集合框架

Author：zhangzhang

Version：1.0.0

一、集合概念

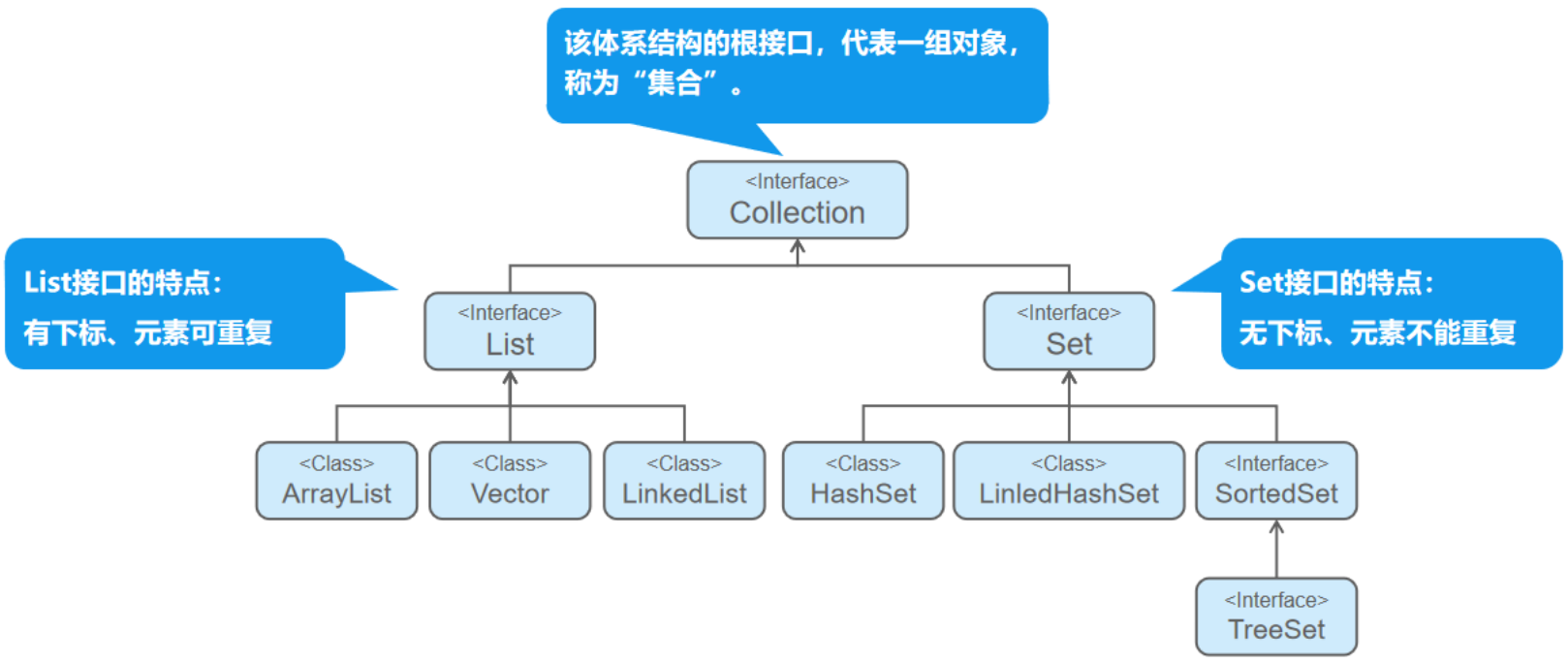
- 对象的容器，定义了对多个对象进行操作的常用方法。可实现数组的功能。
- 位置：java.util.*;

集合和数组区别：

- 数组长度固定，集合长度不固定。
- 数组可以存储基本类型和引用类型，集合只能存储引用类型。

二、Collection体系集合

Collection体系集合



Collection父接口：

- 特点：代表一组任意类型的对象，无序、无下标。

常用方法：

方法	描述
boolean add(Object obj)	添加一个对象数据
boolean addAll(Collection c)	将一个集合中的所有对象添加到此集合中
void clear()	清空此集合中的所有对象
boolean contains(Object o)	检查此集合中是否包含o对象
boolean equals(Object o)	比较此集合是否与指定对象相等
boolean isEmpty()	判断此集合是否为空
boolean remove(Object o)	在此集合中移除o对象
int size()	返回此集合中的元素个数
Object[] toArray()	将此集合转换成数组

