专业语言:理解接口 PDF转换可能丢失图片或格式，建议阅读原文

https://www.100test.com/kao_ti2020/267/2021_2022__E4_B8_93_E4_B8_9A_E8_AF_AD_E8_c104_267892.htm 好的面向对象编程要求类设计人员隐藏那些不需要类的使用人员了解的信息。对于 Java 编程语言，这样的访问可以通过使用关键字 private, protected, 和 public来控制。这些关键字控制类内部的变量和方法是否可见，但是不好的类设计导致太多的可见信息和方法没有被很好的封装。封装的一种方式是通过使用接口(Interface)实现的。接口提供一种途径，使类隐藏其处理的特定事物的细节，仅对外公布它必须支持的属性。对于编程所涉及的，你可以修改类的实现，而不修改它的调用，因为属性本身没有改变，修改的仅仅是类的实现。一个接口被经常用得到的地方是Collection Framework。这个框架定义了一组非常重要的接口，它们是由一组多个类实现的。通过仅仅研究主要的接口，你可以有效的掌握整个框架，因为特别的实现类一般不影响设计的整体。例如，List接口定义了一个有序的元素集合。可用地实现包括ArrayList和LinkedList，它们都实现了List接口。当你的程序需要处理List时，不必考虑它是ArrayList还是LinkedList，只要知道所选用的类的属性即可。这个属性就是接口。通过实现类的接口，并且在类设计时仅对外公布接口，你就有效的封装了类的定义，这样后台实现的变动将对系统其它部分的影响最小。以ArrayList和LinkedList为例。将ArrayList看作一个可增长的数组对象（指是存储对象，而不是原始数据）。当类实现了List的全部接口时，它的特性在特定条件下是可以优化的。例如，如果你

的程序是要队列表中的数据进行频繁的随机访问，（例如，显示第3，12，2，和25项数据）ArrayList类提供对列表中每一个数据快速查询。快速查询是以在列表中间添加和删除数据的速度为代价的。如果后一种行为是你需要的，那么LinkedList类是一个好的选择。它提供快速的顺序访问、添加和删除，但是，代价是慢速随机查询。在处理ArrayList和LinkedList时，有两种方式创建对象：List cityList = new ArrayList() .LinkedList peopleList = new LinkedList() .两个代码段都是正确的，但是这两行代码之间存在的显著的差别。第一行表示我们要创建一个ArrayList，但是我们只需要把它作为一个List来访问。第二行正好相反。是，LinkedList项目被创建了，ArrayList也一样。但是，声明部分说明它只能作为LinkedList来访问，这就数它的最大区别。理解接口真正变的重要是在这两行代码的用户确定“ 查询项目n ”比在位置m处删除（或添加）项目更为重要时。PeopleList变量被声明为LinkedList类型。这不是它本身的问题，因为你研究的更深层次的内容，你将发现peopleList在任何地方都被作为LinkedList对象处理。在你对peopleList使用LinkedList特有的方法的同时，如果你想把它作为ArrayList来处理，将会出现问题。List peopleList = new ArrayList() .通过学习仅使用接口来处理任何对象，你将发现在设计完成之后修改实现，你需要修改的仅仅是声明部分，除此之外，没有任何代码改动。这就是接口的绝妙之处。因为类的最初声明是LinkedList，当类型变为List时意味着象addFirst或addLast这样的方法是无效的，因为新的peopleList的类型是List，它没有这些方法。这种基于接口设计的代码，就像Collection Framework所向大家承诺的那

样，任何人编写的代码是以循环构造方式进行的，而无需知道使用的是哪个Collection。创建的类是被限制为提供接口的完全实现。除此之外，新代码将不能被编译。作为实例，下面的程序创建一组集合。每个集合提供一个系统定义的Iterator这样集合的每个元素可以被访问。这个iterator将被传递到帮助例程，在这里集合的独立元素将被打印。import java.util.*. public class Interfaces { public static void main(String args[]) { Properties props = System.getProperties(). Set keySet = props.keySet(). dumpIterator(keySet.iterator()). List list = Arrays.asList(args). dumpIterator(list.iterator()). } private static void dumpIterator(Iterator itor) { // System.out.println(itor.getClass().getName()). while (itor.hasNext()) { System.out.println(">> " itor.next()). } System.out.println("----"). }}类Iterator的类型是unknown，这正是接口的绝妙之处，而不是问题。真正的事实是iterator方法返回的是一个真实的Iterator对象。然而，dumpIterator通常提供接口的完全实现。如果你去掉dumpIterator中的println行的注释，你将发现真实的iterator类名，对Properties是Hashtable.Enumerator而List是AbstractList.Itr。这个事实不必知道，也不会对你的程序有任何帮助。真正重要的是List和Properties的iterator方法所返回的任何对象，必须实现java.util.Iterator:hasNext, next和remove方法。没有这三种方法中任何两种，dumpIterator方法将永远不能工作。 