

## 知识点列表

编号	名称	描述	级别
1	散列表 Map	理解散列表的相关概念	*
2	HashMap	掌握并能熟练应用 HashMap 的常用 API 方法	***
3	集合框架 (Collection 和 Map)	理解集合框架，尤其是描述集合框架的那张图	*
4	Java 泛型	掌握泛型的使用。	**
5	集合的迭代	理解并掌握 Java 中迭代器的使用方法，尤其注意迭代中的删除操作	**
6	集合的工具类 Collections	掌握集合工具类的常用方法	**
7	Comparable 和 Comparator	理解比较器和可比较的接口	*
8	java 中的包装类	通过案例理解 Java 中的包装类，掌握常用的 API 方法	**

注：    "\*"理解级别    "\*\*\*"掌握级别    "\*\*\*\*"应用级别

## 目录

1. 散列表 Map *	3
2. HashMap ***	3
2.1. 案例：HashMap	4
3. 集合框架 ( Collection 和 Map ) *	5
3.1. 案例：HashSet	6
3.2. 案例：“贪吃蛇” 版本 2	8
3.3. 案例：链表数据结构	10
4. Java 泛型 **	12
4.1. 案例：泛型	12
5. 集合的迭代 **	13
5.1. 案例：Iterator	13
5.2. 案例：使用集合写“扑克牌”	14
6. 集合的工具类 Collections **	16
6.1. 案例：Collections	16
7. Comparable 和 Comparator *	16
7.1. 案例：Comparator	17
8. java 中的包装类 **	18
8.1. 案例：包装类	18

## 1. 散列表 Map \*

### 散列表概念

- 1) 容量 散列表中散列数组大小
- 2) 散列运算 key->散列值(散列数组下标)的算法,  
如: "mm".hashCode()%10->8
- 3) 散列桶 散列值相同的元素的“线性集合”
- 4) 加载因子 就是散列数组加载率,一般小于 75%性能比较理想  
就是元素数量/散列数组大小,如: 7/10=70%
- 5) 散列查找 根据 Key 计算散列值,根据散列值(下标)找到散列桶,在散列桶中  
顺序比较 Key, 如果一样, 就返回 value  
散列表中 Key 不同, Value 可以重复

## 2. HashMap \*\*\*

HashMap 以键-值对 ( 关键字 : 值 ) 的形式存储对象, 关键字 key 是唯一的、不重复的

- 1) key 可以是任何对象, Value 可以是任何对象
- 2) ( key : value ) 成对放置在集合中
- 3) 重复的 key 算一个, 重复添加是替换操作 ( 会覆盖原来的元素 )
- 4) 根据 key 的散列值计算散列表, 元素按照散列值(不可见)排序
- 5) HashMap 默认的容量: 16 , 默认加载因子(加载率) 0.75
- 6) HashMap 根据 key 检索查找 value 值

HashMap 可以在构造时指定参数: 初始容量和加载因子, 一般使用默认

EnumSet

EventListenerProxy

EventObject

FormattableFlags

Formatter

GregorianCalendar

**HashMap**

HashSet

Hashtable

IdentityHashMap

LinkedHashMap

LinkedHashSet

LinkedList

ListResourceBundle

Locale

### 构造方法摘要

<b>HashMap</b> ()	构造一个具有默认初始容量 (16) 和默认加载因子 (0.75) 的空 HashMap。
<b>HashMap</b> (int initialCapacity)	构造一个带指定初始容量和默认加载因子 (0.75) 的空 HashMap。
<b>HashMap</b> (int initialCapacity, float loadFactor)	构造一个带指定初始容量和加载因子的空 HashMap。
<b>HashMap</b> (Map<? extends K, ? extends V> m)	构造一个映射关系与指定 Map 相同的新 HashMap。