> 案例演示：保存简单数据。

```
public class TestCollection1 {
  public static void main(String[] args) {
    //创建集合
    Collection collection=new ArrayList();
    //1添加元素
    collection.add("苹果");
    collection.add("西瓜");
    collection.add("榴莲");
    System.out.println("元素个数:"+collection.size());
    System.out.println(collection);
    //2删除元素
    //collection.remove("榴莲");
    //collection.clear();
    //System.out.println("删除之后:"+collection.size());
    //3遍历元素【重点】
    //3.1使用增强for
    System.out.println("------3.1使用增强for-------");
    for (Object object : collection) {
      System.out.println(object);
    }
    //3.2使用迭代器(迭代器专门用来遍历集合的一种方式)
    //hasNext();有没有下一个元素,
    //next();获取下一个元素
    //remove();删除当前元素
    System.out.println("------3.2使用增强for-------");
    Iterator it=collection.iterator();
    while(it.hasNext()) {
      String s=(String)it.next();
      System.out.println(s);
      //不能使用collection删除方法，因为遍历同时不能使用集合删除方法，否则出现并发修改异常。
            //可以使用迭代器的删除方法
      //collection.remove(s);
      //it.remove();
    }
    System.out.println("元素个数:"+collection.size());
    //4判断
    System.out.println(collection.contains("西瓜"));
    System.out.println(collection.isEmpty());
  }
}
```

注：遍历同时不能使用集合删除方法，否则出现并发修改异常，可使用迭代器的删除方法。

> 案例演示：保存对象数据。

```
public class Student {
  private String name;
  private int age;
  public Student() {
    // TODO Auto-generated constructor stub
  }
  public Student(String name, int age) {
    super();
    this.name = name;
    this.age = age;
  }
  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }
  public int getAge() {
    return age;
```

```
  }
  public void setAge(int age) {
    this.age = age;
  }
  @Override
  public String toString() {
    return "Student [name=" + name + ", age=" + age + "]";
  }

  @Override
  public boolean equals(Object obj) {
    //1判断是不是同一个对象
    if(this==obj) {
      return true;
    }
    //2判断是否为空
    if(obj==null) {
      return false;
    }
    //3判断是否是Student类型
    if(obj instanceof Student) {
      Student s=(Student)obj;
      //4比较属性
      if(this.name.equals(s.getName())&&this.age==s.getAge()) {
        return true;
      }
    }
    //5不满足条件返回false
    return false;
  }

}
```

```
public class TestCollection2 {
  public static void main(String[] args) {
    //新建Collection对象
    Collection collection=new ArrayList();
    Student s1=new Student("张三", 20);
    Student s2=new Student("张无忌", 18);
    Student s3=new Student("王二", 22);
    //1添加数据
    collection.add(s1);
    collection.add(s2);
    collection.add(s3);
    System.out.println("元素个数:"+collection.size());
    System.out.println(collection.toString());
    //2删除
    //collection.remove(s1);
    //collection.remove(new Student("王二", 22));
//    collection.clear();
//    System.out.println("删除之后:"+collection.size());
    //3遍历
    //3.1 增强for
    System.out.println("-------增强for----");
    for (Object object : collection) {
      Student s=(Student)object;
      System.out.println(s.toString());
    }
    //3.2迭代器: hasNext()  next();  remove(); 迭代过程中不能使用使用collection的删除方法
    System.out.println("-------迭代器----");
    Iterator it=collection.iterator();
    while(it.hasNext()) {
      Student s=(Student)it.next();
      System.out.println(s.toString());
    }
    //4判断
    System.out.println(collection.contains(s1));
    System.out.println(collection.isEmpty());

  }
}
```

三、List接口与实现类【 重点 】

3.1 List接口

> - 特点：有序、有下标、元素可以重复。
> - 继承Collection接口。

> 常用方法：

方法	描述
void add(int index, Object o)	在index位置插入对象o。
boolean addAll(int index, Collection c)	将一个集合中的元素添加到此集合中的index位置。
Object get(int index)	返回集合中指定位置的元素。
List subList(int fromIndex, int toIndex)	返回fromIndex和toIndex之间的集合元素。

> 案例演示：

```
public class TestList {
  public static void main(String[] args) {
    //先创建集合对象
    List list=new ArrayList();
    //1添加元素
    list.add("苹果");
    list.add("小米");
    list.add(0, "华为");
    System.out.println("元素个数:"+list.size());
    System.out.println(list.toString());
    //2删除元素
    //list.remove("苹果");
//    list.remove(0);
//    System.out.println("删除之后:"+list.size());
//    System.out.println(list.toString());
    //3遍历
    //3.1使用for遍历
    System.out.println("-----3.1使用for遍历-----");
    for(int i=0;i<list.size();i++) {
      System.out.println(list.get(i));
    }
    //3.2使用增强for
    System.out.println("-----3.2使用增强for-----");
    for (Object object : list) {
      System.out.println(object);
    }
    //3.3使用迭代器
    Iterator it=list.iterator();
    System.out.println("-----3.3使用迭代器-----");
    while(it.hasNext()) {
      System.out.println(it.next());
    }
    //3.4使用列表迭代器 ,和Iterator的区别，ListIterator可以向前或向后遍历，添加、删除、修改元素
    ListIterator lit=list.listIterator();
    System.out.println("------使用列表迭代器从前往后-------");
    while(lit.hasNext()) {
      System.out.println(lit.nextIndex()+":"+lit.next());
    }
    System.out.println("------使用列表迭代器后往前-------");
    while(lit.hasPrevious()) {
      System.out.println(lit.previousIndex()+":"+lit.previous());
    }

    //4判断
    System.out.println(list.contains("苹果"));
    System.out.println(list.isEmpty());

    //5获取位置
    System.out.println(list.indexOf("华为"));
  }
}
```

