

# 集合框架-day02

## 今日学习内容：

- 集合元素迭代操作
- Set接口和实现类
- Map接口和实现类
- 集合的工具类

## 今日学习目标：

- 掌握使用for循环对集合做迭代
- 掌握使用for-each对集合做迭代
- 掌握使用Iterator对集合做迭代
- 掌握Set接口存储特点
- 掌握HashSet类常用方法
- 了解Comparable和Comparator接口
- 掌握Map接口和存储特点
- 掌握HashMap类常用方法
- 掌握Arrays和Collections常用方法

## 27.6. 集合元素迭代（掌握）

### 27.6.1. 集合元素遍历（掌握）

对集合中的每一个元素获取出来。

```
List<String> list = new ArrayList<>();
list.add("西施");
list.add("王昭君");
list.add("貂蝉");
list.add("杨玉环");
```

### 使用for遍历

```
for (int index = 0; index < list.size(); index++) {
    String ele = list.get(index);
    System.out.println(ele);
}
```

### 使用迭代器遍历

Iterator表示迭代器对象，迭代器中拥有一个指针，默认指向第一个元素之前，

- boolean hasNext(): 判断指针后是否存在下一个元素
- Object next(): 获取指针位置下一个元素，获取后指针向后移动一位

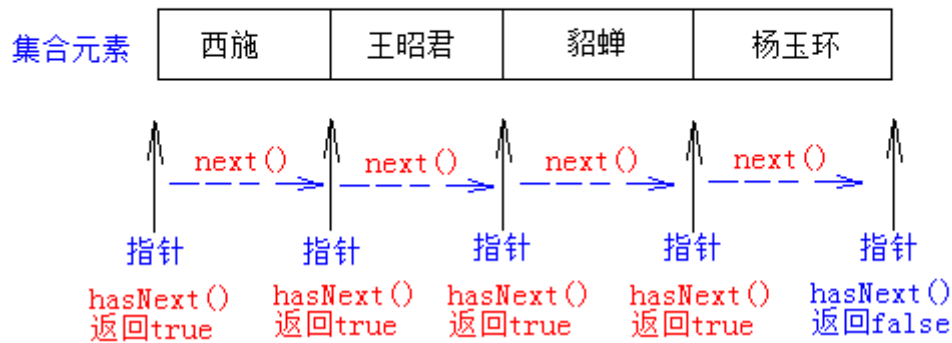
```
Iterator<String> it = list.iterator();
while(it.hasNext()) {
    String ele = it.next();
    System.out.println(ele);
}
```

操作原理如下图：

迭代集合元素：

1：先判断指针位置后面是否有元素 `hasNext()` 为true。

2：如果有，执行`next()`获取下一个元素，且向后移动指针位置。



使用for-each遍历（推荐使用）

语法：

```
for(数据类型 迭代变量： 数组/Iterable实例对象){  
    //TODO  
}
```

这里Iterable实例对象表示Iterable接口实现类对象，其实就是Collection，不包括Map。

```
for (String ele : list) {  
    System.out.println(ele);  
}
```

通过反编译工具会发现，for-each操作集合时，其实底层依然是Iterator，我们直接使用for-each即可。

## 27.6.2. 并发修改异常（了解）

需求：在迭代集合时删除集合元素，比如删除王昭君。

```
List<String> list = new ArrayList<>();  
list.add("西施");  
list.add("王昭君");  
list.add("貂蝉");  
list.add("杨玉环");  
  
System.out.println(list);  
for (String ele : list) {  
    if("王昭君".equals(ele)) {  
        list.remove(ele);  
    }  
}  
System.out.println(list);
```

此时报错`java.util.ConcurrentModificationException`，并发修改异常。

造成该错误的原因是，**不允许在迭代过程中改变集合的长度（不能删除和增加）**。如果要在迭代过程中删除元素，就不能使用集合的remove方法，只能使用迭代器的remove方法，此时只能使用迭代器来操作，不能使用foreach。

```
List<String> list = new ArrayList<>();
list.add("西施");
list.add("王昭君");
list.add("貂蝉");
list.add("杨玉环");
System.out.println(list);

//获取迭代器对象
Iterator<String> it = list.iterator();
while(it.hasNext()) {
    String ele = it.next();
    if("王昭君".equals(ele)) {
        it.remove();
    }
}
System.out.println(list);
```

## 27.7.Set接口（掌握）

Set是Collection子接口，Set接口定义了一种规范，也就是该容器不记录元素的添加顺序，也不允许元素重复，那么Set接口的实现类都遵循这一种规范。

### Set集合存储特点：

- 不允许元素重复
- 不会记录元素的添加先后顺序

Set只包含从Collection继承的方法，不过Set无法记住添加的顺序，不允许包含重复的元素。当试图添加两个相同元素进Set集合，添加操作失败，add()方法返回false。

