

## 08 | 理论五：接口vs抽象类的区别？如何用普通的类模拟抽象类和接口？

王争 · 设计模式之美



在面向对象编程中，抽象类和接口是两个经常被用到的语法概念，是面向对象四大特性，以及很多设计模式、设计思想、设计原则编程实现的基础。比如，我们可以使用接口来实现面向对象的抽象特性、多态特性和基于接口而非实现的设计原则，使用抽象类来实现面向对象的继承特性和模板设计模式等等。

不过，并不是所有的面向对象编程语言都支持这两个语法概念，比如，C++ 这种编程语言只支持抽象类，不支持接口；而像 Python 这样的动态编程语言，既不支持抽象类，也不支持接口。尽管有些编程语言没有提供现成的语法来支持接口和抽象类，我们仍然可以通过一些手段来模拟实现这两个语法概念。

这两个语法概念不仅在工作中经常会被用到，在面试中也经常被提及。比如，“接口和抽象类的区别是什么？什么时候用接口？什么时候用抽象类？抽象类和接口存在的意义是什么？能解决哪些编程问题？”等等。


你可以先试着回答一下，刚刚我提出的几个问题。如果你对某些问题还有些模糊不清，那也没关系，今天，我会带你把这几个问题彻底搞清楚。下面我们就一起来看！

## 什么是抽象类和接口？区别在哪里？

不同的编程语言对接口和抽象类的定义方式可能有些差别，但差别并不会很大。Java 这种编程语言，既支持抽象类，也支持接口，所以，为了让你对这两个语法概念有比较直观的认识，我们拿 Java 这种编程语言来举例讲解。

**首先，我们来看一下，在 Java 这种编程语言中，我们是如何定义抽象类的。**

下面这段代码是一个比较典型的抽象类的使用场景（模板设计模式）。Logger 是一个记录日志的抽象类，FileLogger 和 MessageQueueLogger 继承 Logger，分别实现两种不同的日志记录方式：记录日志到文件中和记录日志到消息队列中。FileLogger 和 MessageQueueLogger 两个子类复用了父类 Logger 中的 name、enabled、minPermittedLevel 属性和 log() 方法，但因为这两个子类写日志的方式不同，它们又各自重写了父类中的 doLog() 方法。

 复制代码

```
1 // 抽象类
2 public abstract class Logger {
3     private String name;
4     private boolean enabled;
5     private Level minPermittedLevel;
6
7     public Logger(String name, boolean enabled, Level minPermittedLevel) {
8         this.name = name;
9         this.enabled = enabled;
10        this.minPermittedLevel = minPermittedLevel;
11    }
12
13    public void log(Level level, String message) {
14        boolean loggable = enabled && (minPermittedLevel.intValue() <= level.intValue()
15        if (!loggable) return;
16        doLog(level, message);
17    }
18
19    protected abstract void doLog(Level level, String message);
20 }
21 // 抽象类的子类：输出日志到文件
```

```

22 public class FileLogger extends Logger {
23     private Writer fileWriter;
24
25     public FileLogger(String name, boolean enabled,
26         Level minPermittedLevel, String filepath) {
27         super(name, enabled, minPermittedLevel);
28         this.fileWriter = new FileWriter(filepath);
29     }
30
31     @Override
32     public void doLog(Level level, String message) {
33         // 格式化level和message,输出到日志文件
34         fileWriter.write(...);
35     }
36 }
37 // 抽象类的子类: 输出日志到消息中间件(比如kafka)
38 public class MessageQueueLogger extends Logger {
39     private MessageQueueClient msgQueueClient;
40
41     public MessageQueueLogger(String name, boolean enabled,
42         Level minPermittedLevel, MessageQueueClient msgQueueClient) {
43         super(name, enabled, minPermittedLevel);
44         this.msgQueueClient = msgQueueClient;
45     }
46
47     @Override
48     protected void doLog(Level level, String message) {
49         // 格式化level和message,输出到消息中间件
50         msgQueueClient.send(...);
51     }
52 }