3.2 List实现类

3.2.1 ArrayList

- 数组结构实现，查询快、增删慢。
- JDK1.2版本、线程不安全。

> 案例演示：ArrayList保存对象。

```
public class TestArrayList {
  public static void main(String[] args) {
    //创建集合  size 0  容量 0, 扩容原来的1.5倍
    ArrayList arrayList=new ArrayList<>();
    //1添加元素
    Student s1=new Student("刘德华", 20);
    Student s2=new Student("郭富城", 22);
    Student s3=new Student("梁朝伟", 18);
    arrayList.add(s1);
    arrayList.add(s2);
    arrayList.add(s3);
    System.out.println("元素个数:"+arrayList.size());
    System.out.println(arrayList.toString());
```

```
    //2删除元素
//    arrayList.remove(new Student("刘德华", 20));//equals(this==obj)
//    System.out.println("删除之后:"+arrayList.size());

    //3遍历元素【重点】
    //3.1使用迭代器
    System.out.println("-------3.1使用迭代器-----");
    Iterator it=arrayList.iterator();
    while(it.hasNext()) {
      Student s=(Student)it.next();
      System.out.println(s.toString());
    }
    //3.2列表迭代器
    ListIterator lit=arrayList.listIterator();
    System.out.println("-------3.2使用列表迭代器---------");
    while(lit.hasNext()) {
      Student s=(Student)lit.next();
      System.out.println(s.toString());
    }

    System.out.println("-------3.2使用列表迭代器逆序---------");
    while(lit.hasPrevious()) {
      Student s=(Student)lit.previous();
      System.out.println(s.toString());
    }

    //4判断
    System.out.println(arrayList.contains(new Student("梁朝伟", 18)));
    System.out.println(arrayList.isEmpty());

    //5查找
    System.out.println(arrayList.indexOf(new Student("梁朝伟", 18)));
  }
}
```

> 案例演示：ArrayList保存包装类型数据。

```
public class Demo4 {
  public static void main(String[] args) {
    //创建集合
    List list=new ArrayList();
    //1添加数字数据(自动装箱)
    list.add(20);
    list.add(30);
    list.add(40);
    list.add(50);
    list.add(60);
    System.out.println("元素个数:"+list.size());
    System.out.println(list.toString());
    //2删除操作
    //list.remove(0);
//    list.remove(new Integer(20));
//    System.out.println("删除元素:"+list.size());
//    System.out.println(list.toString());

    //3补充方法subList：返回子集合,含头不含尾
    List subList=list.subList(1, 3);
    System.out.println(subList.toString());

  }
}
```

3.2.2 LinkedList

> - 链表结构实现，增删快，查询慢。
> - JDK1.2版本、线程不安全。

> 案例演示：LinkedList保存对象数据。

```
public class Demo2 {
  public static void main(String[] args) {
    //创建集合
    LinkedList linkedList=new LinkedList();
    //1添加元素
    Student s1=new Student("刘德华", 20);
    Student s2=new Student("郭富城", 22);
    Student s3=new Student("梁朝伟", 18);
    linkedList.add(s1);
    linkedList.add(s2);
    linkedList.add(s3);

    System.out.println("元素个数:"+linkedList.size());
    System.out.println(linkedList.toString());

    //2删除
```

```
//    linkedList.remove(new Student("刘德华", 20));
//    System.out.println("删除之后:"+linkedList.size());
//    linkedList.clear();
    //3遍历
    //3.1for遍历
    System.out.println("-----for----------");
    for(int i=0;i<linkedList.size();i++) {
      System.out.println(linkedList.get(i));
    }
    //3.2增强for
    System.out.println("-----增强for----------");
    for (Object object : linkedList) {
      Student s=(Student)object;
      System.out.println(s.toString());
    }
    //3.3使用迭代器
    System.out.println("-----使用迭代器----------");
    Iterator it=linkedList.iterator();
    while(it.hasNext()) {
      Student s=(Student)it.next();
      System.out.println(s.toString());
    }
    //3.4-使用列表迭代器
    System.out.println("-----使用列表迭代器----------");
    ListIterator lit=linkedList.listIterator();
    while(lit.hasNext()) {
      Student s=(Student)lit.next();
      System.out.println(s.toString());
    }

    //4判断
    System.out.println(linkedList.contains(s1));
    System.out.println(linkedList.isEmpty());
    //5获取
    System.out.println(linkedList.indexOf(s2));
  }
}
```

