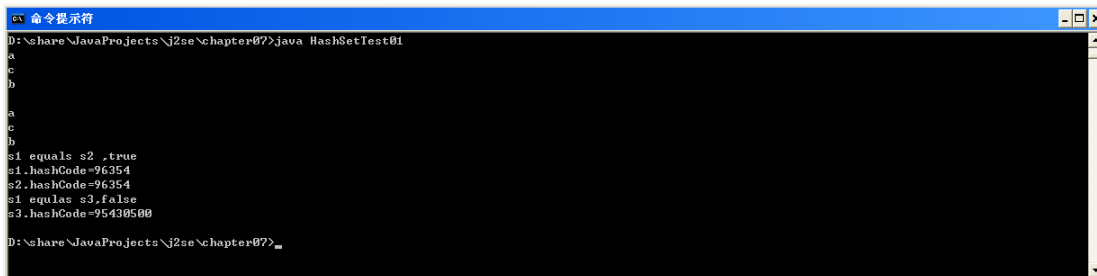
HashSet 中的数据是无序的不可重复的。HashSet 按照哈希算法存取数据的，具有非常好性能，它的工作原理是这样的，当向 HashSet 中插入数据的时候，他会调用对象的 hashCode 得到该对象的哈希码，然后根据哈希码计算出该对象插入到集合中的位置。

```
import java.util.*;  
  
public class HashSetTest01 {  
  
    public static void main(String[] args) {  
  
        //它是无序的，不重复  
        Set set = new HashSet();  
        set.add("a");  
        set.add("b");  
        set.add("c");  
        //输出是无序的  
        for (Iterator iter=set.iterator(); iter.hasNext();) {  
            System.out.println(iter.next());  
        }  
        //加入重复数据  
        set.add("a");  
        System.out.println("");  
        for (Iterator iter=set.iterator(); iter.hasNext();) {
```

```
        System.out.println(iter.next());
    }
    String s1 = "abc";
    String s2 = "abc";
    System.out.println("s1 equals s2 ," + s1.equals(s2));

    //equals 相等，hashCode 一定是相等的
    System.out.println("s1.hashCode=" + s1.hashCode());
    System.out.println("s2.hashCode=" + s2.hashCode());

    String s3 = "dddd";
    System.out.println("s1 equals s3," + s1.equals(s3));
    System.out.println("s3.hashCode=" + s3.hashCode());
}
}
```



```
命令提示符
D:\share\JavaProjects\j2se\chapter07>java HashSetTest01
a
b
c
a
c
b
s1 equals s2 ,true
s1.hashCode=96354
s2.hashCode=96354
s1 equals s3,false
s3.hashCode=95430500
D:\share\JavaProjects\j2se\chapter07>
```

1.4.3 equals 和 hashCode

```
import java.util.*;

public class HashSetTest02 {

    public static void main(String[] args) {
        Person p1 = new Person();
        p1.name = "张三";
```

```
p1.age = 20;

Person p2 = new Person();
p2.name = "李四";
p2.age = 30;

Person p3 = new Person();
p3.name = "张三";
p3.age = 40;

Set set = new HashSet();
set.add(p1);
set.add(p2);
set.add(p3);

for (Iterator iter=set.iterator(); iter.hasNext();) {
    Person p = (Person)iter.next();
    System.out.println("name=" + p.name + ", age=" + p.age);
}

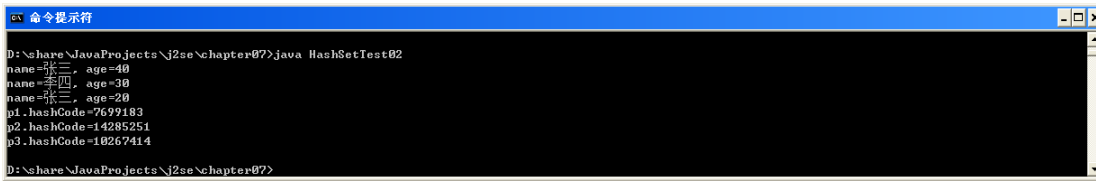
System.out.println("p1.hashCode=" + p1.hashCode());
System.out.println("p2.hashCode=" + p2.hashCode());
System.out.println("p3.hashCode=" + p3.hashCode());
}
}

class Person {

    String name;
```



```
int age;  
}
```



```
D:\share\JavaProjects\j2se\chapter07>java HashSetTest02  
name=张三, age=40  
name=李四, age=30  
name=张三, age=20  
p1.hashCode=7699183  
p2.hashCode=14285251  
p3.hashCode=10267414  
D:\share\JavaProjects\j2se\chapter07>
```

