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

2、具体内容

集合输出实际上从JDK1.8开始就在Iterable接口之中提供有一个foreach()方法，但是这种方法的输出并不是传统意义上的集合输出形式，并且也很难在实际的开发之中出现，对于集合操作而言，一共定义有四种输出形式：iterator迭代输出（95%）、listiterator双向迭代输出（0.1%）、enumeration枚举输出（4.8%）、foreach输出（与iterator相当）。

■Iterator输出

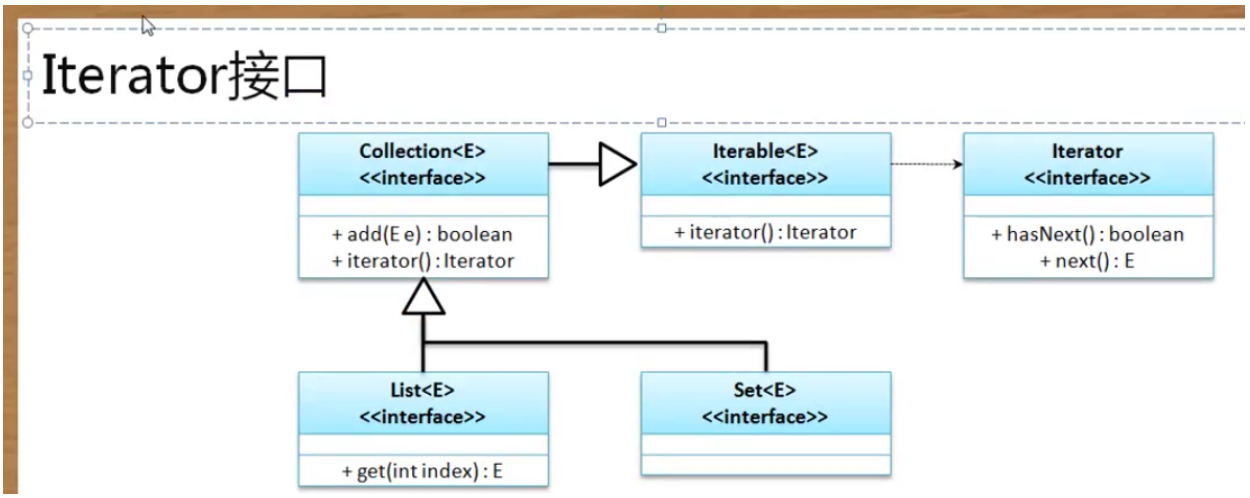
通过Collector接口的继承关系可以发现，从JDK1.5开始其多继承了一个Iterable父接口，并且在这个接口里面定义有一个iterator()操作方法，通过此方法可以获取Iterator接口对象（在JDK1.5之前，这一方法直接定义在Collection接口之中）。

·获取iterator接口对象：public Iterator<T> iterator();

在iterator接口里面定义有如下的方法：

No	方法名称	类型	描述
01	public boolean hasNext()	普通	判断是否有数据
02	public E next()	普通	取出当前数据
03	public default void remove()	普通	删除

在之前使用的java.util.Scanner类就是Iterator接口的子类，所以此时类继承关系如下：



范例：使用Iterator输出

```
package cn.mldn.demo;
import java.util.Iterator;
import java.util.Set;
public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		Set<String> all = Set.of("Hello", "World", "MLDN");
		Iterator<String> iter = all.iterator() ; // 实例化Iterator接口对象
		while (iter.hasNext()) {
			String str = iter.next() ;
			System.out.println(str);
		}
	}
}
```

但是对于iterator接口中的remove()方法的使用需要特别注意以下（如果不是必须不要使用）。实际上在Collection接口里面定义有数据的删除操作方法，但是在进行迭代输出的过程里面如果使用了Collection中的remove()方法会导致迭代失败。

范例：采用Collection集合中的remove()方法删除

```
package cn.mldn.demo;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;
public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		Set<String> all = new HashSet<String>();
		all.add("Hello") ;
		all.add("World") ;
		all.add("MLDN") ;
		Iterator<String> iter = all.iterator() ; // 实例化Iterator接口对象
		while (iter.hasNext()) {
			String str = iter.next() ;
			if ("World".equals(str)) {
				all.remove("World") ; // Collection集合方法
			} else {
				System.out.println(str);
			}
```

```
            }
        }
}
```

程序运行结果	Hello Exception in thread "main" java.util.ConcurrentModificationException