**Set接口定义了一种规范，也就是该容器不记录元素的添加顺序，也不允许元素重复。**

Set接口常用的实现类有：

- HashSet类：底层采用哈希表实现，开发中使用对多的实现类，重点。
- TreeSet类：底层采用红黑树实现，可以对集合中元素排序，使用不多。

Set 接口常用的方法

- add()
- clear() / remove(E)
- size() / contains

### 27.7.1.HashSet类 (重点)

HashSet 是Set接口的实现类，底层数据结构是哈希表，集合容器不记录元素的添加顺序，也不允许元素重复。通常我们也说HashSet中的元素是**无序的、唯一的**。

HashSet 命名规则：底层数据结构 + 实现接口

需求1：操作Set接口常用方法

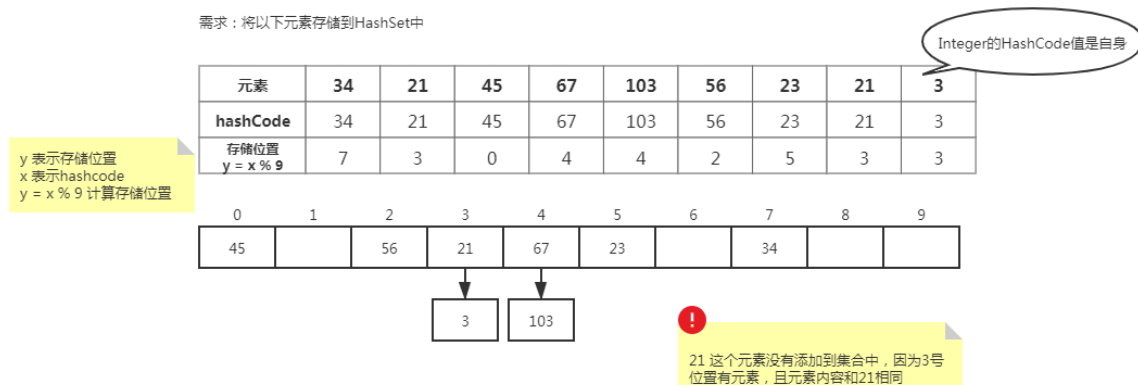
```
public class Test01HashSet {
    public static void main(String[] args) {
        Set<String> set = new HashSet<>();
        //添加操作：向列表中添加4个元素
        set.add("will");
        set.add("wolf");
        set.add("code");
        set.add("Lucy");

        //查询操作：
        System.out.println("集中所有元素：" + set); //[code, wolf, will, Lucy]
        System.out.println("元素数量：" + set.size()); //4
        System.out.println("是否存在某个元素：" + set.contains("code")); //true
        System.out.println("是否存在某个元素：" + set.contains("code2")); //false

        //删除操作：删除code元素
        set.remove("code");
        System.out.println("删除后：" + set); //[wolf, will, Lucy]

        //使用for-each遍历
        for (String ele : set) {
            System.out.println(ele);
        }
        //使用迭代器遍历
        Iterator<String> it = set.iterator();
        while (it.hasNext()) {
            Object ele = it.next();
            System.out.println(ele);
        }
    }
}
```

### 哈希表工作原理



HashSet底层采用哈希表实现，元素对象的hashCode值决定了在哈希表中的存储位置。

向HashSet添加元素时，经过以下过程：

step 1：首先计算要添加元素e的hashCode值，根据  $y = k(x)$  计算出元素在哈希表的存储位置，一般计算位置的函数会选择  $y = \text{hashCode} \% 9$ ，这个函数称为散列函数，可见，散列的位置和添加的位置不一致。

step 2：根据散列出来的位置查看哈希表该位置是否有元素，如果没有元素，直接添加该元素即可。如果有元素，查看e元素和此位置所有元素的是否相等，如果相等，则不添加，如果不相等，在该位置连接e元素即可。

在哈希表中元素对象的hashCode和equals方法的很重要。

每一个存储到哈希表中的对象，都得覆盖hashCode和equals方法用来判断是否是同一个对象，一般的，根据对象的字段数据比较来判断，通常情况下equals为true的时候hashCode也应该相等。

打完收工

IDEA可以根据对象哪些字段做比较而自动生成hashCode和equals方法。

需求2：创建三个User对象，覆盖equals和hashCode方法，存储在HashSet中。

```
class User {
    private String name;
    private int age;

    public User(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + age;
        result = prime * result + ((name == null) ? 0 : name.hashCode());
        return result;
    }

    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        User other = (User) obj;
        if (age != other.age)
            return false;
        if (name == null) {
            if (other.name != null)
                return false;
        } else if (!name.equals(other.name))
            return false;
        return true;
    }

    public String toString() {
```

```

        return "User [name=" + name + ", age=" + age + "]";
    }
}