加入了重复的数据，因为 hashCode 是不同的，所以会根据算出不同的位置，存储格式

Person{张三, 20}	Person{李四, 30}	Person{张三, 40}
7699183	14285251	10267414

进一步完善，覆盖 equals

```
import java.util.*;  
  
public class HashSetTest03 {  
  
    public static void main(String[] args) {  
        Person p1 = new Person();  
        p1.name = "张三";  
        p1.age = 20;  
  
        Person p2 = new Person();  
        p2.name = "李四";  
        p2.age = 30;  
  
        Person p3 = new Person();  
        p3.name = "张三";  
        p3.age = 40;  
  
        System.out.println("p1 equals p2," + p1.equals(p2));  
        System.out.println("p1 equals p3," + p1.equals(p3));  
    }  
}
```

```
Set set = new HashSet();
set.add(p1);
set.add(p2);
set.add(p3);

for (Iterator iter=set.iterator(); iter.hasNext();) {
    Person p = (Person)iter.next();
    System.out.println("name=" + p.name + ", age=" + p.age);
}

System.out.println("p1.hashCode=" + p1.hashCode());
System.out.println("p2.hashCode=" + p2.hashCode());
System.out.println("p3.hashCode=" + p3.hashCode());
}
}

class Person {

    String name;

    int age;

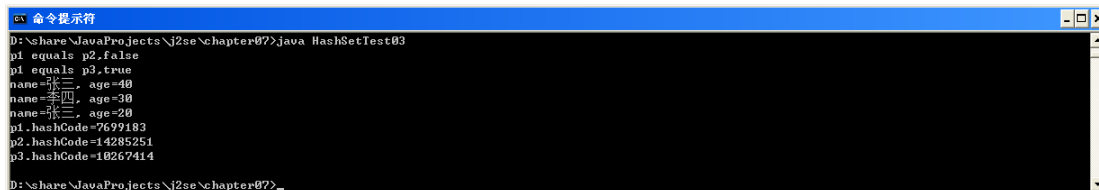
    //覆盖 equals
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj instanceof Person) {
            Person p = (Person)obj;
```

```

        return this.name.equals(p.name);
    }

    return false;
}
}

```



```

C:\ 命令提示符
D:\share\JavaProjects\j2se\chapter07\java HashSetTest03
p1 equals p2.false
p1 equals p3.true
name=张三, age=40
name=李四, age=30
name=张三, age=20
p1.hashCode=7699183
p2.hashCode=14285251
p3.hashCode=10267414
D:\share\JavaProjects\j2se\chapter07>

```

以上仍然存在重复数据，在 Person 中覆盖了 hashCode 方法，能够正确的比较出两个 Person 是相等的还是不等的，但是为什么 HashSet 中还是放入了重复数据？因为 Person 对象的 hashCode 不同，所以它就换算出了不同的位置，让后就会把相关的值放到不同的位置上，就忽略 equals，所以我们必须覆盖 hashCode 方法

Person{ 张三, 20}	Person{ 李四, 30}	Person{ 张三, 40}
7699183	14285251	10267414

【代码示例】，只覆盖 hashCode，不覆盖 equals

```

import java.util.*;

public class HashSetTest04 {

    public static void main(String[] args) {

        Person p1 = new Person();
        p1.name = "张三";
        p1.age = 20;

        Person p2 = new Person();
        p2.name = "李四";
        p2.age = 30;
    }
}

```

```
Person p3 = new Person();
p3.name = "张三";
p3.age = 40;

System.out.println("p1 equals p2," + p1.equals(p2));
System.out.println("p1 equals p3," + p1.equals(p3));

Set set = new HashSet();
set.add(p1);
set.add(p2);
set.add(p3);

for (Iterator iter=set.iterator(); iter.hasNext();) {
    Person p = (Person)iter.next();
    System.out.println("name=" + p.name + ", age=" + p.age);
}

System.out.println("p1.hashCode=" + p1.hashCode());
System.out.println("p2.hashCode=" + p2.hashCode());
System.out.println("p3.hashCode=" + p3.hashCode());
}
}

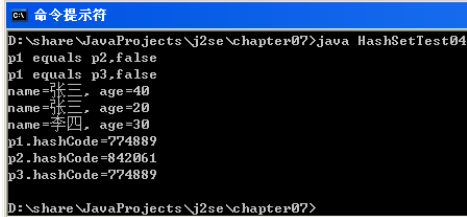
class Person {

    String name;

    int age;

    //覆盖 hashCode
```

```
public int hashCode() {  
    return (name==null)?0:name.hashCode();  
}  
}
```



```
命令提示符  
D:\share\JavaProjects\j2se\chapter07>java HashSetTest04  
p1 equals p2.false  
p1 equals p3.false  
name=张三, age=40  
name=张三, age=20  
name=李四, age=30  
p1.hashCode=774889  
p2.hashCode=842061  
p3.hashCode=774889  
D:\share\JavaProjects\j2se\chapter07>
```

