09-理论六：为什么基于接口而非实现编程？有必要为每个类都定义接口吗？

在上一节课中，我们讲了接口和抽象类，以及各种编程语言是如何支持、实现这两个语法概念的。今天，我们继续讲一个跟“接口”相关的知识点：基于接口而非实现编程。这个原则非常重要，是一种非常有效的提高代码质量的手段，在平时的开发中特别经常被用到。

为了让你理解透彻，并真正掌握这条原则如何应用，今天，我会结合一个有关图片存储的实战案例来讲解。除此之外，这条原则还很容易被过度应用，比如为每一个实现类都定义对应的接口。针对这类问题，在今天的讲解中，我也会告诉你如何来做权衡，怎样恰到好处地应用这条原则。

话不多说，让我们正式开始今天的学习吧！

如何解读原则中的“接口”二字？

“基于接口而非实现编程”这条原则的英文描述是：“Program to an interface, not an implementation”。我们理解这条原则的时候，千万不要一开始就与具体的编程语言挂钩，局限在编程语言的“接口”语法中（比如Java中的interface接口语法）。这条原则最早出现于1994年GoF的《设计模式》这本书，它先于很多编程语言而诞生（比如Java语言），是一条比较抽象、泛化的设计思想。

实际上，理解这条原则的关键，就是理解其中的“接口”两个字。还记得我们上一节课讲的“接口”的定义吗？从本质上来看，“接口”就是一组“协议”或者“约定”，是功能提供者提供给使用者的一个“功能列表”。“接口”在不同的应用场景下会有不同的解读，比如服务端与客户端之间的“接口”，类库提供的“接口”，甚至是一组通信的协议都可以叫作“接口”。刚刚对“接口”的理解，都比较偏上层、偏抽象，与实际的写代码离得有点远。如果落实到具体的编码，“基于接口而非实现编程”这条原则中的“接口”，可以理解为编程语言中的接口或者抽象类。

前面我们提到，这条原则能非常有效地提高代码质量，之所以这么说，那是因为，应用这条原则，可以将接口和实现相分离，封装不稳定的实现，暴露稳定的接口。上游系统面向接口而非实现编程，不依赖不稳定的实现细节，这样当实现发生变化的时候，上游系统的代码基本上不需要做改动，以此来降低耦合性，提高扩展性。

实际上，“基于接口而非实现编程”这条原则的另一个表述方式，是“基于抽象而非实现编程”。后者的表述方式其实更能体现这条原则的设计初衷。在软件开发中，最大的挑战之一就是需求的不断变化，这也是考验代码设计好坏的一个标准。**越抽象、越顶层、越脱离具体某一实现的设计，越能提高代码的灵活性，越能应对未来的需求变化。好的代码设计，不仅能应对当下的需求，而且在将来需求发生变化的时候，仍然能够在不破坏原有代码设计的情况下灵活应对。**而抽象就是提高代码扩展性、灵活性、可维护性最有效的手段之一。

如何将这条原则应用到实战中？

对于这条原则，我们结合一个具体的实战案例来进一步讲解一下。

假设我们的系统中有很多涉及图片处理和存储的业务逻辑。图片经过处理之后被上传到阿里云上。为了代码复用，我们封装了图片存储相关的代码逻辑，提供了一个统一的AliyunImageStore类，供整个系统来使用。具体的代码实现如下所示：

```
public class AliyunImageStore {
```

```
  //...省略属性、构造函数等...

  public void createBucketIfNotExisting(String bucketName) {
    // ...创建bucket代码逻辑...
    // ...失败会抛出异常..
  }

  public String generateAccessToken() {
    // ...根据accesskey/secrectkey等生成access token
  }

  public String uploadToAliyun(Image image, String bucketName, String accessToken) {
    //...上传图片到阿里云...
    //...返回图片存储在阿里云上的地址(url) ...
  }

  public Image downloadFromAliyun(String url, String accessToken) {
    //...从阿里云下载图片...
  }
}

// AliyunImageStore类的使用举例
public class ImageProcessingJob {
  private static final String BUCKET_NAME = "ai_images_bucket";
  //...省略其他无关代码...

  public void process() {
    Image image = ...; //处理图片，并封装为Image对象
    AliyunImageStore imageStore = new AliyunImageStore(/*省略参数*/);
    imageStore.createBucketIfNotExisting(BUCKET_NAME);
    String accessToken = imageStore.generateAccessToken();
    imagestore.uploadToAliyun(image, BUCKET_NAME, accessToken);
  }

}
```