```

通过上面的这个例子，我们来看一下，抽象类具有哪些特性。我总结了下面三点。

抽象类不允许被实例化，只能被继承。也就是说，你不能 new 一个抽象类的对象出来（`Logger logger = new Logger(...);` 会报编译错误）。

抽象类可以包含属性和方法。方法既可以包含代码实现（比如 `Logger` 中的 `log()` 方法），也可以不包含代码实现（比如 `Logger` 中的 `doLog()` 方法）。不包含代码实现的方法叫作抽象方法。

子类继承抽象类，必须实现抽象类中的所有抽象方法。对应到例子代码中就是，所有继承 `Logger` 抽象类的子类，都必须重写 `doLog()` 方法。

刚刚我们讲了如何定义抽象类，现在我们再来看一下，在 Java 这种编程语言中，我们如何定义接口。

 复制代码

```
1 // 接口
2 public interface Filter {
3     void doFilter(RpcRequest req) throws RpcException;
4 }
5 // 接口实现类：鉴权过滤器
6 public class AuthencationFilter implements Filter {
7     @Override
8     public void doFilter(RpcRequest req) throws RpcException {
9         //...鉴权逻辑..
10    }
11 }
12 // 接口实现类：限流过滤器
13 public class RateLimitFilter implements Filter {
14     @Override
15     public void doFilter(RpcRequest req) throws RpcException {
16         //...限流逻辑...
17    }
18 }
19 // 过滤器使用Demo
20 public class Application {
21     // filters.add(new AuthencationFilter());
22     // filters.add(new RateLimitFilter());
23     private List<Filter> filters = new ArrayList<>();
24
25     public void handleRpcRequest(RpcRequest req) {
26         try {
27             for (Filter filter : filters) {
28                 filter.doFilter(req);
29             }
30         } catch (RpcException e) {
31             // ...处理过滤结果...
32         }
33         // ...省略其他处理逻辑...
34     }
35 }
```

上面这段代码是一个比较典型的接口的使用场景。我们通过 Java 中的 interface 关键字定义了一个 Filter 接口。AuthencationFilter 和 RateLimitFilter 是接口的两个实现类，分别实现了对 RPC 请求鉴权和限流的过滤功能。

代码非常简洁。结合代码，我们再来看一下，接口都有哪些特性。我也总结了三点。

接口不能包含属性（也就是成员变量）。

接口只能声明方法，方法不能包含代码实现。

类实现接口的时候，必须实现接口中声明的所有方法。

前面我们讲了抽象类和接口的定义，以及各自的语法特性。从语法特性上对比，这两者有比较大的区别，比如抽象类中可以定义属性、方法的实现，而接口中不能定义属性，方法也不能包含代码实现等等。除了语法特性，从设计的角度，两者也有比较大的区别。

抽象类实际上就是类，只不过是一种特殊的类，这种类不能被实例化为对象，只能被子类继承。我们知道，继承关系是一种 is-a 的关系，那抽象类既然属于类，也表示一种 is-a 的关系。相对于抽象类的 is-a 关系来说，接口表示一种 has-a 关系，表示具有某些功能。对于接口，有一个更加形象的叫法，那就是协议（contract）。

## 抽象类和接口能解决什么编程问题？

刚刚我们学习了抽象类和接口的定义和区别，现在我们再来学习一下，抽象类和接口存在的意义，让你知其然知其所以然。

### 首先，我们来看一下，我们为什么需要抽象类？它能够解决什么编程问题？

刚刚我们讲到，抽象类不能实例化，只能被继承。而前面的章节中，我们还讲到，继承能解决代码复用的问题。所以，抽象类也是为代码复用而生的。多个子类可以继承抽象类中定义的属性和方法，避免在子类中，重复编写相同的代码。