以上示例，张三的 hashCode 相同，当两个对象的 equals 不同，所以认为值是不一样的，那么 java 会随机换算出一个新的位置，放重复数据

Person{ 张三, 20}	Person{ 李四, 30}	Person{ 张三, 40}
774889-1	14285251	774889-2

【代码示例】，进一步改善，覆盖 equals，覆盖 hashCode

```
import java.util.*;  
  
public class HashSetTest05 {  
  
    public static void main(String[] args) {  
        Person p1 = new Person();  
        p1.name = "张三";  
        p1.age = 20;  
  
        Person p2 = new Person();  
        p2.name = "李四";  
        p2.age = 30;  
  
        Person p3 = new Person();  
        p3.name = "张三";  
        p3.age = 40;
```

```
System.out.println("p1 equals p2," + p1.equals(p2));
System.out.println("p1 equals p3," + p1.equals(p3));

Set set = new HashSet();
set.add(p1);
set.add(p2);
set.add(p3);

for (Iterator iter=set.iterator(); iter.hasNext();) {
    Person p = (Person)iter.next();
    System.out.println("name=" + p.name + ", age=" + p.age);
}

System.out.println("p1.hashCode=" + p1.hashCode());
System.out.println("p2.hashCode=" + p2.hashCode());
System.out.println("p3.hashCode=" + p3.hashCode());
}
}

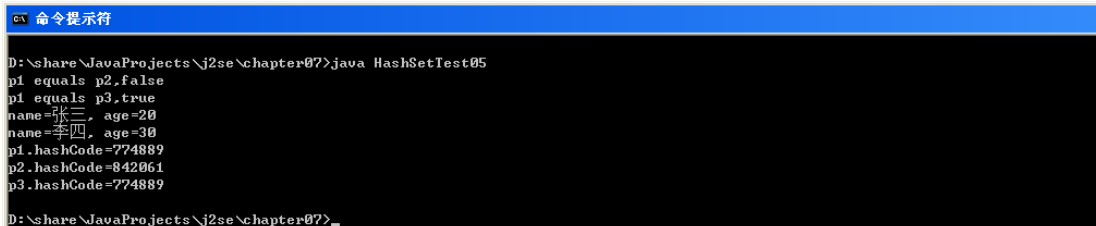
class Person {

    String name;

    int age;

    //覆盖 hashCode
    public int hashCode() {
        return (name==null)?0:name.hashCode();
    }
}
```

```
//覆盖 equals
public boolean equals(Object obj) {
    if (this == obj) {
        return true;
    }
    if (obj instanceof Person) {
        Person p = (Person)obj;
        return this.name.equals(p.name);
    }
    return false;
}
}
```



```
命令提示符
D:\share\JavaProjects\j2se\chapter07>java HashSetTest05
p1 equals p2.false
p1 equals p3.true
name=张三, age=20
name=李四, age=30
p1.hashCode=774889
p2.hashCode=842061
p3.hashCode=774889
D:\share\JavaProjects\j2se\chapter07>
```

以上输出完全正确的，因为覆盖了 `equals` 和 `hashCode`，当 `hashCode` 相同，它会调用 `equals` 进行比较，如果 `equals` 比较相等将不加把此元素加入到 `Set` 中，但 `equals` 比较不相等会重新根据 `hashCode` 换算位置仍然会将该元素加入进去的。

Person{张三, 20}	Person{李四, 30}	
774889	842061	

再次强调：特别是向 `HashSet` 或 `HashMap` 中加入数据时必须同时覆盖 `equals` 和 `hashCode` 方法，应该养成一种习惯覆盖 `equals` 的同时最好同时覆盖 `hashCode`

Java 要求：

两个对象 equals 相等，那么它的 hashCode 相等

两个对象 equals 不相等，那么它的 hashCode 并不要求它不相等，但一般建议不相等
hashCode 相等不代表两个对象相等（采用 equals 比较）

1.4.4 TreeSet

TreeSet 可以对 Set 集合进行排序，默认自然排序（即升序），但也可以做客户化的排序

```
import java.util.*;

public class TreeSetTest01 {

    public static void main(String[] args) {

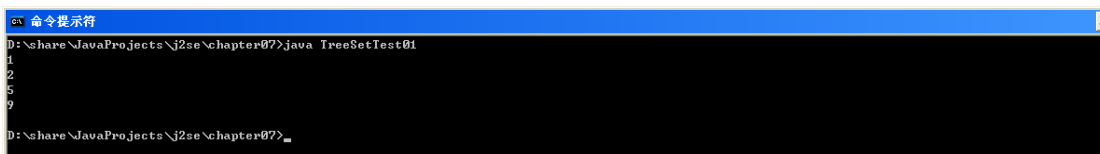
        Set set = new TreeSet();

        set.add(9);
        set.add(2);
        set.add(5);
        set.add(1);

        //不能放入重复数据
        set.add(5);

        for (Iterator iter=set.iterator(); iter.hasNext();) {
            Integer v = (Integer)iter.next();
            System.out.println(v);
        }

    }
}
```



```
命令提示符
D:\share\JavaProjects\j2se\chapter07>java TreeSetTest01
1
2
5
9
D:\share\JavaProjects\j2se\chapter07>
```


以上没有输出重复的，是按自然顺序排序的（升序）

```
import java.util.*;

public class TreeSetTest02 {

    public static void main(String[] args) {

        Set set = new TreeSet();
        set.add(9);
        set.add(2);
        set.add(5);
        set.add(1);

        //不能放入重复数据
        set.add(5);

        //不能加入 abc，加入后无法排序
        //排序只能对一种类型排序
        //set.add("abc");

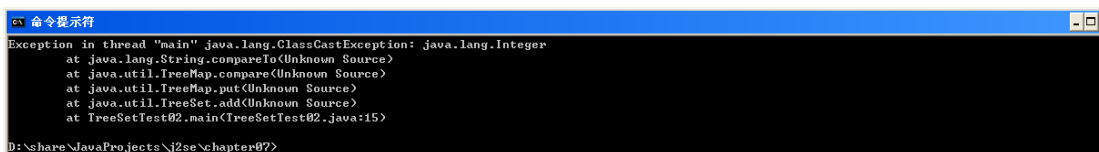
        for (Iterator iter=set.iterator(); iter.hasNext();) {

            //Integer v = (Integer)iter.next();
            //System.out.println(v);
            System.out.println(iter.next());

        }

    }

}
```