ArrayList和LinkedList区别：

- ArrayList存储结构是数据，查找、遍历效率高。
- LinkedList存储结构是双向链表，删除、添加效率高。

3.2.3 Vector

> 数组结构实现，查询快、增删慢。
> JDK1.0版本，线程安全、运行效率比ArrayList较慢。

> 案例演示：使用Vector保存数据。

```
public class TestVector {
  public static void main(String[] args) {
    //创建集合
    Vector vector=new Vector<>();
    //1添加元素
    vector.add("草莓");
    vector.add("芒果");
    vector.add("西瓜");
    System.out.println("元素个数:"+vector.size());
    //2删除
//    vector.remove(0);
//    vector.remove("西瓜");
//    vector.clear();
    //3遍历
    //使用枚举器
    Enumeration en=vector.elements();
    while(en.hasMoreElements()) {
      String  o=(String)en.nextElement();
      System.out.println(o);
    }
    //4判断
    System.out.println(vector.contains("西瓜"));
    System.out.println(vector.isEmpty());
    //5vector其他方法
    //firsetElement、lastElement、elementAt();
  }
}
```

四、泛型

4.1 泛型概念

概念：

- Java泛型是JDK1.5中引入的一个新特性，其本质是参数化类型，把类型作为参数传递。
- 常见形式有泛型类、泛型接口、泛型方法。

语法：

- <T,...> T称为类型占位符，表示一种引用类型。

优点：

- 提高代码的重用性。
- 防止类型转换异常，提高代码的安全性。

案例演示：创建泛型类。

```
public class MyGeneric<T> {
  //使用泛型T
  //1创建变量
  T t;

  //2泛型作为方法的参数
  public void show(T t) {
    System.out.println(t);
  }
  //3泛型作为方法的返回值
  public T getT() {
    return t;
  }
}
```

案例演示：泛型方法。

```
public class MyGenericMethod {

  //泛型方法
  public  <T> T show(T t) {
    System.out.println("泛型方法"+t);
    return t;
  }

}
```

案例演示：泛型接口。

```
public interface MyInterface<T> {
  String name="张三";


  T server(T t);

}
```

测试类：

```
public class TestGeneric {
  public static void main(String[] args) {
    //使用泛型类创建对象
    MyGeneric<String> myGeneric=new MyGeneric<String>();
    myGeneric.t="hello";
    myGeneric.show("大家好,加油");
    String string=myGeneric.getT();

    MyGeneric<Integer> myGeneric2=new MyGeneric<Integer>();
    myGeneric2.t=100;
    myGeneric2.show(200);
    Integer integer=myGeneric2.getT();


    //泛型接口
    MyInterfaceImpl impl=new MyInterfaceImpl();
    impl.server("xxxxxxx");


    MyInterfaceImpl2<Integer> impl2=new MyInterfaceImpl2<>();
    impl2.server(1000);

    //泛型方法

    MyGenericMethod myGenericMethod=new MyGenericMethod();
    myGenericMethod.show("中国加油");
    myGenericMethod.show(200);
```

```
    myGenericMethod.show(3.14);

  }
}
```

注：1、泛型只能使用引用类型。 2、不同泛型类型对象之间不能相互赋值。

4.2 泛型集合

> - 参数化类型、类型安全的集合，强制集合元素的类型必须一致。
> - 特点：
>   - 编译时即可检查，而非运行时抛出异常。
>   - 访问时，不必类型转换（拆箱）。
>   - 不同泛型之间引用不能相互赋值，泛型不存在多态。

```
public class TestArrayList2 {
  public static void main(String[] args) {
    ArrayList<String> arrayList=new ArrayList<String>();
    arrayList.add("xxx");
    arrayList.add("yyy");
//    arrayList.add(10);
//    arrayList.add(20);

    for (String string : arrayList) {
      System.out.println(string);
    }

    ArrayList<Student> arrayList2=new ArrayList<Student>();
    Student s1=new Student("刘德华", 20);
    Student s2=new Student("郭富城", 22);
    Student s3=new Student("梁朝伟", 18);
    arrayList2.add(s1);
    arrayList2.add(s2);
    arrayList2.add(s3);

    Iterator<Student> it=arrayList2.iterator();
    while(it.hasNext()) {
      Student s=it.next();
      System.out.println(s.toString());
    }
  }
}
```

五、Collections工具类

> 集合工具类，定义了除了存取以外的集合常用方法。

> 常见方法：

方法	描述
public static void reverse(List<?> list)	反转集合中元素的顺序
public static void shuffle(List<?> list)	随机重置集合元素的顺序
public static void sort(List list) /	升序排序（元素类型必须实现Comparable接口）
public static int binarySearch(list, T key)	二分查找

> 案例演示：

```
public class TestCollections {
  public static void main(String[] args) {
    List<Integer> list=new ArrayList<>();
    list.add(20);
    list.add(5);
    list.add(12);
    list.add(30);
    list.add(6);
    //sort排序
    System.out.println("排序之前:"+list.toString());
    Collections.sort(list);
    System.out.println("排序之后:"+list.toString());

    //binarySearch二分查找
    int i=Collections.binarySearch(list, 13);
    System.out.println(i);

    //copy复制
    List<Integer> dest=new ArrayList<>();
    for(int k=0;k<list.size();k++) {
      dest.add(0);
```

```
        }
        Collections.copy(dest, list);
        System.out.println(dest.toString());

        //reverse反转

        Collections.reverse(list);
        System.out.println("反转之后:"+list);

        //shuffle 打乱
        Collections.shuffle(list);
        System.out.println("打乱之后:"+list);


        //补充:  list转成数组
        System.out.println("-------list转成数组 -----");
        Integer[] arr=list.toArray(new Integer[10]);
        System.out.println(arr.length);
        System.out.println(Arrays.toString(arr));

        //数组转成集合
        System.out.println("-------数组转成集合 -----");
        String[] names= {"张三","李四","王五"};
        //集合是一个受限集合，不能添加和删除
        List<String> list2=Arrays.asList(names);
        //list2.add("赵六");
        //list2.remove(0);
        System.out.println(list2);
        //把基本类型数组转成集合时，需要修改为包装类型
        Integer[] nums= {100,200,300,400,500};
        List<Integer> list3=Arrays.asList(nums);
        System.out.println(list3);
    }
}
```