## 2.1. 案例：HashMap

```

HashMapDemo.java X
1 package corejava.day06.ch01;
2 import java.util.HashMap;
3
4 /** 散列表演示 */
5 public class HashMapDemo {
6     public static void main(String[] args) {
7         HashMap users = new HashMap();
8         //注册用户
9         users.put("Tom", new User("Tom", "123", 5));
10        users.put("Jerry", new User("Jerry", "123", 6));
11        users.put("Andy", new User("Andy", "abc", 5)); //将被覆盖
12        users.put("Andy", new User("Andy", "123", 8));
13        System.out.println(users);
14
15        //登录查找
16        Scanner s = new Scanner(System.in);
17        while(true) {
18            System.out.print("用户名:");
19            String name = s.nextLine();
20            System.out.print("密码:");
21            String pwd = s.nextLine();
22            if(!users.containsKey(name)) {
23                System.out.println("没有注册!");
24                continue;
25            }
26            User user = (User)users.get(name);
27            if(user.pwd.equals(pwd)) {
28                System.out.println("欢迎"+user.name
29                    + " age:"+user.age);
30                break;
31            }
32        }
33    }
34 }
35 class User{
36     String name;
37     String pwd;
38     int age;
39     public User(String name, String pwd, int age){
40         this.name = name;
41         this.age = age;

```

```

42     this.pwd = pwd;
43     }
44     public String toString() {
45         return name + ":" + age;
46     }
47 }

```

注：

✓ HashMap 还有一些常用方法，请参照 API 练习

- clear()
- containsKey(Object key)
- containsValue(Object value)
- get()
- isEmpty()
- keySet()                      返回所有的 key ( 注意：返回值都放入 set 集合中 )
- put(key, value)              向 Map 中加入元素
- remove(Object o)
- size()

### ● HashMap VS Hashtable

HashMap      新，非线程安全，不检查锁，快

Hashtable    旧 (JDK1.2 版本以前)，线程安全，检查锁，慢一点 ( 差的很小 )

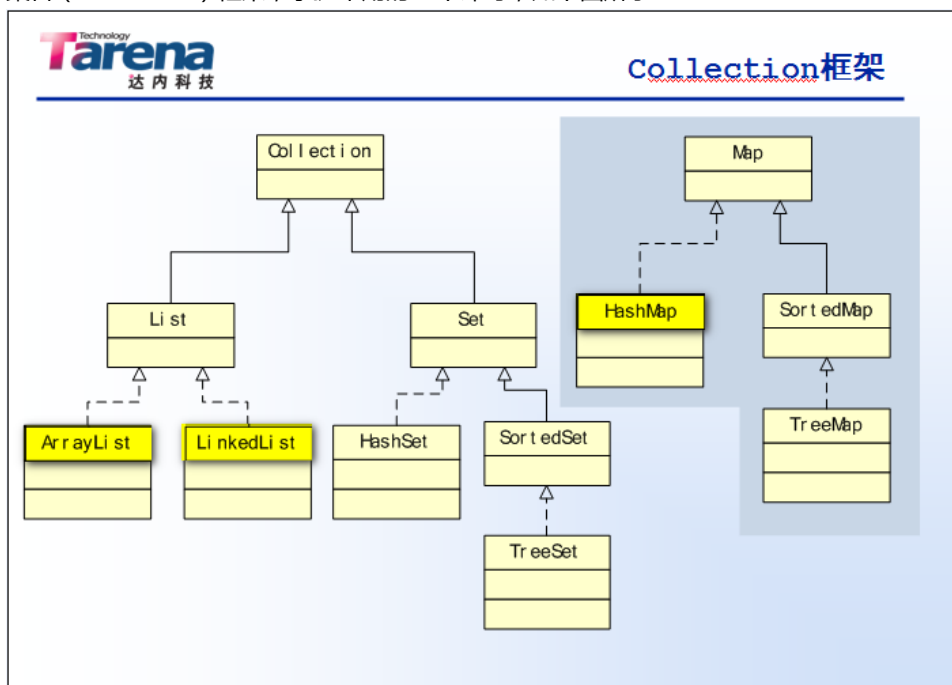
HashMap 较常用，HashMap 和 Hashtable 的比较常出现于面试题

## 3. 集合框架 ( Collection 和 Map ) \*

集合框架包括集合与映射 ( Collection and Map )，以及它们的子类 ( 容器类 )