【代码示例】，对 Person 进行自然排序

```
import java.util.*;

public class TreeSetTest03 {

    public static void main(String[] args) {

        Person p1 = new Person();
        p1.name = "张三";
        p1.age = 20;

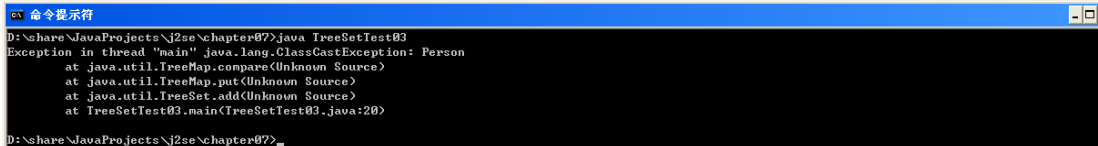
        Person p2 = new Person();
        p2.name = "李四";
        p2.age = 30;

        Person p3 = new Person();
        p3.name = "张三";
        p3.age = 40;

        Set set = new TreeSet();
        set.add(p1);
        set.add(p2);
        set.add(p3);

        for (Iterator iter=set.iterator(); iter.hasNext();) {
            Person p = (Person)iter.next();
            System.out.println("name=" + p.name + ", age=" + p.age);
        }
    }
}
```

```
class Person {  
  
    String name;  
  
    int age;  
  
}
```



```
命令提示符  
D:\share\JavaProjects\j2se\chapter07>java TreeSetTest03  
Exception in thread "main" java.lang.ClassCastException: Person  
    at java.util.TreeMap.compare(Unknown Source)  
    at java.util.TreeMap.put(Unknown Source)  
    at java.util.TreeSet.add(Unknown Source)  
    at TreeSetTest03.main(TreeSetTest03.java:20)  
D:\share\JavaProjects\j2se\chapter07>
```

出现错误，因为放到 TreeSet 中 TreeSet 会对其进行排序，那么必须实现 Comparable 接口，而我们的 Person 没有实现，所以出现了错误，如：基本类型的包装类和 String 他们都是可以排序的，他们都实现 Comparable 接口

1.4.5 实现 Comparable 接口完成排序

```
import java.util.*;  
  
public class TreeSetTest04 {  
  
    public static void main(String[] args) {  
        Person p1 = new Person();  
        p1.name = "张三";  
        p1.age = 20;  
  
        Person p3 = new Person();  
        p3.name = "张三";  
        p3.age = 40;  
  
        Person p2 = new Person();  
        p2.name = "李四";
```

```
p2.age = 30;

Set set = new TreeSet();
set.add(p1);
set.add(p2);
set.add(p3);

for (Iterator iter=set.iterator(); iter.hasNext();) {
    Person p = (Person)iter.next();
    System.out.println("name=" + p.name + ", age=" + p.age);
}
}

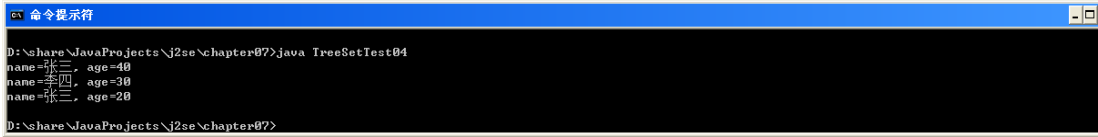
class Person implements Comparable {

    String name;

    int age;

    //如果覆盖了 equals，最好保证 equals 和 compareTo 在
    //相等情况下的比较规则是一致的
    public int compareTo(Object o) {
        if (o instanceof Person) {
            Person p = (Person)o;
            //升序
            //return (this.age - p.age);
            //降序
            return (p.age-this.age);
        }
    }
}
```

```
        throw new IllegalArgumentException("非法参数， o="+ o);  
    }  
}
```



```
C:\ 命令提示符  
D:\share\JavaProjects\j2se\chapter07>java TreeSetTest04  
name=张三, age=40  
name=李四, age=30  
name=张三, age=20  
D:\share\JavaProjects\j2se\chapter07>
```

1.4.6 实现 Comparator 接口完成排序

```
import java.util.*;  
  
public class TreeSetTest05 {  
  
    public static void main(String[] args) {  
        Person p1 = new Person();  
        p1.name = "张三";  
        p1.age = 20;  
  
        Person p3 = new Person();  
        p3.name = "张三";  
        p3.age = 40;  
  
        Person p2 = new Person();  
        p2.name = "李四";  
        p2.age = 30;  
  
        Comparator personComparator = new PersonComparator();  
  
        Set set = new TreeSet(personComparator);  
        set.add(p1);  
    }  
}
```

```
set.add(p2);
set.add(p3);

for (Iterator iter=set.iterator(); iter.hasNext();) {
    Person p = (Person)iter.next();
    System.out.println("name=" + p.name + ", age=" + p.age);
}
}
```

```
class Person {

    String name;

    int age;
}
```

//实现 Person 的比较器

//Comparator 和 Comparable 的区别?