```

其中equals和hashCode方法表示根据User对象的name和age来做对比，两个方法时IDE工具自动生成的，不要求掌握。

```

public class HashSetDemo2 {
    public static void main(String[] args) {
        Set<User> users = new HashSet<>();
        users.add(new User("will", 17));
        users.add(new User("lucy", 18));
        users.add(new User("stef", 19));
        users.add(new User("allen", 20));
        System.out.println(users);
    }
}

```

### 27.7.2.TreeSet类 (了解)

TreeSet类底层才有红黑树算法，会对存储的元素对象默认使用自然排序（从小到大）。

数据类型	排序规则
BigDecimal, BigInteger, Byte, Double, Float, Integer, Long, Short	按数字大小排序
Character	按字符的Unicode值得数字大小排序
String	按字符串中字符的Unicode值排序

注意：必须保证TreeSet集合中的元素对象是相同的数据类型，否则报错。

```

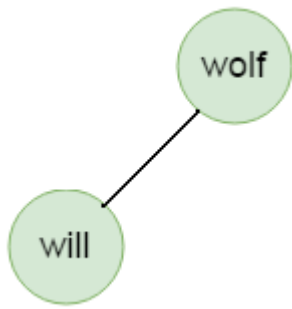
public class TreeSetDemo{
    public static void main(String[] args) {
        Set<String> set = new TreeSet<>();
        set.add("wolf");
        set.add("will");
        set.add("sfef");
        set.add("allen");
        System.out.println(set);// [allen, sfef, will, wolf]
    }
}

```

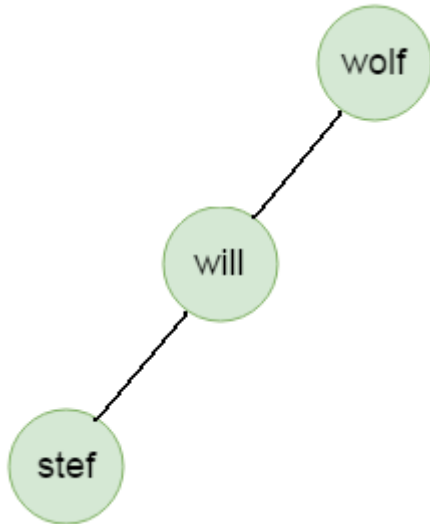
第一步：插入第一个节点，无须比较，直接作为根节点。



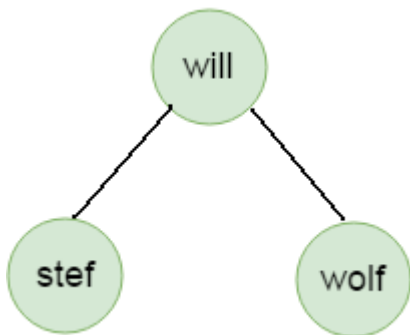
第二步：插入第二个节点，和wolf节点做比较，较小，往左移动。



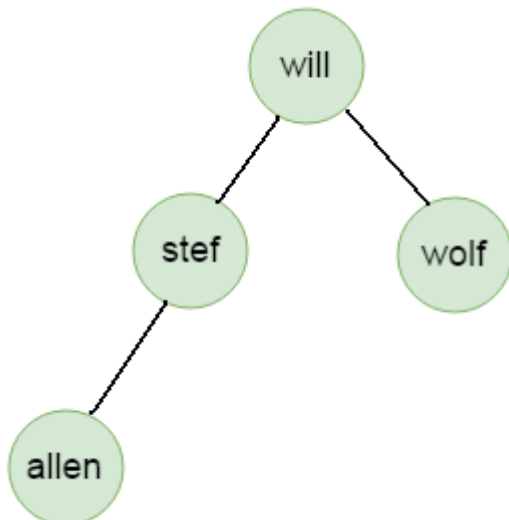
第三步：插入第三个节点，和wolf节点比较，较小，左移，再和will节点比较，较小，再左移。



第四步：由于TreeSet是平衡二叉树，如果树不平衡，会对节点进行调整。



第五步：插入第四个节点，此时和will节点作比较，较小，左移，再和stef节点做比较，较小，左移。



### 27.7.3.Comparable接口 (了解)

缺省情况下, TreeSet中的元素会采用自然排序(从小到大), 此时要求元素对象必须实现 java.util.Comparable接口, 大多数JDK自带的类都实现了该接口, 比如八大包装类和String。