- 1) **List** 元素有先后次序的集合，元素有 index 位置，元素可以重复，继承自 Collection 接口，
  - 实现类: ArrayList, Vector, LinkedList
- 2) **Set** 元素无续，不能重复添加，是数学意义上的集合，继承自 Collection 接口
  - 实现类: HashSet(是一个只有 Key 的 HashMap)
- 3) **Collection** 集概念，没有说明元素是否重复和有序，使用集合的跟接口，很少直接使用  
其他集合都是实现类: ArrayList, HashSet
- 4) **Map** 描述了 ( key : value ) 成对放置的集合，key 不重复，Value 可以重复 ( key 重复算一个 )
  - 实现类: HashMap(散列表算法实现)
  - TreeMap(二叉排序树实现,利用 Key 排序)
  - Map 适合检查查找

集合 (Collection) 框架，掌握常用的 3 个即可，如下图所示：



注：

- ✓ **Collection 接口**      表示集合的概念
  - List 接口
  - Set 接口
- ✓ **List 接口**      表示有序线性表的概念
  - ArrayList      底层实现是数组
  - LinkedList      底层实现是链表
- ✓ **Set 接口**      表示无序不重复的概念
  - HashSet

### 3.1. 案例：HashSet

关于 set 接口 (HashSet) 的测试，如下所示：

```

SetDemo.java
1 package corejava.day06.ch02;
2 import java.util.HashSet;
3 /** Set 集合演示:
4  * 模仿贪吃蛇中的"豆豆"*/
5 public class SetDemo {
6     public static void main(String[] args) {
7         Set<Node> foods = new HashSet<Node>();
8     }
9 }
    
```

```

9      //1. 放入5颗"豆豆"
10     foods.add(new Node(3,4));
11     foods.add(new Node(6,8));
12     foods.add(new Node(5,10));
13     foods.add(new Node(2,12));
14     foods.add(new Node(9,9));
15     System.out.println(foods.size());
16     System.out.println(foods);
17     //2. 吃了1颗
18     foods.remove(new Node(9,9));
19     //3. 在面板WormPane中画出"豆豆"
20     for(int i=0; i<10; i++){
21         for(int j=0; j<20; j++){
22             if(foods.contains(new Node(i,j))){
23                 System.out.print("0");
24             }else{
25                 System.out.print(" ");
26             }
27         }
28         System.out.println();
29     }
30 }
31 }
32 class Node{
33     int i;
34     int j;
35     public Node(int i,int j) {
36         this.i=i;
37         this.j=j;
38     }
39     //注意: 必须写equals()方法
40     public boolean equals(Object obj) {
41         if(obj==null){
42             return false;
43         }
44         if(this==obj)
45             return true;
46         if (obj instanceof Node) {
47             Node o = (Node) obj;
48             return i==o.i && j==o.j;
49         }
50         return false;
51     }

```

```

52 //注意：必须写hashCode()方法
53 public int hashCode() {
54     return (i<<16)|j;
55 }
56 public String toString() {
57     return "["+i+","+j+"]";
58 }
59 }

```

注：

- ✓ Node 类只是课堂演示代码，当练习时要遵守 **Java Bean 规范**

### 3.2. 案例：“贪吃蛇” 版本 2

案例描述：在 day05 “贪吃蛇” 案例中面板（WormPane）中加入“豆豆”

如下图所示：

- **Node.java**  
请参照 day05 代码（略）
- **Worm.java**  
请参照 day05 代码（略）
- **WormDemo.java**  
请参照 day05 代码（略）
- **WormPane.java**

```

WormPane.java
1 package corejava.day06.ch03;
2 import java.util.HashSet;
3
4
5 public class WormPane {
6     private Worm worm;
7     /** 行数 */
8     private int rows = 10;
9     /** 列数 */
10    private int cols = 32;
11