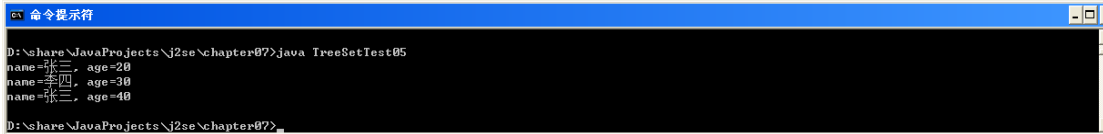
//Comparable 是默认的比较接口, Comparable 和需要比较的对象紧密结合到一起了

//Comparator 可以分离比较规则, 所以它更具灵活性

```
class PersonComparator implements Comparator {

    public int compare(Object o1, Object o2) {
        if (!(o1 instanceof Person)) {
            throw new IllegalArgumentException("非法参数, o1=" + o1);
        }
        if (!(o2 instanceof Person)) {
            throw new IllegalArgumentException("非法参数, o2=" + o2);
        }
    }
}
```

```
    }  
    Person p1 = (Person)o1;  
    Person p2 = (Person)o2;  
    return p1.age - p2.age;  
}  
}
```



1.4.7 采用匿名类完成 Comparator 的实现

```
import java.util.*;  
  
public class TreeSetTest06 {  
  
    public static void main(String[] args) {  
        Person p1 = new Person();  
        p1.name = "张三";  
        p1.age = 20;  
  
        Person p3 = new Person();  
        p3.name = "张三";  
        p3.age = 40;  
  
        Person p2 = new Person();  
        p2.name = "李四";  
        p2.age = 30;  
        //采用匿名类实现比较器  
        Set set = new TreeSet(new Comparator() {
```

```
        public int compare(Object o1, Object o2) {
            if (!(o1 instanceof Person)) {
                throw new IllegalArgumentException("非法参数, o1="+ o1);
            }
            if (!(o2 instanceof Person)) {
                throw new IllegalArgumentException("非法参数, o2="+ o2);
            }
            Person p1 = (Person)o1;
            Person p2 = (Person)o2;
            return p1.age - p2.age;
        }
    });
    set.add(p1);
    set.add(p2);
    set.add(p3);

    for (Iterator iter=set.iterator(); iter.hasNext();) {
        Person p = (Person)iter.next();
        System.out.println("name=" + p.name + ", age=" + p.age);
    }
}

class Person {

    String name;

    int age;
}
```


1.4.8 Comparable 和 Comparator 的区别?

一个类实现了 Comparable 接口则表明这个类的对象之间是可以相互比较的，这个类对象组成的集合就可以直接使用 sort 方法排序。

Comparator 可以看成一种算法的实现，将算法和数据分离，Comparator 也可以在下面两种环境下使用：

- 1、类的没有考虑到比较问题而没有实现 Comparable，可以通过 Comparator 来实现排序而不必改变对象本身
- 2、可以使用多种排序标准，比如升序、降序等

1.5 Map 接口

Map 中可以放置键值对，也就是每一个元素都包含键对象和值对象，Map 实现较常用的为 HashMap，HashMap 对键对象的存取和 HashSet 一样，仍然采用的是哈希算法，所以如果使用自定类作为 Map 的键对象，必须复写 equals 和 hashCode 方法。

1.5.1 HashMap

```
import java.util.*;

public class HashMapTest01 {

    public static void main(String[] args) {
        Map map = new HashMap();
        map.put("1001", "张三");
        map.put("1002", "李四");
        map.put("1003", "王五");

        //采用 entrySet 遍历 Map
        Set entrySet = map.entrySet();
        for (Iterator iter=entrySet.iterator(); iter.hasNext();) {
            Map.Entry entry = (Map.Entry)iter.next();
        }
    }
}
```

```
        System.out.println(entry.getKey() + ", " + entry.getValue());
    }

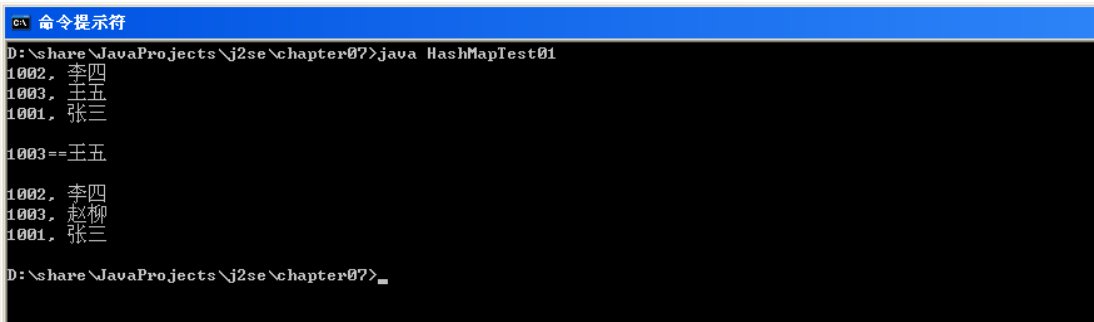
    System.out.println("");
    //取得 map 中指定的 key
    Object v = map.get("1003");
    System.out.println("1003==" + v);
    System.out.println("");

    //如果存在相同的条目，会采用此条目替换
    //但 map 中始终保持的是不重复的数据
    //主要依靠 key 来判断是否重复，和 value 没有任何关系
    map.put("1003", "赵柳");

    //采用 keySet 和 get 取得 map 中的所有数据
    for (Iterator iter=map.keySet().iterator(); iter.hasNext();) {
        String k = (String)iter.next();
        System.out.println(k + ", " + map.get(k));
    }

}

}
```



```
命令提示符
D:\share\JavaProjects\j2se\chapter07>java HashMapTest01
1002, 李四
1003, 王五
1001, 张三

1003==王五

1002, 李四
1003, 赵柳
1001, 张三

D:\share\JavaProjects\j2se\chapter07>
```

