API对象

笔记本:	JAVA		
创建时间:	2018/8/27 22:04	**更新时间:**	2018/8/31 21:46
作者:	Debao Xu		

String

对象也可以比较大小、相等，都是通过方法完成。
对象比较相同是通过Object类中的 boolean **equals**(Object obj) 来完成；对象比较大小是通过方法**compareTo()**，其结果有三种可能情况：正数、负数、零，前者大于后者返回正数，前者小于后者返回负数，前者等于后者返回零。

```
public class StringDemo
{
	public static void main(String[] args)
	{
		String[] ss = { "nba", "Cba", "qq" };
		printArray(ss);
		sort(ss); // 见到“小叉”，按快捷键 Ctrl+1，改变修饰符按Tab键，结束按回车。
		printArray(ss);
	}
	private static void sort(String[] ss) //冒泡排序（大圈套小圈）
	{
		for (int i = 0; i < ss.length-1; i++)
		{
			for (int j = i+1; j < ss.length; j++)
			{
				if(ss[i].compareTo(ss[j])>0)
				{
					swap(ss, i, j);
				}
			}
		}
	}
	private static void swap(String[] ss, int i, int j)
	{
		String temp = ss[i];
		ss[i] = ss[j];
		ss[j] = temp;
	}
	// 代码标准格式化快捷键：Ctrl+Shift+F
	// 光标到指定的位置快捷键：Ctrl+A
	public static void printArray(String[] ss) // 抽取方法快捷方式：右击->重构->抽取方法
	{
		for (int i = 0; i < ss.length; i++)
		{
			if (i != ss.length - 1)
				System.out.print(ss[i] + ",");
			else
				System.out.println(ss[i]);
		}
	}
}
运行结果：
nba,Cba,qq
Cba,nba,qq
```

```
public class String1Demo {
	public static void main(String[] args) {
		/*
		 * 查找一个字符串在整串中出现的次数
```

```
		 * 思路：
		 * 1，要找的子串是否存在，如果存在获取其出现的位置。这个可以使用indexOf完成。
		 * 2，如果找到了，那么就记录出现的位置并在剩余的字符串中继续查找该子串，而剩余字符串的起始位是出现位置+子串的长度.
		 * 3，以此类推，通过循环完成查找，如果找不到就是-1，并对每次找到用计数器记录。*/

		String str = "bahunbauhunabfdnbafdnba";
		String key = "nba";

		int count = getKeyStringCount(str,key);//先写调用的函数名，再用Ctrl+1来创建函数体
		System.out.println("count="+count);
	}
	private static int getKeyStringCount(String str, String key) {
		int count = 0;
		int index = 0;
		while((index = str.indexOf(key, index)) != -1) //indexOf(key, index):返回指定子字符串在此字符串中第一次出现处的索引
		{
			index += key.length();
			count++;
		}
		return count;
	}
}
```

StringBuffer

就是字符串缓冲区，用于存储数据的容器。

特点：

1，长度的可变的。

2，可以存储不同类型数据。

3，最终要转成字符串进行使用。

4，可以对字符串进行修改(增删改查)。

例如创建字符串：

```
String str = "a"+4+"c";
```

上面代码在内存中的过程如下，1、创建一个字符串缓冲区容器。2、将要组成字符串的元素先存储起来。3、最后将缓冲区填充的数据变成字符串（用toString方法），等效于下面的代码：

```
str = new StringBuffer().append("a").append(4).append("c").toString();
```

StringBuffer的操作：

```
StringBuffer sb = new StringBuffer(); //创建缓冲区对象
sb.append("aeryutibc");//追加字符串
System.out.println(sb);//println方法会将所有要打印的数据先转换成字符串再输出，对于对象会自动调用toString方法，然而对于其他情况下必须要加上toString()，如sb.toString()

sb.insert(1, true);//插入一个布尔值true
System.out.println(sb);

sb.delete(1, 4);//删除“包含头，不包含尾”。扩展一下，操作一段数据都是“包含头，不包含尾”
System.out.println(sb);

sb.replace(1, 5, "false");//替换字符
System.out.println(sb);


运行结果：
aeryutibc
atrueeryutibc
aeeryutibc
afalseutibc
```

