

# 集合框架

集合体系结构

Collection

Map

## 单列集合



## 双列集合

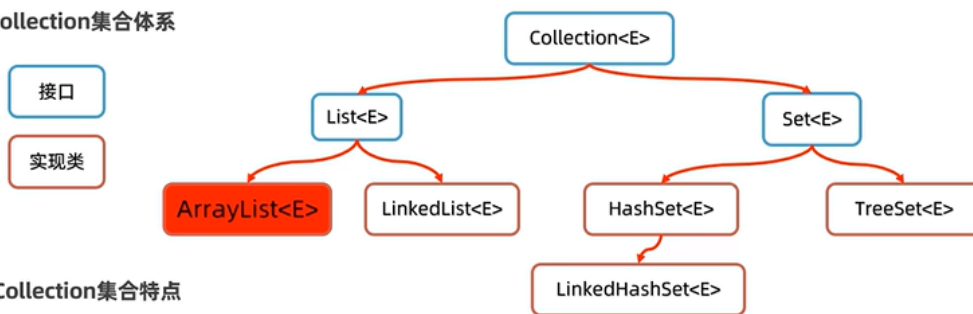
第一个数据

第二个数据

第三个数据



Collection集合体系



Collection集合特点

- **List系列集合**：添加的元素是有序、可重复、有索引。
  - ◆ ArrayList、LinkedList：有序、可重复、有索引。
- **Set系列集合**：添加的元素是无序、不重复、无索引。
  - ◆ HashSet：无序、不重复、无索引；
  - ◆ LinkedHashSet：**有序**、不重复、无索引。
  - ◆ TreeSet：**按照大小默认升序排序**、不重复、无索引。

```
public class demo2 {  
    public static void main(String[] args) {  
        ArrayList<String> list=new ArrayList<>();  
        list.add("java1");  
        list.add("java2");  
        list.add("java1");  
        list.add("java2");  
        list.add("java3");  
        System.out.println(list);  
  
        HashSet<String> set=new HashSet<>();  
        set.add("java1");  
        set.add("java2");  
        set.add("java2");  
    }  
}
```

```

set.add("java1");
set.add("java3");
System.out.println(set);
    }
}

```

```

[java1, java2, java1, java2, java3]
[java3, java2, java1]

```

## 一、Collection集合

它是接口，需要实现类

```

public class demo2 {
    public static void main(String[] args) {
        Collection<String> list=new ArrayList<>();
        //添加函数
        list.add("java1");
        list.add("java2");
        list.add("java1");
        list.add("java2");
        list.add("java3");
        System.out.println(list);

        //清空集合元素
        // list.clear();
        // System.out.println(list);

        //判断是否为空
        System.out.println(list.isEmpty());

        //获取集合大小
        System.out.println(list.size());

        //判断集合是否有某个元素
        System.out.println(list.contains("java1"));
        System.out.println(list.contains("Java1"));

        //删除某个元素，如果有多个重复元素默认删除前面第一个
        System.out.println(list.remove("java1"));
        System.out.println(list);

        //把集合转换为数组
        //toArray转的必是Object类型
        //打印时要转换类型
        Object[] a=list.toArray();
        System.out.println(Arrays.toString(a));//用于将 String[] a 数组转换为字符串形式

        //把一个集合的全部数据倒入到另一个集合中去(拷贝)，两者集合类型要相同
        Collection<String> list1=new ArrayList<>();
        list1.add("java1");
        list1.add("java2");
        list.addAll(list1);//list1到list
        System.out.println(list);
    }
}

```

```
}  
}
```

```
[java1, java2, java1, java2, java3]  
false  
5  
true  
false  
true  
[java2, java1, java2, java3]  
[java2, java1, java2, java3]  
[java2, java1, java2, java3, java1, java2]
```