不过，既然继承本身就能达到代码复用的目的，而继承也并不要求父类一定是抽象类，那我们不使用抽象类，照样也可以实现继承和复用。从这个角度上来讲，我们貌似并不需要抽象类这种语法呀。那抽象类除了解决代码复用的问题，还有什么其他存在的意义吗？

我们还是拿之前那个打印日志的例子来讲解。我们先对上面的代码做下改造。在改造之后的代码中，Logger 不再是抽象类，只是一个普通的父类，删除了 Logger 中 log()、doLog() 方


法，新增了 isLoggable() 方法。FileLogger 和 MessageQueueLogger 还是继承 Logger 父类，以达到代码复用的目的。具体的代码如下：

 复制代码

```
1 // 父类：非抽象类，就是普通的类。删除了log(),doLog(), 新增了isLoggable().
2 public class Logger {
3     private String name;
4     private boolean enabled;
5     private Level minPermittedLevel;
6
7     public Logger(String name, boolean enabled, Level minPermittedLevel) {
8         //...构造函数不变，代码省略...
9     }
10
11     protected boolean isLoggable() {
12         boolean loggable = enabled && (minPermittedLevel.intValue() <= level.intValue
13         return loggable;
14     }
15 }
16 // 子类：输出日志到文件
17 public class FileLogger extends Logger {
18     private Writer fileWriter;
19
20     public FileLogger(String name, boolean enabled,
21         Level minPermittedLevel, String filepath) {
22         //...构造函数不变，代码省略...
23     }
24
25     public void log(Level level, String message) {
26         if (!isLoggable()) return;
27         // 格式化level和message,输出到日志文件
28         fileWriter.write(...);
29     }
30 }
31 // 子类：输出日志到消息中间件(比如kafka)
32 public class MessageQueueLogger extends Logger {
33     private MessageQueueClient msgQueueClient;
34
35     public MessageQueueLogger(String name, boolean enabled,
36         Level minPermittedLevel, MessageQueueClient msgQueueClient) {
37         //...构造函数不变，代码省略...
38     }
39
40     public void log(Level level, String message) {
41         if (!isLoggable()) return;
42         // 格式化level和message,输出到消息中间件
43         msgQueueClient.send(...);
```


```
44     }
45 }
```

这个设计思路虽然达到了代码复用的目的，但是无法使用多态特性了。像下面这样编写代码，就会出现编译错误，因为 `Logger` 中并没有定义 `log()` 方法。

 复制代码

```
1 Logger logger = new FileLogger("access-log", true, Level.WARN, "/users/wangzheng/");
2 logger.log(Level.ERROR, "This is a test log message.");
```

你可能会说，这个问题解决起来很简单啊。我们在 `Logger` 父类中，定义一个空的 `log()` 方法，让子类重写父类的 `log()` 方法，实现自己的记录日志的逻辑，不就可以了么？

 复制代码

```
1 public class Logger {
2     // ...省略部分代码...
3     public void log(Level level, String message) { // do nothing... }
4 }
5 public class FileLogger extends Logger {
6     // ...省略部分代码...
7     @Override
8     public void log(Level level, String message) {
9         if (!isLoggable()) return;
10        // 格式化level和message,输出到日志文件
11        fileWriter.write(...);
12    }
13 }
14 public class MessageQueueLogger extends Logger {
15     // ...省略部分代码...
16     @Override
17     public void log(Level level, String message) {
18         if (!isLoggable()) return;
19        // 格式化level和message,输出到消息中间件
20        msgQueueClient.send(...);
21    }
22 }
```

这个设计思路能用，但是，它显然没有之前通过抽象类的实现思路优雅。我为什么这么说呢？主要有以下几点原因。

在 `Logger` 中定义一个空的方法，会影响代码的可读性。如果我们不熟悉 `Logger` 背后的设计思想，代码注释又不怎么给力，我们在阅读 `Logger` 代码的时候，就可能对为什么定义一个空的 `log()` 方法而感到疑惑，需要查看 `Logger`、`FileLogger`、`MessageQueueLogger` 之间的继承关系，才能弄明白其设计意图。