此时无法进行数据的删除处理操作，那么此时就只能够利用Iterator接口中remove()方法删除。

范例：使用Iterator接口删除方法

```
package cn.mldn.demo;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;
public class JavaAPIDemo {
    public static void main(String[] args) throws Exception {
        Set<String> all = new HashSet<String>();
        all.add("Hello") ;
        all.add("World") ;
        all.add("MLDN") ;
        Iterator<String> iter = all.iterator() ; // 实例化Iterator接口对象
        while (iter.hasNext()) {
            String str = iter.next() ;
            if ("World".equals(str)) {
                iter.remove() ; // 删除当前的数据
            } else {
                System.out.println(str);
            }
        }
        System.out.println("*** " + all);
    }
}
```

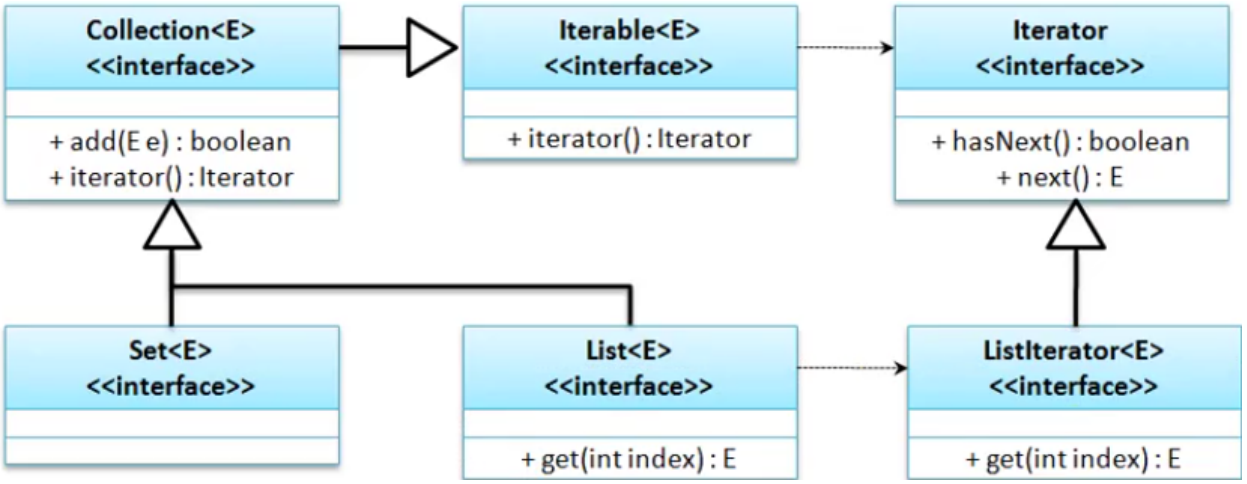
此时程序执行之后没有出现任何的错误，并且可以成功的删除原始集合中的数据。

面试题：请解释Collection.remove()与Iterator.remove()的区别?

·在进行迭代输出的时候如果使用了Collection.remove()则会造成并发更新异常，导致程序删除出错，而此时只能够利用Iterator.remove()方法实现正常的删除处理。

■ListIterator输出

使用Iterator进行的迭代输出操作有一个特点：只允许由前向后实现输出，而如果说现在需要进行双向迭代处理，那么就必须依靠iterator的子接口：ListIterator接口来实现了。需要注意的是如果要想获取ListIterator接口对象Collection并没有定义相关的处理方法，但是list子接口有，也就是说这个输出接口是专门为list集合准备的。



在ListIterator接口里面定义有如下的操作方法：

·判断是否有前一个元素：public boolean hasPrevious();

·获取当前元素：public E previous();

范例：实现双向迭代

```
package cn.mldn.demo;
import java.util.ArrayList;
import java.util.List;
import java.util.ListIterator;
public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		List<String> all = new ArrayList<String>();
		all.add("Hello") ;
		all.add("World") ;
		all.add("MLDN") ;
		ListIterator<String> iter = all.listIterator() ;
		System.out.print("由前向后输出：");
		while(iter.hasNext()) {
			System.out.print(iter.next() + "、");
		}
		System.out.print("\n由后向前输出：");
		while (iter.hasPrevious()) {
			System.out.print(iter.previous() + "、");
		}
	}
}
```