整个上传流程包含三个步骤：创建bucket（你可以简单理解为存储目录）、生成access token访问凭证、携带access token上传图片到指定的bucket中。代码实现非常简单，类中的几个方法定义得都很干净，用起来也很清晰，乍看起来没有太大问题，完全能满足我们将图片存储在阿里云的业务需求。

不过，软件开发中唯一不变的就是变化。过了一段时间后，我们自建了私有云，不再将图片存储到阿里云了，而是将图片存储到自建私有云上。为了满足这样一个需求的变化，我们该如何修改代码呢？

我们需要重新设计实现一个存储图片到私有云的PrivateImageStore类，并用它替换掉项目中所有的AliyunImageStore类对象。这样的修改听起来并不复杂，只是简单替换而已，对整个代码的改动并不大。不过，我们经常说，“细节是魔鬼”。这句话在软件开发中特别适用。实际上，刚刚的设计实现方式，就隐藏了很多容易出问题的“魔鬼细节”，我们一块来看看都有哪些。

新的PrivateImageStore类需要设计实现哪些方法，才能在尽量最小化代码修改的情况下，替换掉AliyunImageStore类呢？这就要求我们必须将AliyunImageStore类中所定义的所有public方法，在PrivateImageStore类中都逐一定义并重新实现一遍。而这样做就会存在一些问题，我总结了下面两点。

首先，AliyunImageStore类中有些函数命名暴露了实现细节，比如，uploadToAliyun()和downloadFromAliyun()。如果开发这个功能的同事没有接口意识、抽象思维，那这种暴露实现细节的命名方式就不足为奇了，毕竟最初我们只考虑将图片存储在阿里云上。而我们把这种包含“aliyun”字眼的方

法，照抄到PrivateImageStore类中，显然是不合适的。如果我们在新类中重新命名uploadToAliyun()、downloadFromAliyun()这些方法，那就意味着，我们要修改项目中所有使用到这两个方法的代码，代码修改量可能就会很大。

其次，将图片存储到阿里云的流程，跟存储到私有云的流程，可能并不是完全一致的。比如，阿里云的图片上传和下载的过程中，需要生产access token，而私有云不需要access token。一方面，AliyunImageStore中定义的generateAccessToken()方法不能照抄到PrivateImageStore中；另一方面，我们在使用AliyunImageStore上传、下载图片的时候，代码中用到了generateAccessToken()方法，如果要改为私有云的上传下载流程，这些代码都需要做调整。

那这两个问题该如何解决呢？解决这个问题的根本方法就是，在编写代码的时候，要遵从“基于接口而非实现编程”的原则，具体来讲，我们需要做到下面这3点。

1. 函数的命名不能暴露任何实现细节。比如，前面提到的uploadToAliyun()就不符合要求，应该改为去掉aliyun这样的字眼，改为更加抽象的命名方式，比如：upload()。
2. 封装具体的实现细节。比如，跟阿里云相关的特殊上传（或下载）流程不应该暴露给调用者。我们对上传（或下载）流程进行封装，对外提供一个包裹所有上传（或下载）细节的方法，给调用者使用。
3. 为实现类定义抽象的接口。具体的实现类都依赖统一的接口定义，遵从一致的上传功能协议。使用者依赖接口，而不是具体的实现类来编程。