## 二、List集合

### 1、基本用法

它也是接口，需要实现类

Collection的方法List都继承了

List有索引，多了很多索引操作方法

方法名称	说明
<code>void add(int index,E element)</code>	在此集合中的指定位置插入指定的元素
<code>E remove(int index)</code>	删除指定索引处的元素，返回被删除的元素
<code>E set(int index,E element)</code>	修改指定索引处的元素，返回被修改的元素
<code>E get(int index)</code>	返回指定索引处的元素

```
package r;  
  
import java.text.DateFormat;  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.Calendar;  
import java.util.Collection;  
import java.util.Date; // 导入 Date 类  
import java.util.Random;  
import java.util.HashSet;  
import java.util.List;  
  
public class demo2 {  
    public static void main(String[] args) {  
List<String> list=new ArrayList<>();  
  
list.add("Java1");  
list.add("Java3");
```

```

list.add("Java4");
list.add("Java5");
System.out.println(list);

list.add(1,"Java2");
System.out.println(list);

list.remove(4);
System.out.println(list);

System.out.println(list.get(2));
System.out.println(list);

list.set(3,"Java5");
System.out.println(list);

    }
}

```

```

[Java1, Java3, Java4, Java5]
[Java1, Java2, Java3, Java4, Java5]
[Java1, Java2, Java3, Java4]
Java3
[Java1, Java2, Java3, Java4]
[Java1, Java2, Java3, Java5]

```

## 2、List的遍历

```

package r;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
//注意导入的包

public class demo2 {
    public static void main(String[] args) {
        List<String> list=new ArrayList<>();

        list.add("Java1");
        list.add("Java2");
        list.add("Java3");

        //1、for循环
        for(int i=0;i<list.size();++i){
            String s=list.get(i);
            System.out.println(s);
        }

        //2、迭代器
    }
}

```

```

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

//3、foreach遍历
for(String s : list){
    System.out.println(s);
}

//4、Lambda表达式
list.forEach(s->{
    System.out.println(s);
});

    }
}

```

### 3、ArrayList类

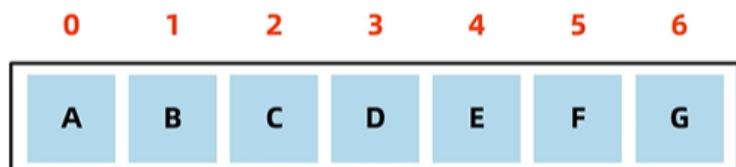
ArrayList与LinkedList底层使用的数据结构不同

#### 1、ArrayList集合的底层原理

- 基于数组实现的。

数组的特点

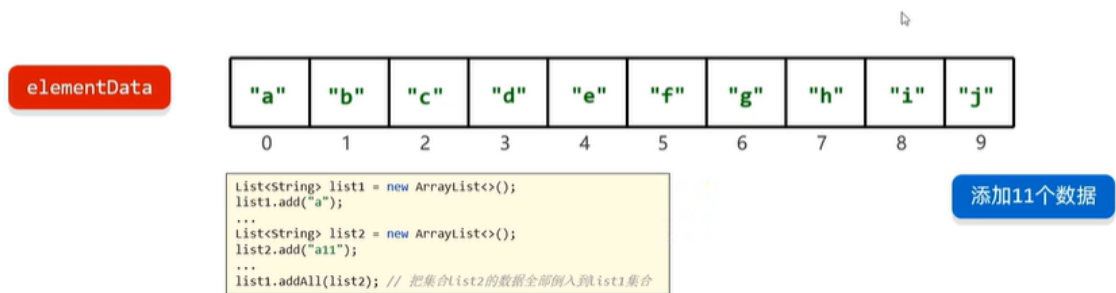
查询快、增删慢



- **查询速度快**（注意：是根据索引查询数据快）：查询数据通过地址值和索引定位，查询任意数据耗时相同。
- **删除效率低**：可能需要把后面很多的数据进行前移。
- **添加效率极低**：可能需要把后面很多的数据后移，再添加元素；或者也可能需要进行数组的扩容。

