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？

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。

```
// 抽象类
public abstract class Logger {
  private String name;
  private boolean enabled;
  private Level minPermittedLevel;

  public Logger(String name, boolean enabled, Level minPermittedLevel) {
    this.name = name;
    this.enabled = enabled;
    this.minPermittedLevel = minPermittedLevel;
  }

  public void log(Level level, String message) {
    boolean loggable = enabled && (minPermittedLevel.intValue() <= level.intValue());
    if (!loggable) return;
    doLog(level, message);
  }

  protected abstract void doLog(Level level, String message);
}
// 抽象类的子类：输出日志到文件
```

```
public class FileLogger extends Logger {
  private Writer fileWriter;

  public FileLogger(String name, boolean enabled,
    Level minPermittedLevel, String filepath) {
    super(name, enabled, minPermittedLevel);
    this.fileWriter = new FileWriter(filepath);
  }

  @Override
  public void doLog(Level level, String mesage) {
    // 格式化level和message,输出到日志文件
    fileWriter.write(...);
  }
}
// 抽象类的子类：输出日志到消息中间件(比如kafka)
public class MessageQueueLogger extends Logger {
  private MessageQueueClient msgQueueClient;

  public MessageQueueLogger(String name, boolean enabled,
    Level minPermittedLevel, MessageQueueClient msgQueueClient) {
    super(name, enabled, minPermittedLevel);
    this.msgQueueClient = msgQueueClient;
  }

  @Override
  protected void doLog(Level level, String mesage) {
    // 格式化level和message,输出到消息中间件
    msgQueueClient.send(...);
  }
}
```

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。

```
// 接口
public interface Filter {
  void doFilter(RpcRequest req) throws RpcException;
}
// 接口实现类：鉴权过滤器
public class AuthencationFilter implements Filter {
  @Override
  public void doFilter(RpcRequest req) throws RpcException {
    //...鉴权逻辑..
  }
}
// 接口实现类：限流过滤器
```

```
public class RateLimitFilter implements Filter {
  @Override
  public void doFilter(RpcRequest req) throws RpcException {
    //...限流逻辑...
  }
}
// 过滤器使用demo
public class Application {
  // filters.add(new AuthencationFilter());
  // filters.add(new RateLimitFilter());
  private List<Filter> filters = new ArrayList<>();

  public void handleRpcRequest(RpcRequest req) {
    try {
      for (Filter filter : fitlers) {
        filter.doFilter(req);
      }
    } catch(RpcException e) {
      // ...处理过滤结果...
    }
    // ...省略其他处理逻辑...
  }
}
```

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：

```
// 父类：非抽象类，就是普通的类．删除了log(),doLog()，新增了isLoggable().
public class Logger {
  private String name;
  private boolean enabled;
  private Level minPermittedLevel;

  public Logger(String name, boolean enabled, Level minPermittedLevel) {
    //...构造函数不变，代码省略...
  }

  protected boolean isLoggable() {
    boolean loggable = enabled && (minPermittedLevel.intValue() <= level.intValue());
    return loggable;
  }
}
// 子类：输出日志到文件
public class FileLogger extends Logger {
  private Writer fileWriter;

  public FileLogger(String name, boolean enabled,
    Level minPermittedLevel, String filepath) {
    //...构造函数不变，代码省略...
  }

  public void log(Level level, String mesage) {
    if (!isLoggable()) return;
    // 格式化level和message,输出到日志文件
    fileWriter.write(...);
  }
}
// 子类：输出日志到消息中间件(比如kafka)
public class MessageQueueLogger extends Logger {
  private MessageQueueClient msgQueueClient;

  public MessageQueueLogger(String name, boolean enabled,
    Level minPermittedLevel, MessageQueueClient msgQueueClient) {
    //...构造函数不变，代码省略...
  }

  public void log(Level level, String mesage) {
    if (!isLoggable()) return;
    // 格式化level和message,输出到消息中间件
    msgQueueClient.send(...);
  }
}
```

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。

```
Logger logger = new FileLogger("access-log", true, Level.WARN, "/users/wangzheng/access.log");
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，不就可以了吗？

```
public class Logger {
  // ...省略部分代码...
  public void log(Level level, String mesage) { // do nothing... }
}
public class FileLogger extends Logger {
  // ...省略部分代码...
  @Override
  public void log(Level level, String mesage) {
    if (!isLoggable()) return;
    // 格式化level和message,输出到日志文件
    fileWriter.write(...);
  }
}
public class MessageQueueLogger extends Logger {
  // ...省略部分代码...
  @Override
  public void log(Level level, String mesage) {
    if (!isLoggable()) return;
    // 格式化level和message,输出到消息中间件
    msgQueueClient.send(...);
  }
}
```

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）。