我们知道字符缓冲区的长度是可变的，那么这是为什么呢？因为字符缓冲区中维护了一个“可变长度的数组”，所谓可变长度的数组其实就是如果超出了内部数组长度后，会新建一个是原数组长度1.x倍的新数组，并将原数组中的元素复制到新数组，另外新的元素也被添加到新数组中，这样就有了更长长度的新数组。

```
public class String3Demo {
	public static void main(String[] args) {
		String rec = draw(5,9);
		System.out.println(rec);

		int[] arr = {23,55,78,99};
		String array = tostring(arr);
		System.out.println(array);
	}

	//通过缓冲区，将要打印的矩形元素*进行存储后，一次性返回，并输出。（先存储，再输出）
	public static String draw(int row, int col) {

		StringBuffer sb = new StringBuffer();//创建缓冲区对象
		for(int i = 0; i < row; i++)
		{
			for(int j = 0; j < col; j++ )
			{
				sb.append("*");
			}
			sb.append("\n"); //换行
		}
		return sb.toString(); //将缓冲区中的内容转换为字符串再返回
	}

	//将int数组的元素转换成字符串，格式为：[23,55,78,99]，代码结构同上
	public static String tostring(int[] array)
	{
		StringBuffer sb = new StringBuffer();
		sb.append("[");
		for (int i = 0; i < array.length; i++) {
			if(i != array.length-1)
				sb.append(array[i]+",");
			else
				sb.append(array[i]+"]");
		}
		return sb.toString();
	}
}
//什么时候用字符串缓冲区StringBuffer?
答：数据很多，个数无所谓确定，类型无所谓确定，只要最后都转换成字符串，就可以使用StringBuffer这个容器
StringBuffer使用的局限性：1、必须最终转换成字符串，2、无法对存储进来的元素进行单独操作，因为存储进来的元素都变成了字符串，“成了一个整体了”
```

```
public class String3Demo {
	public static void main(String[] args) {

		StringBuffer buf1 = new StringBuffer("java");
		StringBuffer buf2 = new StringBuffer("Hello");
		test(buf1,buf2);
		System.out.println(buf1+"..."+buf2);
	}

	public static void test(StringBuffer b1,StringBuffer b2)
	{
```

```
            b1.append(b2);
            b1 = b2;
        }
}
运行结果:
javaHello...Hello
//在函数test中，b2中的内容被追加到了b1中，而"b1 = b2"只是将b2的引用赋给了b1。
```

JDK1.5以后，出现了和StringBuffer一样用法的方法：StringBuilder，这两者的区别就是，StringBuffer是线程同步的，StringBuilder是线程不同步的，**一般建议选择StringBuilder，因为它的速度更快**。

基本数据类型对象包装类

```
public class WrapperDemo {
        public static void main(String[] args) {
            /*
             * 基本数据类型对象包装类
             * 将基本数据类型封装成了对象
             * 好处：可以在对象中定义更多属性和行为，对基本数据进行操作
             *
             *   关键字       类
             *   byte        Byte
             *   short       Short
             *   int         Integer
             *   long        Long
             *   boolean     Boolean
             *   float       Float
             *   double      Double
             *   char        Character
             *
             *   基本数据类型对象包装类的重要功能：在基本数据类型和String类型之间互相转换
             */

            //举例：int的范围最值，只有int自己最清楚，所以找int对应的对象最合适
            System.out.println(Integer.MAX_VALUE);//MAX_VALUE求取Interger的最大值
            System.out.println(Integer.toBinaryString(7));//toBinaryString将十进制转换为二进制

            //字符串转换成基本数据类型
            //使用的是parse基本数据类型(字符串); parseInt  parseByte  parseDouble
parseBoolean
            System.out.println(Integer.parseInt("123")+5);//parseInt将字符串解析为十进制数,这
个方法要抛异常NumberFormatException，但是它是RuntimeException的子类，所以可以不用声明

            System.out.println(Integer.parseInt("110", 8));//可以将其他进制的字符串转为十进制整
数


            //基本数据类型转换为字符串
            System.out.println(""+3+4);
            System.out.println(Integer.toString(3)+8);


            //为了对整数进行更多的操作，可以将整数封装成对象，通过Integer的方法完成
            //有两种方法将整数转换为对象，即 int-->Integer

            Integer i1 = new Integer(4); //构造方法
            Integer i11 = new Integer("4"); //构造方法

            Integer i2 = Integer.valueOf(4); //静态方法
```

```
            //将对象转换为整数，即 Integer-->int
            Integer i3 = new Integer(7); //对象Integer
            int x1 = i3.intValue(); //转换为int型


            //JDK1.5以后，自动装箱/拆箱，就是要像操作int一样操作Integer对象
            Integer i = 4;//自动装箱，等价于 Integer i = Integer.valueOf(4);
            i = i + 6;//等号右边的i自动拆箱，等价于i.intValue() + 6,但是运算结束之后又一次装箱并
赋值给i
        }
}
运行结果:
2147483647
111
128
72
34
38
```