```



```

12 // 【1】
13 /** 食物 */
14 private Set<Node> foods = new HashSet<Node>();
15
16 public WormPane() {
17     worm = new Worm(); //先有蛇
18     // 【3】
19     initFoods(5); //后有"豆豆"
20 }
21
22 // 【2】
23 public void initFoods(int n) {
24     Random r = new Random();
25     while(true) {
26         int i = r.nextInt(rows-2)+1;
27         int j = r.nextInt(cols-2)+1;
28         if(worm.contains(i, j)) {
29             continue;
30         }
31         Node food = new Node(i, j);
32         if(foods.contains(food)) {
33             continue;
34         }
35         foods.add(food);
36         if(foods.size()==n) {
37             break;
38         }
39     }
40 }
41
42 public Worm getWorm() {
43     return worm;
44 }
45
46 /** 画出当前面板 */
47 public void print() {
48     for(int i=0; i<rows; i++) {
49         for(int j=0; j<cols; j++) {
50             if(i==0 || i==rows-1) {
51                 System.out.print("-"); //不能输出回车
52             } else if(j==0 || j==cols-1) {

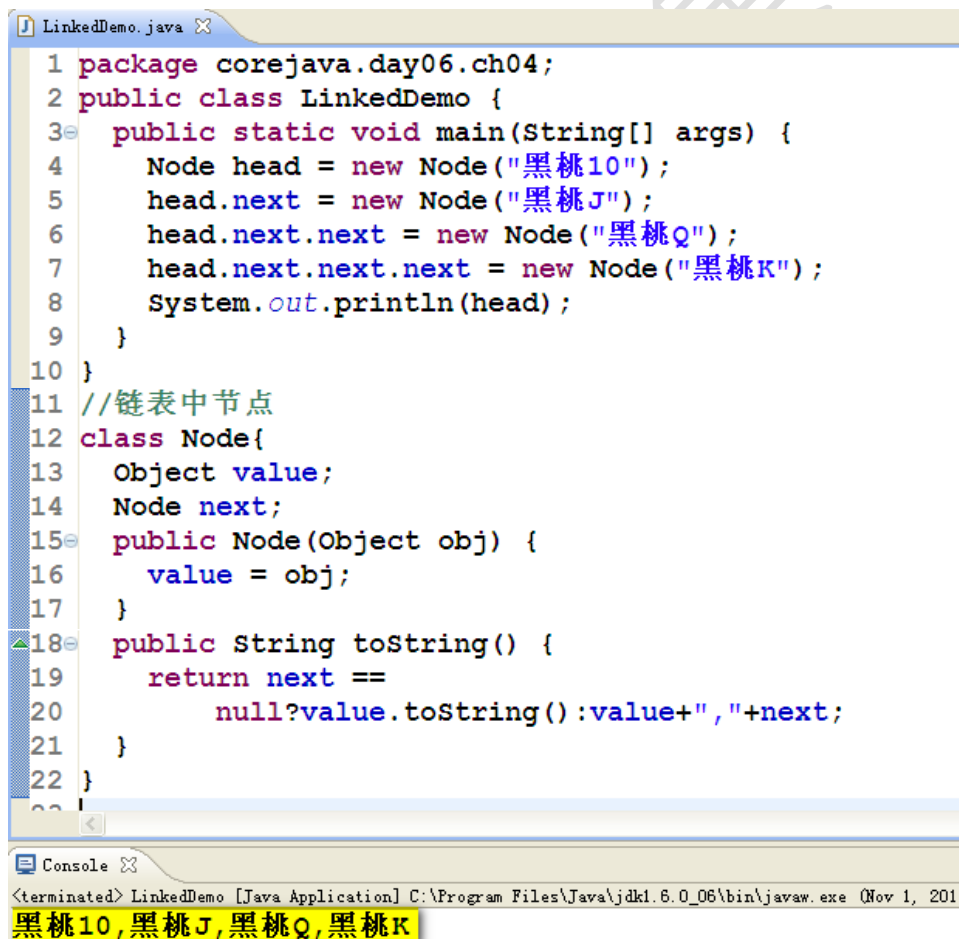
```

```

53         System.out.print("|");
54     }else if(worm.contains(i, j)){ //【4】
55         System.out.print("#");//"豆豆"不能在蛇身上
56     }else if(foods.contains(new Node(i, j))){
57         System.out.print("o");//"豆豆"
58     }else{
59         System.out.print(" ");
60     }
61 }
62 System.out.println(); //一行结束以后画回车
63 }
64 }
65 }