## 1、ArrayList集合的底层原理

- ① 利用无参构造器创建的集合，会在底层创建一个默认长度为0的数组
- ② 添加第一个元素时，底层会创建一个新的长度为10的数组
- ③ 存满时，会扩容1.5倍
- ④ 如果一次添加多个元素，1.5倍还放不下，则新创建数组的长度以实际为准



## 4、LinkedList

他是以双向链表储存的

查询慢，增删快

它新增了很多首尾操作的特有方法

方法名称	说明
<code>public void addFirst(E e)</code>	在该列表开头插入指定的元素
<code>public void addLast(E e)</code>	将指定的元素追加到此列表的末尾
<code>public E getFirst()</code>	返回此列表中的第一个元素
<code>public E getLast()</code>	返回此列表中的最后一个元素
<code>public E removeFirst()</code>	从此列表中删除并返回第一个元素
<code>public E removeLast()</code>	从此列表中删除并返回最后一个元素

注意：`addFirst()`和`push()`差不多。

`removeFirst()`和`pop()`差不多

## 三、Set集合

无序，不重复，无索引

```
import java.util.HashSet;
import java.util.LinkedHashSet;
import java.util.Set;
import java.util.TreeSet;

public class w {

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

```

Set<Integer> s=new HashSet<>();//无序不重复，无索引
s.add(1);
s.add(8);
s.add(1);
s.add(2);
s.add(3);
s.add(8);
System.out.println(s);

Set<Integer> l=new LinkedHashSet<>();//有序不重复，无索引
l.add(1);
l.add(8);
l.add(1);
l.add(2);
l.add(3);
l.add(8);
System.out.println(l);

Set<Integer> t=new TreeSet<>();//可序不重复，无索引 升序排序
t.add(1);
t.add(8);
t.add(1);
t.add(2);
t.add(3);
t.add(8);
System.out.println(t);

