集合类2

笔记本:	JAVA		
创建时间:	2018/9/5 9:31	**更新时间:**	2018/10/31 15:30
作者:	Debao Xu		

练习4：对多个字符串（**不重复**）按照长度排序（**由短到长**）。
思路:
1、多个字符串，需要容器存储
2、选择哪个容器，对象是字符串，可以选择集合，而且不重复 ，可以选择Set集合
3、还需要排序，可以选择TreeSet集合

```
package cn.itcast.set.test;
import java.util.Comparator;
public class ComparatorByLength implements Comparator {
	public int compare(Object o1, Object o2) {

		//向下转型，对字符串按照长度比较
		String s1 = (String)o1;
		String s2 = (String)o2;

		//比较长度，如果长度相同，再按照字典顺序比较
		int temp = s1.length()-s2.length();
		return temp==0?s1.compareTo(s2):temp;
	}
}
```

```
package cn.itcast.set.test;
import java.util.Comparator;
import java.util.Set;
import java.util.TreeSet;
public class Test {
	public static void main(String[] args) {
		sortStringByLength();
	}

	public static void sortStringByLength()
	{
		Set set = new TreeSet(new ComparatorByLength());

		set.add("fgdr");
		set.add("fgde");
		set.add("hghdyte");
		set.add("nab");
		set.add("de");

		for(Object obj : set) //增强型for循环
		{
			System.out.println(obj);
		}
	}
}
```

运行结果:

```
de
nab
fgde
fgdr
hghdyte
```

练习5：对多个字符串(**有重复**)，按照长度排序（**由短到长**）。
思路：
1、能使用TreeSet吗？不能，因为有重复的元素
2、可以存储到数组，list，这里先选择数组

```
package cn.itcast.set.test;
import java.util.Comparator;
public class ComparatorByLength implements Comparator {

	public int compare(Object o1, Object o2) {

		//向下转型，对字符串按照长度比较
		String s1 = (String)o1;
		String s2 = (String)o2;

		//比较长度，如果长度相同，再按照字典顺序比较
		int temp = s1.length()-s2.length();
		return temp==0?s1.compareTo(s2):temp;
	}
}
```

```
package cn.itcast.set.test;
import java.util.Comparator;
import java.util.Set;
import java.util.TreeSet;
public class Test {
	public static void main(String[] args) {
		sortStringByLength2();
	}
	public static void sortStringByLength2()
	{
		String[] strs = {"nbate","pto","nbate","cctv","zs","cctv"};

		//自然排序可以使用String类中的compareTo()方法
		//但是对于长度排序，这就需要比较器，因此定义一个按照长度排序的比较器对象
		Comparator comp = new ComparatorByLength();

		for(int x = 0;x < strs.length;x++)
		{
			for(int y = x+1;y < strs.length;y++)
			{
				//if(strs[x].compareTo(strs[y])>0) 对象调用自己的方法，即对象自身具有比较
的功能
				if(comp.compare(strs[x], strs[y])>0)
				{
					swap(strs,x,y);
				}
			}
		}
		for(String s : strs) //增强型for循环
		{
			System.out.println(s);
		}
	}
	private static void swap(String[] strs, int x, int y) {
		String temp = strs[x];
```

```
                strs[x] = strs[y];
                strs[y] = temp;
        }
}
```

运行结果：

```
zs
pto
cctv
cctv
nbate
nbate
```

泛型

泛型的由来：

在jdk1.4之前，容器什么对象都可以存储，但是在取出时，需要用到对象的特有内容时，需要向下转型，但是由于对象的类型不一致，就会导致向下转型发生了ClassCastException异常，为了避免这个问题，只能主观上控制，让集合中存储的对象的类型保持一致。

而在jdk1.5以后就解决了该问题，即在定义集合时，就直接明确集合中存储元素的具体类型。这样，编译器在编译时，就可以对集合中存储的元素的类型进行检查，一旦发现类型不匹配，就会编译失败，这就是泛型技术。

泛型的好处

1、将**运行时期的问题转移到了编译时期**，可以更好的让程序员发现问题并解决问题

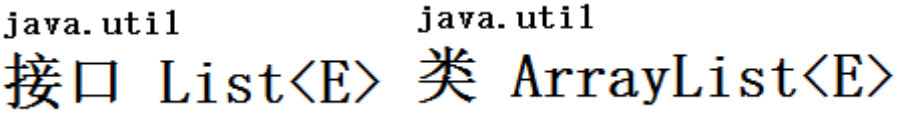
2、避免了向下转型的麻烦（因为**类型已经明确**，所以不用再向下转型了）

以前在定义数组时，有 int[] arr = new int[2]；这就表明了该数组只能存储整型元素，如果输入 arr[0] = 2.0；就会报错，这样做的好处是能够**在编译时就发现类型错误**。同理，为了运行时不出现“类型异常”，可以在定义容器时，就明确容器中元素（对象）的类型，并用符号<>来表示，这种技术就称为“泛型”。具体的做法就是在容器后加上一个参数，该参数就表明了容器中所存储的对象的类型。

```
package cn.itcast.generic.demo;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
public class GenericDemo {
        public static void main(String[] args) {

                //List list = new ArrayList(); 这是不使用泛型技术的情况

                List<String> list = new ArrayList<String>(); //使用泛型技术<String>明确容器中存储的是String类型的元素
                list.add("abc");
                //list.add(4); 通过上面泛型技术明确该容器中只能存储String类型时，这里的add(4)就不能够存入容器中，这样编译时就会报错

                for (Iterator<String> it = list.iterator(); it.hasNext();) {
                        //在迭代器Iterator中也要明确迭代出的是String类型
                        String str = it.next();
                        System.out.println(str.length());
                }
        }
}
```