六、Set集合

6.1 Set接口

- 特点：无序、无下标、元素不可重复。
- 方法：全部继承自Collection中的方法。
- 使用foreach循环遍历：
  for(数据类型 局部变量 : 集合名){
    //循环内部的局部变量，代表当次循环从集合中取出的对象
  }

案例演示：使用Set接口保存数据。

```
public class Demo1 {
  public static void main(String[] args) {
    //创建集合
    Set<String> set=new HashSet<>();
    //1添加数据
    set.add("小米");
    set.add("苹果");
    set.add("华为");
    //set.add("华为");
    System.out.println("数据个数:"+set.size());
    System.out.println(set.toString());
    //2删除数据
//    set.remove("小米");
//    System.out.println(set.toString());
    //3遍历 【重点】
    //3.1使用增强for
    System.out.println("-----增强for-------");
    for (String string : set) {
      System.out.println(string);
    }
    //3.2使用迭代器
    System.out.println("-----使用迭代器------");
    Iterator<String> it=set.iterator();
    while(it.hasNext()) {
      System.out.println(it.next());
    }
    //4判断
    System.out.println(set.contains("华为"));
    System.out.println(set.isEmpty());
  }
}
```

6.2 Set实现类

6.2.1 HashSet

> - 基于HashCode实现元素不重复。
> - 当存入元素的哈希码相同时，会调用==或equals进行确认，结果为true，拒绝后者存入。

> 案例演示：HashSet保存简单数据。

```
public class TestHashSet1 {
  public static void main(String[] args) {
    //新建集合
    HashSet<String> hashSet=new HashSet<String>();
    //1添加元素
    hashSet.add("刘德华");
    hashSet.add("梁朝伟");
    hashSet.add("林志玲");
    hashSet.add("周润发");
    //hashSet.add("刘德华");
    System.out.println("元素个数:"+hashSet.size());
    System.out.println(hashSet.toString());
    //2删除数据
//    hashSet.remove("刘德华");
//    System.out.println("删除之后:"+hashSet.size());
    //3遍历操作
    //3.1增强for
    System.out.println("--------3.1增强for--------");
    for (String string : hashSet) {
      System.out.println(string);
    }
    //3.2使用迭代器
    System.out.println("-------3.2迭代器--------");
    Iterator<String> it=hashSet.iterator();
    while(it.hasNext()) {
      System.out.println(it.next());
    }
    //4判断
    System.out.println(hashSet.contains("郭富城"));
    System.out.println(hashSet.isEmpty());

  }
}
```

> 案例演示：HashSet保存对象数据。

```
public class TestHashSet2 {
  public static void main(String[] args) {
    //新建集合
    HashSet<String> hashSet=new HashSet<String>();
    //1添加元素
    hashSet.add("刘德华");
    hashSet.add("梁朝伟");
    hashSet.add("林志玲");
    hashSet.add("周润发");
    //hashSet.add("刘德华");
    System.out.println("元素个数:"+hashSet.size());
    System.out.println(hashSet.toString());
    //2删除数据
//    hashSet.remove("刘德华");
//    System.out.println("删除之后:"+hashSet.size());
    //3遍历操作
    //3.1增强for
    System.out.println("--------3.1增强for--------");
    for (String string : hashSet) {
      System.out.println(string);
    }
    //3.2使用迭代器
    System.out.println("-------3.2迭代器--------");
    Iterator<String> it=hashSet.iterator();
    while(it.hasNext()) {
      System.out.println(it.next());
    }
    //4判断
    System.out.println(hashSet.contains("郭富城"));
    System.out.println(hashSet.isEmpty());

  }
}
```

```
public class Person{
  private String name;
  private int age;
  public Person() {
    // TODO Auto-generated constructor stub
  }
  public Person(String name, int age) {
    super();
    this.name = name;
```

```
    this.age = age;
  }
  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }
  public int getAge() {
    return age;
  }
  public void setAge(int age) {
    this.age = age;
  }
  @Override
  public String toString() {
    return "Person [name=" + name + ", age=" + age + "]";
  }


  @Override
  public int hashCode() {
    int n1=this.name.hashCode();
    int n2=this.age;

    return n1+n2;

  }

  @Override
  public boolean equals(Object obj) {
    if(this==obj) {
      return true;
    }
    if(obj==null) {
      return false;
    }
    if(obj instanceof Person) {
      Person p=(Person)obj;

      if(this.name.equals(p.getName())&&this.age==p.getAge()) {
        return true;
      }
    }
      return false;
    }
  }
```