```

### 3.3. 案例：链表数据结构



```

1 package corejava.day06.ch04;
2 public class LinkedDemo {
3     public static void main(String[] args) {
4         Node head = new Node("黑桃10");
5         head.next = new Node("黑桃J");
6         head.next.next = new Node("黑桃Q");
7         head.next.next.next = new Node("黑桃K");
8         System.out.println(head);
9     }
10 }
11 //链表中节点
12 class Node{
13     Object value;
14     Node next;
15     public Node(Object obj) {
16         value = obj;
17     }
18     public String toString() {
19         return next ==
20             null?value.toString():value+","+next;
21     }
22 }

```

Console

```

<terminated> LinkedDemo [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Nov 1, 201
黑桃10,黑桃J,黑桃Q,黑桃K

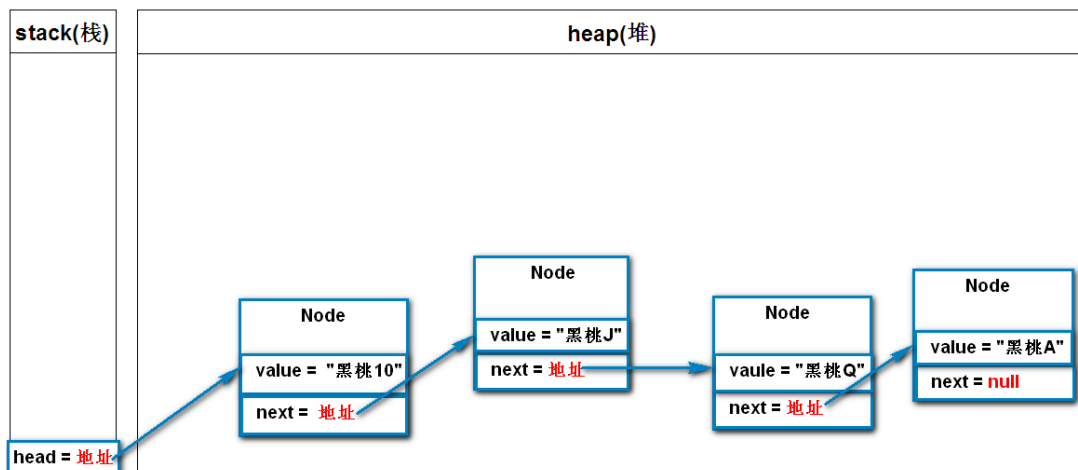
```

注：

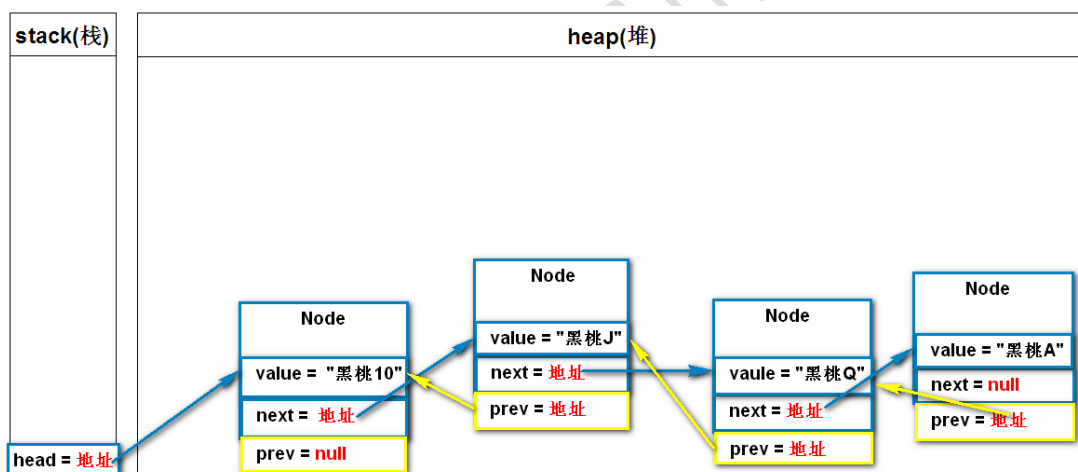
- ✓ 递归的写法：return next == null ? value.toString() : value+","+next;

