# Collection

Collection是一个接口，不能直接创建他的对象



添加元素应注意的细节：

细节1:如果我们要往List系列集合中添加数据，那么方法永远返回true，因为List系列的是允许元素重复的。

细节2:如果我们要往set系列集合中添加数据，如果当前要添加元素不存在，方法返回true，表示添加成功。

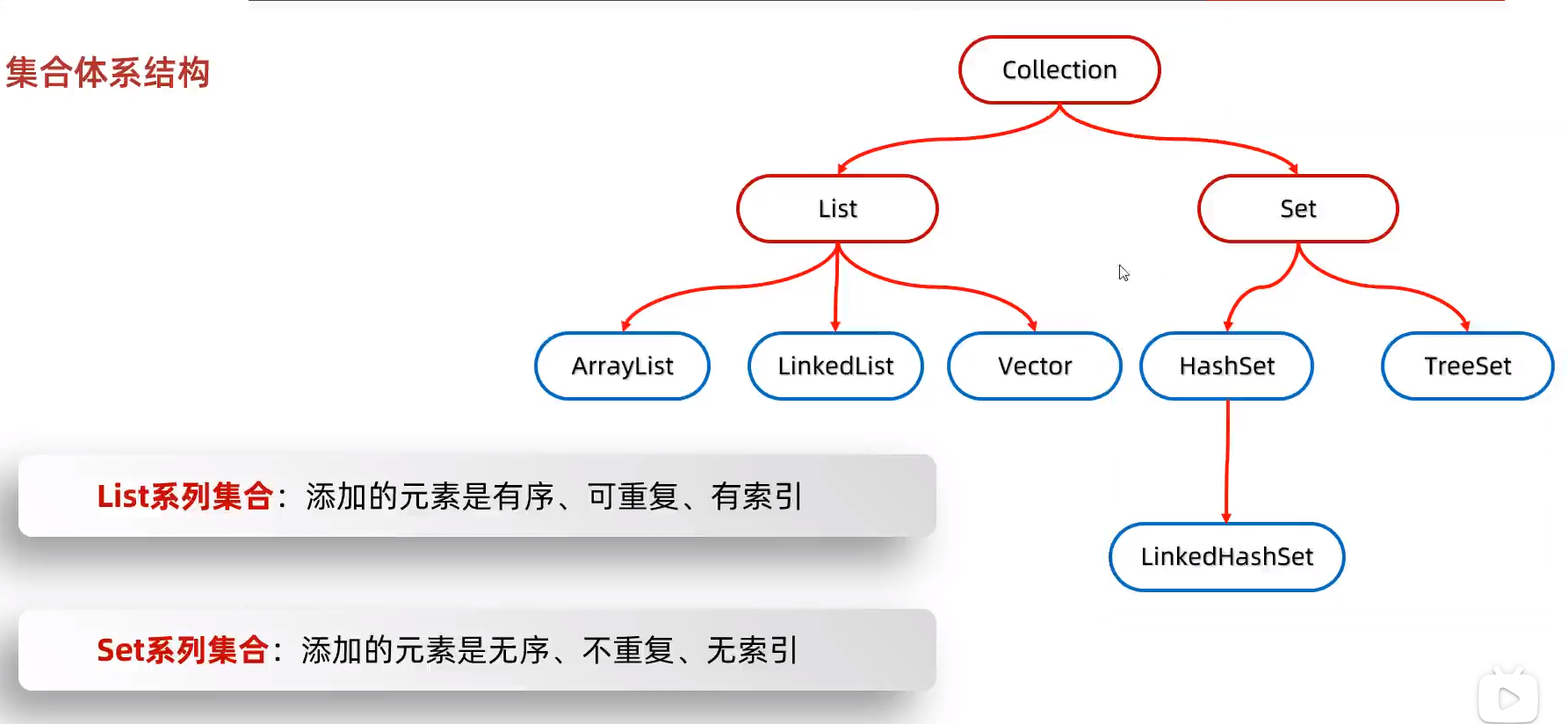
如果当前要添加的元素已经存在，方法返回false，表示添加失败。

因为set系列的集合不允许重复。



注意如果是自定义对象要判断是否相等，需要重写equals方法，因为在java底层默认是用地址值来判断对象是否相等。

# 集合体系结构

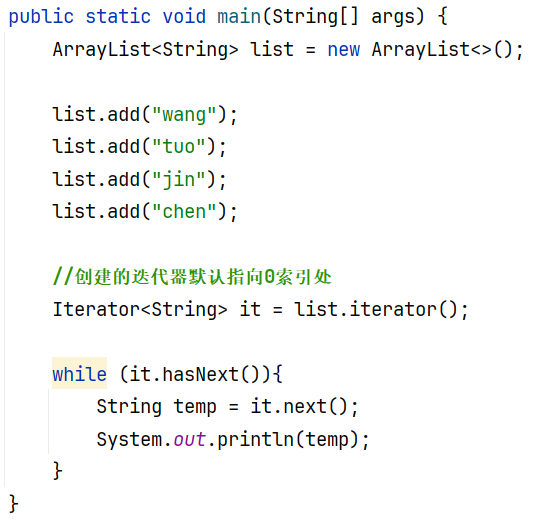


Set无索引，遍历需要使用迭代器

# 迭代器遍历

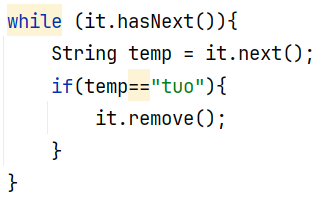


使用迭代器遍历集合的一般方法：