当创建一个新的子类继承 `Logger` 父类的时候，我们有可能会忘记重新实现 `log()` 方法。之前基于抽象类的设计思路，编译器会强制要求子类重写 `log()` 方法，否则会报编译错误。你可能会说，我既然要定义一个新的 `Logger` 子类，怎么会忘记重新实现 `log()` 方法呢？我们举的例子比较简单，`Logger` 中的方法不多，代码行数也很少。但是，如果 `Logger` 有几百行，有  $n$  多方法，除非你对 `Logger` 的设计非常熟悉，否则忘记重新实现 `log()` 方法，也不是不可能的。

`Logger` 可以被实例化，换句话说，我们可以 `new` 一个 `Logger` 出来，并且调用空的 `log()` 方法。这也增加了类被误用的风险。当然，这个问题可以通过设置私有的构造函数的方式来解决。不过，显然没有通过抽象类来的优雅。

**其次，我们再来看一下，我们为什么需要接口？它能够解决什么编程问题？**

抽象类更多的是为了代码复用，而接口就更侧重于解耦。接口是对行为的一种抽象，相当于一组协议或者契约，你可以联想类比一下 API 接口。调用者只需要关注抽象的接口，不需要了解具体的实现，具体的实现代码对调用者透明。接口实现了约定和实现相分离，可以降低代码间的耦合性，提高代码的可扩展性。

实际上，接口是一个比抽象类应用更加广泛、更加重要的知识点。比如，我们经常提到的“基于接口而非实现编程”，就是一条几乎天天会用到，并且能极大地提高代码的灵活性、扩展性的设计思想。关于接口这个知识点，我会单独再用一节课的时间，更加详细全面的讲解，这里就不展开了。

**如何模拟抽象类和接口两个语法概念？**



在前面举的例子中，我们使用 Java 的接口语法实现了一个 Filter 过滤器。不过，如果你熟悉的是 C++ 这种编程语言，你可能会说，C++ 只有抽象类，并没有接口，那从代码实现的角度上来说，是不是就无法实现 Filter 的设计思路了呢？

实际上，我们可以通过抽象类来模拟接口。怎么来模拟呢？这是一个不错的面试题，你可以先思考一下，然后再来看我的讲解。


我们先来回忆一下接口的定义：接口中没有成员变量，只有方法声明，没有方法实现，实现接口的类必须实现接口中的所有方法。只要满足这样几点，从设计的角度上来说，我们就可以把它叫作接口。实际上，要满足接口的这些语法特性并不难。在下面这段 C++ 代码中，我们就用抽象类模拟了一个接口（下面这段代码实际上是策略模式中的一段代码）。

 复制代码

```
1 class Strategy { // 用抽象类模拟接口
2     public:
3         ~Strategy();
4         virtual void algorithm()=0;
5     protected:
6         Strategy();
7 };
```

抽象类 Strategy 没有定义任何属性，并且所有的方法都声明为 virtual 类型（等同于 Java 中的 abstract 关键字），这样，所有的方法都不能有代码实现，并且所有继承这个抽象类的子类，都要实现这些方法。从语法特性上来看，这个抽象类就相当于一个接口。

不过，如果你熟悉的既不是 Java，也不是 C++，而是现在比较流行的动态编程语言，比如 Python、Ruby 等，你可能还会有疑问：在这些动态语言中，不仅没有接口的概念，也没有类似 abstract、virtual 这样的关键字来定义抽象类，那该如何实现上面的讲到的 Filter、Logger 的设计思路呢？实际上，除了用抽象类来模拟接口之外，我们还可以用普通类来模拟接口。具体的 Java 代码实现如下所示。

 复制代码

```
1 public class MockInterface {
2     protected MockInterface() {}
```

```
3     public void funcA() {  
4         throw new MethodUnsupportedException();  
5     }  
6 }
```