泛型的擦除：编译器通过泛型对文件类型进行检查，只要检查通过，就会生成class文件。但是有可能虚拟机可能并没有进行“升级”，因此，在生成的class文件中，**会将泛型标识（例如 <String>）去掉**，这就是泛型的擦除。

泛型的表现：泛型技术在集合框架中应用的范围很大。什么时候需要用泛型呢？查阅API文档，例如下，



只要看到类或者接口在描述时右边定义了尖括号<E>，就需要使用泛型。尖括号中的参数E实际就是该容器中存储数据的类型。

泛型在比较器和比较方法上的应用

```
package cn.itcast.generic.demo;

//定义Person类来实现Comaparable
public class Person implements Comparable<Person> {
	private String name;
	private int age;
	public Person() {
		super();
	}
	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;
	}

	 //覆盖compareTo方法
	public int compareTo(Person o) {
		int temp = this.age-o.age;
		return temp == 0?this.name.compareTo(o.name):temp;
	}
	public String toString()
	{
		return "Person [name = " + name + ", age = "+ age + "]";
	}
}
```

```
package cn.itcast.Comparator2;

import java.util.Comparator;
import cn.itcast.generic.demo.Person;

//定义比较器，按姓名进行比较，实现Comparator，采用泛型<Pesrson>
public class ComparatorByName implements Comparator<Person> {
	public int compare(Person o1, Person o2) {
		int temp = o1.getName().compareTo(o2.getName());
		return temp == 0?o1.getAge() - o2.getAge() : temp;
	}
}
```

```
package cn.itcast.generic.demo;
import java.util.Set;
import java.util.TreeSet;
import cn.itcast.Comparator2.ComparatorByName;
public class GenericDemo2 {
	public static void main(String[] args) {

		//采用泛型，并传入比较器 new ComparatorByName() (按姓名进行排序)
		Set<Person> set = new TreeSet<Person>(new ComparatorByName());

		set.add(new Person("rex",25));
		set.add(new Person("ass",24));
		set.add(new Person("bex",28));
		set.add(new Person("hex",21));

		for(Person person : set)
			System.out.println(person);
	}
}
运行结果:
Person [name = ass, age = 24]
Person [name = bex, age = 28]
Person [name = hex, age = 21]
Person [name = rex, age = 25]
```

泛型类

创建一个用于操作Student对象的工具类，对对象进行设置和获取，下面的方法只能操作Student对象，太有局限性了，可不可以定义一个能操作所有对象的工具呢?

```
//该类只能操作Student
class Tool
{
	private Student stu;
	public Student getStu() {
		return stu;
	}
	public void setStu(Student stu) {
		this.stu = stu;
	}
}
```

可以将类型向上抽取，如果要操作的对象类型不确定的时候，为了扩展，可以使用Object类型来完成，但是这种方法有一些弊端，**容易出现ClassCastException（尤其在向下转型的时候）**，例如下，

```
class Tool
{
	private Object obj;
	public Object getObj() {
		return obj;
	}
	public void setObj(Object obj) {
		this.obj = obj;
	}
}
```

jdk1.5以后，对于类型不确定时，可以对外提供参数，由**使用者通过参数传递的形式完成类型的确定**，在定义类时就明确参数，由使用该类的调用者，来传递具体的类型，例如下

```
package cn.itcast.generic.demo;
import cn.itcast.domain.Student;
public class GenericDemo3 {
	public static void main(String[] args) {

/*		Tool tool = new Tool(); //不使用泛型类
		tool.setObj(new Student());
		Student stu = (Student)tool.getObj(); 通过(Student)来进行强制向下转型
		System.out.println(stu);*/

		Util<Student> util = new Util<Student>(); //使用泛型类
		util.setObj(new Student()); //如果类型不匹配（例如用了new Worker()），编译直接报错
		Student stu = util.getObj(); //避免了向下转型（此处不需要进行向下转型）
		System.out.println(stu);
	}
}
class Util<W> //把泛型定义在类上，这就是泛型类，即类上多了参数
{
	private W obj;
	public W getObj() {
		return obj;
	}
	public void setObj(W obj) {
		this.obj = obj;
	}
}
```

```
//描述Student类
package cn.itcast.domain;
public class Student extends Person {
	public Student() {
		super();
	}
	public Student(String name, int age) {
		super(name, age);
	}
	public String toString()
	{
		return "Student [name = " + getName() + ", age = "+ getAge() + "]";
	}
}
```

```
//描述Person类
package cn.itcast.domain;
public class Person {
	private String name;
	private int age;
	public Person() {
		super();
	}
	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;
	}
	public boolean equals(Object obj) //需要建立对象自己的equals方法。因为如果不建立，则使用Object的equals方法
	{
		if(!(obj instanceof Person))
		{
			throw new ClassCastException("类型错误！ ");
		}
		Person p = (Person)obj;
		return this.name .equals(p.name) && this.age == p.age;
	}
	public String toString()
	{
		return "Person [name = " + name + ", age = "+ age + "]";
	}
}
```