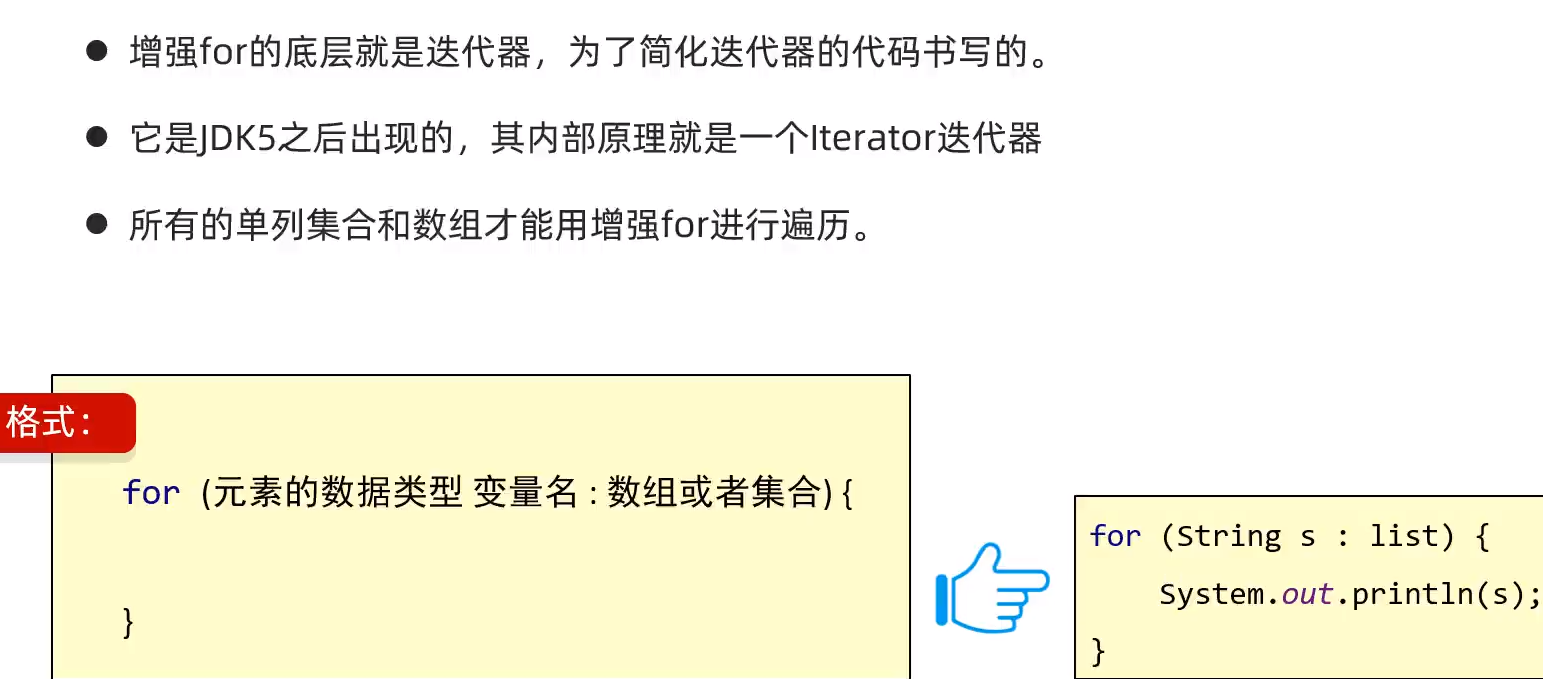
使用迭代器删除元素需要注意：

不能直接用集合的思维去删除，而是要使用迭代器提供的方法进行删除：



迭代器遍历时，不能用集合的方法添加或删除元素。

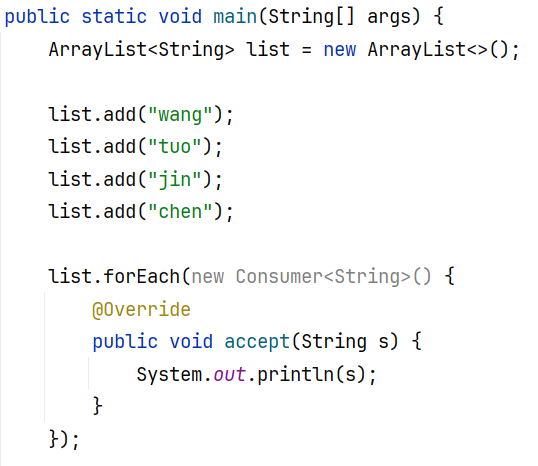
# 增强for遍历



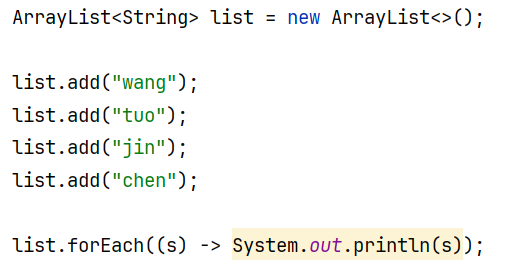
修改增强for中的变量，不会改变原有数据

# Foreach遍历方式

Foreach底层会遍历集合，每一轮获取的元素会传递给accept方法，也就是参数s，至于要对元素进行什么操作，取决于程序员



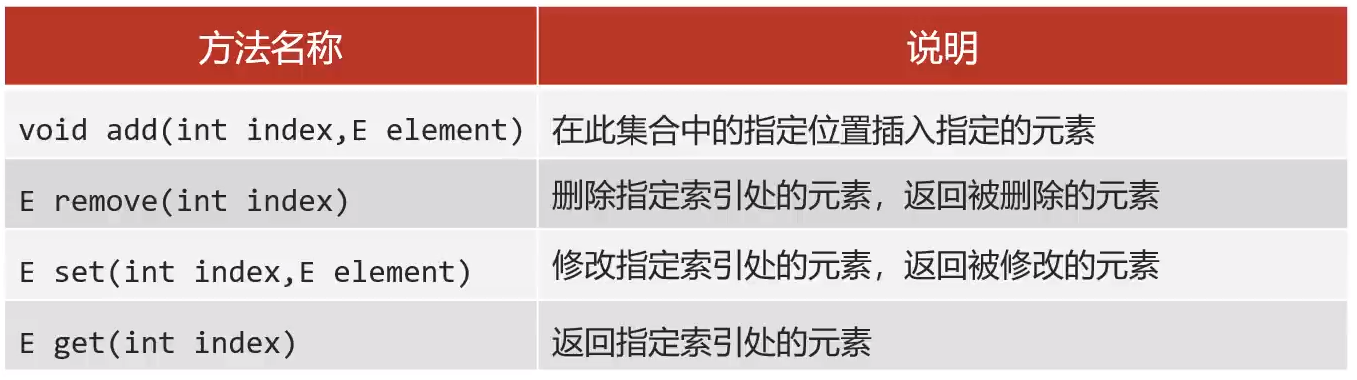
Foreach的lambda表达式写法



方法体只有一行可以省略大括号，参数可以省略类型

# List集合特有的方法

List集合因为有了索引，所以多了一些索引操作的方法



# List的遍历方式

List系列集合的五种遍历方式:

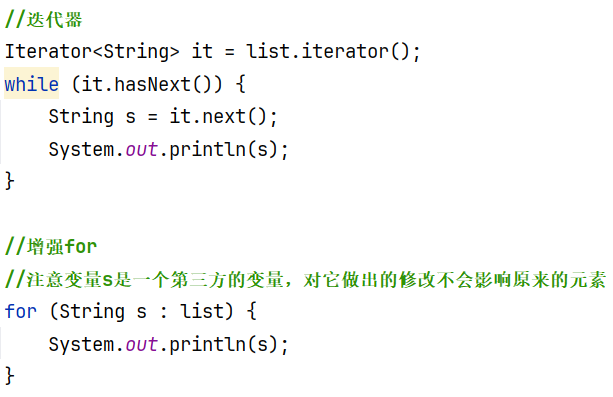
1.迭代器

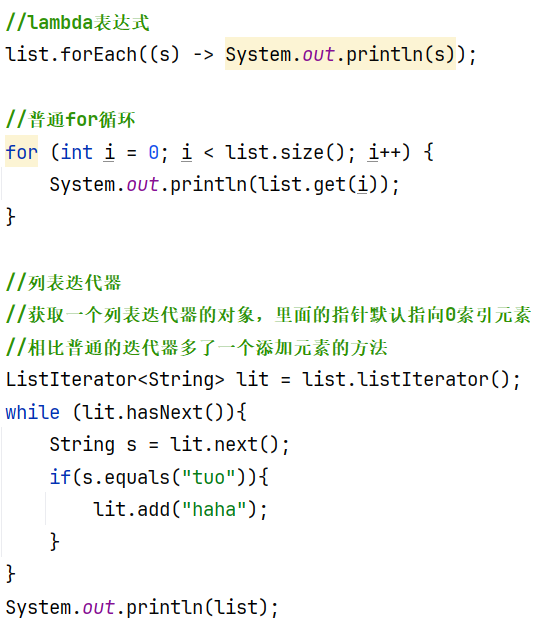
2.列表迭代器

3.增强for

4.Lambda表达式

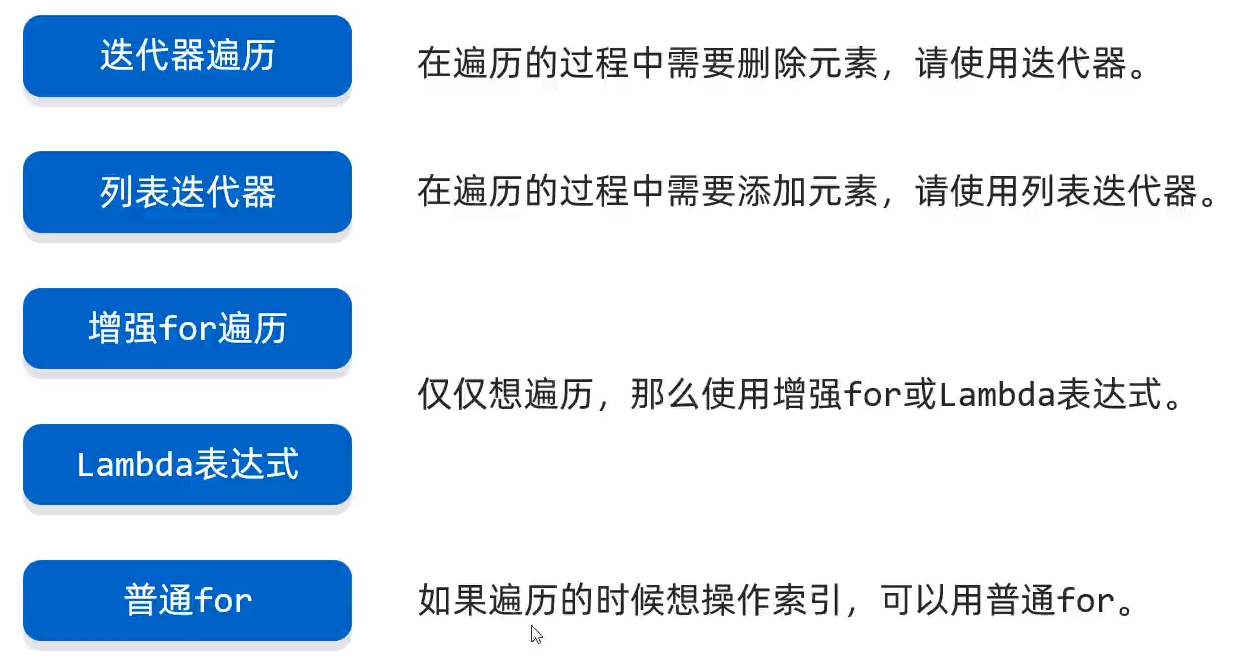
5.普通for循环





列表迭代器处的结果：

什么时候用什么遍历方式？

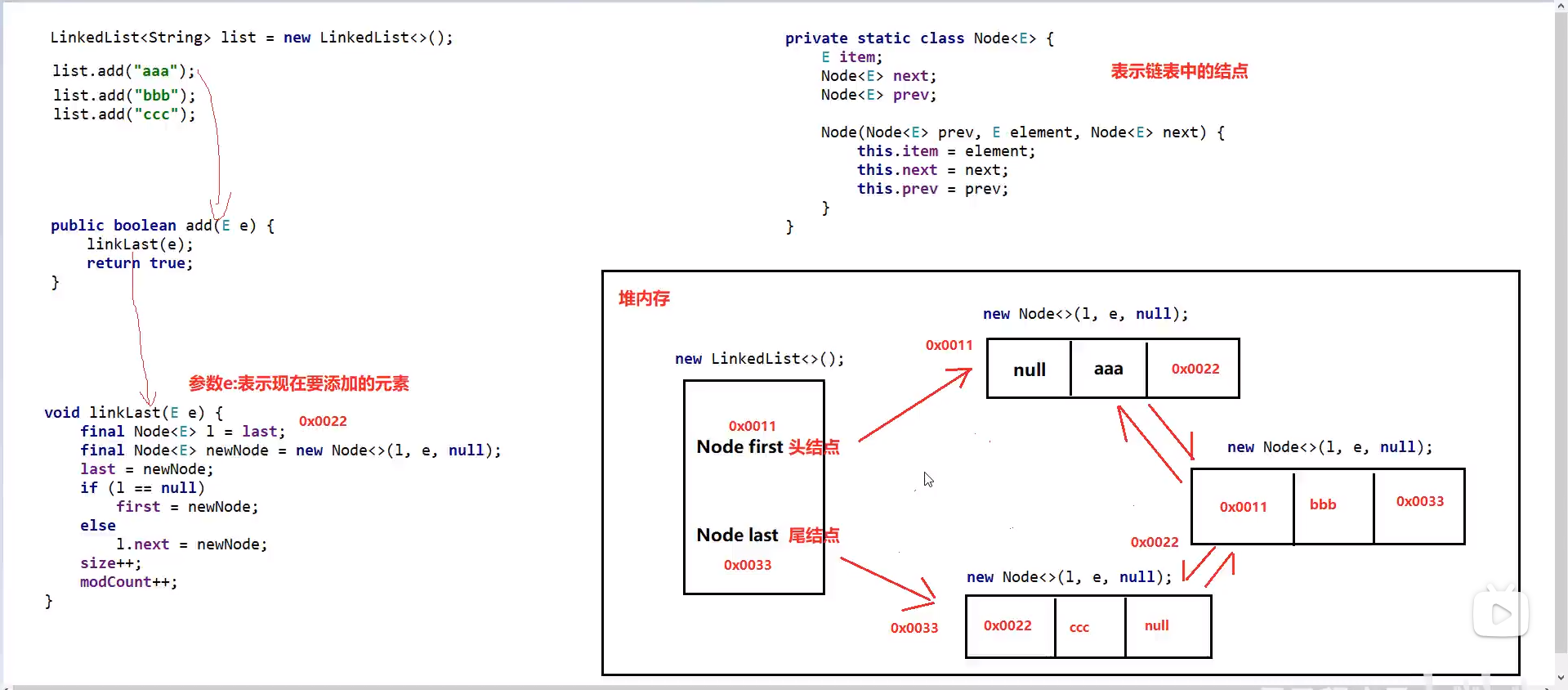


# Linkedlist集合

底层数据结构是双链表，查询慢，首尾操作的速度很快，所以多了很多首尾操作的特有API



# Linkedlist底层是如何实现增加结点的？



# 迭代器iterator底层是如何实现的？

# 

# 

通过变量modcount来记录集合变化的次数来检查此时是否发生**并发修改异常**