我们知道类中的方法必须包含实现，这个不符合接口的定义。但是，我们可以让类中的方法抛出 `MethodUnsupportedException` 异常，来模拟不包含实现的接口，并且能强迫子类在继承这个父类的时候，都去主动实现父类的方法，否则就会在运行时抛出异常。我们将构造函数设置成 `protected` 属性的，这样就能避免非同包下的类去实例化 `MockInterface`。不过，这样还是无法避免同包中的类去实例化 `MockInterface`。为了解决这个问题，我们可以学习 Google Guava 中 `@VisibleForTesting` 注解的做法，自定义一个注解，人为表明不可实例化。

刚刚我们讲了如何用抽象类来模拟接口，以及如何用普通类来模拟接口，那如何用普通类来模拟抽象类呢？这个问题留给你自己思考，你可以留言说说你的实现方法。

实际上，对于动态编程语言来说，还有一种对接口支持的策略，那就是 `duck-typing`。我们在上一节课中讲到多态的时候也有讲过，你可以再回忆一下。

## 如何决定该用抽象类还是接口？

刚刚的讲解可能有些偏理论，现在，我们就从真实项目开发的角度来看一下，在代码设计、编程开发的时候，什么时候该用抽象类？什么时候该用接口？

实际上，判断的标准很简单。如果我们要表示一种 `is-a` 的关系，并且是为了解决代码复用的问题，我们就用抽象类；如果我们要表示一种 `has-a` 关系，并且是为了解决抽象而非代码复用的问题，那我们就可以使用接口。

从类的继承层次上来看，抽象类是一种自下而上的设计思路，先有子类的代码重复，然后再抽象成上层的父类（也就是抽象类）。而接口正好相反，它是一种自上而下的设计思路。我们在编程的时候，一般都是先设计接口，再去考虑具体的实现。

## 重点回顾

好了，今天内容就讲完了，我们一块来总结回顾一下，你需要掌握的重点内容。

## 1. 抽象类和接口的语法特性

抽象类不允许被实例化，只能被继承。它可以包含属性和方法。方法既可以包含代码实现，也可以不包含代码实现。不包含代码实现的方法叫作抽象方法。子类继承抽象类，必须实现抽象类中的所有抽象方法。接口不能包含属性，只能声明方法，方法不能包含代码实现。类实现接口的时候，必须实现接口中声明的所有方法。

## 2. 抽象类和接口存在的意义

抽象类是对成员变量和方法的抽象，是一种 is-a 关系，是为了解决代码复用问题。接口仅仅是对方法的抽象，是一种 has-a 关系，表示具有某一组行为特性，是为了解决解耦问题，隔离接口和具体的实现，提高代码的扩展性。

## 3. 抽象类和接口的应用场景区别

什么时候该用抽象类？什么时候该用接口？实际上，判断的标准很简单。如果要表示一种 is-a 的关系，并且是为了解决代码复用问题，我们就用抽象类；如果要表示一种 has-a 关系，并且是为了解决抽象而非代码复用问题，那我们就用接口。

## 课堂讨论

1. 你熟悉的编程语言，是否有现成的语法支持接口和抽象类呢？具体是如何定义的呢？
2. 前面我们提到，接口和抽象类是两个经常在面试中被问到的概念。学习完今天的内容之后，你是否对抽象类和接口有一个新的认识呢？如果面试官再让你聊聊接口和抽象类，你会如何回答呢？

欢迎在留言区写下你的答案，和同学一起交流和分享。如果有收获，也欢迎你把这篇文章分享给你的朋友。

## AI智能总结

抽象类和接口是面向对象编程中常用的两个语法概念。抽象类是一种特殊的类，不能被实例化，只能被继承，可以包含属性和方法的实现，子类必须实现抽象方法。而接口则不能包含属性，方法不能包含代码实现，类实现接口时必须实现接口中声明的所有方法。从设计角度来看，抽象类表示 is-a 关系，而接口表示 has-a 关系，也被称为协议。抽象类主要解决代码复用问题，而接口更侧重于解耦，提高代码的可扩展性。