递归结束条件是 `next == null`

程序演示了单向链表，内存结构图如下所示：



双向链表内存结构图如下所示：



注：

- ✓ 双向链表，即为 Node 对象添加 1 个属性 `private Node prev` (上一个)；

**LinkedList** 就是用这种双向链表的数据结构存储集合元素的，可以参看 Java 源码。

## 4. Java 泛型 \*\*

泛型是 Java5 以后提出的语法现象，作用是在编译期检查的类型约束（运行期不检查泛型）

泛型可以用来约束类中元素的类型

### 4.1. 案例：泛型

```
ShopDemo.java
1 package corejava.day06.ch05;
2 /** 泛型演示:
3  * 约束商店商品类型 */
4 public class ShopDemo {
5     public static void main(String[] args) {
6         //食品店
7         Shop<Food> foodShop=new Shop<Food>(new Food("土豆"));
8         //宠物店
9         Shop<Pet> petShop = new Shop<Pet>(new Pet("旺财"));
10        System.out.println(foodShop.buy());
11        System.out.println(petShop.buy());
12
13        //如果不使用泛型, 默认泛型是<Object>
14        Shop shop = new Shop(new Object());
15    }
16 }
17 /** P 代表商品类型 */
18 class Shop<P>{
19     P product;    //销售产品
20     public Shop(P p){ product = p; }
21     //进货方法
22     P buy(){ return product; }
23 }
24 class Food{
25     String name;
26     public Food(String name) { this.name = name; }
27     public String toString() { return name; }
28 }
29
30 class Pet{
31     String name;
32     public Pet(String name) { this.name = name; }
33     public String toString() { return name; }
34 }
```

## 5. 集合的迭代 \*\*

集合的迭代，是一种遍历算法。

- 1) 迭代操作举例：播放列表的“逐个播放”；将扑克牌“逐一发放”
- 2) java 使用 Iterator 接口描述了迭代模式操作  
Iterator 中的方法，专门为 **while 循环**设计
- 3) Iterator 的实例可以从**集合对象**获得，是这个集合的一个元素序列视图，默认包含一个**操作游标**（在第一个元素之前）
  - **hasNext()方法**，可以检查游标是否有下一个元素
  - **next() 方法**，移动游标到下一个元素，并且返回这个元素引用
 使用 **while 循环**配合这个两个方法，可以迭代处理集合的所有元素
- 4) 迭代时可以使用迭代器 **remove()** 方法删除刚刚迭代的元素在迭代过程中
  - **迭代时不能使用集合方法**(add, remove, set) 更改集合元素！