```
class Strategy { // 用抽象类模拟接口
  public:
    ~Strategy();
    virtual void algorithm()=0;
  protected:
    Strategy();
};
```

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。

```
public class MockInteface {
  protected MockInteface() {}
  public void funcA() {
    throw new MethodUnSupportedException();
  }
```

```
}
```

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。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。

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。

精选留言：

- 辣么大 2019-11-20 06:38:20
  如果让我聊聊接口和抽象类，我会这么聊：定义、区别（是什么），存在意义（从哪来），应用（到哪去）。
  1、定义：
  抽象类：不允许实例化，只能被继承；可包含属性和方法，包含抽象方法；子类继承抽象类必须重写抽象方法。
  接口：不允许实例化，只能被实现；不包含属性和普通方法，包含抽象方法、静态方法、default 方法；类实现接口时，必须实现抽象方法。
  2、意义：
  抽象类：解决复用问题，适用于is-a的关系。
  接口：解决抽象问题，适用于has-a的关系。
  3、应用：
  例如：
  解决复用问题：java中的子类FileInputStream和PipeInputStream等继承抽象类InputStream。重写了read(source)方法，InputStream 中还包含其他方法，FileInputStream继承抽象类复用了父类的其他方法。
  解决抽象问题：抽象类InputStream实现了Closeable接口，该接口中包含close()抽象方法。Closeable这个接口还在很多其他类中实现了，例如Channel，Socket中都有close() 关闭这个功能，但具体实现每个类又各有不同的实现，这个就是抽象。

  4、补充知识点（语法）：
  Java接口中可以定义静态方法、default方法，枚举类型，接口中还可以定义接口（嵌套）。
  public interface ILog {
  enum Type {
  LOW,
  MEDIUM,
  HIGH
  }
  interface InILog{
  void initInLog();
  }
  default void init() {
  Type t = Type.LOW;
  System.out.println(t.ordinal());
  }
  static void OS() {
  System.out.println(System.getProperty("os.name", "linux"));

```
}
void log(OutputStream out);
}
class ConsoleLog implements ILog {
@Override
public void log(OutputStream out) {
System.out.println("ConsoleLog...");
}
}
```
[42赞]

- NoAsk 2019-11-20 07:14:49
  Java使用abstract表示抽象类，interface表示接口。
  老师讲的很好，我补充一下使用；
  1.java中抽象类是类，而java不支持多继承，当想要实现多继承的时候只能使用表示has-a的接口来实现。
  2.在java8之前，定义的接口不能有具体实现，这会导致在后续维护的时候如果想要在接口中新增一个方法，必须在所有实现类中都实现一遍，并且只有几个新的实现类可能要去具体实现，其他的都只是加上默认实现，这样比较麻烦。在java8中接口可以用使用关键字default，来实现一个默认方法，这样就解决了上述的麻烦。 [20赞]

- 侯金彪 2019-11-20 07:58:09
  接口强调具有什么能力(has-a)，
  抽象类强调是什么(is-a)。 [12赞]

- Daiver 2019-11-20 00:51:54
  go和java 都有接口设计，但go的设计是飞入侵入性（duck type），而java必须显式实现该借口，这一点go做的真的好了。 [8赞]

- 梦倚栏杆 2019-11-20 07:45:08
  关于抽象类和接口有一个疑问，也可能是对业务模型不够了解，同一件事其实可以表达成is a也可以表达成has a，这个就看你的语言描述到底是个名词还是行为。
  举例说明：
  宠物猫和宠物狗都是宠物。
  宠物猫和宠物狗都有会被宠。
  is a基类定义：宠物类
  has a 接口定义：可被宠的
  如果取决于需求的语言阐述，感觉就太依赖个人表达习惯了，这不是一个团队持续迭代好项目的方式吧
  希望老师可以帮忙解惑 [5赞]

- Smallfly 2019-11-20 08:44:27
  抽象类 vs 接口

  抽象类 接口
  实例化 否 否
  属性 是 否
  方法 是 是
  实现 是 否
  目的 复用 扩展
  意义 is-a has-a [4赞]

- 熊斌 2019-11-20 07:13:43
  举个例子吧，附件上传服务端实现

  需求：支持上传、下载、删除以及文件的存储，存储的话需要支持存本地和云端，涉及附件上传的业务点有十几个。

  设计思路：
  1、接口设计：上传、下载、删除是公共行为，抽象到接口中

  2、存储方法是一大块公共代码，写到抽象类里面

  3、每个业务的附件上传子类实现接口、继承抽象类 [4赞]

- 大牛凯 2019-11-20 03:14:23
  Python有抽象类吧？abc不是可以定义抽象类么 [4赞]

- 编程界的小学生 2019-11-20 09:39:52
  1.普通类模拟抽象类方法：
  私有构造器，protected修饰的方法和成员变量，模拟抽象方法的时候在实现体内直接抛出异常并写好交由子类具体实现的注释说明
  2.用的JAVA。天然支持，abstract和interface
  3.抽象类：
  不支持多继承，为了解决代码复用问题而产生，他可以内部定义方法实现、公用属性以及需要子类各自实现的抽象方法，应用场景最典型的就是模板模式，is a的关系。爸爸儿子的关系。
  接口：
  支持多继承，为了让代码更加解耦合，更加灵活而产生，他所定义的方法都是抽象的，has a的关系，好比一种协议，一种规范，应用场景贼广泛，比如各大设计模式，设计原则。再比如责任链模式 [2赞]

- Uncle.席 2019-11-20 07:05:49
  接口强调某一方面的统一，抽象强调共性的复用 [2赞]

- 香蕉派2号 2019-11-20 05:50:14
  问题1:
  c#接口两种实现方式：
  // 直接实现接口
  interface IAttack
  {
  void Attck();
  }
  class Gun : IAttack
  {
  public void Attck()
  {
  throw new NotImplementedException();
  }
  }
  class Rifle : IAttack
  {
  public void Attck()