泛型方法（即泛型加在了方法上）

泛型方法存在的**原因**：当有了泛型类之后，即泛型加在了类上，这会使得类中所有方法的泛型都要和类的泛型保持一致（**例如，类的泛型为String，而类中的某些方法只能接收int类型的数据**），这就限定了类中方法的灵活性，因为类中有些方法的类型有时并不和类保持一致。因此，一个解决办法就是对于那些不和类的泛型保持一致的方法，我们将会单独为其设定泛型，这就是泛型方法。

```
package cn.itcast.generic.demo;
public class GenericDemo4 {
	public static void main(String[] args) {
		Demo<String> d = new Demo<String>();
		d.show("abc"); //show方法的类型要和类Demo“保持一致”，都为泛型<W>，此处为String类型

		d.print(12); //而print方法中参数的类型为泛型<Q>，并不和类Demo“保持一致”，因此可以接受任何类型
		d.print("dec");
	}
}
class Demo<W> //泛型加在了类上，即“泛型类”
{
	public void show(W w)
	{
		System.out.println("show:"+w);
	}

	public <Q> void print(Q q) //泛型<Q>加在了方法上，即得到“泛型方法”。**说白了，这里的泛型Q就是相当于Object类型**
	{
		System.out.println("print:"+q);
	}
}
```

运行结果：

```
show:abc
```

```
print:12
print:dec
```

泛型接口

```
package cn.itcast.generic.demo;
public class GenericDemo5 {
	public static void main(String[] args) {
		SubDemo d = new SubDemo();
		d.show("abc"); //该子类的泛型参数已经是String，故只能输入字符串
	}
}
interface Inter1<T> //将泛型定义在接口上，即泛型接口
{
	public void show(T t);
}
class InterImp<W> implements Inter1<W>{ //实现接口的类也有泛型W，并将之传递给接口Inter1
	public void show(W t) {
		System.out.println("show:"+t);
	}
}
class SubDemo extends InterImp<String>{ //子类继承父类，并明确子类的类型为String

}
```

泛型中的通配符？

在泛型中如果有类型不明确的情况，可以使用通配符？来进行表示

```
package cn.itcast.generic.demo;
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Set;
import cn.itcast.domain.Student;
public class GenericDemo6 {
	public static void main(String[] args) {
		List<Student> list = new ArrayList<Student>();//list中的是Sutdent类型
		list.add(new Student("rex",23));
		list.add(new Student("debr",27));
		list.add(new Student("rrthyex",26));
		printCollection(list);

		Set<String> set = new HashSet<String>(); //set中的是String类型
		set.add("xde");
		set.add("red");
		set.add("bede");
		printCollection(set);
	}
	private static void printCollection(Collection<?> coll) {
		//参数为Collection，因为List和Set的向上提取就是Collection接口，这样就可以接收所有类型的集合，上面的list和set中的类型不一样，这样在类型不明确的情况下，可以使用通配符？来表示

		for (Iterator<?> it = coll.iterator(); it.hasNext();) {
			Object obj = it.next(); //这里的迭代器有可能迭代出不同的类型对象，所以用Object修饰
			System.out.println(obj);
		}
```

```
        }
}
运行结果:
Student [name = rex, age = 23]
Student [name = debr, age = 27]
Student [name = rrthyex, age = 26]
red
xde
bede
```

泛型的限定

在上一个程序中，为了解决类型不确定的问题，使用了通配符？，这样就可以接收任何类型。现在我们需要缩小这个接收的范围（因为由上面的程序中，我们已经发现只是定义了两种类型Student和String，而在迭代器中却接收任何类型，这样就显得不是很好了）。因此，我们想要的就是，如果只定义了一部分类型，那么在接收的时候也只能接收一部分类型，这样就比较合适了，这就是**泛型的限定**。（即既不是只接收一种类型，又不是接受所有的类型，而是只接收需要的那一部分类型。）

泛型限定（有两种）：
1、 **? extends E** 意为接收E类型或者E的子类型（泛型的上限）
2、 **? super E** 意为接收E类型或者E的父类型（泛型的下限）

```
package cn.itcast.generic.demo;
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Set;
import cn.itcast.domain.Student;
import cn.itcast.domain.Worker;
public class GenericDemo6 {
	public static void main(String[] args) {
		List<Student> list = new ArrayList<Student>();
		list.add(new Student("rex",23));  //Student类型
		list.add(new Student("debr",27));
		list.add(new Student("rrthyex",26));
		printCollection(list);

		Set<Worker> set = new HashSet<Worker>();
		set.add(new Worker("xde",28)); //Worker类型
		set.add(new Worker("red",58));
		set.add(new Worker("bede",29));
		printCollection(set);
	}
	private static void printCollection(Collection<? extends Person> coll) {
		 //这里将原来的Collection<?>改为 Collection<? extends Person>，这里的泛型限定的意思就是只接收从Person中继承来的类型。

		   //for循环中也要进行泛型的限定
		for (Iterator<? extends Person> it = coll.iterator(); it.hasNext();) {
			Person obj = it.next();
			System.out.println(obj.getName()+":"+obj.getAge());
		}
	}
}
```

