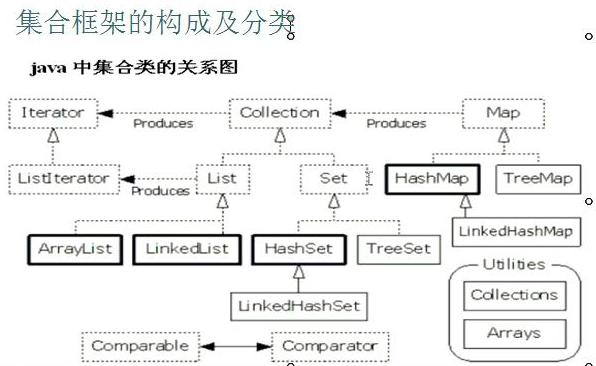
# Collections

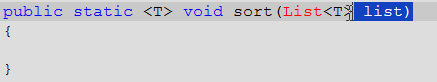


## sort方法

Collections.sort()方法声明如下：



看到上面的声明会觉得很复杂，所以先简单化如下：



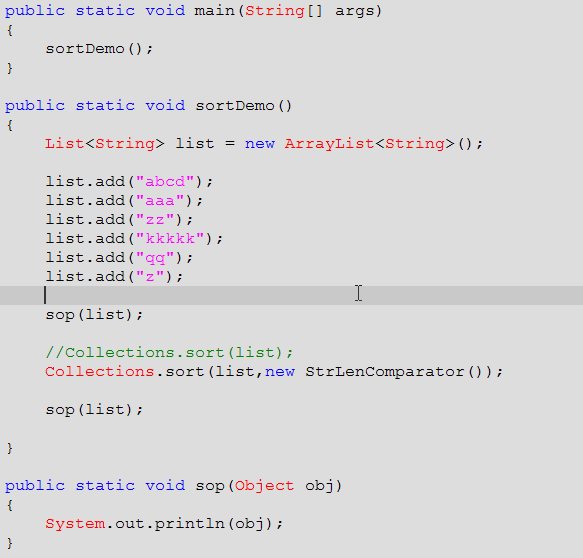
到这里想一个问题，TreeSet集合能对元素进行排序是因为元素实现Comparable，而Collections中的sort方法能对List集合中元素排序的话那也是一样的道理，应该要求List集合中的元素也要实现Comparable，所以“<T extends Comparable>”这个泛型的声明就限定了T这个元素必须是Comparable或者是Comarable的子类，而Comarable本身也是泛型类，通过前面学习泛型的时候也有例子，Comparable是比较两个元素，Comparable<T>这是普通的限定，而Comparable<? Super T>则限定了比较器可以比较的元素可以是T或者是T的父类。

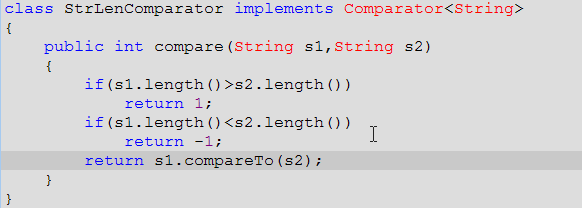
如果元素实现了Comparable，但是它的比较规则不是你所需要的，则你可以自己定义一个比较器，然后调用如下方法：



为什么这个方法中“static <T> void sort(….)”中的<T>不需要像前面的sort方法一样的限定呢，因为这里的排序限定已经在参数里的“Comarator<? Super T>”中进行了限定了。

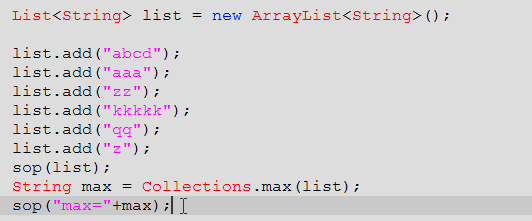
实例如下：





## max方法

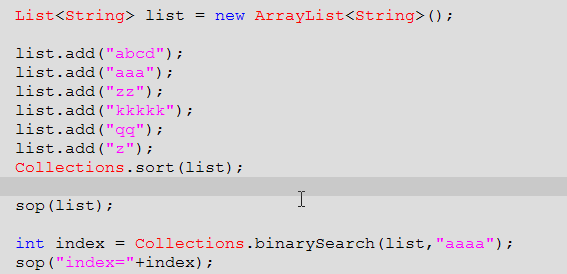
获取最大值，要拿最大值就需要比较，所以元素必须具体比较性，也就是必须实现Comparable接口