在以后如何避免并发修改异常？

在使用迭代器或者是增强for遍历集合的过程中，不要使用集合的方法去添加或者删除元素即可。

# 泛型

什么是泛型？



以这个集合为例，<String>就是泛型

没有泛型的时候，集合如何存储数据？

如果我们没有给集合指定类型，默认认为所有的数据类型都是object类型

此时可以往集合添加任意的数据类型。

这样会带来一个坏处:我们在获取数据的时候，无法使用他的特有行为。

于是推出了泛型，可以在添加数据的时候就把类型进行统一

而且我们在获取数据的时候，也省的强转了，非常的方便。

注意：

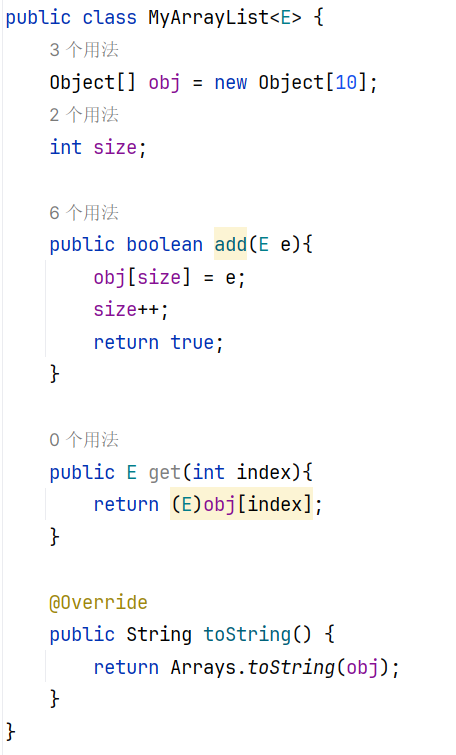
1. 泛型中不能写基本数据类型

2、指定泛型的具体类型后，传递数据时，可以传入该类类型或者其子类类型