Map集合

Map集合与前面的Collection集合有很多相似之处。Map：双列集合，一次存一对（键、值对），并且要保证键的唯一性。之前学习的Collection为单列集合。
Map的共性功能：
1、添加。 v put(key,value); (put方法返回的v是与key关联的旧值) putAll(Map<k,v> map);
2、删除。 void clear(); v remove(key);
3、判断。 boolean containsKey(object); boolean containsValue(object); boolean isEmpty();
4、获取。 v get(key); int size();

```
package cn.itcast.map;
import java.util.HashMap;
import java.util.Map;
public class MapDemo {
	public static void main(String[] args) {
		Map<Integer,String> map = new HashMap<Integer,String>();
		methodDemo(map);
	}
	 //Map中的泛型有两个参数 k，v。     Map<k,v>
	public static void methodDemo(Map<Integer,String> map) {
		//存储键值对,如果键相同，会出现值覆盖
		System.out.println(map.put(3,"rex")); //put方法返回的v是与key关联的旧值，第一次为空
		System.out.println(map.put(3,"debao"));//相同的键，第二次返回的就是与该键相关联的上一次的值，即rex
		map.put(5,"frde");
		map.put(8,"rexall");

		//System.out.println(map.remove(5)); remove操作会改变容器长度，即获取的同时也会删除元素
		System.out.println(map.get(5)); //get方法只获取元素，不改变容器的长度
		System.out.println(map);
	}
}
运行结果：
null
rex
frde
{3=debao, 5=frde, 8=rexall}
```

如何取出Map集合中的所有元素

方法1：map集合没有迭代器，但是可以将map集合转成set集合，再使用迭代器，就可以迭代出所有的键对应的值

```
package cn.itcast.map;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;
public class MapDemo2 {
	public static void main(String[] args) {

		//取出map中所有的元素
		Map<String,String> map = new HashMap<String,String>();
		map.put("qiang", "beijing");
		map.put("hong", "sichuan");
		map.put("ling", "tianjing");
		map.put("wei", "nanjing");

		Set<String> keySet = map.keySet(); //keySet方法取出所有的键，并存储到set集合中
		for (Iterator<String> it = keySet.iterator(); it.hasNext();) { //用迭代器迭代出所有的键对应的值
			String key = it.next();
			String value = map.get(key);
```

```
                System.out.println(key+":"+value);
            }
        }
}
运行结果:
hong:sichuan
ling:tianjing
wei:nanjing
qiang:beijing
```

方法2：这里通过将map集合转换为set集合，再通过**entrySet**方法，entrySet()方法取出的是“键值关系”，该泛型为Map.Entry<E,E>，得到“键值关系”Map.Entry<E,E>之后，再调用它的getKey和getValue方法即可得到键和值

```
package cn.itcast.map;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;
public class MapDemo2 {
        public static void main(String[] args) {

                //取出map中所有的元素
                Map<String,String> map = new HashMap<String,String>();
                map.put("qiang", "beijing");
                map.put("hong", "sichuan");
                map.put("ling", "tianjing");
                map.put("wei", "nanjing");

                //Map.Entry其实就是一个Map接口中的内部接口

                Set<Map.Entry<String, String>> entrySet = map.entrySet();

                for (Iterator<Map.Entry<String, String>> it = entrySet.iterator(); it.hasNext();) {
                        Map.Entry<String, String> me = it.next();
                        String key = me.getKey();
                        String value = me.getValue();
                        System.out.println(key+":::"+value);
                }
        }
}
运行结果:
hong:::sichuan
ling:::tianjing
wei:::nanjing
qiang:::beijing
```

方法3：可以通过values方法来取出所有的“值”，但是不能得到相应的“键”。

```
package cn.itcast.map;
import java.util.Collection;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
public class MapDemo2 {
        public static void main(String[] args) {

                Map<String,String> map = new HashMap<String,String>();
                map.put("qiang", "beijing");
                map.put("hong", "sichuan");
```

```
            map.put("ling", "tianjing");
            map.put("wei", "nanjing");

            //通过values方法得到所有的值  Collection<V> values()
            Collection<String> values = map.values();
            for (Iterator<String> it = values.iterator(); it.hasNext();) {
                  String value = it.next();
                  System.out.println(value);
            }
      }
}
```

运行结果:

```
sichuan
tianjing
nanjing
beijing
```

Map的一些实现类

Hashtable：哈希表，是同步的，不允许null作为键，不允许mull作为值
HashMap：哈希表，是不同步的，允许null作为键，null作为值
TreeMap：二叉树，是不同步的，可以对map集合中的键进行排序

练习:
学生对象（姓名、年龄）都有对应的归属地　key = Student　value = String
1、将学生和归属存储到**HashMap**集合中并取出，同姓名同年龄视为同一个学生

```
package cn.itcast.domain;
public class Employee {
      private String name;
      private int age;
      public Employee(String name, int age) {
            super();
            this.name = name;
            this.age = age;
      }
      public Employee() {
            super();
      }
      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;
      }
      public String toString()
      {
            return "Employee [name = " + name + ", 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;
		Employee other = (Employee) 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;
	}
}
```

```
package cn.itcast.map;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;
import cn.itcast.domain.Employee;
public class MapDemo3 {
	public static void main(String[] args) {
		Map<Employee,String> map = new HashMap<Employee,String>();

		//put方法添加元素
		map.put(new Employee("zhang",24),"北京");
		map.put(new Employee("wang",28),"济南");
		map.put(new Employee("li",34),"苏州");
		map.put(new Employee("li",34),"广州"); //这个与上面的元素是同姓名同年龄，所以在Employee中要覆盖hashCode和equals方法
		map.put(new Employee("yang",19),"南京");

		Set<Employee> keySet = map.keySet(); //map集合转换为set集合

		for(Employee employee : keySet) { //增强型for循环进行元素的遍历
			String value = map.get(employee);
			System.out.println(employee.getName()+" "+employee.getAge()+" "+value);
		}
	}
}
运行结果:
zhang 24 北京
yang 19 南京
li 34 广州  //这个将put(new Employee("li",34),"苏州")覆盖掉了
wang 28 济南
```