TreeSet会调用元素的compareTo方法来比较元素的大小关系,然后将集合元素按照升序排列。

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

比较规则, 拿当前元素和另一个元素做比较:

- this > o: 返回正整数 1, 优先级较高
- this < o: 返回负整数 -1, 优先级较低
- this == o: 返回 0, 此时认为两个对象为同一个对象。

此时compareTo方法返回0, 则认为两个对象是同一个对象, 返回正数排前面, 返回负数排后面。

如果我们自定义一个类, 需要存储到TreeSet中, 此时我们需要让该类实现Comparable接口, 并覆盖compareTo方法, 在该方法编写比较规则。

需求: 按照用户的年龄从小到大排序

```
class User implements java.lang.Comparable<User> {  
    private String name;  
    private int age;  
    public User(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
    //比较规则  
    public int compareTo(User o) {  
        if (this.age > o.age) {  
            return 1;  
        } else if (this.age < o.age) {  
            return -1;  
        }  
        return 0;  
    }  
    public String toString() {  
        return "User [name=" + name + ", age=" + age + "];"  
    }  
}  
  
public class ComparableDemo{  
    public static void main(String[] args) {  
        Set<User> set = new TreeSet<>();  
        set.add(new User("stef", 28));  
        set.add(new User("will", 17));  
        set.add(new User("allen", 15));  
        set.add(new User("Lucy", 22));  
        System.out.println(set);  
    }  
}
```

输出结果:

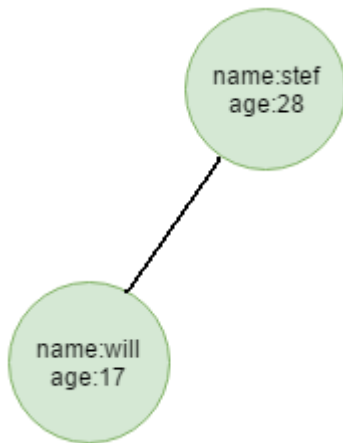


```
[User [name=allen, age=15], User [name=will, age=17], User [name=Lucy, age=22],  
User [name=stef, age=28]]
```

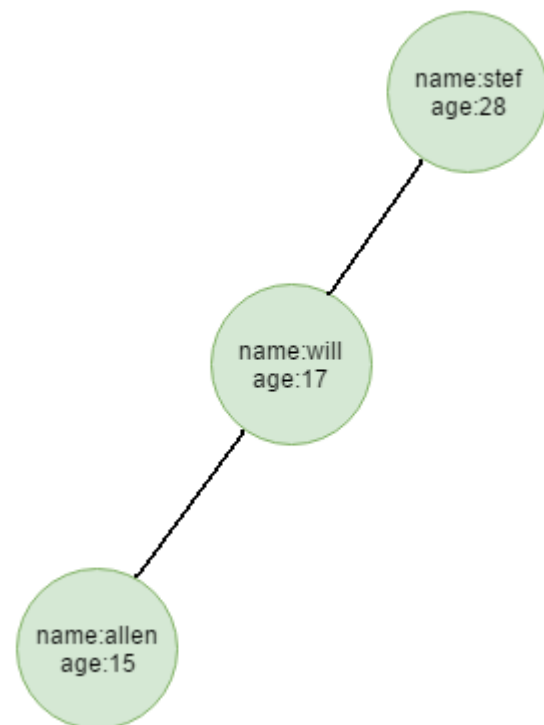
第一步：第一个节点进去，直接作为根节点，无需比较



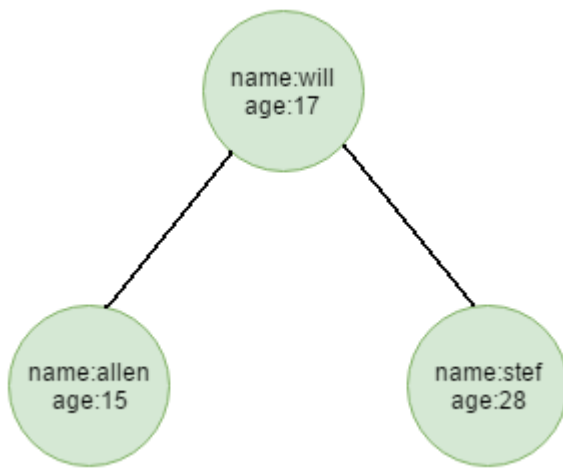
第二步：第二个节点进去和第一个节点作比较，此时的this表示第二个节点，如果compareTo返回-1，则插入节点往左边走。