### 5.1. 案例：Iterator

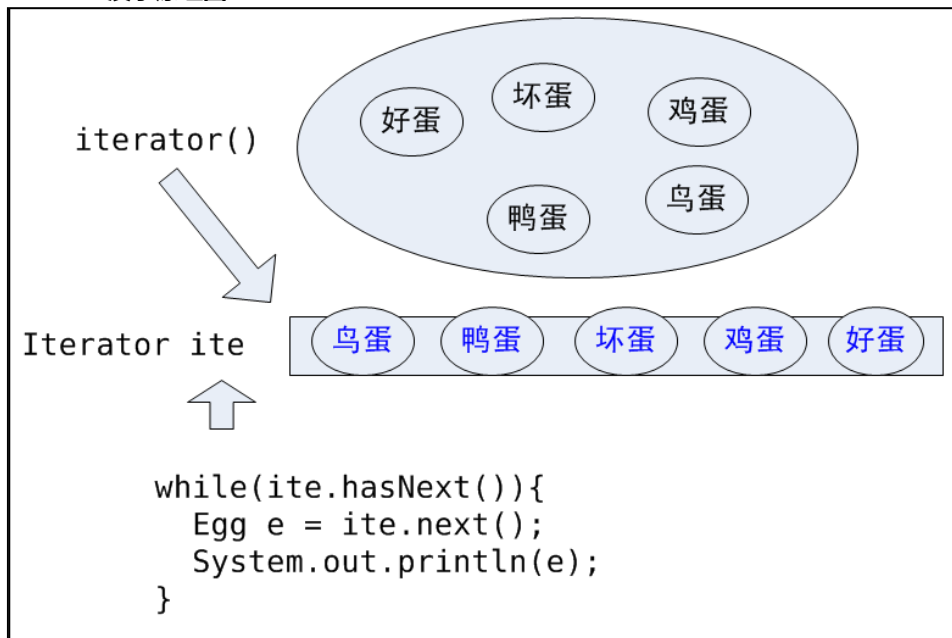
```

1 package corejava.day06.ch06;
2 import java.util.Iterator;
3
4 /** Iterator 演示 */
5
6 public class IteratorDemo {
7     public static void main(String[] args) {
8         Collection<String> eggs = new HashSet<String>();
9         eggs.add("鸟蛋");
10        eggs.add("鸭蛋");
11        eggs.add("坏蛋");
12        eggs.add("鸡蛋");
13        eggs.add("好蛋");
14        //ite 是eggs的集合的视图(另外一种外观)
15        Iterator<String> ite = eggs.iterator();
16        //ite 是顺序结构，含有一个游标，在第一个元素之前
17        //ite =["鸟蛋", "鸭蛋", "坏蛋", "鸡蛋", "好蛋"]
18        //      ^
19        while(ite.hasNext()){//检查当前游标是否有下一个
20            String egg = ite.next();//移动游标，返回下一个
21            System.out.println(ite.next());
22        }
23    }
24 }
    
```

注：

- ✓ hasNext()和 next()方法是模式化的，配合 while 循环使用
- ✓ 注意：每调用一次 next()，游标会向后移动一位

Iterator 演示原理图



## 5.2. 案例：使用集合写“扑克牌”

**案例描述：**使用集合完成洗牌、发牌等操作

- **Card.java**

请参考 day04 代码（略）

- **Player.java**

请参考 day04 代码（略）

- **CardDemo.java**

```

1 package corejava.day06.ch07;
2 import java.util.ArrayList;
3
4 public class CardDemo {
5     public static void main(String[] args) {
6         //1. 创建一副扑克牌
7         List<Card> cards = new ArrayList<Card>();
8         for(int rank=Card.THREE;rank<=Card.DEUCE; rank++){
9             cards.add(new Card(Card.DIAMOND, rank));
10            cards.add(new Card(Card.CLUB, rank));
11            cards.add(new Card(Card.SPADE, rank));
12            cards.add(new Card(Card.HEART, rank));
13        }
14        cards.add(new Card(Card.JOKER, Card.BLACK));
15        cards.add(new Card(Card.JOKER, Card.COLOR));
16
17        //2. 洗牌操作 (将集合cards内容打乱)
18        Collections.shuffle(cards); //Java提供的工具方法
19
20        //3. 创建3个玩家
21        List<Player> players = new LinkedList<Player>();
22        players.add(new Player(1,"王菲"));
23        players.add(new Player(2,"张飞"));
24        players.add(new Player(3,"刘亦菲"));
25
26        //4. 发牌(迭代)
27        Iterator<Card> ite = cards.iterator();
28        int i=0;
29        while(ite.hasNext()){
30            //4.1 c代表每一张牌
31            Card c = ite.next();
32            //4.2 发给某个人
33            players.get(i++%players.size()).add(c);
34            //4.3 从cards集合中删除刚刚发过(迭代过)的牌
35            ite.remove();
36            //cards.remove(c); //错误的写法! 运行异常
37            //4.4 剩下三张不发
38            if(cards.size()==3){
39                break;
40            }
41        }
42        System.out.println(players.get(0));
43        System.out.println(players.get(1));
44        System.out.println(players.get(2));
45        System.out.println(cards); //剩余的牌
46    }
47 }

