



博客：<https://www.cnblogs.com/HOsystem/p/14116443.html>

## 2、具体内容

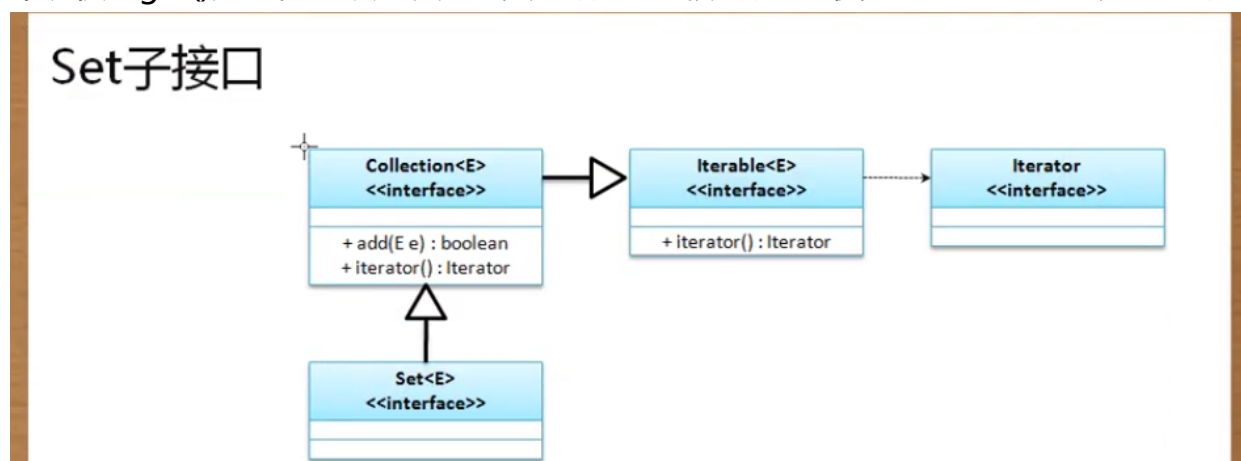
Set集合最大的特点就是不允许保存重复元素，其也是Collection子接口。

### ■Set接口简介

在JDK1.9以前Set集合与Collection集合的定义并无差别，Set继续使用了Collection接口中提供的方法进行操作，但是从JDK1.9之后，Set集合也像List集合一样扩充了一些static方法。Set集合的定义如下：

```
public interface Set<E> extends Collection<E>
```

需要注意的是Set集合并不想List集合那样扩充了许多的新方法，所以无法使用List集合中提供的get()方法，也就是说无法实现指定索引数据的获取，Set接口的继承关系如下：



从JDK1.9之后，Set集合也提供像List集合之中类似的of()的静态方法，下面就使用此方法进行Set集合特点的验证。

#### 范例：验证Set集合特征

```
package cn.mldn.demo;
import java.util.Set;
public class JavaAPIDemo {
```

<pre> public static void main(String[] args) throws Exception {     // 进行Set集合数据的保存，并且设置有重复的内容     Set&lt;String&gt; all = Set.of("Hello", "World", "MLDN", "Hello", "World");     all.forEach(System.out::println);    // 直接输出 } </pre>	
程序运行结果	Exception in thread "main" java.lang.IllegalArgumentException: duplicate element : Hello