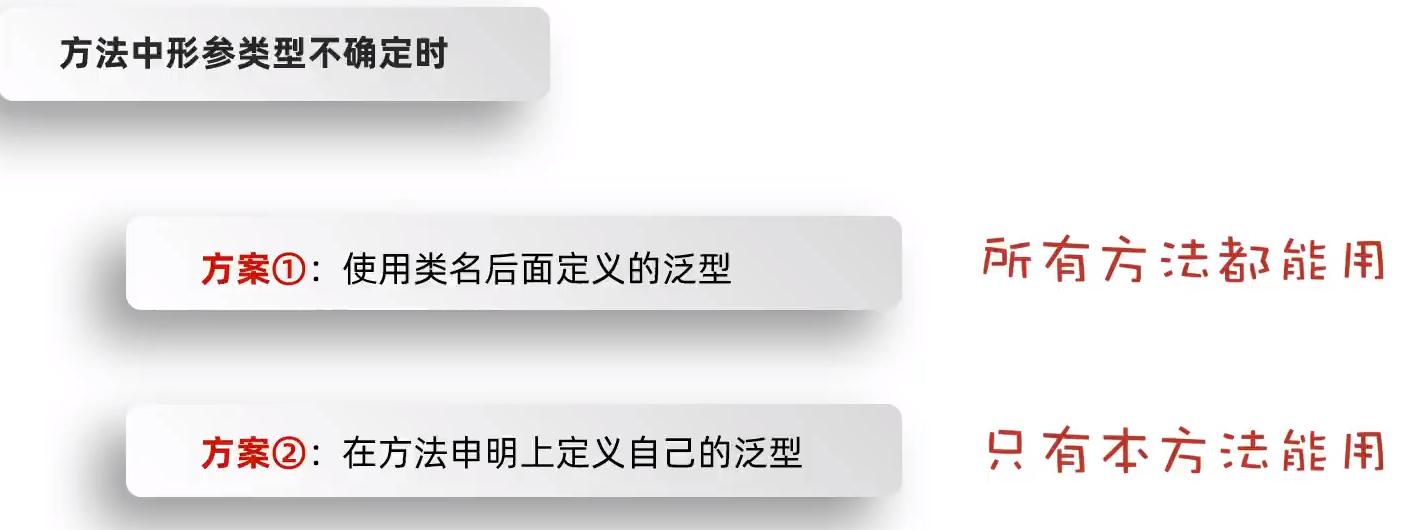
3、如果不写泛型，类型默认是Object

# 泛型类

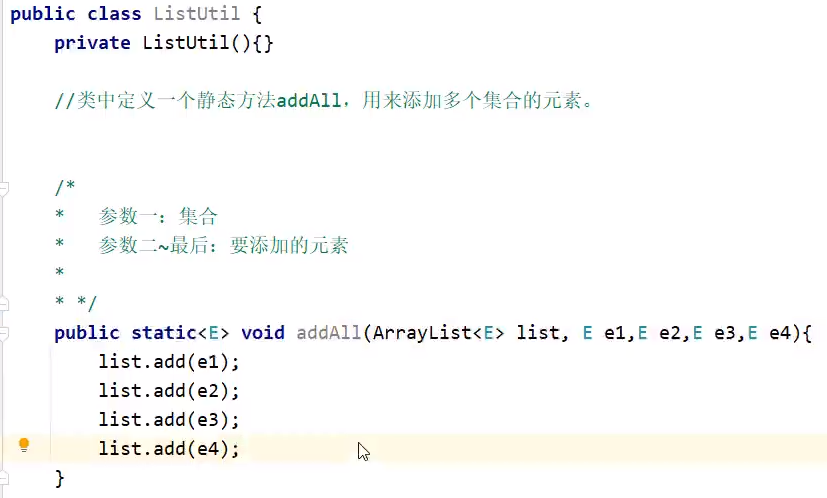
当编写一个类时，如果不确定，可以把这个类定义为泛型类



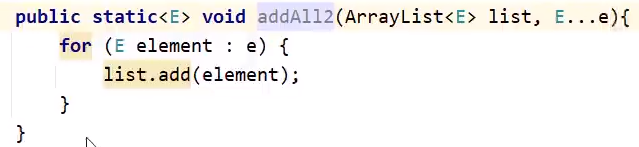
# 泛型方法



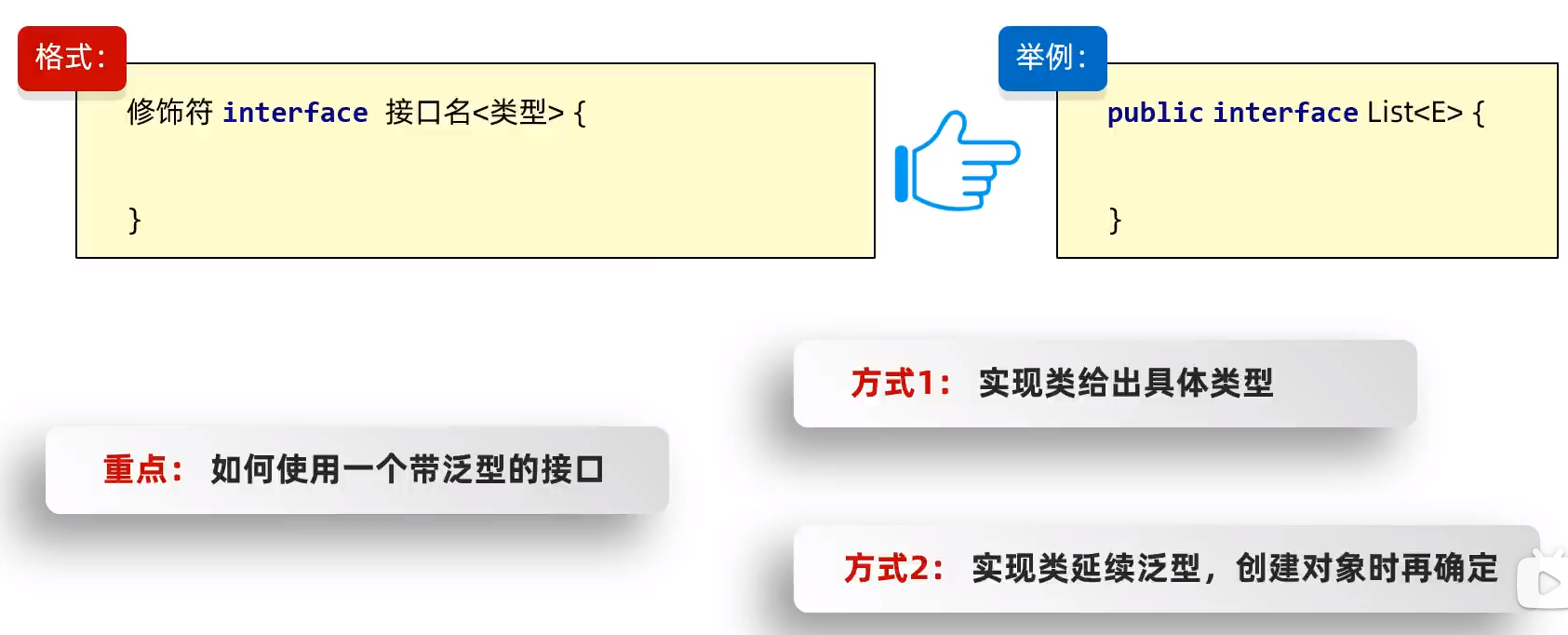
例子：



想要添加任意数量的元素怎么办？



# 泛型接口



注意：泛型不具有继承性，对于方法而言，泛型里面是什么类型就只能传递什么类型的数据，比如现在有ye，fu，zi三个类，他们的继承关系可想而知，但是如果一个方法的泛型里面是ye，你不能传fu和zi进去。