```

注：

- ✓ remove()方法一定要在 next()方法后执行，删除的是 next()返回的元素

## 6. 集合的工具类 Collections \*\*

同数组的工具类 Arrays 相同，集合的工具类为 Collections，其中提供了许多的方法，诸如排序、二分查找、打乱、填充等操作。

### 6.1. 案例：Collections

```

1 package corejava.day06.ch08;
2 import java.util.ArrayList;
3 import java.util.Collections;
4 import java.util.List;
5 /** Collections 演示 */
6 public class CollectionsDemo {
7     public static void main(String[] args) {
8         List<String> names = new ArrayList<String>();
9         names.add("Tom");
10        names.add("Jerry");
11        names.add("Black");
12        names.add("Andy");
13        names.add("Lee");
14        //1. 排序
15        Collections.sort(names);
16        System.out.println(names);
17        //2. 二分法查找
18        int index =
19            Collections.binarySearch(names, "Jerry");
20        System.out.println(index);
21        //3. 乱序
22        Collections.shuffle(names);
23        System.out.println(names);
24    }
25 }

```

注：

- ✓ Collections.sort()底层调用了 str.compareTo()方法比较大小

## 7. Comparable 和 Comparator \*

### Comparable

- ✓ 表示可以比较的（用于类实现）



- ✓ 实现这个接口表示：这个类的实例可以比较大小，可以进行自然排序
- ✓ compareTo() 返回正数表示大，返回负数表示小，返回 0 表示相等
- ✓ Comparable 的实现必须与 equals() 的结果一致，就是相等的对象时候，比较结果一定是 0！

### Comparator

- ✓ 比较工具
- ✓ 用于临时定义比较规则，不是默认比较规则

## 7.1. 案例：Comparator

```

1 package corejava.day06.ch09;
2 import java.util.ArrayList;
3
4
5
6
7 public class Demo {
8     public static void main(String[] args) {
9         //1. 演示
10        //String重写的compareTo()方法
11        int a = "Tom".compareTo("Jerry");
12        System.out.println(a); //结果>0
13        a = "Tom".compareTo("Tom");
14        System.out.println(a); //结果==0
15        a = "Jerry".compareTo("Tom");
16        System.out.println(a); //结果<0
17
18        List<String> names = new ArrayList<String>();
19        names.add("Tom");
20        names.add("Jerry");
21        names.add("Black");
22        names.add("Andy");
23        names.add("Lee");
24        ByLength byLength = new ByLength();
25        //2. 演示
26        //按照自定义比较规则排序: byLength
27        Collections.sort(names, byLength);
28        System.out.println(names);
29    }
30 }
31 //自定义比较规则：按照字符串长度比较
32 class ByLength implements Comparator<String>{
33     public int compare(String o1, String o2) {
34         return -(o1.length() - o2.length()); //按照长度比
35     }
36 }
    
```

## 8. java 中的包装类 \*\*

**包装类**可以把基本类型包装为对象类型。

1) 共有 8 种包装类

- int                    **Integer**
- long                  Long
- byte                  Byte
- short                Short
- float                Float
- double              Double
- boolean            Boolean
- char                 **Character**

2) 包装类提供了对应数据类型的**工具方法**

- Integer.toHexString()
- Integer.toString(int)
- Integer.toBinaryString()
- Integer.parseInt(String)
- Integer.parseInt(String, int)
- Double.parseDouble(String str);

3) 自动包装(auto boxing / unboxing)

- java5 以后可以

4) 注意点

- 包装类是 final 的类
- 包装类对象是不变的, 与字符串类似(不变模式)  
Integer a = 1;  
Integer b = 2;  
a = a+b;  
a = new Integer(a.intValue() + b.intValue())
- 包装类覆盖了 toString()、equals()、hashCode()、compareTo()方法

### 8.1. 案例：包装类

```
public class WarpClass {  
    public static void main(String[] args) {  
        //ArrayList  
        //包装  
        Integer i = new Integer(1);  
        //拆包  
        int a = i.intValue();  
        //java5 提供自动拆包//包装  
        Double d = 2.3;//new Double(2.3)  
        double b = d;//d.doubleValue()  
  
        double x = 2 * d + i;
```