文章通过Java语言的例子详细介绍了抽象类和接口的定义和特性，以及它们在面向对象编程中的应用场景和意义。此外，文章还探讨了如何模拟抽象类和接口的语法概念，包括用抽象类模拟接口、用普通类模拟接口以及动态编程语言中的duck-typing策略。

总的来说，本文深入浅出地介绍了抽象类和接口的区别及使用方法，适合读者快速了解这两个重要的面向对象编程概念。文章内容简洁明了，通过实际项目开发的角度解释了抽象类和接口的选择标准，帮助读者更好地理解并应用这些概念。同时，通过课堂讨论和重点回顾，读者可以加深对抽象类和接口的理解，并在面试等场景中更加自信地表达自己的观点。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

## 全部留言 (285)

最新 精选



helloworld

2019-11-21

『那又如何避免这个类被实例化呢？实际上很简单，我们只需要将这个类的构造函数声明为 protected 访问权限就可以了。』当把一个类的构造方法声明为protected后，在同一个包的其他类中还是可以new这个类的对象的。老师，是我想错了吗？请老师指正

作者回复: 好像我写的是有点问题，稍后更正下，多谢指出

共 17 条评论 >

👍 38



我爱布丁

2019-11-21

老师举的Filter接口的例子可以用抽象类实现吗？

我觉得可以说 AuthencationFilter is a Filter, RateLimitFilter is a Filter. 如果两个Filters在d oFilter里会预先做一些通用的对于RpcRequest的validation工作，或者有一些shared state s，那么这样是不是就应该用抽象类实现呢？

作者回复: 可以使用抽象类。但也可以使用组合来实现代码复用。后面会讲到



👍 5



ye feng

2020-07-29

老师老师，我想补充一下，什么时候使用抽象类这个问题。文章中说如果要is-a关系就用抽象类，感觉这个答案有点不标准，is-a关系也可以用普通类来实现代码复用问题。更标准的答案是不是如果既有代码复用问题，子类又一定要实现多态关系的时候才用抽象类。这个答案是不是准确，请老师指点一下。

作者回复: 嗯嗯 你的表述更准确些

共 3 条评论 >

👍 4



初学者

2020-11-24

抽象类做于代码复用，接口做于代码扩展

作者回复: 没错



👍 3



胡子高兴了

2019-11-20

老师，接口中可以定义静态成员变量吗？

作者回复: 可以的



👍 3



小乙哥

2019-11-25

接口举例的地方，添加实例Filter的地方被注释掉了

作者回复: 那是故意那么写的😂，因为那两行代码放到那里并不合适



👍 2



阿玛铭

2019-11-21

问问题外话：老师对响应式编程有研究吗？想要一门这样的课程。理由：1.josh long打造，可以代表开发技术的趋势。 2.了解的不够，高并发，类似nety 3.语言隔阂短时间无法突破。还有一个重要的理由是我觉得老师你讲内容的思路容易理解和接受。

作者回复: 😊 没研究过响应式编程 帮不上你了

共 2 条评论 >



2



DY

2020-06-10

既然接口能实现的功能抽象类都能实现，为什么还要用接口？

作者回复: 在某些场景下，接口更好用，更能表达业务含义。这就类似，吃馒头能吃饱，为啥还要吃肉一个道理，更香！

共 2 条评论 >



1



longslee

2019-12-04

咦，老师，“接口不能包含属性（也就是成员变量）”，是怎么回事？可以的呀

作者回复: 只能包含静态常量吧



1



刘永超

2020-11-14

从语法特性区分抽象类和接口是比较实在的，对于语法特性，不能死记，但是理解记忆是必须的。也许对于高手来讲，语法太浅显了，但对于刚入门的人来说，语法是重要的基础。

作者回复: 嗯嗯 是的