现在我们希望对于一个方法，只能传递ye，fu，zi这三种有继承关系的类，于是有了泛型通配符。

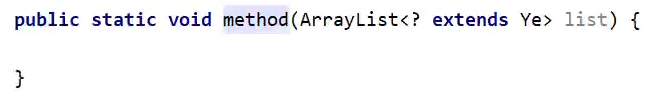
?也表示不确定的类型

他可以进行类型的限定

? Extends E:表示可以传递E或者E所有的子类类型

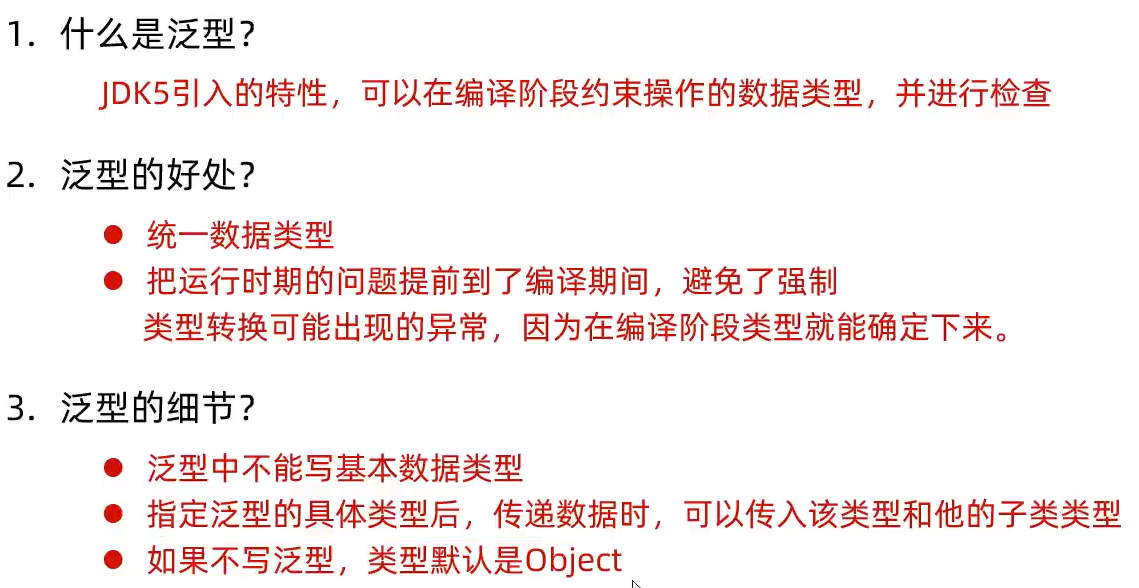
? super E:表示可以传递E或者E所有的父类类型

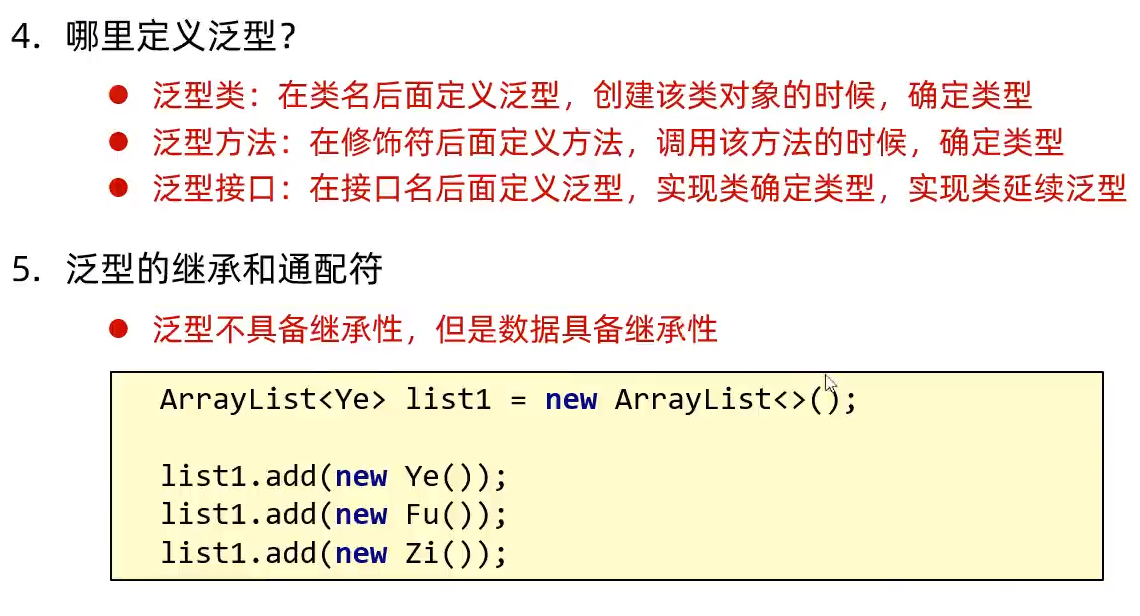
例子：



泛型通配符的作用就是限定类型的范围

# 泛型总结





# Set系列集合

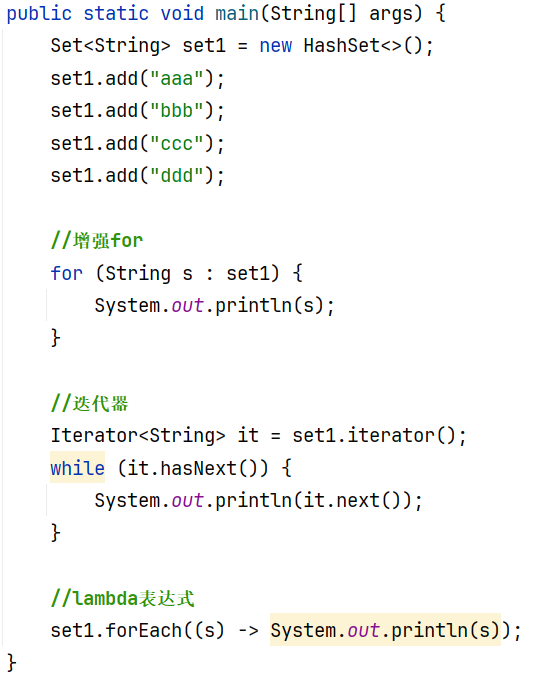


# Set遍历方式

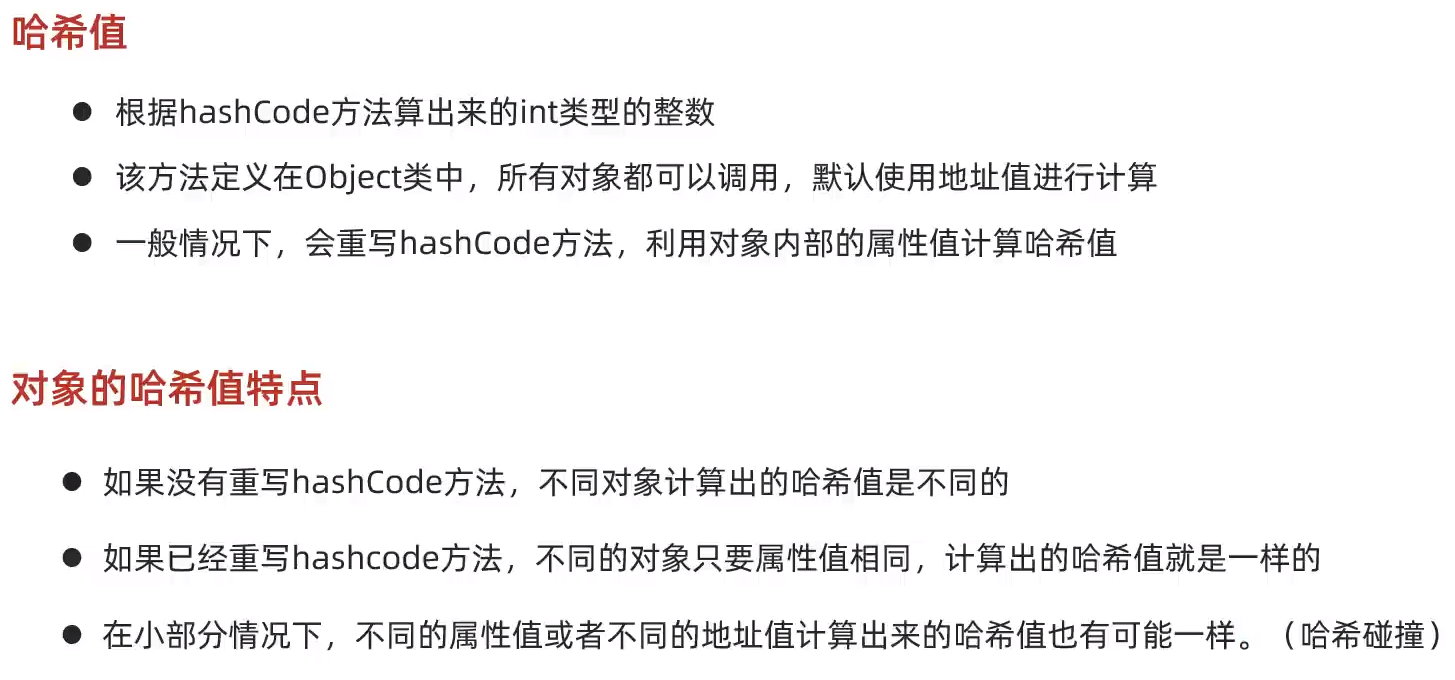
迭代器

增强for

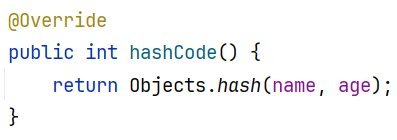
Lambda表达式



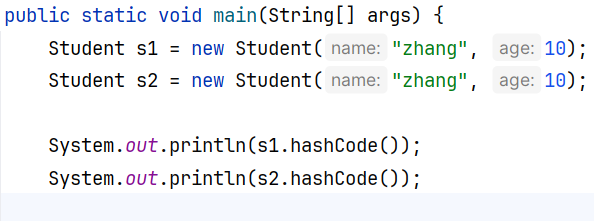
# Hashset



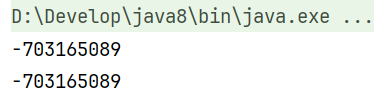