练习:

```
/*
 * 对字符串中的数值进行升序排序，生成一个数值有序的字符串
 * 思路:
 * 1、对整数数值进行排序
 * 2、排序的数值都在字符串中，如何取出?
 * 3、找String类的功能，可以发现数字之间的间隔都是相同的空格，有规律，结果找到方法String[]
split(String)
 * 4、将获取到的数字格式的字符串转换成数字存储到数组中
 * 5、对数组排序
 * 6、将数组转成字符串
 */
import java.util.Arrays;
public class String4Demo {
        public static void main(String[] args) {
                String numStr = "23 10 -4 0 9 87 32 109";
                String sortStr = sortNumberString(numStr);
                System.out.println(sortStr);
        }

        public static String sortNumberString(String numStr)
        {
                //1、将给定的字符串分解成多个数字格式字符串
                String[] numStrs = toStringArray(numStr);

                //2、将字符串数组转换成int数组
                int[] nums = toIntArray(numStrs);

                //3、对数组排序
                sort(nums);

                //4、将int数组转换成字符串
                return toString(nums);
        }


    //将int数组转成字符串
    private static String toString(int[] nums) {
                //1、定义一个字符串缓冲区
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < nums.length; i++) {
                        if(i != nums.length-1)
```

```
                        sb.append(nums[i]+" ");
                else
                        sb.append(nums[i]);
        }
    return sb.toString();
    }

    //对int数组进行升序排序
    private static void sort(int[] nums) {
    Arrays.sort(nums);
    }

    //将字符串数组转换成int型数组
    private static int[] toIntArray(String[] numStrs) {
        //1、创建一个int类型的数组，长度和字符串数组的长度一致
        int[] nums = new int[numStrs.length];

        //2、对字符串数组进行遍历
        for (int i = 0; i < numStrs.length; i++) {
            //3、将字符串数组中的元素通过parseInt转换后，赋值给int类型的数组
            nums[i] = Integer.parseInt(numStrs[i]);
        }
        return nums;
    }

    //将字符串按照指定的分隔，转换成字符串数组
    private static String[] toStringArray(String numStr) {
        //使用字符串的split(regex)
        return numStr.split(" ");
    }
}
运行结果:
-4 0 9 10 23 32 87 109
```

集合类

为什么出现集合类?

面向对象语言对事物的体现都是以对象的形式，所以为了方便对多个对象的操作，就**对对象进行存储**，**集合就是存储对象最常用的一种方式**。

数组和集合类同是容器，**有何不同**?