当使用of()这个新方法的时候如果发现集合之中存在有重复元素则会直接抛出异常。这与传统的Set集合不保存重复元素的特点相一致，只不过自己抛出了异常而已。

Set集合的常规使用形式一定是依靠子类进行实例化的，所以Set接口之中有两个常用子类：HashSet、TreeSet。

## ■HashSet子类

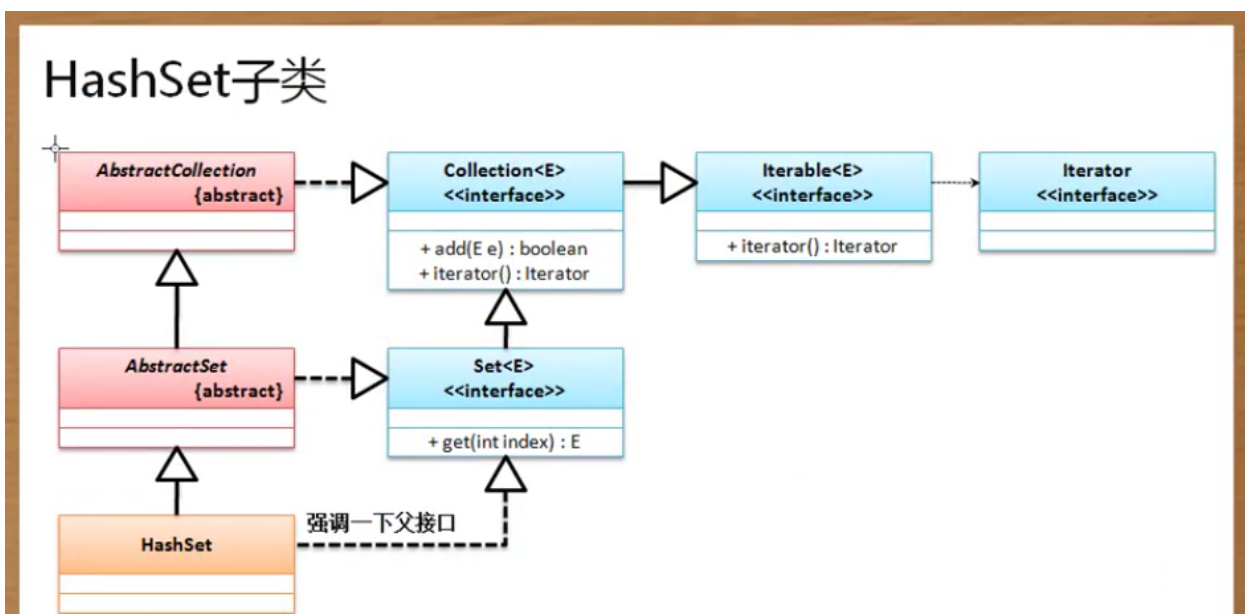
HashSet是Set接口里面使用最多的一个子类，其最大的特点就是保存的数据是无须的，而HashSet子类的继承关系如下：

```

public class HashSet<E> extends AbstractSet<E> implements Set<E>, Cloneable,
Serializable

```

这种继承的形式和之前的ArrayList是非常相似的，那么现在来观察一下类的继承结构：



### 范例：观察HashSet子类

```

package cn.mldn.demo;
import java.util.HashSet;
import java.util.Set;
public class JavaAPIDemo {
    public static void main(String[] args) throws Exception {
        Set<String> all = new HashSet<String>();
        all.add("MLDN");
    }
}

```

```

        all.add("NiHao");
        all.add("Hello");
        all.add("Hello"); // 重复元素
        all.add("World");
        all.forEach(System.out::println); // 直接输出
    }
}

```

通过执行结果就可以发现HashSet子类的操作特点：不允许保存重复元素（Set接口定义的），另外一点HashSet之中保存的数据是无序的。

## ■TreeSet子类

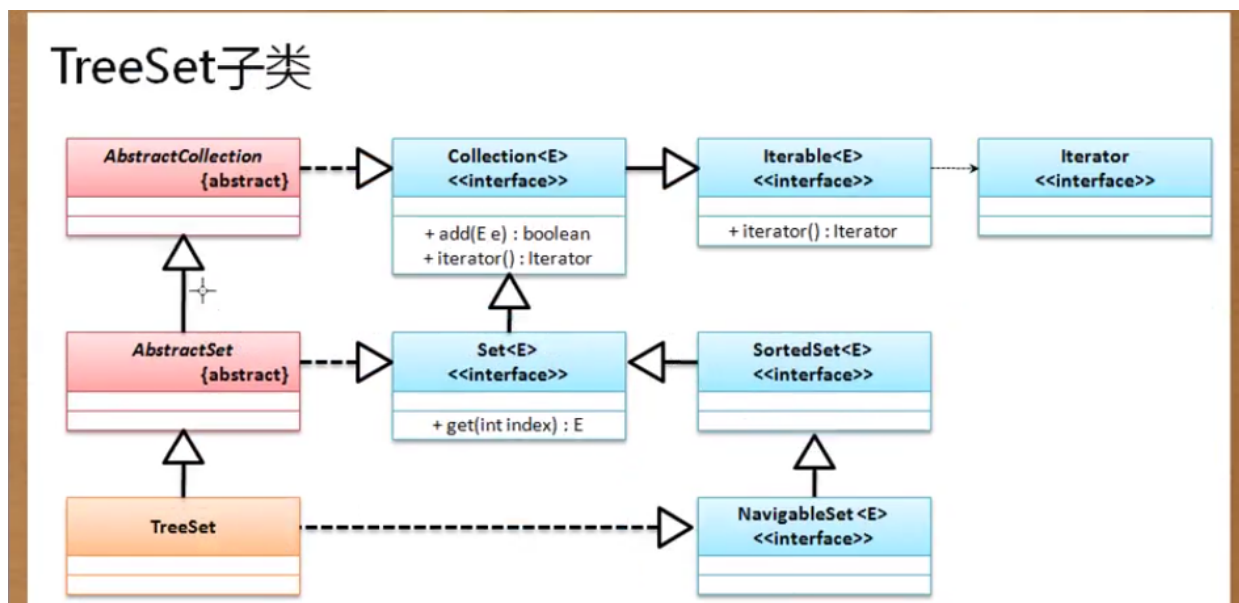
Set接口的另外一个子类就是TreeSet，与HashSet最大的区别在于TreeSet集合里面所保存的数据是有序的，首先来观察一下TreeSet类的定义：

```

public class TreeSet<E> extends AbstractSet<E> implements NavigableSet<E>, Cloneable,
Serializable