在student类中重写hashcode方法



这样一来，如果2个student对象的属性都相同，那么他们的哈希值也相同



结果：



# Hashset底层原理

1. 创建一个默认长度16，默认加载因子0.75的数组，数组名table

Ps：加载因子是数组扩容的时机，当数组内容到达16\*0.75=12时，数组长度翻倍

2、根据元素的哈希值跟数组的长度计算出应存入的位置

3、判断当前位置是否为nul，如果是nul直接存入

4、如果位置不为nul，表示有元素，则调用equals方法比较属性值

5、发生哈希碰撞时：

元素属性值一样：不存 元素属性值不一样：存入数组，形成链表

JDK8以前：新元素存入数组，老元素挂在新元素下面

JDK8以后：新元素直接挂在老元素下面

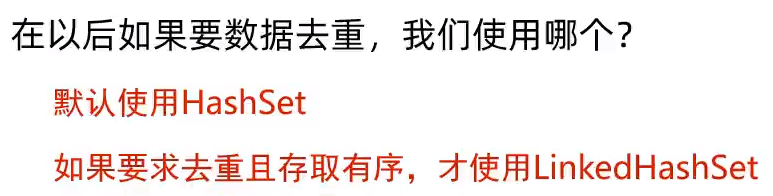
**两个注意点：**

1、JDK8以后，当链表长度超过8而且数组长度大于等于64时，自动转换为红黑树

2、如果集合中存储的是自定义对象，必须要重写hashcode和equals方法

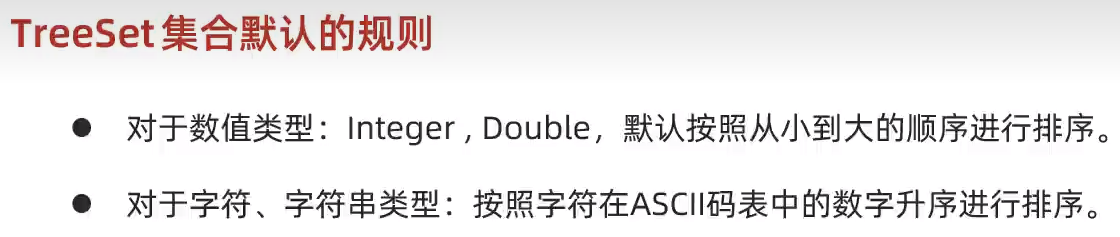