2、将学生和归属存储到**TreeMap**中，按照学生的年龄进行升序排序并取出

```
package cn.itcast.domain;
public class Employee implements Comparable<Employee>{ //实现Comparable方法
```

```
	private String name;
	private int age;
	public Employee(String name, int age) {
		super();
		this.name = name;
		this.age = age;
	}
	public Employee() {
		super();
	}
	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;
	}
	public String toString()
	{
		return "Employee [name = " + name + ", 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;
		Employee other = (Employee) 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;
	}
	@Override
	public int compareTo(Employee o) { //覆盖compareTo方法
		int temp = this.age-o.age;
		return temp==0?this.name.compareTo(o.name):temp;
	}
}
```

```
package cn.itcast.map;
```

```
import java.util.Map;
import java.util.Set;
import java.util.TreeMap;
import cn.itcast.domain.Employee;
public class MapDemo3 {
	public static void main(String[] args) {
		Map<Employee,String> map = new TreeMap<Employee,String>(); //存储到TreeMap中，因此需要进行排序

		//put方法添加元素
		map.put(new Employee("zhang",24),"北京");
		map.put(new Employee("wang",28),"济南");
		map.put(new Employee("li",34),"苏州");
		map.put(new Employee("li",34),"广州"); // 这个与上面的元素是同姓名同年龄，所以在Employee中要覆盖hashCode和equals方法
		map.put(new Employee("yang",19),"南京");

		Set<Map.Entry<Employee,String>> entrySet = map.entrySet(); //map集合转换为entrySet集合

		for(Map.Entry<Employee,String> me : entrySet) { //增强型for循环进行元素的遍历
			Employee key = me.getKey();
			String value = me.getValue();
			System.out.println(key.getName()+" "+key.getAge()+" "+value);
		}
	}
}
```

运行结果：//按照年龄进行排序

```
yang 19 南京
zhang 24 北京
wang 28 济南
li 34 广州
```

什么时候使用Map集合?

当需求中出现映射（对应）关系时，应该最先想到map集合

练习：根据数字编号获得中文星期，再根据中文星期来获得英文星期

```
package cn.itcast.map;
public class NoWeekException extends RuntimeException {
	public NoWeekException() {
		super();
	}
	public NoWeekException(String arg0, Throwable arg1, boolean arg2, boolean arg3) {
		super(arg0, arg1, arg2, arg3);
	}
	public NoWeekException(String arg0, Throwable arg1) {
		super(arg0, arg1);
	}
	public NoWeekException(String arg0) {
		super(arg0);
	}
	public NoWeekException(Throwable arg0) {
		super(arg0);
	}
}
```

```
package cn.itcast.map;
import java.util.HashMap;
import java.util.Map;
```

```
public class MapDemo4 {
	public static void main(String[] args) {
		String cnWeek = getCnWeek(3); //根据数字编号获得中文的星期
		System.out.println(cnWeek);

		String enWeek = getEnWeek(cnWeek); //根据中文的星期获得英文的星期
		System.out.println(enWeek);
	}

	//根据中文星期，获得对应的英文星期。中文与英文相对应，可以建立表，没有有序的编号，只能通过map集合
	public static String getEnWeek(String cnWeek) {

		//创建一个表
		Map<String,String> map = new HashMap<String,String>();
		map.put("星期一", "Monday");
		map.put("星期二", "Tuesday");
		map.put("星期三", "Wednesday");
		map.put("星期四", "Thursday");
		map.put("星期五", "Friday");
		map.put("星期六", "Saturday");
		map.put("星期日", "Sunday");

		return map.get(cnWeek);
	}

	public static String getCnWeek(int num) {
		if(num<=0 || num>7) {
			throw new NoWeekException(num+"没有对应的星期"); //抛出自定义的异常
		}
		String[] cnWeeks = {"","星期一","星期二","星期三","星期四","星期五","星期六","星期日",};
		return cnWeeks[num];
	}
}
运行结果:
星期三
Wednesday
```

集合框架的工具类（Java已经做好了的）

1、Collections：定义的都是操作Collections的静态方法。对list排序（sort）：
public static <T> void sort(List<T> list, Comparator<? super T> c) //根据指定比较器产生的顺序对指定列表进行排序。

```
package cn.itcast.comparator;
import java.util.Comparator;

//指定比较器，按照长度进行排序
public class ComparatorByLength implements Comparator<String> {
	public int compare(String o1,String o2) {
		int temp = o1.length()-o2.length();
		return temp==0?o1.compareTo(o2):temp;
	}
}
```

```
package cn.itcast.collections;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
```

```
import cn.itcast.comparator.ComparatorByLength;
public class CollectionsDemo {
	public static void main(String[] args) {
		methodDemo1();
	}

	private static void methodDemo1() {
		List<String> list = new ArrayList<String>();
		list.add("cd");
		list.add("jdd");
		list.add("rterd");
		list.add("u");
		list.add("hghjh");

		System.out.println(list);

		//对list排序，自然排序，使用的是元素的compareTo方法
		Collections.sort(list);
		System.out.println(list);

		//按照长度排序
		//reverseOrder(Comparator<T> cmp); 返回一个比较器，它强行逆转指定比较器的顺序。
		Collections.sort(list, Collections.reverseOrder(new ComparatorByLength()));
		System.out.println(list);
	}
}
```