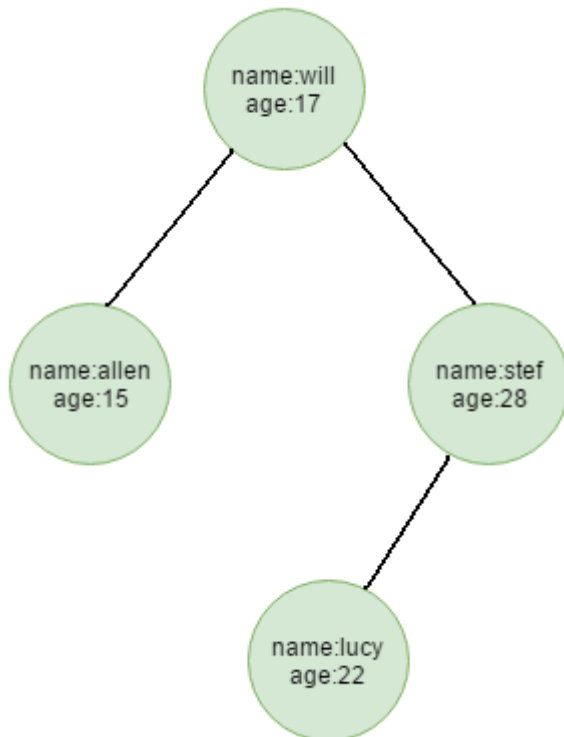
第三步：第三个节点进去和第一个节点做比较，返回-1，再和第二个节点做比较。



第四步：由于TreeSet是平衡二叉树，如果树不平衡，会对节点进行调整。



第五步：第四个节点进去先和will比较，再和stef比较。



#### 27.7.4.Comparator接口（了解）

TreeSet除了默认支持自然排序外，还支持自定义排序，此时需要在构建TreeSet对象时传递java.util.Comparator接口的实现类对象，Comparator表示比较器，里面封装比较规则。

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

此时compare方法返回0，则认为两个对象是同一个对象，返回正数排前面，返回负数排后面。

需求：根据用户的名字长度排序，如果名字相同按照年龄排序。

```
class User {  
    private String name;  
    private int age;  
    public User(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
}
```

```

    }
    public int getAge() {
        return age;
    }
    public String getName() {
        return name;
    }
    public String toString() {
        return "User [name=" + name + ", age=" + age + "]";
    }
}

public class App {
    public static void main(String[] args) {
        Set<User> set = new TreeSet<>(new NameLengthComparator());
        set.add(new User("James", 30));
        set.add(new User("Bryant", 22));
        set.add(new User("Allen", 28));
        set.add(new User("Will", 17));
        System.out.println(set);
    }
}

class NameLengthComparator implements java.util.Comparator<User> {
    public int compare(User o1, User o2) {
        int ret = o1.getName().length() - o2.getName().length();
        if (ret > 0) {
            return 1;
        } else if (ret < 0) {
            return -1;
        } else {
            if (o1.getAge() > o2.getAge()) {
                return 1;
            } else if (o1.getAge() < o2.getAge()) {
                return -1;
            } else {
                return 0;
            }
        }
    }
}

```

输出结果:

```
[User [name=Will, age=17], User [name=Allen, age=28], User [name=James, age=30],
User [name=Bryant, age=22]]
```