# LinkedHashSet底层原理

# 

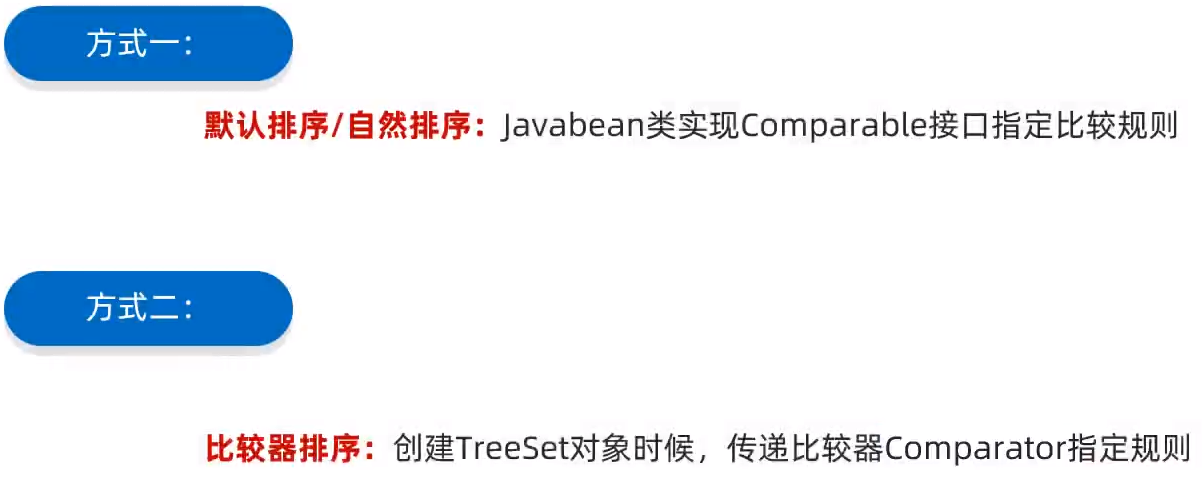


# Treeset

Treeset是默认自己排序的，它底层是红黑树结构



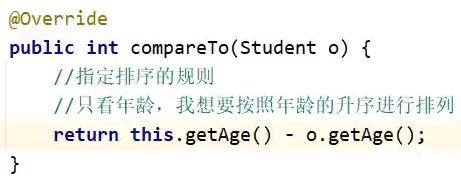
# Treeset的两种比较方式

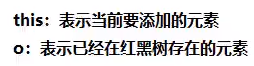


方式一

对于自定义对象，如果想存入Treeset，需要在类中实现comparable接口，重写compareTo方法来自己写排序的规则

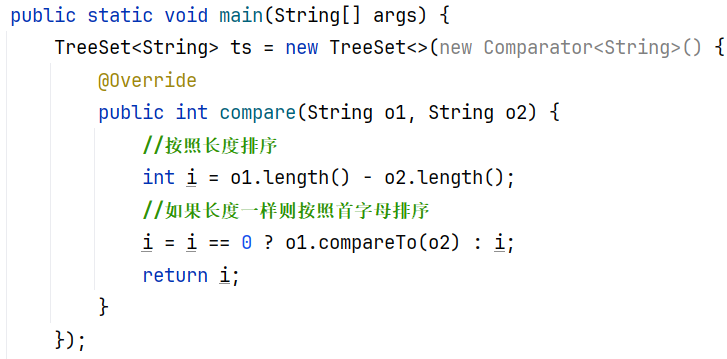


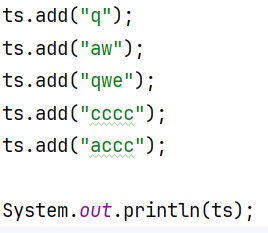




方式二

在创建treeset时就自定义Comparator对象，重写里面的compare方法

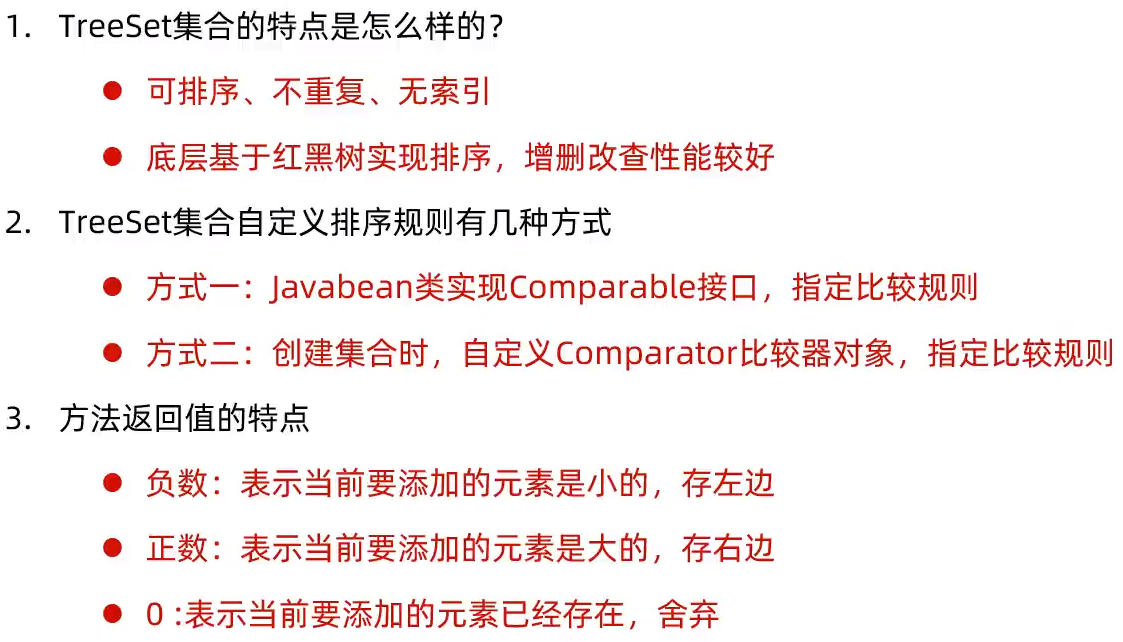




结果：

# Treeset总结

能用方式一就不用方式二，当方式一和方式二两种规则同时存在，以方式二为准



# 集合使用总结