运行结果：

```
[cd, jdd, rterd, u, hghjh]
[cd, hghjh, jdd, rterd, u]
[rterd, hghjh, jdd, cd, u]
```

模拟获取集合中最大值的功能

```
package cn.itcast.collections;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;
import cn.itcast.comparator.ComparatorByLength;
public class CollectionsDemo {
	public static void main(String[] args) {

		Collection<String> coll = new ArrayList<String>();
		coll.add("abcd");
		coll.add("erd");
		coll.add("yujfd");
		coll.add("zs");
		coll.add("lofdj");
		String max = getMax(coll);
		System.out.println(max);
	}

	public static String getMax(Collection<String> coll) {

		//通过迭代器来遍历容器中的元素
		Iterator<String> it = coll.iterator();

		//定义变量，记录容器中的一个元素
		String max = it.next();
```

```
            //遍历容器中的所有元素
            while(it.hasNext()) {
                    String temp = it.next();
                    if(temp.compareTo(max)>0) {
                            max = temp;
                    }
            }
            //返回最大值
            return max;
    }
}
运行结果:
max = zs
```

Collections中有一个可以将非同步集合转换成同步集合的方法:
同步集合 synchronized集合 (非同步集合);
例如: <T> Collection<T> synchronizedCollection(Collection<T> c)

2、Arrays，用来**操作数组**的工具类，方法都是静态的

```
package cn.itcast.collections;
import java.util.Arrays;
public class Test {
    public static void main(String[] args) {
            int[] arr = {12,25,35,46,78,99};
            System.out.println(arr);

            //static String toString(double[] a), 返回指定数组内容的字符串表示形式。
            System.out.println(Arrays.toString(arr));
    }
}
运行结果:
[I@161cd475
[12, 25, 35, 46, 78, 99]
```

数组转换成集合（用类Arrays中的asList方法）
想要判断数组中是否包含某一个元素，之前的做法是对数组进行遍历并进行比较。现在发现集合中就有判断是否包含的这个方法（contains）。因此，我们只需要将数组转换成集合就行了，然后就可以使用集合中的某些方法对数组进行操作（但是**不能使用“增删”**方法），因为**数组的长度不能被改变**。

```
package cn.itcast.collections;
import java.util.Arrays;
import java.util.List;
public class Test {
    public static void main(String[] args) {

            String[] strs = {"abs","ed","grf","debao"};
            List<String> list = Arrays.asList(strs); //1、利用Arrays中的asList方法，将数组转换成集
合
            System.out.println(list.contains("debao")); //2、应用集合中的contains方法来判断“是否包
含”
```

```
            int[] arr = {12,25,36,46};//如果数组中都是基本数据类型，转成集合时，会将整个数组（数组对象）作为集合中的元素
            List<int[]> list2 = Arrays.asList(arr);
            System.out.println(list2);//上面的arr中都是int型数据，故arr在转换时，被当成一个数组对象，所以输出的结果是 [[I@161cd475]


            Integer[] arr2 = {12,25,36,46};//如果数组中都是引用数据类型，转成集合时，数组元素直接作为集合元素
            List list3 = Arrays.asList(arr2);
            System.out.println(list3.get(0)); //arr2中都是引用数据类型，其中的每一个元素都是Integer型的对象，所以在被转换成集合时，被一个一个地转换成了集合中的对象。因此get(0)得到的就是第一个对象，即 12。

      }
}
运行结果:
true
[[I@161cd475]
12
```

集合转换成数组（用Collection接口中的toArray方法）

为什么要把集合转成数组呢？就是为了限定对元素的操作（例如，转成数组之后就不能够进行“增删”了）

```
package cn.itcast.collections;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
public class CollectionToArray {
      public static void main(String[] args) {

            List<String> list = new ArrayList<String>();
            list.add("abc");
            list.add("def");
            list.add("bgh");

            //<T> T[] toArray(T[] a) ，将集合转成数组
            //这里的[]中为数组的长度，如果传入的数组的长度小于集合的长度，方法中会创建一个新的长度和集合长度一致的数组；如果传入的数组的长度大于等于集合的长度，就会使用传入的数组。所以建议长度定义为集合的size();

            //String[] arr = list.toArray(new String[0]);//这里数组长度为0，集合长度为3，所以会新建一个数组来替代原有的数组

            String[] arr = list.toArray(new String[list.size()]);//长度定义为集合的size();
            System.out.println(Arrays.toString(arr));
      }
}
运行结果:
[abc, def, bgh]
```

可变参数 ...（自动创建数组，书写方便）

```
package cn.itcast.collections;
public class ParamDemo {
      public static void main(String[] args) {

            /*int[] arr = {1,2,5,7};
```

```
		int sum = add(arr);
		System.out.println(sum);*/

		int sum = add(1,2,5,7); //本行的作用与 int[] arr = {1,2,5,7};  int sum = add(arr); 这两句话的作用一样，即在使用了可变参数之后，编译器会自动的创建数组，将数组元素装进数组
		System.out.println(sum);
	}
	private static int add(int... arr) { //这里的...就是可变参数，表示数组中有很多int型元素。可变参数需要注意，它只能定义在参数列表的最后，假如参数列表中有多个参数，如下：
	  //只能这样定义：add(int x,int... arr) 不能这样定义：add(int... arr,int x)

		int sum = 0;
		for (int i = 0; i < arr.length; i++) {
			sum = sum+arr[i];
		}
		return sum;
	}

	/*private static int add(int[] arr) {
		int sum = 0;
		for (int i = 0; i < arr.length; i++) {
			sum = sum+arr[i];
		}
		return sum;
	}*/
}
运行结果：
15
```