}
}

```

```

[1, 2, 3, 8]
[1, 8, 2, 3]
[1, 2, 3, 8]

```

**注意：**

**Set要用到的常用方法，基本上就是Collection提供的！！**

**自己几乎没有额外新增一些常用功能！**

## 1、HashSet集合

## 哈希值

- 就是一个int类型的数值，Java中每个对象都有一个哈希值。
- Java中的所有对象，都可以调用Object类提供的hashCode方法，返回该对象自己的哈希值。

`public int hashCode()`: 返回对象的哈希码值。

## 对象哈希值的特点

- 同一个对象多次调用hashCode()方法返回的哈希值是相同的。
- 不同的对象，它们的哈希值一般不相同，但也有可能会相同(哈希碰撞)。

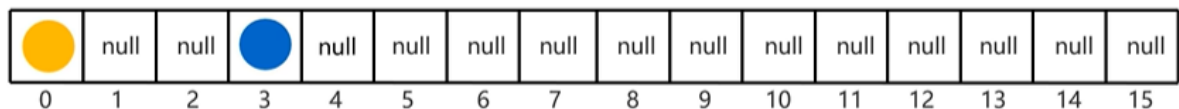
int (-21亿多 ~ 21亿多)

45亿个对象

JDK8前：数组+链表

JDK8之前HashSet集合的底层原理，基于哈希表：数组+链表

table[]



- ① 创建一个默认长度16的数组，默认加载因子为0.75，数组名table
- ② 使用元素的哈希值对数组的长度求余计算出应存入的位置
- ③ 判断当前位置是否为null，如果是null直接存入
- ④ 如果不为null，表示有元素，则调用equals方法比较

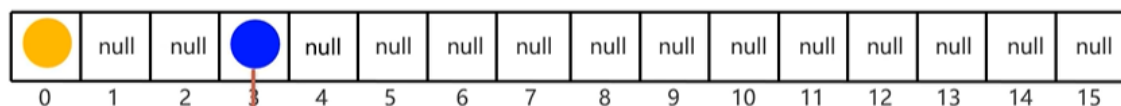
相等，则不存；不相等，则存入数组

- JDK 8之前，新元素存入数组，占老元素位置，老元素挂下面
- JDK 8开始之后，新元素直接挂在老元素下面

哈希表是一种增删改查数据都较好的数据结构

JDK8之前HashSet集合的底层原理，基于哈希表：数组+链表

table[]



- ① 创建一个默认长度16的数组，默认加载因子为0.75，数组名table
- ② 使用元素的哈希值  $16 * 0.75 = 12$  的位置
- ③ 判断当前位置是否为null，如果是null直接存入
- ④ 如果不为null，表示有元素，则调用equals方法比较

相等，则不存；不相等，则存入数组

、如果数组快占满了，会出什么问题？该咋办？

链表会过长，导致查询性能降低

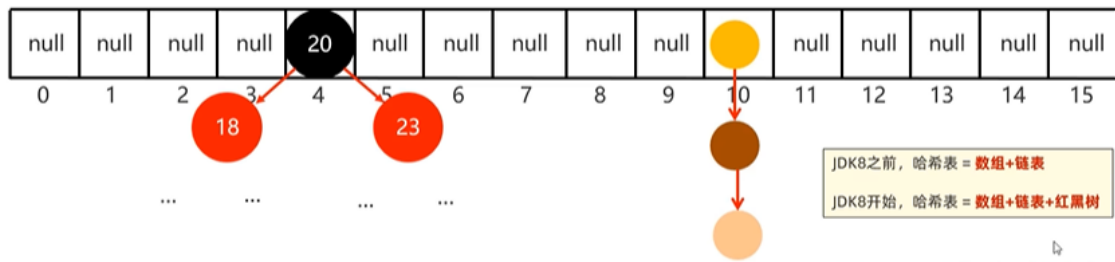
扩容

- JDK 8之前，新元素存入数组，占老元素位置，老元素挂下面
- JDK 8开始之后，新元素直接挂在老元素下面



JDK8开始HashSet集合的底层原理，基于哈希表：数组 + 链表 + 红黑树

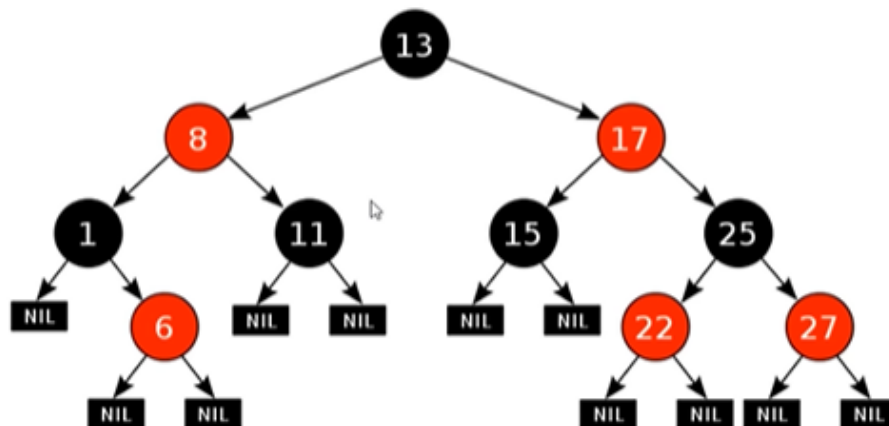
Segment[]



JDK8开始，当链表长度超过8，且数组长度 $\geq 64$ 时，自动将链表转成红黑树

小结：JDK8开始后，哈希表中引入了红黑树后，进一步提高了操作数据的性能。

- 红黑树，就是可以自平衡的二叉树
- 红黑树是一种增删改查数据性能相对都较好的结构。

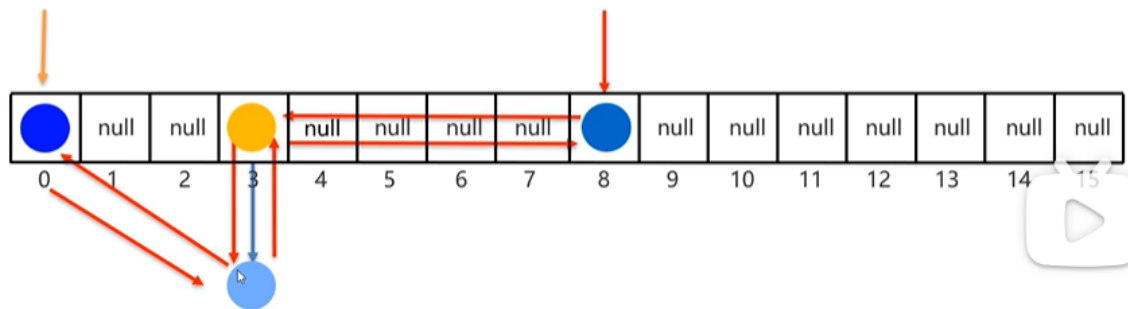


HashSet集合默认不能对内容一样的两个不同的对象去重复

## 2、LinkedHashSet集合

## LinkedHashSet底层原理

- 依然是基于哈希表(数组、链表、红黑树)实现的。
- 但是，它的每个元素都额外的多了一个双链表的机制记录它前后元素的位置。



内存大一点

## 3、TreeSet集合

### TreeSet

- 特点：不重复、无索引、可排序（默认升序排序，按照元素的大小，由小到大排序）
- 底层是基于红黑树实现的排序。

对于Integer,Double，默认按照数值本身大小进行升序排序

字符串编号排序

对于自定义对象，默认无法排序

### 自定义排序规则

- TreeSet集合存储自定义类型的对象时，必须指定排序规则，支持如下两种方式来指定比较规则。

#### 方式一

- 让自定义的类（如学生类）实现Comparable接口，重写里面的compareTo方法来指定比较规则。

#### 方式二

- 通过调用TreeSet集合有参数构造器，可以设置Comparator对象（比较器对象，用于指定比较规则）。