注意：

- **根据hashcode计算保存的位置，如果此位置为空，则直接保存，如果不为空执行第二步。**
- **再执行equals方法，如果equals方法为true，则认为是重复，否则，形成链表。**
- **JDK1.8之后引入红黑树，提高效率。**

6.2.2 LinkedHashSet

> - 链表实现的HashSet，按照链表进行存储，即可保留元素的插入顺序。

6.2.3 TreeSet

> - 基于排列顺序实现元素不重复。
> - 实现了SortedSet接口，对集合元素自动排序。
> - 元素对象的类型必须实现Comparable接口，指定排序规则（自然排序）。
> - 通过CompareTo方法确定是否为重复元素。

> 案例演示：使用TreeSet保存简单数据。

```
public class TestTreeSet1 {
  public static void main(String[] args) {
    //创建集合
    TreeSet<String> treeSet=new TreeSet<>();
    //1添加元素
    treeSet.add("xyz");
    treeSet.add("abc");
    treeSet.add("hello");
    treeSet.add("xyz");
    System.out.println("元素个数:"+treeSet.size());
    System.out.println(treeSet.toString());

    //2删除
//    treeSet.remove("xyz");
//    System.out.println("删除之后:"+treeSet.size());
    //3遍历
    //3.1使用增强for
    for (String string : treeSet) {
```

```
      System.out.println(string);
    }
    System.out.println("-----------");
    //3.2使用迭代器
    Iterator<String> it=treeSet.iterator();
    while(it.hasNext()) {
      System.out.println(it.next());
    }
    //4判断
    System.out.println(treeSet.contains("abc"));
  }
}
```

> 案例演示：使用TreeSet保存对象数据。

```
public class Demo5 {
  public static void main(String[] args) {

    //创建集合
    TreeSet<Person> persons=new TreeSet<>();
    //1添加元素
    Person p1=new Person("xyz", 20);
    Person p2=new Person("hello", 22);
    Person p3=new Person("zhangsan", 25);
    Person p4=new Person("zhangsan", 20);
    persons.add(p1);
    persons.add(p2);
    persons.add(p3);
    persons.add(p4);
    System.out.println("元素个数:"+persons.size());
    System.out.println(persons.toString());
    //2删除
//    persons.remove(p1);
//    System.out.println(persons.size());
    //3遍历
    //3.1 使用增强for
    for (Person person : persons) {
      System.out.println(person.toString());
    }
    System.out.println("------------");
    //3.2使用迭代器
    Iterator<Person> it=persons.iterator();
    while(it.hasNext()) {
      System.out.println(it.next());
    }
    //4判断
    System.out.println(persons.contains(new Person("zhangsan", 20)));
  }
}
```

注：元素必须要实现Comparable接口,compareTo()方法返回值为0,认为是重复元素。

> Person类：

```
public class Person implements Comparable<Person>{
  private String name;
  private int age;
  public Person() {
    // TODO Auto-generated constructor stub
  }
  public Person(String name, int age) {
    super();
    this.name = name;
    this.age = age;
  }
  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }
  public int getAge() {
    return age;
  }
  public void setAge(int age) {
    this.age = age;
  }
  @Override
  public String toString() {
    return "Person [name=" + name + ", age=" + age + "]";
  }
  //先按姓名比，然后再按年龄比
  @Override
  public int compareTo(Person o) {
    int n1=this.getName().compareTo(o.getName());
    int n2=this.age-o.getAge();
```

```
    return n1==0?n2:n1;
  }
}
```

> Comparator接口：
>
> - 比较器，实现定制比较。
> - compare(o1,o2)方法的返回值0，表示重复。

> 案例演示：比较器的使用。

```
public class TestComparator {
  public static void main(String[] args) {
    //创建集合,并指定比较规则
    TreeSet<Person> persons=new TreeSet<>(new Comparator<Person>() {
      @Override
      public int compare(Person o1, Person o2) {
        int n1=o1.getAge()-o2.getAge();
        int n2=o1.getName().compareTo(o2.getName());
        return n1==0?n2:n1;
      }
    });

    Person p1=new Person("xyz", 20);
    Person p2=new Person("hello", 22);
    Person p3=new Person("zhangsan", 25);
    Person p4=new Person("lisi", 25);

    persons.add(p1);
    persons.add(p2);
    persons.add(p3);
    persons.add(p4);
    System.out.println(persons.toString());
  }
}
```

