<http://dev.yesky.com/436/7581936.shtml>

在[**Java**](http://search.yesky.com/search.do?wd=java&appType=1&columnId=324&sortType=0&btnsearch=%CB%D1)语言中， abstract class 和[**interface**](http://search.yesky.com/search.do?wd=%BD%D3%BF%DA&appType=1&columnId=324&sortType=0&btnsearch=%CB%D1)是支持[**抽象类**](http://search.yesky.com/search.do?wd=%B3%E9%CF%F3%C0%E0&appType=1&columnId=324&sortType=0&btnsearch=%CB%D1)定义的两种机制。正是由于这两种机制的存在，才赋予了Java强大的 面向对象能力。abstract class和interface之间在对于抽象类定义的支持方面具有很大的相似性，甚至可以相互替换，因此很多开发者在进 行抽象类定义时对于abstract class和interface的选择显得比较随意。其实，两者之间还是有很大的区别的，对于它们的选择甚至反映出对 于问题领域本质的理解、对于设计意图的理解是否正确、合理。本文将对它们之间的区别进行一番剖析，试图给开发者提供一个在二者之间进行选择的依据。   
  
　　**理解抽象类**  
  
　　abstract class和interface在Java语言中都是用来进行抽象类（本文 中的抽象类并非从abstract class翻译而来，它表示的是一个抽象体，而abstract class为Java语言中用于定义抽象类的一种方法， 请读者注意区分）定义的，那么什么是抽象类，使用抽象类能为我们带来什么好处呢？  
  
　　在 面向对象的概念中，我们知道所有的对象都是通过类来描绘的，但是反过来却不是这样。并不是 所有的类都是用来描绘对象的，如果一个类中没有包含足够的信息来描绘一个具体的对象，这样的类就是抽象类。抽象类往往用来表征我们在对问题领域进行分析、 设计中得出的抽象概念，是对一系列看上去不同，但是本质上相同的具体概念的抽象。比如：如果我们进行一个图形编辑软件的开发，就会发现问题领域存在着圆、 三角形这样一些具体概念，它们是不同的，但是它们又都属于形状这样一个概念，形状这个概念在问题领域是不存在的，它就是一个抽象概念。正是因为抽象的概念 在问题领域没有对应的具体概念，所以用以表征抽象概念的抽象类是不能够实例化的。  
  
　　在面向对象领域，抽象类主要用来进行类型隐藏。 我们可以构造出一个固定的一组行为的抽象描 述，但是这组行为却能够有任意个可能的具体实现方式。这个抽象描述就是抽象类，而这一组任意个可能的具体实现则表现为所有可能的派生类。模块可以操作一个 抽象体。由于模块依赖于一个固定的抽象体，因此它可以是不允许修改的；同时，通过从这个抽象体派生，也可扩展此模块的行为功能。熟悉OCP的读者一定知 道，为了能够实现面向对象设计的一个最核心的原则OCP(Open-Closed Principle)，抽象类是其中的关键所在。  
  
　　**从语法定义层面看abstract class 和 interface**  
  
　　在语法层面，Java语言对于abstract class和interface给出了不同的定义方式，下面以定义一个名为Demo的抽象类为例来说明这种不同。  
  
　　使用abstract class的方式定义Demo抽象类的方式如下：

|  |
| --- |
| abstract class Demo｛ abstract void method1(); abstract void method2(); … ｝ |

　　使用interface的方式定义Demo抽象类的方式如下：

|  |
| --- |
| interface Demo{ void method1(); void method2(); … } |

　　在abstract class方式中，Demo可以有自己的数据成员，也可以有非 abstract的成员方法，而在interface方式的实现中，Demo只能够有静态的不能被修改的数据成员（也就是必须是static final 的，不过在interface中一般不定义数据成员），所有的成员方法都是abstract的。从某种意义上说，interface是一种特殊形式的 abstract class。  
  
　　从编程的角度来看，abstract class和interface都可以用来实现 "design by contract" 的思想。但是在具体的使用上面还是有一些区别的。  
  