1.5.2 HashMap, 采用自定义类作为 key

```
import java.util.*;

public class HashMapTest02 {

    public static void main(String[] args) {

        IdCard idCard1 = new IdCard();
        idCard1.cardNo = 223243244243243L;
        Person person1 = new Person();
        person1.name = "张三";

        IdCard idCard2 = new IdCard();
        idCard2.cardNo = 223243244244343L;
        Person person2 = new Person();
        person2.name = "李四";

        IdCard idCard3 = new IdCard();
        idCard3.cardNo = 223243244243243L;
        Person person3 = new Person();
        person3.name = "张三";

        Map map = new HashMap();
        map.put(idCard1, person1);
        map.put(idCard2, person2);
        map.put(idCard3, person3);
```

```
        for (Iterator iter=map.entrySet().iterator(); iter.hasNext();) {
            Map.Entry entry = (Map.Entry)iter.next();
            IdCard idCard = (IdCard)entry.getKey();
            Person person = (Person)entry.getValue();
            System.out.println(idCard.cardNo + ", " + person.name);
        }

    }

}

class IdCard {

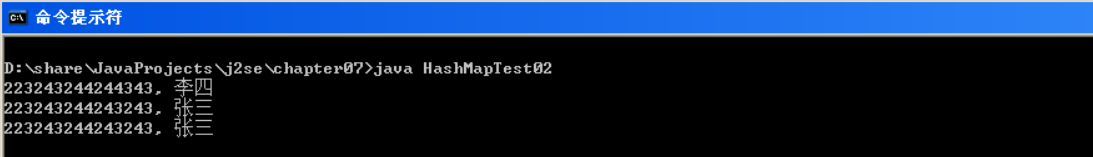
    long cardNo;

    //.....
}

class Person {

    String name;

}
```



```
命令提示符
D:\share\JavaProjects\j2se\chapter07>java HashMapTest02
223243244244343, 李四
223243244243243, 张三
223243244243243, 张三
```

加入了重复的数据，因为 HashMap 的底层实现采用的是 hash 表，所以 Map 的 key 必须覆盖 hashCode 和 equals 方法

1.5.3 HashMap, 覆盖 IdCard 的 equals 和 hashCode 方法

```
import java.util.*;

public class HashMapTest03 {

    public static void main(String[] args) {

        IdCard idCard1 = new IdCard();
        idCard1.cardNo = 223243244243243L;
        Person person1 = new Person();
        person1.name = "张三";

        IdCard idCard2 = new IdCard();
        idCard2.cardNo = 223243244244343L;
        Person person2 = new Person();
        person2.name = "李四";

        IdCard idCard3 = new IdCard();
        idCard3.cardNo = 223243244243243L;
        Person person3 = new Person();
        person3.name = "张三";

        Map map = new HashMap();
        map.put(idCard1, person1);
        map.put(idCard2, person2);
        map.put(idCard3, person3);
```

```
        for (Iterator iter=map.entrySet().iterator(); iter.hasNext();) {
            Map.Entry entry = (Map.Entry)iter.next();
            IdCard idCard = (IdCard)entry.getKey();
            Person person = (Person)entry.getValue();
            System.out.println(idCard.cardNo + ", " + person.name);
        }

    }

}

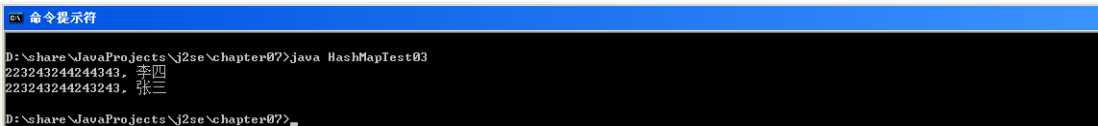
class IdCard {

    long cardNo;

    //.....

    public boolean equals(Object obj) {
        if (obj == this) {
            return true;
        }
        if (obj instanceof IdCard) {
            IdCard idCard = (IdCard)obj;
            if (this.cardNo == idCard.cardNo) {
                return true;
            }
        }
        return false;
    }
}
```

```
public int hashCode() {  
    return new Long(cardNo).hashCode();  
}  
}  
  
class Person {  
  
    String name;  
}
```



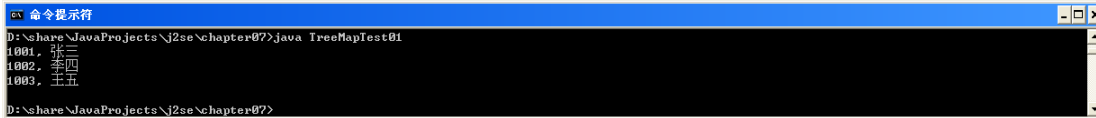
以上没有加入重复的数据，因为覆盖了 equals 和 hashCode 方法

1.5.4 TreeMap

treeMap 可以对 Map 中的 **key** 进行排序，如果 map 中的 key 采用的是自定类那么需要实现 Comparable 或 Comparator 接口完成排序

```
import java.util.*;  
  
public class TreeMapTest01 {  
  
    public static void main(String[] args) {  
        Map map = new TreeMap();  
        map.put("1003", "王五");  
        map.put("1001", "张三");  
        map.put("1002", "李四");  
  
        for (Iterator iter=map.entrySet().iterator(); iter.hasNext();) {  
            Map.Entry entry = (Map.Entry)iter.next();  
            System.out.println(entry.getKey() + ", " + entry.getValue());  
        }  
    }  
}
```

```
}  
  
}  
  
}
```