数组虽然也可以存储对象，但长度是固定的；集合长度是可变的。数组中可以存储基本数据类型，集合只能存储对象。

集合类的特点

集合只用于存储对象，集合长度是可变的，集合可以存储不同类型的对象。

因为容器中数据结构不同，容器有很多种，可以对各种容器的共性功能抽取，这样就形成了集合体系，即集合框架。集合框架的顶层称之为Collection接口。Collection接口中定义了集合的基本功能。

Collection接口中的共性功能

1、添加:

boolean add(Object obj)；一次添加一个

boolean addAll(Collection c); 将指定容器中的所有元素添加，因为参数就是一个Collection

2、删除:

void clear(); 清空集合

boolean remove(Object o)；删除一个对象，返回值是布尔类型的，因为要确认是否删除成功。

boolean remove(Collection c)；删除一部分对象

3、获取长度

int size(); //**对应于字符串、缓冲区等int length()方法**

4、判断

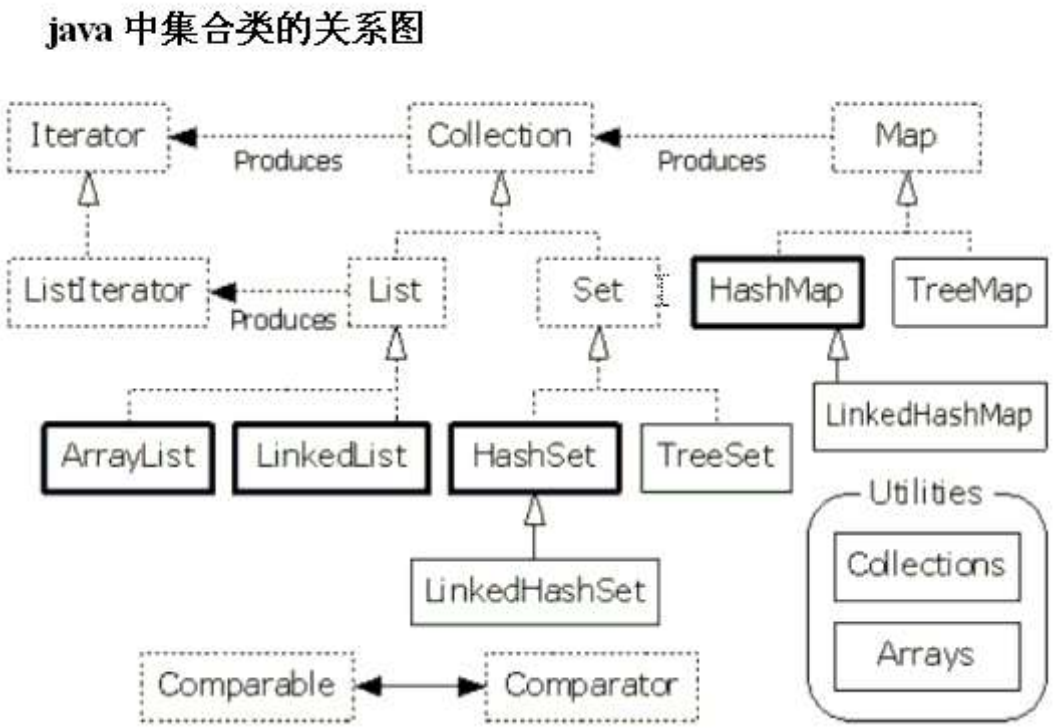
boolean isEmpty(); 判断是否为空
boolean contains(Object o)；判断是否包含某一个对象
boolean contains(Collection c)；判断是否包含某一批对象

5、将集合转成数组
toArray();
toArray([]);