```
public TreeSet(Comparator<? super E> comparator)
```

## 四、Collections的其它知识

## 1、可变参数

```
public class w {

    public static void main(String[] args) {
        //特点:
        test();//不传数据
        test(10);//传输一个数据给它
        test(new int[]{10,20,30,40});//传输一个数值可变参数

    }

    //一个形参列表中，只能有一个可变参数
    //可变参数必须放在形参列表的最后面
    public static void test (int...nums/*可变参数 */){
        //可变参数在方法内部，本质就是一个数组
        System.out.println(nums.length);
        System.out.println(Arrays.toString(nums));
        System.out.println("-----");
    }
}
```

## 2、Collections工具类

### Collections

- 是一个用来操作集合的工具类

### Collections提供的常用静态方法

方法名称	说明
<code>public static &lt;T&gt; boolean addAll(Collection&lt;? super T&gt; c, T... elements)</code>	给集合批量添加元素
<code>public static void shuffle(List&lt;?&gt; list)</code>	打乱List集合中的元素顺序
<code>public static &lt;T&gt; void sort(List&lt;T&gt; list)</code>	对List集合中的元素进行升序排序
<code>public static &lt;T&gt; void sort(List&lt;T&gt; list, Comparator&lt;? super T&gt; c)</code>	对List集合中元素，按照比较器对象指定的规则进行排序

```
public class w {

    public static void main(String[] args) {
        List<String> l=new ArrayList<>();
        Collections.addAll(l,"java1","java2","java3","java4","java5");
        System.out.println(l);
    }
}
```

```

Collections.shuffle(l);
System.out.println(l);