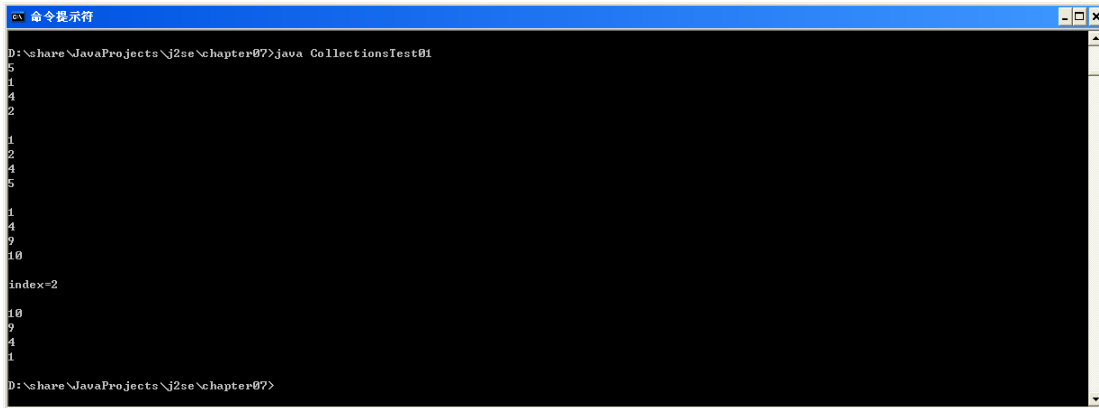
1.6 Collections 工具类

Collections 位于 java.util 包中，提供了一系列实用的方法，如：对集合排序，对集合中的内容查找等

```
import java.util.*;  
  
public class CollectionsTest01 {  
  
    public static void main(String[] args) {  
        List l = new ArrayList();  
        l.add(5);  
        l.add(1);  
        l.add(4);  
        l.add(2);  
        for (Iterator iter=l.iterator(); iter.hasNext();) {  
            System.out.println(iter.next());  
        }  
        System.out.println("");  
  
        Collections.sort(l);  
        for (Iterator iter=l.iterator(); iter.hasNext();) {  
            System.out.println(iter.next());  
        }  
    }  
}
```



```
}  
System.out.println("");  
  
Set set = new HashSet();  
set.add(10);  
set.add(1);  
set.add(4);  
set.add(9);  
  
//不能直接对 set 排序  
//Collections.sort(set);  
List setList = new ArrayList(set);  
Collections.sort(setList);  
for (Iterator iter=setList.iterator(); iter.hasNext();) {  
    System.out.println(iter.next());  
}  
System.out.println("");  
int index = Collections.binarySearch(setList, 9);  
System.out.println("index=" + index);  
  
System.out.println("");  
Collections.reverse(setList);  
for (Iterator iter=setList.iterator(); iter.hasNext();) {  
    System.out.println(iter.next());  
}  
  
}  
  
}
```



1.7 泛型初步

泛型能更早的发现错误，如类型转换错误，通常在运行期才会发现，如果使用泛型，那么在编译期将会发现，通常错误发现的越早，越容易调试，越容易减少成本

1.7.1 为什么使用泛型

```
import java.util.*;
```

```
public class GenericTest01 {
```

```
    public static void main(String[] args) {
```

```
        List l = new ArrayList();
```

```
        l.add(1);
```

```
        l.add(2);
```

```
        l.add(3);
```

```
        for (Iterator iter=l.iterator(); iter.hasNext();) {
```

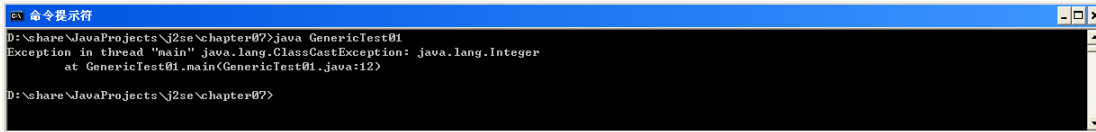
```
            //出现了 java.lang.ClassCastException 异常
```

```
            //这种转型错误时运行期发现了
```

```
            //错误发现的越早越好，最好在编译器能发现类似的错误
```

```
            //如果想在编译器发现类似的错误，必须使用泛型
```

```
String s = (String)iter.next();  
System.out.println(s);  
}  
}  
}
```



1.7.2 使用泛型解决示例一

```
import java.util.*;  
  
public class GenericTest02 {  
  
    public static void main(String[] args) {  
        List<Integer> l = new ArrayList<Integer>();  
        l.add(1);  
        l.add(2);  
        l.add(3);  
  
        //不能将 abc 放到集合中，因为使用泛型，在编译器就可以发现错误  
        //l.add("abc");  
  
        for (Iterator<Integer> iter=l.iterator(); iter.hasNext();) {  
  
            //因为使用泛型，在编译器就可以发现错误  
            //String s = (String)iter.next();  
  
            //使用泛型可以不用进行强制转换
```

```
//Integer s = (Integer)iter.next();

//可以直接取得相应的元素，使用泛型返回的是真正的类型
Integer s = iter.next();
System.out.println(s);
    }
}
}
```