我们按照这个思路，把代码重构一下。重构后的代码如下所示：

```
public interface ImageStore {
  String upload(Image image, String bucketName);
  Image download(String url);
}

public class AliyunImageStore implements ImageStore {
  //...省略属性、构造函数等...

  public String upload(Image image, String bucketName) {
    createBucketIfNotExisting(bucketName);
    String accessToken = generateAccessToken();
    //...上传图片到阿里云...
    //...返回图片在阿里云上的地址(url)...
  }

  public Image download(String url) {
    String accessToken = generateAccessToken();
    //...从阿里云下载图片...
  }

  private void createBucketIfNotExisting(String bucketName) {
    // ...创建bucket...
    // ...失败会抛出异常..
  }

  private String generateAccessToken() {
    // ...根据accesskey/secrectkey等生成access token
  }
}

// 上传下载流程改变：私有云不需要支持access token
public class PrivateImageStore implements ImageStore  {
```

```
  public String upload(Image image, String bucketName) {
    createBucketIfNotExisting(bucketName);
    //...上传图片到私有云...
    //...返回图片的url...
  }

  public Image download(String url) {
    //...从私有云下载图片...
  }

  private void createBucketIfNotExisting(String bucketName) {
    // ...创建bucket...
    // ...失败会抛出异常..
  }
}

// ImageStore的使用举例
public class ImageProcessingJob {
  private static final String BUCKET_NAME = "ai_images_bucket";
  //...省略其他无关代码...

  public void process() {
    Image image = ...;//处理图片，并封装为Image对象
    ImageStore imageStore = new PrivateImageStore(...);
    imagestore.upload(image, BUCKET_NAME);
  }
}
```

除此之外，很多人在定义接口的时候，希望通过实现类来反推接口的定义。先把实现类写好，然后看实现类中有哪些方法，照抄到接口定义中。如果按照这种思考方式，就有可能导致接口定义不够抽象，依赖具体的实现。这样的接口设计就没有意义了。不过，如果你觉得这种思考方式更加顺畅，那也没问题，只是将实现类的方法搬移到接口定义中的时候，要有选择性的搬移，不要将跟具体实现相关的方法搬移到接口中，比如AliyunImageStore中的generateAccessToken()方法。

总结一下，我们在做软件开发的时候，一定要有抽象意识、封装意识、接口意识。在定义接口的时候，不要暴露任何实现细节。接口的定义只表明做什么，而不是怎么做。而且，在设计接口的时候，我们要多思考一下，这样的接口设计是否足够通用，是否能够做到在替换具体的接口实现的时候，不需要任何接口定义的改动。

是否需要为每个类定义接口？

看了刚刚的讲解，你可能会有这样的疑问：为了满足这条原则，我是不是需要给每个实现类都定义对应的接口呢？在开发的时候，是不是任何代码都要只依赖接口，完全不依赖实现编程呢？

做任何事情都要讲求一个“度”，过度使用这条原则，非得给每个类都定义接口，接口满天飞，也会导致不必要的开发负担。至于什么时候，该为某个类定义接口，实现基于接口的编程，什么时候不需要定义接口，直接使用实现类编程，我们做权衡的根本依据，还是要回归到设计原则诞生的初衷上来。只要搞清楚了这条原则是为了解决什么样的问题而产生的，你就会发现，很多之前模棱两可的问题，都会变得豁然开朗。

前面我们也提到，这条原则的设计初衷是，将接口和实现相分离，封装不稳定的实现，暴露稳定的接口。上游系统面向接口而非实现编程，不依赖不稳定的实现细节，这样当实现发生变化的时候，上游系统的代码基本上不需要做改动，以此来降低代码间的耦合性，提高代码的扩展性。

从这个设计初衷上来看，如果在我们的业务场景中，某个功能只有一种实现方式，未来也不可能被其他实现方式替换，那我们就没有必要为其设计接口，也没有必要基于接口编程，直接使用实现类就可以了。

除此之外，越是不稳定的系统，我们越是要在代码的扩展性、维护性上下功夫。相反，如果某个系统特别稳定，在开发完之后，基本上不需要做维护，那我们就没有必要为其扩展性，投入不必要的开发时间。

重点回顾

今天的内容到此就讲完了。我们来一块总结回顾一下，你需要掌握的重点内容。

1.“基于接口而非实现编程”，这条原则的另一个表述方式，是“基于抽象而非实现编程”。后者的表述方式其实更能体现这条原则的设计初衷。我们在做软件开发的时候，一定要有抽象意识、封装意识、接口意识。越抽象、越顶层、越脱离具体某一实现的设计，越能提高代码的灵活性、扩展性、可维护性。