6、**取出集合元素**
iterator，获取集合元素上迭代功能的迭代器对象。
迭代：获取元素的一种方式。有没有啊？有！取一个。还有没有啊？有！再取一个。还有没有啊？没有了！算了，不取了。
迭代器：具备着迭代功能的对象，而**迭代器对象不需要new**，直接通过iterator()方法获取即可。
迭代器是取出Collection集合中元素的公共方法

```
import java.util.ArrayList;
import java.util.Collection;
public class CollectionDemo {
	public static void main(String[] args) {
		Collection coll = new ArrayList(); //接口的多态表示，因为ArrayLIst中已经实现了Collection接口
		//methodDemo(coll);
		methodAllDemo();
	}
	public static void methodDemo(Collection coll) //单个对象的操作方法
	{
		//添加
		coll.add("abc1");
		coll.add("abc2");
		coll.add("abc3");

		//删除
		//coll.remove("abc2");

		//清空
		//coll.clear();

		//包含
		System.out.println("contains:"+coll.contains("abc2"));
		System.out.println(coll.toString());
	}
	public static void methodAllDemo() //所有对象的操作方法，都有All
	{
		//1、创建两个容器
		Collection c1 = new ArrayList();
		Collection c2 = new ArrayList();

		//2、添加元素
		c1.add("abc1");
		c1.add("abc2");
		c1.add("abc3");
		c1.add("abc4");

		c2.add("abc5");
		c2.add("abc1");
		c2.add("abc9");

		//3、往c1中添加c2
		//c1.addAll(c2);
		//System.out.println(c1);

		//4、判断c1中是否包含c2中的所有元素
		//boolean b = c1.containsAll(c2);
		//System.out.println("b="+b);
```

```
		//5、从c1中删除c2
		//c1.removeAll(c2);//将c1中和c2中相同的元素从c1中删除
		//c1.retainAll(c2);//将c1中和c2中不同的元素从c1中删除，和remove方法相反
		System.out.println(c1);
	}
}
```



上面展示了常用的集合，在Collection中，有两个重要的接口，List和Set。
List：有序（存入的顺序和取出的顺序是一致的），有索引，允许重复元素
Set：不允许重复元素

重点是List接口中特有的方法：它的特有方法都是围绕索引定义的。支持“增删改查”
增：add(index , element)
删：remove(index)
改：set(index , newelement)
查：int indexOf(element)　element get(index)

演示List的**特有的方法**

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
public class ListDemo {
	public static void main(String[] args) {

		List list = new ArrayList(); //导入所有的包 Ctrl+Shift+O
		methodDemo(list);
	}

	//演示List特有的方法
	public static void methodDemo(List list)
	{
		//1、添加常规元素
		list.add("abc1");
		list.add("abc2");
		list.add("abc3");

		//2、插入元素
```

```
		//list.add(1,"haha");

		//3、删除
		//list.remove(2);

		//4、获取
		//System.out.println(list.get(1));
		//System.out.println(list.indexOf("abc3"));

		//5、修改
		//list.set(1, "yuy");
		//System.out.println(list);

		//取出集合中所有的元素,用迭代器
		/*for (Iterator iterator = list.iterator(); iterator.hasNext();) {
			System.out.println(iterator.next());
		}*/

		//取出集合中所有的元素,用list集合特有的取出方式。list没有length方法，而是用size方法，然后用get方法取出
		for (int i = 0; i < list.size(); i++) {
			System.out.println(list.get(i));
		}
	}
}
```

列表迭代器ListIterator（在迭代过程中使用）

```
import java.util.ArrayList;
import java.util.List;
import java.util.ListIterator;
public class ListIteratorDemo {
	public static void main(String[] args) {
		List list = new ArrayList();
		list.add("abc1");
		list.add("abc2");
		list.add("abc3");
		list.add("abc4");

		//在遍历的过程中，如果遍历到abc2，添加一个元素haha
		/*for (Iterator it = list.iterator(); it.hasNext();) {
			Object obj = it.next();//java.util.ConcurrentModificationException，迭代过程中使用了集合对象同时对元素进行操作，导致了迭代的不确定性，从而引发了该异常
			//解决方法：在迭代过程中，想要执行一些操作，使用迭代器的方法就可以了
			if(obj.equals("abc2"))
				list.add("haha");
		}*/


		//上面的for循环不能完成任务，可以用List集合特有的迭代器 ListIterator，通过List集合的方法listIterator()获取该迭代器对象
		//ListIterator可以实现迭代过程中的增删改查。因此在迭代过程中想要对元素进行操作的时候就用列表迭代器
		for (ListIterator it = list.listIterator(); it.hasNext();) {
			Object obj = it.next();
			if(obj.equals("abc2"));
				it.add("haha");
		}

		System.out.println(list);
	}
}
```

运行结果：

```
[abc1, haha, abc2, haha, abc3, haha, abc4, haha]
```