1.7.3 采用泛型来改善自定义比较器

```
import java.util.*;

public class GenericTest03 {

    public static void main(String[] args) {
        Person p1 = new Person();
        p1.name = "张三";
        p1.age = 20;

        Person p3 = new Person();
        p3.name = "张三";
        p3.age = 40;

        Person p2 = new Person();
        p2.name = "李四";
        p2.age = 30;

        Set<Person> set = new TreeSet<Person>();
        set.add(p1);
```

```
set.add(p2);
set.add(p3);

for (Iterator<Person> iter=set.iterator(); iter.hasNext();) {
    Person p = iter.next();
    System.out.println("name=" + p.name + ", age=" + p.age);
}
}
}

class Person implements Comparable<Person> {

    String name;

    int age;

    //使用了泛型类似的 instanceof 就不用再写了
    public int compareTo(Person o) {
        return (this.age - o.age);
    }
}
```

1.7.4 采用泛型改造 Map

```
import java.util.*;

public class GenericTest04 {

    public static void main(String[] args) {
```

```
IdCard idCard1 = new IdCard();
idCard1.cardNo = 223243244243243L;
Person person1 = new Person();
person1.name = "张三";
```

```
IdCard idCard2 = new IdCard();
idCard2.cardNo = 223243244244343L;
Person person2 = new Person();
person2.name = "李四";
```

```
IdCard idCard3 = new IdCard();
idCard3.cardNo = 223243244243243L;
Person person3 = new Person();
person3.name = "张三";
```

```
Map<IdCard, Person> map = new HashMap<IdCard, Person>();
map.put(idCard1, person1);
map.put(idCard2, person2);
map.put(idCard3, person3);
```

```
//不能编译
```

```
//map.put("1001", "王五");
```

```
for (Iterator<Map.Entry<IdCard, Person>> iter=map.entrySet().iterator();
iter.hasNext();) {
    /*
    Map.Entry entry = (Map.Entry)iter.next();
    IdCard idCard = (IdCard)entry.getKey();
    Person person = (Person)entry.getValue();
```

```
        */  
        Map.Entry<IdCard, Person> entry = iter.next();  
  
        //不能转换  
        //String s = (String)entry.getKey();  
        IdCard idCard = entry.getKey();  
        Person person = entry.getValue();  
  
        System.out.println(idCard.cardNo + ", " + person.name);  
    }  
  
}  
  
}  
  
class IdCard {  
  
    long cardNo;  
  
    //.....  
  
    public boolean equals(Object obj) {  
        if (obj == this) {  
            return true;  
        }  
        if (obj instanceof IdCard) {  
            IdCard idCard = (IdCard)obj;  
            if (this.cardNo == idCard.cardNo) {  
                return true;  
            }  
        }  
    }  
}
```

```
    }  
    return false;  
}  
  
public int hashCode() {  
    return new Long(cardNo).hashCode();  
}  
}  
  
class Person {  
  
    String name;  
}
```

1.7.5 自定义泛型

```
import java.util.*;  
  
public class GenericTest05 {  
  
    private Object obj;  
  
    public void setObj(Object obj) {  
        this.obj = obj;  
    }  
  
    public Object getObj() {  
        return obj;  
    }  
}
```



```
public static void main(String[] args) {  
  
    GenericTest05 g = new GenericTest05();  
  
    g.setObj("abcd");  
  
    //抛出 java.lang.ClassCastException 错误  
    //因为不知道 Object 到底是什么类型  
    Integer i = (Integer)g.getObj();  
}  
  
}
```

【示例代码】，自定泛型

```
import java.util.*;  
  
public class GenericTest06<T> {  
  
    private T obj;  
  
    public void setObj(T obj) {  
        this.obj = obj;  
    }  
  
    public T getObj() {  
        return obj;  
    }  
  
    public static void main(String[] args) {  
  
        GenericTest06<String> g = new GenericTest06<String>();  
  
    }  
}
```

```
g.setObj("abcd");

//不能设置 int 类型
//因为使用泛型规定只能设置为 String 类型
//g.setObj(123);

//不能转换，因为 String 类型
//Integer i = (Integer)g.getObj();

//使用泛型后不用再进行强制转换了
//它返回的就是真正的类型
String s = g.getObj();
}
}
```

【示例代码】，修改泛型标识

```
import java.util.*;

//泛型的标示符没有限制，只有符合 java 的标示符命名规范即可
//最好和 JDK 的泛型标识一样
public class GenericTest07<AAA> {

    private AAA obj;

    public void setObj(AAA obj) {
        this.obj = obj;
    }

    public AAA getObj() {
```

```
        return obj;
    }

    public static void main(String[] args) {

        GenericTest07<String> g = new GenericTest07<String>();

        g.setObj("abcd");

        String s = g.getObj();
    }
}
```

1.8 遗留类对比表

遗留类	缺点	取代类
Vector	方法都是同步的影响性能	ArrayList 和 LinkedList
Hashtable	方法都是同步的影响性能	HashMap
Stack	因为 Stack 继承了 Vector, 同样影响性能	LinkedList
Enumeration	只能与历史集合使用	Iterator