2.我们在定义接口的时候，一方面，命名要足够通用，不能包含跟具体实现相关的字眼；另一方面，与特定实现有关的方法不要定义在接口中。

3.“基于接口而非实现编程”这条原则，不仅仅可以指导非常细节的编程开发，还能指导更加上层的架构设计、系统设计等。比如，服务端与客户端之间的“接口”设计、类库的“接口”设计。

课堂讨论

在今天举的代码例子中，尽管我们通过接口来隔离了两个具体的实现。但是，在项目中很多地方，我们都是通过下面第8行的方式来使用接口的。这就会产生一个问题，那就是，如果我们要替换图片存储方式，还是需要修改很多类似第8行那样的代码。这样的设计还是不够完美，对此，你有更好的实现思路吗？

```
// ImageStore的使用举例
public class ImageProcessingJob {
  private static final String BUCKET_NAME = "ai_images_bucket";
  //...省略其他无关代码...

  public void process() {
    Image image = ...;//处理图片，并封装为Image对象
    ImageStore imageStore = new PrivateImageStore(/*省略构造函数*/);
    imagestore.upload(image, BUCKET_NAME);
  }
```

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：

- 香蕉派2号 2019-11-22 05:40:02
  思考题
  解决方案=配置文件+反射+工厂模式 [12赞]

- 失火的夏天 2019-11-22 00:16:06
  思考题估计就是要引出工厂模式了吧 [6赞]

- 编程界的小学生 2019-11-22 00:14:19
首先这篇文章受益匪浅，尤其是第二点，与特定实现有关的方法不要暴露到接口中，深有体会。

其次问题解答
我个人的解决方案是这种情况不要去直接new，而是用工厂类去管理这个对象，然后名字可以起成getInstance这类不包含某个具体实现的含义的抽象名称。将来修改直接修改工厂类的getInstance方法即可，这种方式可取吗？还有其他更好的方式吗？求老师点评。 [6赞]

- 辣么大 2019-11-22 07:48:01
关于思考题我想出两种方法改进：简单工厂方法和使用反射。

1、简单工厂方法
ImageStore imageStore = ImageStoreFactory.newInstance(SOTRE_TYPE_CONFIG);
config文件可以写类似properties的文件，使用key-value存储。

缺点：再新增另一种存储手段时，需要修改工厂类和添加新的类。修改工厂类，违反了开放-封闭原则。

那有没有更好一点的方法呢？

2、使用反射。
在配置文件中定义需要的image store类型。
在ProcessJob中
ImageStore store = (ImageStore) Class.forName(STORE_CLASS)
.newInstance();

缺点：使用反射，在大量创建对象时会有性能损失。

关于减少ProcessJob中的修改，还有没有更好的方法呢？我只是抛砖引玉，希望和大家一起讨论。具体实现：https://github.com/gdhucoder/Algorithms4/tree/master/geekbang/designpattern/u009

补充：
关于access token：Aliyun的AccessToken时有expireTime时限的。不需要每次重新获取，过期时重新获取即可。 [5赞]

- 秋惊蛰 2019-11-22 02:21:41
依赖注入，从外部构建具体类的对象，传入使用的地方 [5赞]

- YouCompleteMe 2019-11-22 01:50:48
抽象工厂，把创建具体类型放到工厂类里 [4赞]

- 红酒花生 2019-11-22 00:20:24
存储图片的方式写入到配置文件，第8行改用传入类型参数来实例化不同的对象，明天补上代码。 [4赞]

- NoAsk 2019-11-22 07:14:17
关于什么时候定义接口的一些拙见：
当方法会有其他实现，或者不稳定的时候需要定义接口；
1.不稳定的方法一般能事先确定，用接口能提高可维护性
2.但在开发时往往不确定是否需要其他实现，我的原则是等到需要使用接口的时候再去实现。所以根据kiss原则一般我会先用方法实现，如果有一天真的需要有新的实现的时候再重新抽象出接口对代码进行小重构。

就老师的例子进行一下说明：
刚开始只需要阿里云进行图片上传下载功能，我就先只实现阿里云的图片上传下载方法。
后期发现需要有私有云的上传下载方法的话，那就对这个功能通过接口进行抽象。但是你永远不知道到底是新的图片上传下载功能先来到还是其他阿里接口先来到，如果是新的阿里接口，也是用的这一套token方法，那就用抽象方法或接口对token部分实现抽象。