如果要想实现由后向前的编历，那么首先要实现的是由前向后实现编历处理。

■Enumeration输出

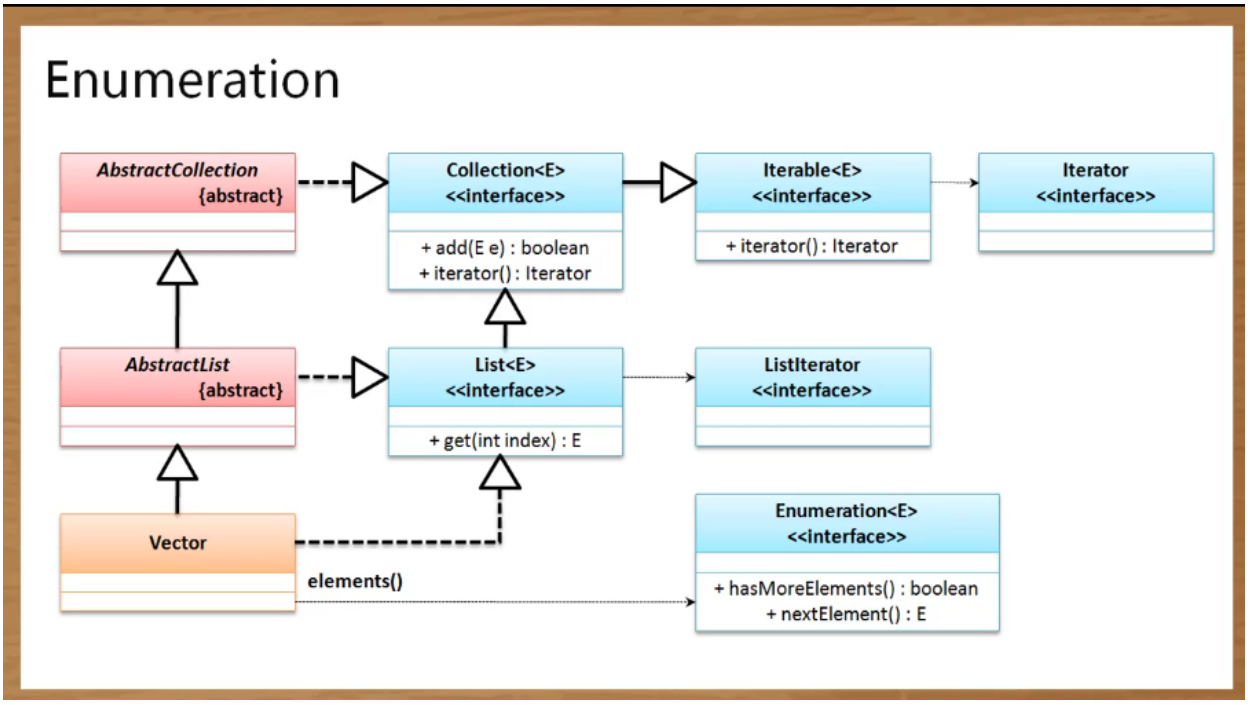
Enumeration的时候就使用的输出接口，这个输出接口主要是为了Vector类提供输出服务的，一直到后续的JDK的发展，Enumeration依然只为Vector一个类服务，如果要想获取Enumeration接口对象，就必须依靠Vector来提供的方法：

·获取Enumeration：public Enumeration<E> elements();

在Enumeration接口之中定义有两个操作方法：

·判断是否有下一个元素：public boolean hasMoreElements();

·获取当前元素：public E nextElement();



范例：使用Enumeration实现输出

```
package cn.mldn.demo;
import java.util.Enumeration;
import java.util.Vector;
public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		Vector<String> all = new Vector<String>();
		all.add("Hello") ;
		all.add("World") ;
		all.add("MLDN") ;
		Enumeration<String> enu = all.elements() ;
		while (enu.hasMoreElements()) {
			String str = enu.nextElement() ;
			System.out.print(str + "、");
		}
	}
}
```

由于该接口出现的时间比较长了，所以在一些比较早的开发过程之中，也有部分的方法只支持Enumeration输出操作，但是随着类方法的不断完善，大部分的操作都直接利用Iterator实现了。

■foreach输出

除了使用迭代接口实现输出之外，从JDK1.5开始加强型for循环也可以实现集合的输出了。这种输出的形式与数组的输出操作形式类似。

范例：使用foreach输出

```
package cn.mldn.demo;
import java.util.ArrayList;
import java.util.List;
public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		List<String> all = new ArrayList<String>();
		all.add("Hello");
		all.add("World");
		all.add("MLDN");
		for (String str : all) {
			System.out.print(str + "、");
		}
	}
}
```

这种输出最初出现的时候很多人并不建议使用，因为标准的集合操作还是应该以Iterator为主，但是毕竟JDK1.5都已经推出十多年了，很多的语法也开始被大部分人所习惯。