静态导入

当我们在写这样的代码时：List list = new ArrayList(); 会发现编译器报错（因为显示红色波浪线），这种情况实际上是我们的类名“写错了”，我们必须要写全名（带上java.util）才能不报错，如下：

```
java.util.List list = new java.util.ArrayList();
```

然而实际上我们在编程时，是通过快捷键Ctrl+Shift+O来导入java.util包的。

```
package cn.itcast.collections;
import java.util.ArrayList;
//import java.util.Collections; //这样是导入Collections这个类
import static java.util.Collections.*; //这样是导入Collections这个类中的内容，这就是静态导入（加了关键字static)
import java.util.List;
public class StaticImportDemo {
	public static void main(String[] args) {
			List<String> list = new ArrayList<String>();
			list.add("debao");
			list.add("rex");
			//Collections.sort(list);
			//Collections.max(list);

			//这样就可以将上面的写法进行简化，不用每次都带上Collections
			//sort(list);
			System.out.println(max(list));
	}
}
运行结果：
rex
```

System类

System：全是静态的属性和行为
属性：
out：标准输出流，默认对应的设备是显示器
in：标准输入流，默认对应的设备是键盘

```
package cn.itcast.system;
import java.util.Properties;
public class SystemDemo {

	private static final String FILE_SEPARATOR = System.getProperty("file.separator"); //键file.separator对应的值为：文件分隔符
	private static final String LINE_SEPARATOE = System.getProperty("line.separator"); //键line.separator对应的值为：换行符

	public static void main(String[] args) {
		/*long time = System.currentTimeMillis(); //currentTimeMillis返回的是当前时间与协调世界时 1970 年 1 月 1 日午夜之间的时间差（以毫秒为单位测量）。
		System.out.println(time);*/

		//public static Properties getProperties() 确定当前的系统属性。该方法返回系统属性的集合，该集合使用“键值对”来存储(Map)
		Properties prop = System.getProperties();
		System.out.println(prop.get("os.name")); //通过键os.name，得到它的值，即该程序运行在什么操作系统上


		/*如何取出属性集，取出过程就是读取Map集合元素的过程
		Set<String> keySet = prop.stringPropertyNames();
		for(String key : keySet) {
			String value = prop.getProperty(key);
			System.out.println(key+":"+value);
		}*/

		 System.out.println("C:"+FILE_SEPARATOR+"abc"+FILE_SEPARATOR+"1.txt"); //文件分隔操作

		//System.out.println("hello\nworld");
		System.out.println("hello"+LINE_SEPARATOE+"world"); //文件换行
	}
}
```

运行结果：

```
Windows 10
C:\abc\1.txt
hello
world
```

Runtime类（该类就是**单例设计模式**）

```
package cn.itcast.otherapi;
import java.io.IOException;
public class RuntimeDemo {
	public static void main(String[] args) throws IOException {

		Runtime r = Runtime.getRuntime();
```

```
            //public Process exec(String command) throws IOException , 在单独的进程中执行指定的字符串命令。

            //r.exec("notepad.exe"); //打开记事本
            r.exec("E:\\CAJ\\CAJVieweru.exe F:\\基于小型四旋翼的多无人机编队飞行控制系统设计_沈俊楠.caj"); //使用CAJVieweru.exe来打开F盘中的文件“基于小型四旋翼的多无人机编队飞行控制系统设计_沈俊楠”
        }
}
```

Math类

```
package cn.itcast.otherapi;
import java.util.Random;
public class MathDemo {
        public static void main(String[] args) {

            /*//Math：数学类，都是静态成员
            double d1 = Math.ceil(12.34); //ceil:大于参数的最小整数
            double d2 = Math.floor(12.34); //floor：小于参数的最大整数
            double d3 = Math.round(12.34); //round：四舍五入
            System.out.println(d1);
            System.out.println(d2);
            System.out.println(d3);

            System.out.println(Math.pow(10, 3)); //pow(a,b)：返回a的b次幂 */

            /*for(int x = 0;x < 10;x++) {

                //double d = Math.random();  //返回一个大于等于 0.0 且小于 1.0的伪随机数
                int d = (int)Math.ceil(Math.random()*6); //返回1到6
                System.out.println(d);
            }*/


            //在上面的Math类中调用了random方法来生成随机数，而实际上Random被封装成了对象，在java.util包中可以查到，利用Random类中的方法可以更加方便地生成各种随机数
            Random r = new Random();
            for(int x = 0;x < 5; x++) {
                int d = r.nextInt(6)+1;  //int nextInt(int n)：随机生成一个0到n(不包括n)之间的整数
                System.out.println("random:"+d);
            }
        }
}
运行结果：
random:5
random:6
random:2
random:6
random:3
```