课后问题：
简单工厂，工厂模式可以提高可扩展性，维护性。
java spring项目可以使用注入的方式。 [2赞]

- 秋惊蛰 2019-11-22 02:32:26
这一节解决了一些疑惑，但是还有三个问题想问一下：
1. 看到有些Spring Boot的代码，会给每个业务逻辑分别定义一个service接口，并用一个类来实现这个接口，然后在controller里像第8行那样调用。
这种是不是就过度使用接口了，还是说有必要考虑每个业务逻辑可能的实现方式。
2. 能不能具体的说一下哪些典型场景适合用接口抽象，比如输入输出，这两个地方肯定不止一种方式，用接口很有必要，还有其他场景吗？
3. Python这种动态语言怎么做到面向抽象编程？ [2赞]

- William 2019-11-22 01:39:05
所以思考题，想到的是，将接口作为构造函数中的参数，传递进来，再调用. [2赞]

- bearlu 2019-11-22 08:52:58
老师，希望能把示例代码和问题代码也放到Github上。 [1赞]

作者回复2019-11-22 09:15:13
👏 我抽空整理一下放上去
https://github.com/wangzheng0822

- Monday 2019-11-22 08:50:34
依赖注入可以解决思考题，基于接口的实现有多种时，注入处也需要指明是哪咤实现 [1赞]

- 超威丶 2019-11-22 08:21:32
个人觉得维护map是最好的选择，实现类型和具体实现对应。 [1赞]

- 黄林晴 2019-11-22 08:08:02
打卡✔ [1赞]

- 海 2019-11-22 07:51:30
我是搞Java的，关于思考题，个人感觉主要依靠控制反转，即把对象的构造权交给容器，而非代码中直接写死。像Spring那种依赖注入的方式就可以，或者不使用Spring，可以把具体的实现类全路径名配置到配置文件中，代码中以Class.forName的方式得到Class，然后再用Class的newInstance方法得到实例并缓存起来以便后面使用避免重复构造实例。以后替换实现的时候只需要替换配置文件中的类全路径名即可。当然前提是这个Class需要实现统一抽象出来的接口，使用逻辑中也是 [1赞]

- 业余爱好者 2019-11-22 07:43:49
关于抽象和函数命名的问题，不知道哪个大佬说过这么一句话:

每个优秀的程序员都知道，不应该定义一个attackBaghdad() ‘袭击巴格达‘的方法，而是应该把城市

作为函数的参数 attack(city)。 [1赞]

- Paul Shan 2019-11-22 04:55:52
  基于抽象而非具体体现了信息隐藏和分离代码中稳定性不同的部分。在一个上传图片的部分不需要知道图片是如何上传的，阿里云以及token就属于过多的信息，有必要隐藏这些信息。另外一方面上传图片这件事比阿里云实现要稳定的多，不上传图片的概率低于不用阿里云上传图片的概率。这里有必要分离图片上传这个接口和用阿里云上传这个实现。
  不过原来的实现也没什么问题，毕竟谁也不能未卜先知，将来一定会替换阿里云。如果我拿到这个变更需求，我会先用同名接口替换原来的实现（原来的阿里类实现清晰，功能单一，只是不适合直接调用），然后用adapter来转接口，然后一步一步实现接口和实现的分离，目标是接口能够隐藏信息，实现能够清晰明了，每一步都能用IDE工具重构，每一步都能编译和测试。 [1赞]

- 传说中的成大大 2019-11-22 11:09:00
  我也想说思考题 是要引出设计模式了哈哈哈哈

- Jesse 2019-11-22 10:48:05
  思考题
  用抽象工厂模式解决将process()中的具体的ImageStore类解耦出来，我觉得组合模式也可以，ImageProcessingJob持有一个ImageStore 引用，能达到同样的效果。

- 塔兹米 2019-11-22 10:47:35
  打卡，
  关于是思考题，我们用 Java 的解决方法就是spring的依赖注入了，在配置文件里，配置好实现类的地址，
  将接口声明成，成员变量然后注入即可。