输出: max = zz，在max方法中也可以传一个比较器来自定义比较的规则。

## binarySearch方法

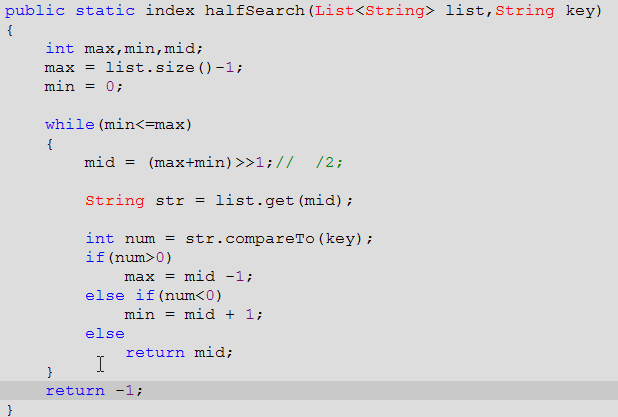
查找返回的是角标，所以方法接受的参数一定是带脚标的，List中的元素是有脚标的。



这里返回的是-2，因为没有查找到所以是负数，aaaa如果插入进去的话，排序后应该在角标为1的位置（即为aaaa的插入点），-2即为负插入点-1，这个也可以从JDK文档了解到：



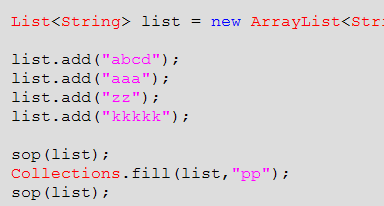
折半查找法模拟：



如果想模拟上面的在没找到的时候返回插入点，则可以把最后一个return改成这样：



## fill方法



它的功能为将集合中的所有元素都替换为“PP”元素。

练习，把List集合中指定一段连续的元素替换为指定元素：

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

List<String> list = **new** ArrayList<String>();

list.add("a");

list.add("b");

list.add("c");

list.add("d");

list.add("e");

list.add("f");

list.add("g");

System.*out*.println(list);

*fill*(list, "xx", 1, 6);

System.*out*.println(list);

}

**public** **static** **void** fill(List<String> list, String key, **int** start, **int** end) {

List<String> subList = list.subList(start, end);

Collections.*fill*(subList, key);

}

## replaceAll方法



## reverse方法

返转元素的顺序

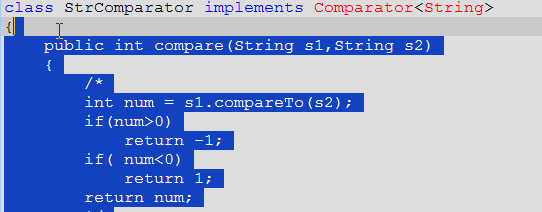


## reverseOrder方法

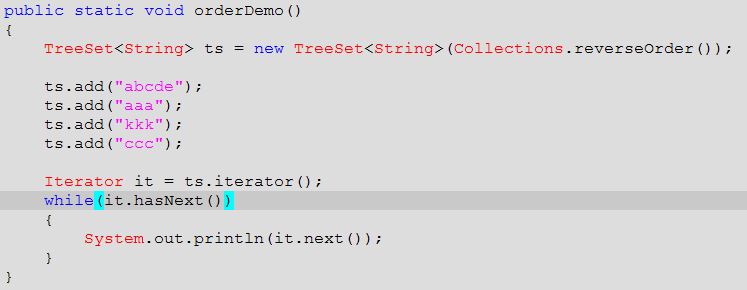
之前学的TreeSet是有顺序的，如果TreeSet存String，想使用自然顺序的倒序，则要传一个比较器，而比较器自己写比较麻烦，有没有一种办法可以把一个已经存在的比较器的顺利反转呢？在Collections中就有提供这样的方法：



其实原理也很简单，拿已存在的比较器进行比较，如果原来的比较器返回值大于0，则我们返回-1，如果小于0则返回1，否则返回0，这样就是把原先的顺序反过来了，模拟代码如下：



实例：



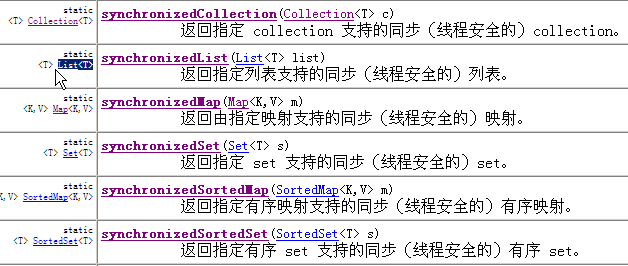
reverseOder还有一个重载方法，它可以把一个比较器反转：



这样有什么好处，如写好了一个比较器是按字符串长度排序的，按短到长排，但是如果有时需要按长到短排呢，如果去修改原来写好的比较器，这样不好，因为要的功能只是反转顺序，所以可以用reversOrder生成一个反转顺序的比较器，如：Comparator comp = Collections.reverseOrder(new MyComparator());