　　首先，abstract class 在 Java 语言中表示的是一种继承关系，一个类只能使用一次继承关系(因为Java不支持多继承 -- 转注)。但是，一个类却可以实现多个interface。也许，这是Java语言的设计者在考虑Java对于多重继承的支持方面的一种折中考虑吧。  
  
　　其次，在abstract class的定义中，我们可以赋予方法的默认行为。但是在interface的定义中，方法却不能拥有默认行为，为了绕过这个限制，必须使用委托，但是这会增加一些复杂性，有时会造成很大的麻烦。  
  
　　在 抽象类中不能定义默认行为还存在另一个比较严重的问题，那就是可能会造成维护上的麻烦。因 为如果后来想修改类的界面（一般通过 abstract class 或者interface来表示）以适应新的情况（比如，添加新的方法或者给已用的方法中添 加新的参数）时，就会非常的麻烦，可能要花费很多的时间（对于派生类很多的情况，尤为如此）。但是如果界面是通过abstract class来实现的，那 么可能就只需要修改定义在abstract class中的默认行为就可以了。  
  
　　同样，如果不能在抽象类中定义默认行为，就会导致同样的方法实现出现在该抽象类的每一个派生类中，违反了 "one rule，one place" 原则，造成代码重复，同样不利于以后的维护。因此，在abstract class和interface间进行选择时要非常的小心。  
  
　　**从设计理念层面看 abstract class 和 interface**  
  
　　上面主要从语法定义和编程的角度论述了abstract class和interface的区 别，这些层面的区别是比较低层次的、非本质的。本小节将从另一个层面：abstract class和interface所反映出的设计理念，来分析一下二者的区别。作者认为，从这个层面进行分析才能理解二者概念的本质所在。  
  
　　前面已经提到过，abstract class在Java语言中体现了一种继承关系，要想使得 继承关系合理，父类和派生类之间必须存在"[is](http://mydown.yesky.com/soft/network/chat/475/444475.shtml)-a"关系，即父类和派生类在概念本质上应该是相同的。对于interface来说则不然，并不要求interface的实现者和interface定义在概念本质上是一致的， 仅仅是实现了interface定义的契约而已。为了使论述便于理解，下面将通过一个简单的实例进行说明。  
  
　　考虑这样一个例子，假设在我们的问题领域中有一个关于Door的抽象概念，该Door具有执行两个动作open和close，此时我们可以通过abstract class或者interface来定义一个表示该抽象概念的类型，定义方式分别如下所示：  
  
　　使用abstract class方式定义Door：

|  |
| --- |
| abstract class Door{ abstract void open(); abstract void close()； } |

　　使用interface方式定义Door：

|  |
| --- |
| interface Door{ void open(); void close(); } |

　　其他具体的Door类型可以extends使用abstract class方式定义的Door或者implements使用interface方式定义的Door。看起来好像使用abstract class和interface没有大的区别。  
  
　　如果现在要求Door还要具有报警的功能。我们该如何设计针对该例子的类结构呢（在本例中， 主要是为了展示 abstract class 和interface 反映在设计理念上的区别，其他方面无关的问题都做了简化或者忽略）？下面将罗列出可能的解 决方案，并从设计理念层面对这些不同的方案进行分析。  
  
　　解决方案一：  
  
　　简单的在Door的定义中增加一个alarm方法，如下：

|  |
| --- |
| abstract class Door{ abstract void open(); abstract void close()； abstract void alarm(); } |

　　或者

|  |
| --- |
| interface Door{ void open(); void close(); void alarm(); } |

　　那么具有报警功能的AlarmDoor的定义方式如下：

|  |
| --- |
| class AlarmDoor extends Door{ void open(){…} void close(){…} void alarm(){…} } |

　　或者

|  |
| --- |
| class AlarmDoor implements Door｛ void open(){…} void close(){…} void alarm(){…} ｝ |