> 案例演示：使用TreeSet集合实现字符串按照长度进行排序。

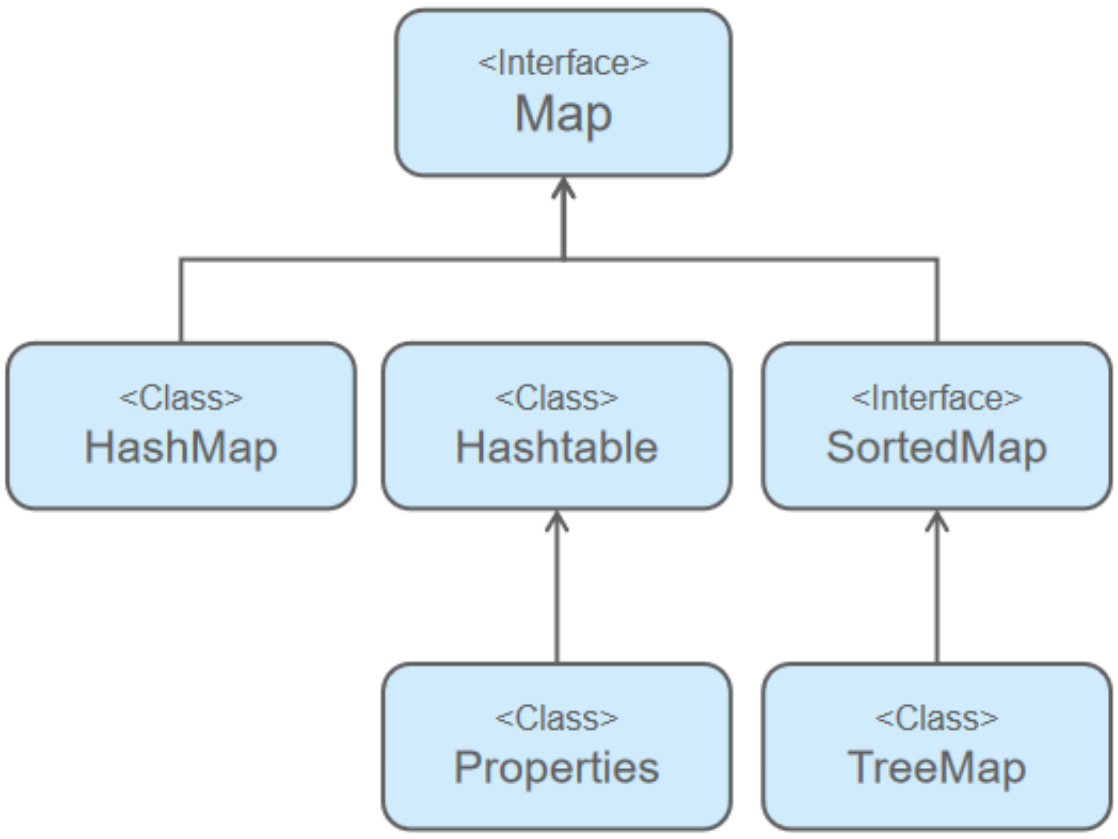
```
public class TestComparator2 {
  public static void main(String[] args) {
    //创建集合，并指定比较规则
    TreeSet<String> treeSet=new TreeSet<>(new Comparator<String>() {
      @Override
      public int compare(String o1, String o2) {
        int n1=o1.length()-o2.length();
        int n2=o1.compareTo(o2);
        return n1==0?n2:n1;
      }
    });
    //添加数据
    treeSet.add("helloworld");
    treeSet.add("pingguo");
    treeSet.add("lisi");
    treeSet.add("zhangsan");
    treeSet.add("beijing");
    treeSet.add("cat");
    treeSet.add("nanjing");
    treeSet.add("xian");

    System.out.println(treeSet.toString());


  }
}
```

七、Map集合

Map集合



> Map接口的特点：
>
> - 用于存储任意键值对(Key-Value)。
> - 键：无下标、不可以重复（唯一）。
> - 值：无下标、可以重复 。

7.1 Map接口

> 特点：
>
> - 称为“映射”存储一对数据(Key-Value)，键不可重复，值可以重复。

> 常用方法：

方法名	描述
V put(K key,V value)	将对象存入到集合中，关联键值。key重复则覆盖原值。
Object get(Object key)	根据键获取对应的值。
Set keySet()	返回所有key。
Collection values()	返回包含所有值的Collection集合。
Set<Map.Entry<K,V>> entrySet()	键值匹配的Set集合。

> 案例演示：Map集合的使用。

```
public class TestMap {
  public static void main(String[] args) {
    //创建Map集合
    Map<String, String> map=new HashMap<>();
    //1添加元素
    map.put("cn", "中国");
    map.put("uk", "英国");
    map.put("usa", "美国");
    map.put("cn", "zhongguo");

    System.out.println("元素个数:"+map.size());
    System.out.println(map.toString());

    //2删除
//    map.remove("usa");
//    System.out.println("删除之后:"+map.size());
    //3遍历
    //3.1使用keySet();
    System.out.println("------keySet()--------");
    //Set<String> keyset=map.keySet();
    for (String key : map.keySet()) {
      System.out.println(key+"-----"+map.get(key));
    }
    //3.2使用entrySet()方法
    System.out.println("------entrySet()-----");
    //Set<Map.Entry<String, String>> entries=map.entrySet();
    for (Map.Entry<String, String> entry : map.entrySet()) {
      System.out.println(entry.getKey()+"---------"+entry.getValue());
    }
    //4判断
    System.out.println(map.containsKey("cn"));
```

```
    System.out.println(map.containsValue("泰国"));

  }
}
```