```
{
throw new NotImplementedException();
}
}

// 显示实现接口（当多个接口的中需要实现的方法名称一样的时候）
interface IAttack
{
void Attack();
}
interface ISpecialAttack
{
void Attack();
}
class Gun : IAttack, ISpecialAttack
{
public void Attack()
{
Console.WriteLine("普通攻击");
throw new NotImplementedException();
}
void ISpecialAttack.Attack() // 接口的显示实现
{
Console.WriteLine("特殊攻击");
throw new NotImplementedException();
}
}
class Rifle : IAttack, ISpecialAttack
{
public void Attack()
{
Console.WriteLine("普通攻击");
throw new NotImplementedException();
}
void ISpecialAttack.Attack() // 接口的显示实现
{
Console.WriteLine("特殊攻击");
throw new NotImplementedException();
}
}
public class TestWeapons
{
public void TestMethod()
{
ISpecialAttack desertEagle = new Gun();
desertEagle.Attack();

IAttack rifle = new Rifle();
rifle.Attack();
}
```

```
}
```

c#抽象类定义以及实现：

```
abstract class Weapon{
public float attackRange;
public float damage;
public virtual void Attack(Enemy enmey)
{
enmey.health -= damage;
Console.WriteLine("attack...");
}
}

class Gun: Weapon
{
public override void Attack(Enemy enmey)
{
base.Attack(enmey);
}
}

public class TestWeapon02
{
public static void main02(string[] args)
{
Enemy enemy = new Enemy();
Weapon gun = new Gun()
{
damage = 5,
attackRange = 100,
};
gun.Attack(enemy);
Console.WriteLine("enemy health: " + enemy.health);
}
}
```

问题2:
总的来说，抽象类就是类别上的总称；接口就是行为上的规范

[2赞]

- 未来、尽在我手 2019-11-20 00:29:45
  在学Java时，我一直在疑惑什么时候用抽象类，什么时候用接口。现在学完对此有一点理解，也明白他们之间的关系及用法。下来需要多实践与学习。 [2赞]

- 五岳寻仙 2019-11-21 20:27:30

  Python中抽象类是通过 abc 模块中的 abstractmethod 和 ABCMeta

```
from abc import abstractmethod, ABCMeta

class BaseClass(metaclass=ABCMeta):
@abstractmethod
def fun(self, x): pass
```

因为 Python 需要天然支持多继承，所以可以直接用上述的抽象类实现接口 [1赞]

- Maurice 2019-11-21 15:32:54

嗯，刚刚看了一下评论区，小弟给大家初学设计模式以及刚刚接触项目的同学一些建议，其实在学这一章更以及往后正式的20多种设计模式时候更加锻炼的是一种抽象思维，以我个人的一点经验，就像老师一开始说的，在学习的时候，不要总是类比生活中的一些阿猫啊狗的例子，没错初次是很好理解，而且你会感觉好像懂了，但是实际上你还是不懂（可能是以为我笨），建议大家没学习一种模式的时候，思考一下现在在做的项目中那些业务可以进行优化，如果可以的话，可以进行重构，这样对你的学习才有很大的帮助；举个栗子，例如策略模式，我们在代码是不是经常调用第三方接口，每个接口都有它的规范之类的，如果你按照low 的方式就是按照规范组织一些报文请求，但是这个时候，你可以用的你的抽象思维，比如，在这类业务中，无外乎就是，1、构建报文，2、请求结构，3、解析返回，那么我就可以这样设计：