　　这种方法违反了面向对象设计中的一个核心原则 ISP (Interface Segregation Principle)，在Door的定义中把Door概念本身固有的行为方法和另外一个概念"报警器"的行为方 法混在了一起。这样引起的一个问题是那些仅仅依赖于Door这个概念的模块会因为"报警器"这个概念的改变（比如：修改alarm方法的参数）而改变，反 之依然。  
  
　　解决方案二：  
  
　　既然open、close和alarm属于两个不同的概念，根据ISP原则应该把它们分别定 义在代表这两个概念的抽象类中。定义方式有：这两个概念都使用 abstract class 方式定义；两个概念都使用interface方式定义；一个概念 使用 abstract class 方式定义，另一个概念使用interface方式定义。  
  
　　显然，由于Java语言不支持多重继承，所以两个概念都使用abstract class方式定义是不可行的。后面两种方式都是可行的，但是对于它们的选择却反映出对于问题领域中的概念本质的理解、对于设计意图的反映是否正确、合理。我们一一来分析、说明。  
  
　　如果两个概念都使用interface方式来定义，那么就反映出两个问题：1、我们可能没有 理解清楚问题领域，AlarmDoor在概念本质上到底是Door还是报警器？2、如果我们对于问题领域的理解没有问题，比如：我们通过对于问题领域的分 析发现AlarmDoor在概念本质上和Door是一致的，那么我们在实现时就没有能够正确的揭示我们的设计意图，因为在这两个概念的定义上（均使用 interface方式定义）反映不出上述含义。  
  
　　如果我们对于问题领域的理解是：AlarmDoor在概念本质上是Door，同时它有具有报 警的功能。我们该如何来设计、实现来明确的反映出我们的意思呢？前面已经说过，abstract class在Java语言中表示一种继承关系，而继承关系 在本质上是"is-a"关系。所以对于Door这个概念，我们应该使用abstarct class方式来定义。另外，AlarmDoor又具有报警功能，说 明它又能够完成报警概念中定义的行为，所以报警概念可以通过interface方式定义。如下所示：

|  |
| --- |
| abstract class Door{ abstract void open(); abstract void close()； } interface Alarm{ void alarm(); } class Alarm Door extends Door implements Alarm{ void open(){…} void close(){…} void alarm(){…} } |

　　这种实现方式基本上能够明确的反映出我们对于问题领域的理解，正确的揭示我们的设计意图。其 实abstract class表示的是"is-a"关系，interface表示的是"like-a"关系，大家在选择时可以作为一个依据，当然这是建立在对问题领域的理解上的，比如：如果我们认为AlarmDoor在概念本质上是报警器，同时又具有Door的功能，那么上述的定义方式就要反过来了。  
  
　　**小结**  
  
　　1.abstract class 在 Java 语言中表示的是一种继承关系，一个类只能使用一次继承关系。但是，一个类却可以实现多个interface。  
  
　　2.在abstract class 中可以有自己的数据成员，也可以有非abstarct的成员方法，而在interface中，只能够有静态的不能被修改的数据成员（也就是必须是static final的，不过在 interface中一般不定义数据成员），所有的成员方法都是abstract的。  
  
　　3.abstract class和interface所反映出的设计理念不同。其实abstract class表示的是"is-a"关系，interface表示的是"like-a"关系。   
  
　　4.实现抽象类和接口的类必须实现其中的所有方法。抽象类中可以有非抽象方法。接口中则不能有实现方法。  
  
　　5.接口中定义的变量默认是public static final 型，且必须给其初值，所以实现类中不能重新定义，也不能改变其值。  
  
　　6.抽象类中的变量默认是 friendly 型，其值可以在子类中重新定义，也可以重新赋值。   
  
　　7.接口中的方法默认都是 public,abstract 类型的。  
  
　　**结论**  
  
　　abstract class 和 interface 是 Java语言中的两种定义抽象类的方式，它们之间有很大的相似性。但是对于它们的选择却又往往反映出对于问题领域中的概 念本质的理解、对于设计意图的反映是否正确、合理，因为它们表现了概念间的不同的关系（虽然都能够实现需求的功能）。这其实也是语言的一种的惯用法，希望读者朋友能够细细体会。