Date的日期格式化

```
package cn.itcast.otherapi;
import java.text.SimpleDateFormat;
import java.util.Date;
public class DateDemo {
        public static void main(String[] args) {
```

```
            //Date对象
            Date date = new Date();
            //System.out.println(date); //默认格式:  Wed Oct 31 09:00:05 CST 2018

            //想要把日期按照我们的习惯格式化一下，找到了DateFormat

            //通过DateFormat类中的静态工厂方法获取实例,并可以使用不同的风格(如: FULL、LONG、MEDIUM等)
            //DateFormat dateFormat = DateFormat.getDateInstance(DateFormat.FULL); //2018年10月
31日星期三
            //DateFormat dateFormat = DateFormat.getDateTimeInstance(DateFormat.FULL,
DateFormat.FULL); //2018年10月31日星期三 中国标准时间 上午9:22:44

            //通过SimpleDateFormat类可以设定为自定义的日期格式
            SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd hh:mm:ss");
//2018/10/31 09:42:19，SimpleDateFormat中的参数含义（例如y、M、D、d等）可以查询API得到


            //使用DateFormat的format方法，对日期对象进行格式化,将日期对象转成日期格式的字符串
            String str_date = dateFormat.format(date);
            System.out.println(str_date);
        }
}
```

字符串格式的日期的解析

```
package cn.itcast.otherapi;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
public class CalendarTest {
        public static void main(String[] args) throws ParseException {

                String str_date = "2017-07-17"; //定义一个字符串格式的日期

          //DateFormat dateFormat = DateFormat.getDateInstance(); 这个得到的是默认风格

                DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");//上面"2017-07-17"不一定
是“默认风格”的类型，因此要先自定义类型将其进行解析
          Date date = dateFormat.parse(str_date); //返回日期对象

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

运行结果:

```
Mon Jul 17 00:00:00 CST 2017
```

日期对象和毫秒值之间相互转换

```
package cn.itcast.otherapi;
import java.text.DateFormat;
import java.util.Date;
public class DateDemo3 {
        public static void main(String[] args) {
```

```
            //毫秒值—>日期对象
            long time = System.currentTimeMillis(); //生成一个毫秒数值（即“协调时间”）

            //public Date(long date)，根据指定的毫秒值来指定日期对象
            Date date = new Date(time); //将毫秒值转换为日期对象

            DateFormat dateFormat = DateFormat.getDateInstance(DateFormat.LONG); //将日期对象格式
化
            String str_date = dateFormat.format(date); //将格式化后的值转换成字符串
            System.out.println(str_date);  //2018年10月31日



            //日期对象—>毫秒值，用Date对象的getTime方法，日期对象转换成毫秒值的作用在于可以对时间进
行“加减”操作
            long time2 = date.getTime();
            System.out.println(time2); //1540952584714
        }
}
```

日历（Calendar）对象

```
package cn.itcast.otherapi;
import java.util.Calendar;
public class CalendarDemo {
        public static void main(String[] args) {

            Calendar c = Calendar.getInstance(); //Calendar(日历)对象是一个Map集合，它将日历中包含的
所有信息封装成了一个集合，然后通过“键”来获取对应的值
            //System.out.println(c);

            int year = c.get(Calendar.YEAR);
            int month = c.get(Calendar.MONTH)+1; //因为计算机中的月为0到11月，所以要加1，将之变为1到
12月
            int day = c.get(Calendar.DAY_OF_MONTH); //注意：计算机中，星期日是第一天，星期六才是最后
一天
            String week = getWeek(c.get(Calendar.DAY_OF_WEEK)); //为了显示结果较好，可以通过查数组得
到对应的星期值

            System.out.println(year+"年"+month+"月"+day+"日"+week);
        }
        private static String getWeek(int i) {

            String[] weeks = {"","星期日","星期一","星期二","星期三","星期四","星期五","星期六"};
            return weeks[i];
        }
}
```

运行结果：
2018年10月31日星期三

练习：
求： "2012/3/17"，"2012-5-28" 之间间隔多少天？**思路**：求间隔时间，必然要**相减**，只有**毫秒值**之间才能相减，所以要进行**转换，字符串—>日期对象—>毫秒值**

```
package cn.itcast.otherapi;
import java.text.DateFormat;
import java.text.ParseException;
```

```java
import java.text.SimpleDateFormat;
import java.util.Date;
public class Test {
	public static void main(String[] args) throws ParseException {
		// 1、只有毫秒可以相减。
		// 2、获取毫秒值，字符串—>日期对象—>毫秒值
		getDays();
	}
	private static void getDays() throws ParseException {
		String str_date1 = "2012/3/17";
		String str_date2 = "2012-5-28";

		//如何将日期格式字符串解析成日期对象呢？，通过DateFormat  parse
		//然而上面两种格式不一定是“默认风格”的类型，因此要先自定义类型将其进行解析
		DateFormat dateFormat1 = new SimpleDateFormat("yyyy/MM/dd");
		DateFormat dateFormat2 = new SimpleDateFormat("yyyy-MM-dd");

		Date date1 = dateFormat1.parse(str_date1);
		Date date2 = dateFormat2.parse(str_date2);

		//通过日期对象获取毫秒值
		long time1 = date1.getTime();
		long time2 = date2.getTime();

		//相减
		long time = Math.abs(time1-time2);

		//毫秒数转换为天数
		int day = transDay(time);
		System.out.println(day);
	}
	private static int transDay(long time) {
		return (int)(time/1000/60/60/24);
	}
}
```

运行结果：

```
72
```