```
/**
* 请求统一接口
*/
public interface Fun<T> {
/**
* 发送请求报文
*/
String constructionRequest();
/**
* 请求报文
*/
String doPost(String request);
/**
* 解析回执报文
*/
T resolveResponse(String responseText);
}
```

然后再写一个策略的算法簇（应该是这么叫）

```
/**
* 统一实现接口
*/
public class DealWithHandle {

/**
* 描述 create.
*
* @param <T> the type parameter
* @param fun the fun
* @return the t
* @author Maurice Mao
* @created 2018 -07-13 10:58:32 Handle t.
*/
public static <T> T handle(Fun<T> fun) {
```

String request = fun.constructionRequest();

String response = fun.doPost(request);

T t = fun.resolveResponse(response);

return t;

}
}
这个时候，假设你做支付宝扫码支付的时候（或者微信以及第三方支付），是不是可以这么调用：
public class OfficialAliScan {

private static Logger logger = LoggerFactory.getLogger(OfficialAliScan.class);

public static ApiTradeResultDto operate(final TradeOrderDO orderDO, final SubMerchatAuthDO sub
MerchatAuthDO, final String authCode) {
return DealwithHandle.handle(new DealwithHandle.Fun<ApiTradeResultDto>() {
@Override
public String constructionRequest() {
///todo
}

@Override
public String doPost(String bizContent) {
///todo
}

@Override
public ApiTradeResultDto resolveResponse(String responseTxt) {
///todo
}
});
}
这样的话，你的业务代码组织的就很清晰，也便于调试维护
[1赞]

- 张飞online 2019-11-20 23:46:56
谈谈c吧，linux内核的设备驱动总线模型，就但说设备吧，

抽象类:
首先来个通用设备结构体，到了具体设备把通用设备结构体包含了，这是典型的抽象类思想，提炼公共部分做基类，提高复用度。
接口，接口实现就是函数指针，不管你的驱动如何实现，接口不变，你就按照函数指针的定义实现函数，任何人都要受控这个函数定义协议。
[1赞]

- nelson 2019-11-20 23:30:42
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）
这里关于C++的说明有点问题，virtual是虚函数和abstract没有对应关系，换句话说，java中成员函数都是virtual的。Java中的abstract关键字对应“virtual void algorithm()=0”当中的“=0”，也就是所谓的纯虚函数，即在本类中没有实现，派生类必须实现 [1赞]

- tt 2019-11-20 20:07:48
思想最重要，用什么工具都可以实现特定的思想。
抽象类体现了IS-A关系，接口体现了HAS-A关系，体现了协议，也即为了解耦行为和实现。

同时，抽象类和接口也体现了方法论，一个是自底向上，一个是自顶向下。

学完这节课，确实对它们的却别和联系更加清楚了。 [1赞]

- 李小四 2019-11-20 19:57:37
设计模式_08
1. 文中讲到用Java普通类实现抽象类，抽象类的限制比接口少，所以在接口的基础上添加一些普通类的功能即可。
public class MockAbstractClass {
protected MockAbstractClass() {}
public void funcAbstract() {
throw new MethodUnSupportedException();
}
public void Object field;
public void funcImpl() {
//function implement code
}
}
2. 今天的内容，印象最深刻的就是**is-a ==> 抽象类， has-a ==> 接口**，坦白讲，之前不是很清楚，看感觉来使用抽象类和方法。 [1赞]

- Monday 2019-11-20 16:50:27
java抽象类与接口区别
1、前者可以包含成员属性和方法实现，后者不可以
2、is-a与has-a区别
3、前者单继承，后者多实现 [1赞]

- Yo nací para quererte. 2019-11-20 16:05:52
记录一下C#的抽象类和接口：
C#中既有virtual关键字（用于定义虚方法）也有abstract关键字（用于定义抽象类或抽象方法）也有interface关键字（用于定义接口）。
子类可以使用new关键字覆盖父类方法，可以使用override关键字重写父类方法。
总结：
1. 不管重写还是覆盖都不会影响父类自身的功能（除非父类代码被改变）
2. 当用子类创建父类对象时，如Animal animal = new Cat();。重写会改变父类的功能，即调用子类的方法，而覆盖不会，扔调用父类方法。
3. 虚方法和实方法都可以被覆盖（new），抽象方法和接口不可以。
4. 抽象方法和接口和虚方法可以被重写（override），实方法不可以。
5. 重写使用的频率比较高，用于实现多态；覆盖用的频率比较低，用于对以前无法修改的类进行继承的时候。 [1赞]