小结: HashSet做等值查询效率高, TreeSet做范围查询效率高, 在开发中一般使用HashSet就可以了。

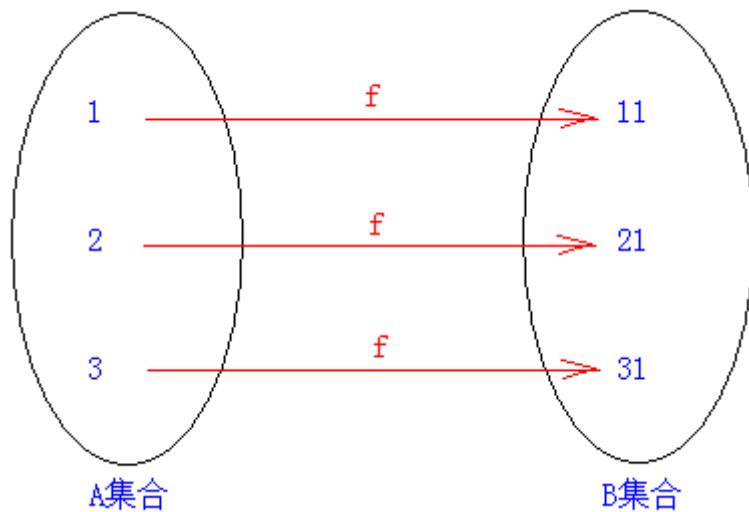
time : 16.30

## 27.8.Map接口（重点）

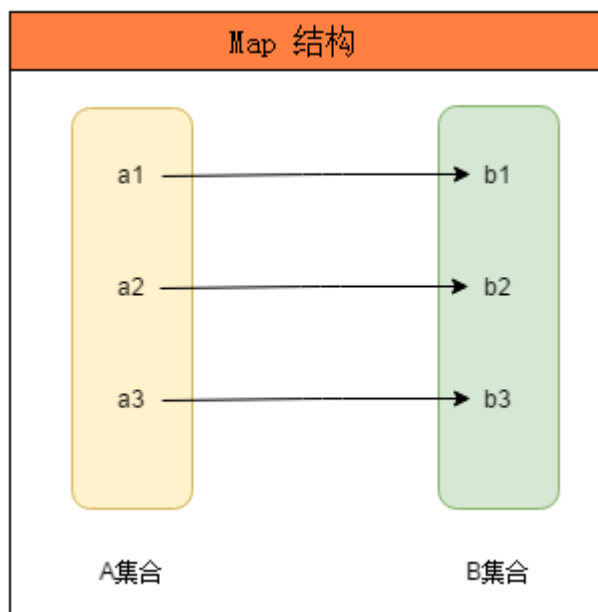
### 27.8.1. 认识Map（理解）

Map，翻译为映射，在数学中的解释为：

设A、B是两个非空集合，如果存在一个法则f，使得A中的每个元素a，按法则f，在B中有唯一确定的元素b与之对应，则称f为从A到B的映射，记作 $f: A \rightarrow B$ 。

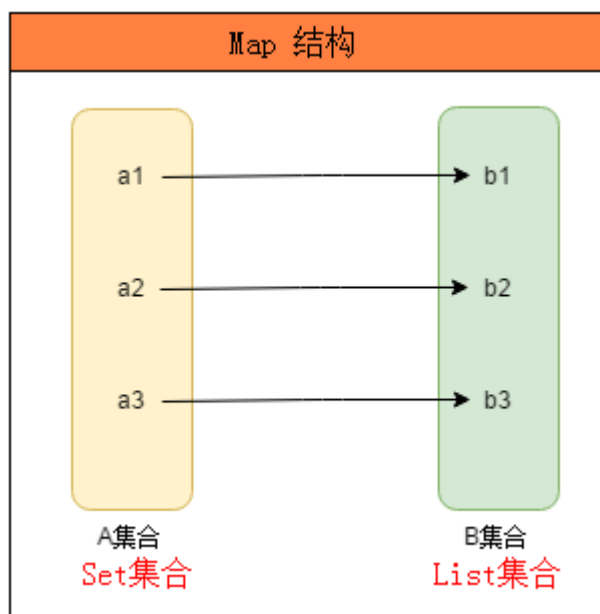


也就是说映射表示两个集合之间各自元素的一种“对应”的关系，在面向对象中使用Map来封装和表示这种关系。



从定义和结构图上，可以看出Map并不是集合，而表示两个集合之间的一种关系，故Map没有实现Collection接口。

在Map中，要求A集合中的每一个元素都可以在B集合中找到唯一的一个值与之对应。这句话可以解读为一个A集合元素只能对应一个B集合元素，也就说A集合中的元素是不允许重复的，B集合中的元素可以重复，也可不重复。那么不难推断出A集合应该是一个Set集合，B集合应该是List集合。

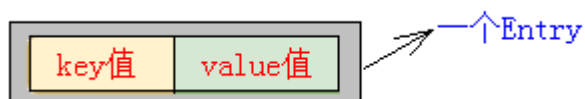


我们把A集合中的元素称之为key，把B集合中的元素称之为value。

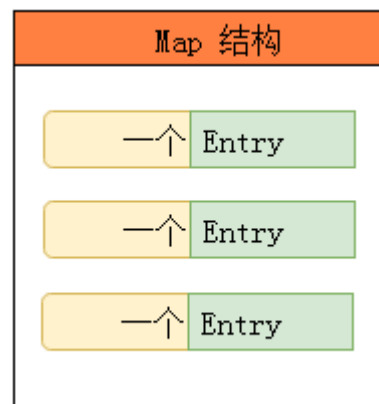
**Map 结构**

key	value
a1	b1
a2	b2
a3	b3
...	...

