

为什么Java中继承多数是有害的-java基础 PDF转换可能丢失图片或格式，建议阅读原文

[https://www.100test.com/kao\\_ti2020/287/2021\\_2022\\_\\_E4\\_B8\\_BA\\_E4\\_BB\\_80\\_E4\\_B9\\_88J\\_c104\\_287882.htm](https://www.100test.com/kao_ti2020/287/2021_2022__E4_B8_BA_E4_BB_80_E4_B9_88J_c104_287882.htm) 多数好的设计者象躲避瘟疫一样来避免使用实现继承(extends 关系)。实际上80%的代码应该完全用interfaces写，而不是通过extends。“Java设计模式”一书详细阐述了怎样用接口继承代替实现继承。这篇文章描述设计者为什么会这么作。Extends是有害的；也许对于Charles Manson这个级别的不是，但是足够糟糕的它应该在任何可能的时候被避开。“JAVA设计模式”一书花了很大的部分讨论用interface继承代替实现继承。好的设计者在他的代码中，大部分用interface，而不是具体的基类。本文讨论为什么设计者会这样选择，并且也介绍一些基于interface的编程基础。接口（Interface）和类（Class）？一次，我参加一个Java用户组的会议。在会议中，Jams Gosling（Java之父）做发起人讲话。在那令人难忘的Q&A部分中，有人问他：“如果你重新构造Java，你想改变什么？”。“我想抛弃classes”他回答。在笑声平息后，它解释说，真正的问题不是由于class本身，而是实现继承(extends) 关系。接口继承（implements关系）是更好的。你应该尽可能的避免实现继承。失去了灵活性 为什么你应该避免实现继承呢？第一个问题是明确的使用具体类名将你固定到特定的实现，给底层的改变增加了不必要的困难。在当前的敏捷编程方法中，核心是并行的设计和开发的概念。在你详细设计程序前，你开始编程。这个技术不同于传统方法的形式----传统的方式是设计应该在编码开始前完成----但是许多成功的项目已经证明你能够

更快速的开发高质量代码，相对于传统的按部就班的方法。但是在并行开发的核心是主张灵活性。你不得以某一种方式写你的代码以至于最新发现的需求能够尽可能没有痛苦的合并到已有的代码中。胜于实现你也许需要的特征，你只需实现你明确需要的特征，而且适度的对变化的包容。如果你没有这种灵活，并行的开发，那简直不可能。对于Interface的编程是灵活结构的核心。为了说明为什么，让我们看一下当使用它们的时候，会发生什么。考虑下面的代码：

```
f() {  
    LinkedList list = new LinkedList(). //... g( list ). }  
g( LinkedList list ) {  
    list.add( ... ). g2( list ) }  
}
```

假设一个对于快速查询的需求被提出，以至于这个LinkedList不能够解决。你需要用HashSet来代替它。在已有代码中，变化不能够局部化，因为你不仅仅需要修改f()也需要修改g()（它带有LinkedList参数），并且还有g()把列表传递给的任何代码。象下面这样重写代码：

```
f() {  
    Collection list = new LinkedList(). //... g( list ). }  
g( Collection list ) {  
    list.add( ... ). g2( list ) }  
}
```

这样修改LinkedList成hash，可能只是简单的用new HashSet()代替new LinkedList()。就这样。没有其他的需要修改的地方。作为另一个例子，比较下面两段代码：

```
f() {  
    Collection c = new HashSet(). //... g( c ). }  
g( Collection c ) {  
    for( Iterator i = c.iterator(). i.hasNext() )  
        do_something_with( i.next() ). }  
}
```

```
f2() {  
    Collection c = new HashSet(). //... g2( c.iterator() ). }  
g2( Iterator i ) {  
    while( i.hasNext() )  
        do_something_with( i.next() ). }  
}
```

g2()方法现在能够遍历Collection的派生，就像你能够从Map中得到的键值对。事实上，你能够写iterator，它产生数据，代替遍历一个Collection。你能够写iterator，它从测试的框架或者文件中得到信息。这会有巨大的灵活性。 100Test 下载频道

开通，各类考试题目直接下载。详细请访问 [www.100test.com](http://www.100test.com)