```

在这个子类里面依然继承了AbstractSet父抽象类，同时又实现了一个NavigableSet父接口。



### 范例：使用TreeSet子类

```

package cn.mldn.demo;
import java.util.Set;
import java.util.TreeSet;
public class JavaAPIDemo {
    public static void main(String[] args) throws Exception {
        Set<String> all = new TreeSet<String>();
        all.add("MLDN");
        all.add("NiHao");
        all.add("Hello");
        all.add("Hello"); // 重复元素
        all.add("World");
        all.forEach(System.out::println); // 直接输出
    }
}

```

```
}  
}
```

当利用TreeSet保存数据时候的所有数据都将按照数据的升序进行自动排序处理。

## ■treeset排序说明

经过分析之后TreeSet子类之中保存的数是允许排序的，但是这个类必须要实现Comparable接口，只有实现了此接口才能够确认出对象的大小关系。

提示：TreeSet本质是利用TreeSet子类实现的集合数据的存储，而TreeMap（树）就需要根据Comparable来确定大小关系。

那么下面就使用一个自定义的类来实现排序的处理操作。

### 范例：实现自定义类排序

```
package cn.mldn.demo;  
import java.util.Set;  
import java.util.TreeSet;  
class Person implements Comparable <Person> { // 比较器  
    private String name ;  
    private int age ;  
    public Person(String name,int age) {  
        this.name = name ;  
        this.age = age ;  
    }  
    public String toString() {  
        return "姓名: " + this.name + "、年龄: " + this.age ;  
    }  
    @Override  
    public int compareTo(Person per) {  
        if (this.age < per.age) {  
            return -1 ;  
        } else if (this.age > per.age) {  
            return 1 ;  
        } else {  
            return this.name.compareTo(per.name) ;  
        }  
    }  
}  
public class JavaAPIDemo {  
    public static void main(String[] args) throws Exception {  
        Set<Person> all = new TreeSet<Person>(); // 为List父接口进行实例化  
        all.add(new Person("张三",19));  
        all.add(new Person("李四",19)); // 年龄相同，但是姓名不同  
        all.add(new Person("王五",20)); // 数据重复  
        all.add(new Person("王五",20)); // 数据重复  
        all.add(new Person("小强",78));  
        all.forEach(System.out::println);  
    }  
}
```

在使用自定义类对象进行比较处理的时候一定要将该类之中的所有属性都依次进行大小关系的匹配，否则如果某一个或某几个属性相同的时候它也会认为是重复元素，所以TreeSet是利用了Comparable接口来确认重复数据的。

由于TreeSet在操作过程之中需要将类中的所有属性进行比对，这样的实现难度太高了，那么在实际的开发之中应该首选HashSet子类进行存储。

---

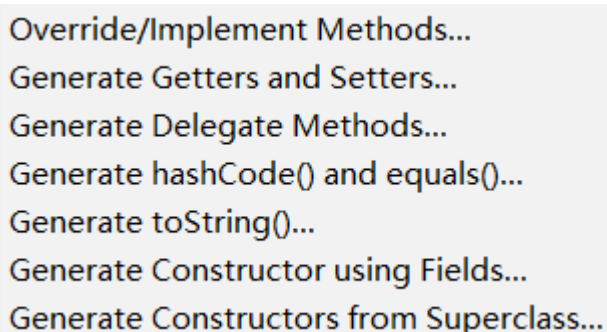
## ■关于重复元素的说明

TreeSet子类是利用了Comparable来实现了重复元素的判断，但是set集合的整体特征是不允许保存重复元素。但是HashSet判断重复元素的方式并不是利用Comparable接口完成的，它利用的是Object类中提供的方法实现的：