Collections.sort(l);
System.out.println(l);
}
}

```

```

[java1, java2, java3, java4, java5]
[java3, java2, java1, java5, java4]
[java1, java2, java3, java4, java5]

```

#### 排序方式1:

方法名称	说明
<code>public static &lt;T&gt; void sort(List&lt;T&gt; list)</code>	对List集合中元素按照默认规则排序

注意：本方法可以直接对自定义类型的List集合排序，但自定义类型必须实现了Comparable接口，指定了比较规则才可以。

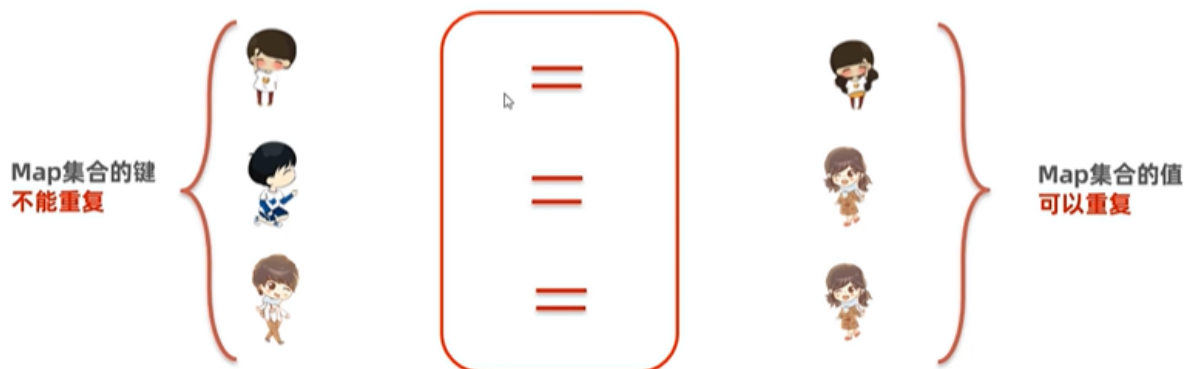
#### 排序方式2:

方法名称	说明
<code>public static &lt;T&gt; void sort(List&lt;T&gt; list, Comparator&lt;? super T&gt; c)</code>	对List集合中元素，按照比较器对象指定的规则进行排序

## 五、Map集合

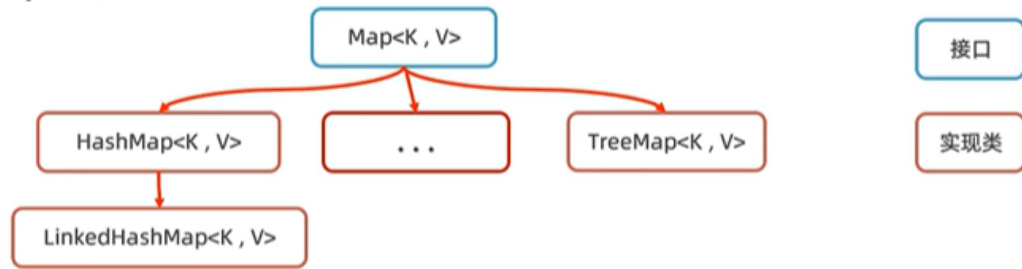
### 认识Map集合

- Map集合称为双列集合，格式：{key1=value1, key2=value2, key3=value3, ...}，一次需要存一对数据做为一个元素。
- Map集合的每个元素“key=value”称为一个键值对/键值对对象/一个Entry对象，Map集合也被叫做“**键值对集合**”
- Map集合的所有键是不允许重复的，但值可以重复，键和值是一一对应的，每一个键只能找到自己对应的值



存储一一对应的数据时，就可以考虑用Map集合来做

## Map集合体系



## Map集合体系的特点

注意：Map系列集合的特点都是由键决定的，值只是一个附属品，值是不做要求的

- HashMap（由键决定特点）：无序、不重复、无索引；（用的最多）
- LinkedHashMap（由键决定特点）：由键决定的特点：有序、不重复、无索引。
- TreeMap（由键决定特点）：按照大小默认升序排序、不重复、无索引。