其实能看出一个Map其实就有多个key-value（键值对）组成的，每一个键值对我们使用Entry表示。



不难发现，一个Map结构也可以理解是Entry的集合，即Set。



一般的，我们依然习惯把Map称之为集合，不过要区分下，Set和List是单元素集合，Map是双元素集合。

- 单元素集合：每次只能存储一个元素，比如Set和List。
- 双元素集合：每次需要存储两个元素（一个key和一个value），比如Map。

注意：

- Map接口并没有继承于Collection接口也没有继承于Iterable接口，所以不能直接对Map使用for-each操作。
- 如果不能理解Map的结构，就直接记住Map每次需要存储两个值，一个是key，一个是value，其中value表示存储的数据，而key就是这一个value的名字。

### 27.8.2.Map常用的API方法（记住）

添加操作

- **boolean put(Object key,Object value): 存储一个键值对到Map中**
- **boolean putAll(Map m): 把m中的所有键值对添加到当前Map中**

删除操作

- **Object remove(Object key): 从Map中删除指定key的键值对，并返回被删除key对应的value**

修改操作

- 无专门的方法，可以调用put方法，存储相同key，不同value的键值对，可以覆盖原来的。

查询操作

- **int size(): 返回当前Map中键值对个数**
- **boolean isEmpty(): 判断当前Map中键值对个数是否为0.**
- **Object get(Object key): 返回Map中指定key对应的value值，如果不存在该key，返回null**
- **boolean containsKey(Object key):判断Map中是否包含指定key**
- **boolean containsValue(Object value):判断Map中是否包含指定value**
- **Set keySet(): 返回Map中所有key所组成的Set集合**
- **Collection values(): 返回Map中所有value所组成的Collection集合**
- **Set entrySet(): 返回Map中所有键值对所组成的Set集合**

注意，标红的是重度使用的方法。

### 27.8.3.HashMap（重点）

HashMap底层基于哈希表算法，Map中存储的key对象的hashCode值决定了在哈希表中的存储位置，因为Map中的key是Set，所以不能保证添加的先后顺序，也不允许重复。

**HashMap key的底层数据结构是哈希表**

需求1：操作Map接口常用方法

```
public class HashMapDemo1{
    public static void main(String[] args) {
        Map<String, String> map = new HashMap<>();
        map.put("girl1", "西施");
        map.put("girl2", "王昭君");
        map.put("girl3", "貂蝉");
        map.put("girl4", "杨玉环");
        System.out.println("map中有多少键值对: "+map.size());
        System.out.println(map);
        System.out.println("是否包含key为girl1: "+map.containsKey("girl1"));
        System.out.println("是否包含value为貂蝉: "+map.containsValue("貂蝉"));
        //替换key为girl3的value值
        map.put("girl3", "小乔");
    }
}
```

```

        System.out.println(map);
        //删除key为girl3的键值对
        map.remove("girl3");
        System.out.println(map);
    }
}

```

Map的迭代遍历:

```

//获取Map中所有的key
Set<String> keys = map.keySet();
System.out.println("Map中所有key: "+keys);

//获取Map中所有的value
Collection<String> values = map.values();
System.out.println("Map中所有value: "+values);

//获取Map中所有的key-value(键值对)
Set<Entry<String, String>> entrys = map.entrySet();
for (Entry<String, String> entry : entrys) {
    String key = entry.getKey();
    String value = entry.getValue();
    System.out.println(key+"->"+value);
}

```

需求2: 统计一个字符串中每个字符出现次数

```

public class HashMapDemo2{
    public static void main(String[] args) {
        String str = "ABCDEFABCDEABCDABCABA";
        //把字符串转换为char数组
        char[] charArray = str.toCharArray();
        //Map的key存储字符, value存储出现的次数
        Map<Character, Integer> map = new HashMap<>();
        //迭代每一个字符
        for (char ch : charArray) {
            //判断Map中是否已经存储该字符
            if (map.containsKey(ch)) {
                Integer count = map.get(ch);
                //如果已经存储该字符, 则把出现次数加上1
                map.put(ch, count+1);
            }else {
                //如果没有存储该字符, 则把设置次数为1
                map.put(ch, 1);
            }
        }
        System.out.println(map);
    }
}

```

#### 27.8.4.TreeMap (了解)

TreeMap key底层基于红黑树算法，因为Map中的key是Set，所以不能保证添加的先后顺序，也不允许重复，但是Map中存储的key会默认使用自然排序（从小到大），和TreeSet一样，除了可以使用自然排序也可以自定义排序。

需求：测试HashMap和TreeMap中key的顺序

```
public class App {
    public static void main(String[] args) {
        Map<String, String> map = new HashMap<>();
        map.put("girl4", "杨玉环");
        map.put("girl2", "王昭君");
        map.put("key1", "西施");
        map.put("key3", "貂蝉");
        System.out.println(map);
        //-----
        map = new TreeMap<>(map);
        System.out.println(map);
    }
}
```

输出结果：

```
{key1=西施, girl4=杨玉环, key3=貂蝉, girl2=王昭君}
{girl2=王昭君, girl4=杨玉环, key1=西施, key3=貂蝉}
```