## 同步集合方法

如果需要线程安全，则需要使集合实现线程同步，通过Collections可以轻松实现：



查看同步的原理：

**public** **static** <T> List<T> synchronizedList(List<T> list) {

**return** (list **instanceof** RandomAccess ?

**new** SynchronizedRandomAccessList<T>(list) :

**new** SynchronizedList<T>(list));

}

查看SynchronizedList是如何实现同步的：

构造函数：

SynchronizedList(List<E> list) {

**super**(list);

**this**.list = list;

}

同步封装：

**public** **void** add(**int** index, E element) {

**synchronized**(mutex) {list.add(index, element);}

}

**public** E remove(**int** index) {

**synchronized**(mutex) {**return** list.remove(index);}

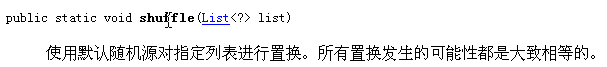
}

## swap方法

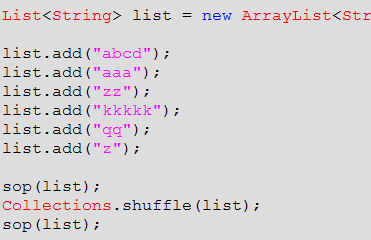


reverse方法底层用的是swap方法来实现的

## shuffle方法



示例：



打印结果：



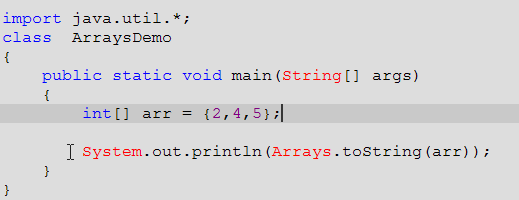
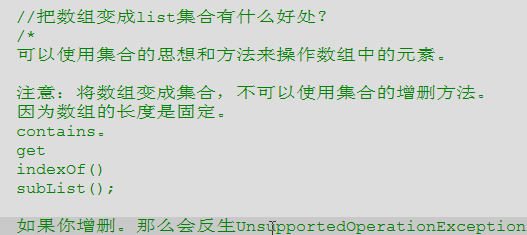
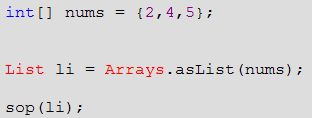
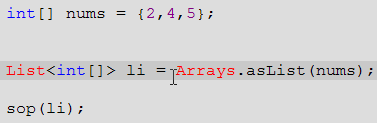
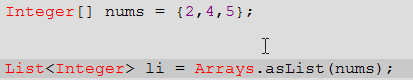
如果再运行一下结果如下：



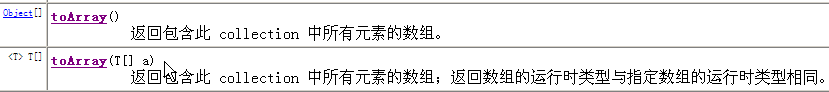
应用：

如斗地主洗牌

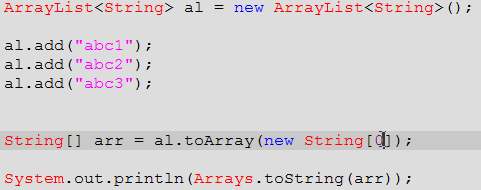
# Arrays

* [**binarySearch**](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/util/Arrays.html#binarySearch(byte[], byte))二分查找
* [**copyOf**](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/util/Arrays.html#copyOf(byte[], int)) 复制
* [**copyOfRange**](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/util/Arrays.html#copyOfRange(boolean[], int, int)) 范围复制
* [**equals**](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/util/Arrays.html#equals(byte[], byte[]))比较数组中的内容是否相同
* [**deepEquals**](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/util/Arrays.html#deepEquals(java.lang.Object[], java.lang.Object[]))深度比较，比如数组里装了4个容器，不但比较数组里的4个容器是否相同，还要比较这4个容器里面的内容是否相同
* [**fill**](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/util/Arrays.html#fill(boolean[], int, int, boolean))替换
* [**hashCode**](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/util/Arrays.html#hashCode(boolean[]))返回数据的哈希值
* [**sort**](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/util/Arrays.html#sort(byte[], int, int))排序
* [**toString**](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/util/Arrays.html#toString(boolean[]))把数组变成字符串  
  应用：  
    