```
public class w {  
  
    public static void main(String[] args) {  
        Map<String,Integer> m0=new HashMap<>(); //无序，不重复索引  
        m0.put("Java1", 9);  
        m0.put("Java2", 8);  
        m0.put("Java3", 7);  
        m0.put("Java4", 6);  
        m0.put("Java5", 5);  
        m0.put(null, null);  
        System.out.println(m0);  
  
        Map<String,Integer> m1=new LinkedHashMap<>(); //有序，不重复索引  
        m1.put("Java1", 9);  
        m1.put("Java2", 8);  
        m1.put("Java3", 7);  
        m1.put("Java4", 6);  
        m1.put("Java5", 5);  
        m1.put(null, null);  
        System.out.println(m1);  
  
        //注：TreeMap不允许键为null  
        Map<String,Integer> m2=new TreeMap<>(); //可排序，不重复索引  
        m2.put("Java1", 9);  
        m2.put("Java2", 8);  
        m2.put("Java3", 7);  
        m2.put("Java4", 6);  
        m2.put("Java5", 5);  
        System.out.println(m2);  
    }  
}
```

## 1、常用方法

方法名称	说明
<code>public V put(K key,V value)</code>	添加元素
<code>public int size()</code>	获取集合的大小
<code>public void clear()</code>	清空集合
<code>public boolean isEmpty()</code>	判断集合是否为空，为空返回true，反之
<code>public V get(Object key)</code>	根据键获取对应值
<code>public V remove(Object key)</code>	根据键删除整个元素
<code>public boolean containsKey(Object key)</code>	判断是否包含某个键
<code>public boolean containsValue(Object value)</code>	判断是否包含某个值
<code>public Set&lt;K&gt; keySet()</code>	获取全部键的集合
<code>public Collection&lt;V&gt; values()</code>	获取Map集合的全部值

```
public class w {  
  
    public static void main(String[] args) {  
        Map<String,Integer> m0=new HashMap<>(); //无序，不重复索引  
        m0.put("Java1", 9);  
        m0.put("Java2", 8);  
        m0.put("Java3", 7);  
        m0.put("Java4", 6);  
        m0.put("Java5", 5);  
        m0.put(null, null);  
        System.out.println(m0);  
  
        System.out.println(m0.size());  
  
        System.out.println(m0.isEmpty());  
  
        System.out.println(m0.get("Java3"));  
        System.out.println(m0.get("Java6"));  
  
        System.out.println(m0.remove("Java5"));  
        System.out.println(m0);  
  
        System.out.println(m0.containsKey("Java1"));  
        System.out.println(m0.containsKey("Java6"));  
  
        System.out.println(m0.containsKey("9"));  
  
        Set<String> s=m0.keySet();  
        System.out.println(s);  
  
        Collection<Integer> c=m0.values();  
        System.out.println(c);  
  
    }  
}
```

```
}
```

```
{Java2=8, null=null, Java3=7, Java4=6, Java5=5, Java1=9}  
6  
false  
7  
null  
5  
{Java2=8, null=null, Java3=7, Java4=6, Java1=9}  
true  
false  
false  
[Java2, null, Java3, Java4, Java1]  
[8, null, 7, 6, 9]
```

## 2、遍历方式

### Map集合的遍历方式

#### 键找值 01

先获取Map集合全部的  
键，再通过遍历键来找值

#### 键值对 02

把“键值对”看成一个整  
体进行遍历（难度较大）

#### Lambda 03

JDK 1.8开始之后的新技  
术（非常的简单）



## Map集合的遍历方式一

需要用到Map的如下方法：

### 键找值 01

先获取Map集合全部的  
键，再通过遍历键来找值

方法名称	说明
<code>public Set&lt;K&gt; keySet()</code>	获取所有键的集合
<code>public V get(Object key)</code>	根据键获取其对应的值

```
public class w {  
  
    public static void main(String[] args) {  
        Map<String,Integer> m0=new HashMap<>();  
        m0.put("Java1", 9);  
        m0.put("Java2", 8);  
        m0.put("Java3", 7);  
        m0.put("Java4", 6);  
        m0.put("Java5", 5);  
        //m0.put(null, null);  
        System.out.println(m0);  
  
