设计模式学习笔记之适配器模式

作者：hjljy
原文链接：https://ld246.com/article/1565019085053
来源网站：链滴

<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile
com/bing/20190515.jpg?imageView2/1/w/960/h/540/interlace/1/q/100">
<h2 id="前言">前言</h2>
这是一篇学习笔记，内容很多是来源于网上的资料，然后按照自己学习情况进行的总结，有些是
身的感受，有些是网上比较好的资料的引用。
<b>如果有人看到我写的笔记有不对的地方欢迎留言指出来，是真的欢迎指出来，因为我可能会
很久，然后才发现。学习技术不能闭门造车，要多交流，多讨论，多思考才能成长的快，学的快。</
> <b>
我的个人博客：<a href="https://ld246.com/forward?goto=https%3A%2F%2Fwww.hjljy.cn"
target="_blank" rel="nofollow ugc">海加尔金鹰
<h2 id="什么是适配器模式">什么是适配器模式</h2>
<h2 id="适配器模式的定义">适配器模式的定义</h2>
适配器模式有时也称包装器模式，指的是将一个类的接口转换成我们需要的接口。根据不同的实
方式适配器可以分为三种：继承类的叫类适配器，持有类的叫对象适配器，实现接口的叫接口适配器

<h2 id="适配器模式适用场景">适配器模式适用场景</h2>
<blockquote>

系统需要使用现有的类，但现有的类却不兼容。
需要建立一个可以重复使用的类，用于一些彼此关系不大的类，并易于扩展，以便于面对将来会
现的类.
需要一个统一的输出接口，但是输入类型却不可预知<sup class="footnotes-ref" id="footnote
-ref-1">1</sup>

</blockquote>
<h2 id="适配器模式的结构">适配器模式的结构</h2>

目标接口(Target)：调用方所期待得到的接口。
适配器(Adaper)：核心角色，适配器把源接口转换成目标接口。
被适配者(Adaptee)：即真正的接口，需要通过适配器进行调用。

<h2 id="类适配器">类适配器</h2>
被适配者(Adaptee)
<pre><code class="highlight-chroma"><span class="highlight
cl">public class HQB {
    //源接口，不符
目前的需求需要进行适配修改。
    public String sh
w(String msg){
        return "霍去
："+msg;
    }
}
</code></pre>
目标接口(Target)
<pre><code class="highlight-chroma"><span class="highlight
cl">public interface HQUBService {
    //客户端希望得
的新的接口
    String newShow
String name,String msg);
}
</code></pre>
适配器(Adaper)

```
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">public class HQBAdapter extends HQB implements HQUBService {
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">    @Override
</span></span><span class="highlight-line"><span class="highlight-cl">    public String n
wShow(String name,String msg) {
</span></span><span class="highlight-line"><span class="highlight-cl">        //这是原来的
口
</span></span><span class="highlight-line"><span class="highlight-cl">        String old =
how(msg);
</span></span><span class="highlight-line"><span class="highlight-cl">        System.out.pr
ntln(old);
</span></span><span class="highlight-line"><span class="highlight-cl">        //新接口需要
原来的接口上进行部分修改
</span></span><span class="highlight-line"><span class="highlight-cl">        String replace
= old.replace("霍去病", name);
</span></span><span class="highlight-line"><span class="highlight-cl">        return replac
;
</span></span><span class="highlight-line"><span class="highlight-cl">    }
</span></span><span class="highlight-line"><span class="highlight-cl">}
</span></span></code></pre>
<h2 id="对象适配器">对象适配器</h2>
<p>在上述代码基础上新增一个适配器</p>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">public class HQBAdapter2 {
</span></span><span class="highlight-line"><span class="highlight-cl">    //将被适配者当
属性持有
</span></span><span class="highlight-line"><span class="highlight-cl">    private HQB hq
;
</span></span><span class="highlight-line"><span class="highlight-cl">    //通过构造器方
注入  不一定使用这种方式，但一定要保证被适配者不为NULL，不然怎么进行适配
</span></span><span class="highlight-line"><span class="highlight-cl">    public HQBAda
ter2(HQB hqb){
</span></span><span class="highlight-line"><span class="highlight-cl">        this.hqb=hqb

</span></span><span class="highlight-line"><span class="highlight-cl">    }
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">    public void ne
Show(String name,String msg){
</span></span><span class="highlight-line"><span class="highlight-cl">        //原来的处理
式
</span></span><span class="highlight-line"><span class="highlight-cl">        String show
 hqb.show(msg);
</span></span><span class="highlight-line"><span class="highlight-cl">        System.out.pr
ntln(show);
</span></span><span class="highlight-line"><span class="highlight-cl">        //新处理方式
</span></span><span class="highlight-line"><span class="highlight-cl">        System.out.pr
ntln(show.replace("霍去病",name));
</span></span><span class="highlight-line"><span class="highlight-cl">    }
</span></span><span class="highlight-line"><span class="highlight-cl">}
</span></span></code></pre>
<h2 id="类适配器和对象适配器的调用方式和结果">类适配器和对象适配器的调用方式和结果</h2>
<p>调用方式：</p>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
```