7.2 Map实现类

7.2.1 HashMap【 重点 】

> - JDK1.2版本，线程不安全，运行效率快。
> - 允许用null 作为key或是value。
> - 存储结构：哈希表。

> 案例演示：HashMap保存对象数据。

```
public class Student {
  private String name;
  private int stuNo;

public Student() {
  // TODO Auto-generated constructor stub
}

public Student(String name, int stuNo) {
  super();
  this.name = name;
  this.stuNo = stuNo;
}

public String getName() {
  return name;
}

public void setName(String name) {
  this.name = name;
}

public int getStuNo() {
  return stuNo;
}

public void setStuNo(int stuNo) {
  this.stuNo = stuNo;
}



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

@Override
public boolean equals(Object obj) {
  if (this == obj)
    return true;
  if (obj == null)
    return false;
  if (getClass() != obj.getClass())
    return false;
  Student other = (Student) obj;
  if (name == null) {
    if (other.name != null)
      return false;
  } else if (!name.equals(other.name))
    return false;
  if (stuNo != other.stuNo)
    return false;
  return true;
}

  @Override
  public String toString() {
    return "Student [name=" + name + ", stuNo=" + stuNo + "]";
  }
}
```

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

    //创建集合
    HashMap<Student, String> students=new HashMap<Student,String>();
```

```
    //刚创建hashmap之后没有添加元素 table=null   size=0 目的节省空间
    //1添加元素
    Student s1=new Student("孙悟空", 100);
    Student s2=new Student("猪八戒", 101);
    Student s3=new Student("沙和尚", 102);
    students.put(s1, "北京");
    students.put(s2, "上海");
    students.put(s3, "杭州");
    //students.put(s3, "南京");
    students.put(new Student("沙和尚", 102), "杭州");
    System.out.println("元素个数:"+students.size());
    System.out.println(students.toString());
    //2删除
//    students.remove(s1);
//    System.out.println("删除之后"+students.size());
    //3遍历
    System.out.println("--------keySet---------");
    //3.1使用keySet();
    for (Student key : students.keySet()) {
      System.out.println(key.toString()+"=========="+students.get(key));
    }
    System.out.println("--------entrySet---------");
    //3.2使用entrySet();
    for (Map.Entry<Student, String> entry : students.entrySet()) {
      System.out.println(entry.getKey()+"---------"+entry.getValue());
    }
    //4判断
    System.out.println(students.containsKey(new Student("孙悟空", 100)));
    System.out.println(students.containsValue("杭州"));

  }
}
```

7.2.2 TreeMap

> 实现了SortedMap接口(Map的子接口)，可以对key自动排序，Key需实现Comparable接口。

> 案例演示：使用TreeMap保存数据。

```
public class Student implements Comparable<Student>{
  private String name;
  private int stuNo;

  public Student() {
    // TODO Auto-generated constructor stub
  }

  public Student(String name, int stuNo) {
    super();
    this.name = name;
    this.stuNo = stuNo;
  }

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public int getStuNo() {
    return stuNo;
  }

  public void setStuNo(int stuNo) {
    this.stuNo = stuNo;
  }

  @Override
  public String toString() {
    return "Student [name=" + name + ", stuNo=" + stuNo + "]";
  }

  @Override
  public int compareTo(Student o) {
    int n2=this.stuNo-o.getStuNo();
    return n2;
  }
}
```

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

```
    //新建集合(定制比较)
    TreeMap<Student, String> treeMap=new TreeMap<Student,String>();
    //1添加元素
    Student s1=new Student("孙悟空", 100);
    Student s2=new Student("猪八戒", 101);
    Student s3=new Student("沙和尚", 102);
    treeMap.put(s1, "北京");
    treeMap.put(s2, "上海");
    treeMap.put(s3, "深圳");
    treeMap.put(new Student("沙和尚", 102), "南京");
    System.out.println("元素个数:"+treeMap.size());
    System.out.println(treeMap.toString());
    //2删除
//    treeMap.remove(new Student("猪八戒", 101));
//    System.out.println(treeMap.size());
    //3遍历
    //3.1使用keySet
    System.out.println("-----keySet()-------");
    for (Student key : treeMap.keySet()) {
      System.out.println(key+"-------"+treeMap.get(key));
    }
    System.out.println("------entrySet()--------");
    for(Map.Entry<Student, String> entry : treeMap.entrySet()) {
      System.out.println(entry.getKey()+"--------"+entry.getValue());
    }

    //4判断
    System.out.println(treeMap.containsKey(new Student("沙和尚", 102)));
  }
}
```

7.2.3 其他实现类

> Hashtable：
>
> - JDK1.0版本，线程安全，运行效率慢；不允许null作为key或是value。
>
> Properties：
>
> - Hashtable的子类，要求key和value都是String。通常用于配置文件的读取。