        Set<String> s=m0.keySet();  
        //foreach遍历  
        for(String i:s){  
            int k=m0.get(i);  
            System.out.println(i+"==>" +k);  
        }  
  
    }  
}
```



## Map集合的遍历方式二：键值对

```
Map<String, Double> map = new HashMap<>();
map.put("蜘蛛精", 169.8);
map.put("紫霞", 165.8);
map.put("至尊宝", 169.5);
map.put("牛魔王", 183.6);
System.out.println(map);
// map = {蜘蛛精=169.8, 牛魔王=183.6, 至尊宝=169.5, 紫霞=165.8}
```

```
Set< Map.Entry<String, Double> > entries = map.entrySet();
```

```
for (Map.Entry<String, Double> entry : entries) {
    String key = entry.getKey();
    double value = entry.getValue();
    System.out.println(key + "====>" + value);
}
```

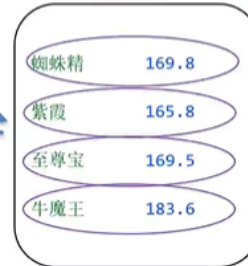
### Map提供的方法

Map提供的方法	说明
Set<Map.Entry<K, V>> entrySet()	获取所有“键值对”的集合

### Map.Entry提供的方法

Map.Entry提供的方法	说明
K getKey()	获取键
V getValue()	获取值

Set< Map.Entry<String, Double> >



Entry对象

Map.Entry<String, Double>

```
public class w {
```

```
    public static void main(String[] args) {
        Map<String,Integer> m0=new HashMap<>();
        m0.put("Java1", 9);
        m0.put("Java2", 8);
        m0.put("Java3", 7);
        m0.put("Java4", 6);
        m0.put("Java5", 5);
        //m0.put(null, null);
    }
}
```

```
System.out.println(m0);
```

//1、调用Map集合提供entrySet方法，把Map集合转换为键值对类型的set集合

```
Set<Map.Entry<String,Integer>> e=m0.entrySet();
```

```
for(Map.Entry<String,Integer> i:e){
    String key=i.getKey();
    int value=i.getValue();
    system.out.println(key+"==>" +value);
}
```

```
}
```

```
}
```

```
}
```

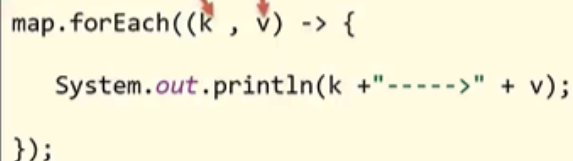
## Map集合的遍历方式三：Lambda

- 需要用到Map的如下方法

方法名称	说明
<code>default void forEach(BiConsumer&lt;? super K, ? super V&gt; action)</code>	结合lambda遍历Map集合

### 流程

**map = {蜘蛛精=169.8, 紫霞=165.8, 至尊宝=169.5, 牛魔王=183.6}**



```
map.forEach((k , v) -> {  
    System.out.println(k + "---->" + v);  
});
```

## 3、HashMap集合

### HashMap集合的底层原理

- HashMap跟HashSet的底层原理是一模一样的，都是基于哈希表实现的。

**实际上：原来学的Set系列集合的底层就是基于Map实现的，只是Set集合中的元素只要键数据，不要值数据而已。**

```
public HashSet() {  
    map = new HashMap<>();  
}
```

### 哈希表

- JDK8之前，哈希表 = 数组+链表
- JDK8开始，哈希表 = 数组+链表+红黑树
- 哈希表是一种增删改查数据，性能都较好的数据结构。

利用键计算哈希值，跟值无关

## 4、LinkedHashMap集合

底层原理好像和LinkedHashSet是一样的

## 5、TreeMap集合

底层原理和TreeSet是一样的