cl">public class HQBTest {
    public static vo
d main(String[] args) {
        //类适配器
        String newS
ow = new HQBAdapter().newShow("两分刘", "真的短");
        System.out.pr
ntln(newShow);
        //对象适配器
        new HQBAd
pter2(new HQB()).newShow("乔碧螺陛下","榜一");
    }
}
</code></pre>
结果：
<pre><code class="highlight-chroma"><span class="highlight
cl">霍去病：真的短
两分刘：真的短
霍去病：榜一
乔碧螺陛下：榜一
</code></pre>
<h2 id="类适配器和对象适配器的权衡">类适配器和对象适配器的权衡</h2>
<blockquote>

类适配器使用对象继承的方式，是静态的定义方式；而对象适配器使用对象组合的方式，是动态合的方式。
对于类适配器由于适配器直接继承了 Adaptee，使得适配器不能和 Adaptee 的子类一起工作，为继承是静态的关系，当适配器继承了 Adaptee 后，就不可能再去处理 Adaptee 的子类了。
对于对象适配器一个适配器可以把多种不同的源适配到同一个目标。换言之，同一个适配器可以源类和它的子类都适配到目标接口。因为对象适配器采用的是对象组合的关系，只要对象类型正确，不是子类都无所谓。
对于类适配器适配器可以重定义 Adaptee 的部分行为，相当于子类覆盖父类的部分实现方法。</l
>
对于对象适配器要重定义 Adaptee 的行为比较困难，这种情况下，需要定义 Adaptee 的子类来现重定义，然后让适配器组合子类。虽然重定义 Adaptee 的行为比较困难，但是想要增加一些新的为则方便的很，而且新增加的行为可同时适用于所有的源。
对于类适配器，仅仅引入了一个对象，并不需要额外的引用来间接得到 Adaptee。
对于对象适配器，需要额外的引用来间接得到 Adaptee。
'
建议尽量使用对象适配器的实现方式，多用合成或聚合、少用继承。当然，具体问题具体分析，根据要来选用实现方式，最适合的才是最好的。^{2}

</blockquote>
<h2 id="接口适配器">接口适配器</h2>
当一个接口当中有很多的方法，但是只想实现其中的几个，如果直接实现接口，就需要实现所有方法，会显得实现类很臃肿，这个时候可以通过一个抽象类来实现接口，将不需要实现的方法置空，后写一个类来继承抽象类就可以了。
<pre><code class="highlight-chroma"><span class="highlight
cl">   代码略
</code></pre>
个人感觉接口适配器更像一种接口实现的优化方式。
同时如果说一个接口里面有太多的方法，按照设计原则上来说，是应该对这个接口里的方法进行下细的。

<h2 id="适配器适用场景">适配器适用场景</h2>
1 在不修改原来接口源码的条件下，将原来的接口转换成新的接口。
2 提供一个高重用的类，可以供多处使用。例如一个手机壳，当我去小米公司上班，通过手机壳将手伪装成小米手机，去华为就伪装成华为手机。但实际上我用的还是诺基亚。
<h2 id="适配器优缺点">适配器优缺点</h2>
优点：提高了复用性以及扩展性
缺点：提高代码复杂性，方法调用不是很直观，明明调用的 A，结果调用的是 B 也可能是 C。反正能用就少用，如果用也要写好注释。
<hr>
<li id="footnotes-def-1"> 适配器模式(三种)简单使用 ↩

<li id="footnotes-def-2"> 《JAVA 与模式》之适配器模式 ↩