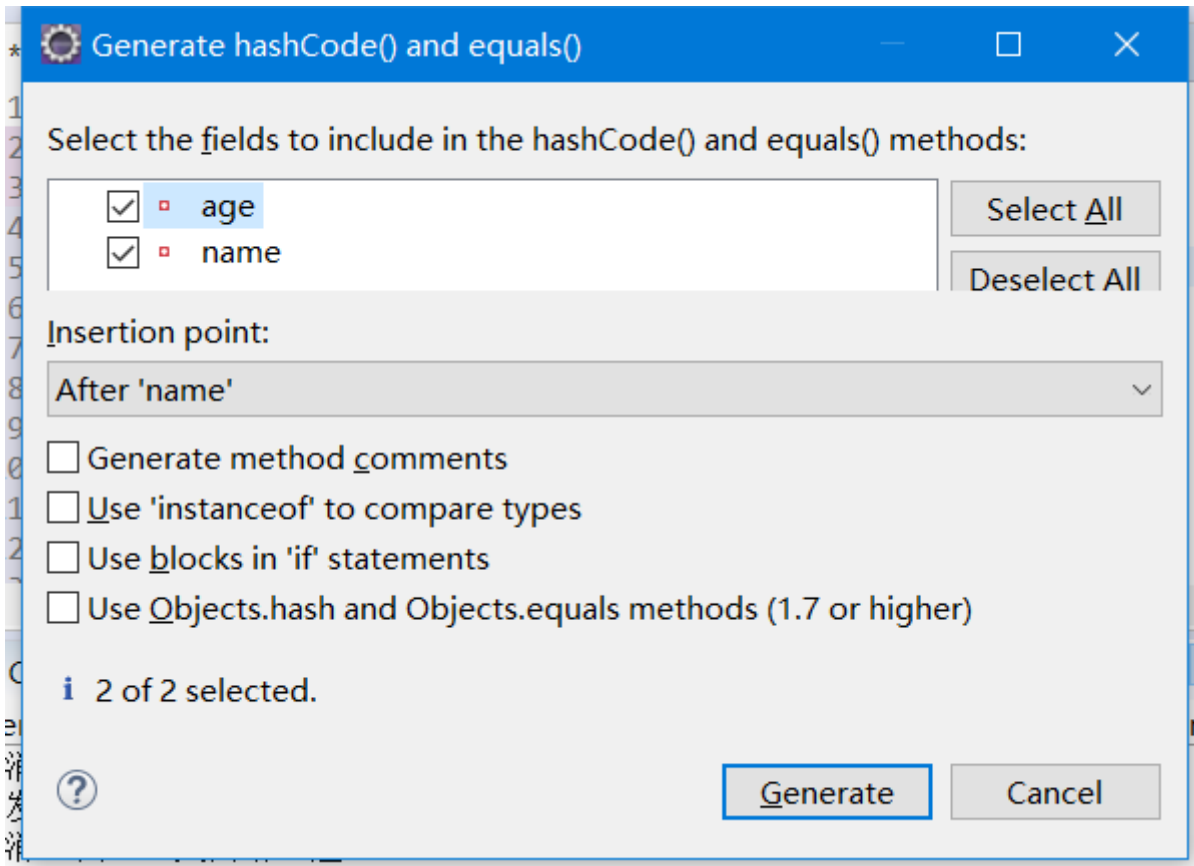
·**对象编码**：public int hashCode();

·**对象比较**：public boolean equals(Object obj);

在进行重复元素判断的时候首先利用hashCode()进行编码的匹配，如果该编码不存在则表示数据不存在，证明没有重复，如果该编码存在了，则进一步进行对象比较处理，如果发现重复了，则此数据是不允许保存的。如果使用的是eclipse开发工具，则可以帮助开发者自动创建hashCode()与equals()方法以简化开发。



Override/Implement Methods...  
Generate Getters and Setters...  
Generate Delegate Methods...  
Generate hashCode() and equals()...  
Generate toString()...  
Generate Constructor using Fields...  
Generate Constructors from Superclass...



### 范例：实现重复元素处理

```
package cn.mldn.demo;
import java.util.HashSet;
import java.util.Set;
class Person { // 比较器
    private String name ;
    private int age ;
    public Person(String name,int age) {
        this.name = name ;
        this.age = age ;
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + age;
        result = prime * result + ((name == null) ? 0 : name.hashCode());
        return result;
    }
    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
```

```

        Person other = (Person) 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 "姓名: " + this.name + "、年龄: " + this.age ;
    }
}
public class JavaAPIDemo {
    public static void main(String[] args) throws Exception {
        Set<Person> all = new HashSet<Person>(); // 为List父接口进行实例化
        all.add(new Person("张三",19));
        all.add(new Person("李四",19)); // 年龄相同, 但是姓名不同
        all.add(new Person("王五",20)); // 数据重复
        all.add(new Person("王五",20)); // 数据重复
        all.add(new Person("小强",78));
        all.forEach(System.out::println);
    }
}

```

在Java程序之中真正的重复元素的判断处理利用的就是hashCode()与equals()两个方法共同作用完成的, 而只有在排序要求的情况下 (TreeSet) 才会利用Comparable接口来实现。