## 28. 集合框架工具类和方法

### 28.1.Arrays (掌握)

Arrays类是数组的工具类，其中有一个方法比较常用。

- public static List asList(T... a): 该方法可以把一个Object数组转换为List集合。

```
public class ArraysDemo{
    public static void main(String[] args) {
        //把Integer[]转换为List<Integer>
        List<Integer> list1 = Arrays.asList(1, 2, 3);
        System.out.println(list1);
        //把String[]转换为List<String>
        List<String> list2 = Arrays.asList("A", "B", "C");
        System.out.println(list2);
    }
}
```

注意：通过Arrays.asList方法得到的List对象的长度是固定的，不能增，也不能减。



## 28.2.Collections (了解)

Collections是集合的工具类，封装了Set、List、Map操作的工具方法，比如拷贝、排序、搜索、比较大小等。

## 28.3. 斗地主发牌案例 (了解)

按照斗地主游戏的规则，模拟对54张扑克牌的洗牌和发牌以及对手中牌排序操作。

具体规则：

- 将54张不同花色的扑克牌打乱 (♠♥♦♣)
- 三个玩家参与游戏，三人交替摸牌，每人17张牌，最后三张留作底牌
- 查看三人各自手中的牌（按照牌的大小排序）、底牌
- 手中扑克牌从大到小的摆放顺序：大王、小王、2、A、K、Q、J、10、9、8、7、6、5、4、3

```
public class App {
    public static void main(String[] args) {
        //花色♠♥♦♣
        List<String> colors = new ArrayList<>();
        colors.add("♠");
        colors.add("♥");
        colors.add("♦");
        colors.add("♣");

        //数字345678910JQKA2
        List<String> numbers = new ArrayList<>();
        Collections.addAll(numbers, "3", "4", "5", "6", "7", "8", "9", "10", "J",
            "Q", "K", "A", "2");

        //大小王： 大♠ 小♠
        //定义Map，用数字来记录每一张牌
        Map<Integer, String> map = new HashMap<>();
        Integer sequence = 0; //牌的顺序
        //加入其它牌
        for (String number : numbers) {
            for (String color : colors) {
                //一副牌54张，序号范围在[0,51]之间
                if (sequence <= 51) {
                    map.put(sequence, number + "(" + color + ")");
                    sequence++;
                }
            }
        }
        map.put(sequence, "小♠");
        sequence++;
        map.put(sequence, "大♠");
        sequence++;

        //使用0~53来记录每一张牌
        List<Integer> cards = new ArrayList<>(map.keySet());
        Collections.shuffle(cards);
        System.out.println(cards);

        //三个玩家和底牌
        List<Integer> player1 = new ArrayList<Integer>();
```

```

List<Integer> player2 = new ArrayList<Integer>();
List<Integer> player3 = new ArrayList<Integer>();
List<Integer> end = new ArrayList<Integer>();

//发牌
for (int index = 0; index < cards.size(); index++) {
    if (index < 51) {
        int mod = index % 3;//index除以3的余数
        int card = cards.get(index);
        if (mod == 1) {
            player1.add(card);
        } else if (mod == 2) {
            player2.add(card);
        } else {
            player3.add(card);
        }
    } else {
        end.add(cards.get(index));
    }
}

System.out.println(player1);
System.out.println(player2);
System.out.println(player3);
System.out.println(end);

//排序
Collections.sort(player1);
Collections.sort(player2);
Collections.sort(player3);
Collections.sort(end);

//各自手上的牌
List<String> player1Cards = new ArrayList<>();
List<String> player2Cards = new ArrayList<>();
List<String> player3Cards = new ArrayList<>();
List<String> endCards = new ArrayList<>();

for (Integer key : player1) {
    player1Cards.add(map.get(key));
}
for (Integer key : player2) {
    player2Cards.add(map.get(key));
}
for (Integer key : player3) {
    player3Cards.add(map.get(key));
}
for (Integer key : end) {
    endCards.add(map.get(key));
}

//看牌
System.out.println("玩家1: " + player1Cards);
System.out.println("玩家2: " + player2Cards);
System.out.println("玩家3: " + player3Cards);
System.out.println("底牌 : " + endCards);
}

```

```

}

```

---

## 28.4. 集合框架小结

List、Set、Map选用

一般的在存储元素时候，是否需要给元素起一个名字：

- 需要：此时使用Map。
- 不需：存储的元素使用需要保证先后添加的顺序
  - 需要：此时使用List
  - 不需：此时使用Set（如果需要保证集合元素不重复，也选用Set）

2> 根据具体业务场景选择这些接口具体的实现类。