  输出如下：  
    
  如果直接打印数组打印出来的是“数组类型@数组的hashCode”。
* [**asList**](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/util/Arrays.html#asList(T...)) 把数组转换为List  
    
    
    
  这打印的将是数组的哈希值，如果加泛型是这样：  
    
  这说明了List集合中装的是一个int数组，而不是一个一个的int值。  
    
  所以应该这样做：  
  

# 集合转换为数组



第二个toArray方法可以指定泛型，示例如下：

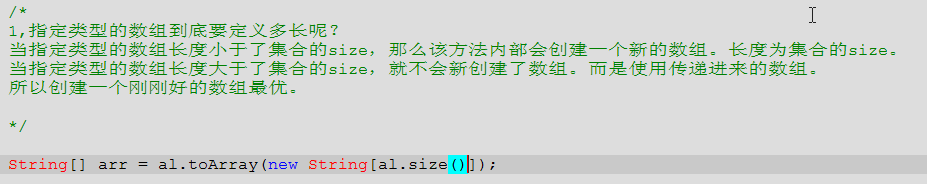


如上，集合变数组的时候不需要类型强转。

上面的一句代码改成这样：



则arr中将会有5个元素，最后两个元素为null，原因如下：





当你返回的数据只能用于别人读取，而不允许别人进行修改的时候可以这样做。比如你写了一个方法，可以返回一个集合，而如果别人对这个集合进行修改那方法内对应的集合也会被修改，所以可以先把集合转成数组再返回给方法，这样别人可以修改这个数组，但是对原集合中的数据没有影响，如：一个目录下有5个文件，这5个文件保存在一个集合中，别人要获取信息时就先转换为数组在返回，这要别人把数组中的文件删除或者增加一个都没关系，因为集合中的文件还在，不会多也不会少，这样别人只能读取，而增删的操作只能自己来做，别人做不了。

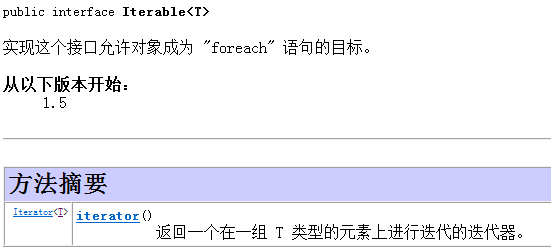
# 增强for循环

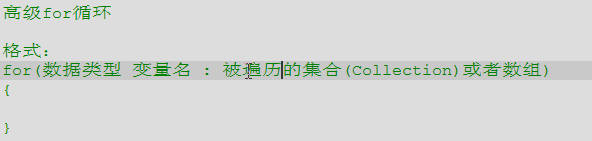
Jdk1.2 ~ 1.4，Collection中的iterator是封装在方法中的：Collection.iterator()

Jdk1.5开始把iterator抽取到了类上面，看JDK文档可知，如下：



再看Iterable接口的文档如下：





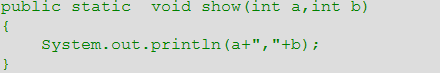
Map不支持高级for循环

高级for循环优点：书写方便

缺点：只能读取，而Iterator多了一个remove方法，如果是ListIterator则增删改查都可以。

注：如果集合没有使用泛型，则在使用高级for循环的时候只能把对象声明为Object

# 可变参数

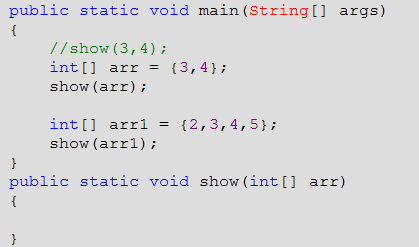


上面的方法为打印a和b

如果再要打印多一个c呢，这时需要重载方法，如下：

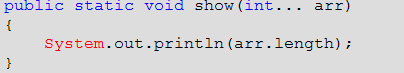


如果还有d、e、f，那就需要很多重载方法，所以要想办法解决这个麻烦，我们看方法的参数都是一个类型，因此可以使用数组：





所以可变参数就解决了这个问题



从上图可发现可变参数中的arr实际上还是数组，但是我们在调用这个方法的时候就不需要再new 一个数组来传数据了，可以这样：show(1,2,3,4)；这里封装成数组的操作由虚拟机来完成。

也可以直接show()；不传任何参数，则方法内的arr为长度为0的数组



如：public static void show(String name, int… arr)，因为在调用这个方法的时候可以确定第一个参数是String，后面的所有参数都是int数据可以封闭为int数组传递给arr，如果把String name参数放在最后如：show(int… arr,String name)这样会有问题，因为在调用show方法的时候无法确定前面多少个数据是int。

# 静态导入



