黑客派

依赖倒置原则

作者：LieBrother

原文链接：https://hacpai.com/article/1546345948900

来源网站：黑客派

<img src="https://static.hacpai.com/images/img-loading.svg" alt="雪" data-src="http://ww.liebrother.com/upload/e4307f405e974a3a8e580392aebd1a92_0014_01.jpg">
2019 新年第一篇。
<blockquote>
设计模式六大原则之三：依赖倒置原则。
</blockquote>
<h2 id="简介">简介</h2>
姓名：依赖倒置原则
英文名：Dependence Inversion Principle
价值观：大男子主义的典型代表，什么都得通过老大或者老爸同意。
伴侣：一定是个温柔体贴的女子。
个人介绍：

High level modules should not depend upon low level modules.Both should depend upon abstractions. 高层模块不应该依赖低层模块，两者都应该依赖其抽象（模块间的依赖通过抽象发生实现类之间不发生直接的依赖关系，其依赖关系是通过接口或抽象类产生的）
Abstractions should not depend upon details. 抽象不应该依赖细节（接口或抽象类不依赖实现类）
Details should depend upon abstractions. 细节应该依赖抽象（实现类依赖接口或抽象类）

给大家讲个故事，我胡乱想的，如有雷同，肯定是英雄所见略同。那必须交个朋友。
一个小村里，有两家饭馆，虽然挂着不同的牌子，挨在一起，但是老板确是表兄弟。这两兄弟抠很，为了节省成本，密谋了一个想法：在两家饭馆谁家忙的时候，可以让不忙的那家的员工过去支援下。这样子，本来每家饭馆都需要 2 个洗碗工，总共需要 4 个，他们就只招了 3 个，省了 1 个洗碗的成本，当然不止洗碗工，还有服务员等等。两兄弟约定了规则：

A 饭馆需要支援的时候，B 饭馆老板，让 B 饭馆老板选哪个员工去支援，不能直接让 A 饭馆的工直接找 B 饭馆的员工去帮忙，但可以让 A 饭馆员工找 B 饭馆老板告知需要支援。
虽然老板权利大，但是也不能说 A 饭馆老板直接叫 B 饭馆的员工去帮忙。
员工没有真实的老板，今天为 A 饭馆工作就是 A 饭馆的员工，没有跟定哪个老板。

大概通过这个小故事，描述了依赖倒置原则的基本内容。
<h2 id="代码体现">代码体现</h2>
下面通过代码来模拟这个故事。
<h3 id="错误的示范">错误的示范</h3>
这个错误的示范将就看哈，可能有些问题没描述清楚。
<h4 id="老板和员工抽象">老板和员工抽象</h4>
<pre><code class="language-java highlight-chroma">abstract class Boss {

}

abstract class Staff {

}
</code></pre>

<h4 id="老板具体类">老板具体类</h4>

<pre><code class="language-java highlight-chroma">class</sp
n> BossA extends
Boss {

}

class BossB <span
lass="highlight-kd">extends Boss <span class=
highlight-o">{

}
</code></pre>

<h4 id="员工具体类">员工具体类</h4>
<pre><code class="language-java highlight-chroma">class</sp
n> StaffA extends
Staff {

}

class StaffB <span
lass="highlight-kd">extends Staff <span class=
highlight-o">{

}
</code></pre>

<h4 id="测试代码">测试代码</h4>
<pre><code class="language-java highlight-chroma">/** 初始
老板和员工 */
StaffA staffA <span c
ass="highlight-o">= new <span class="highligh
-n">StaffA("A 员工"<
span>);
StaffB staffB <span c
ass="highlight-o">= new <span class="highligh
-n">StaffB(" B 员工"<
span>);
Boss bossA <span cl
ss="highlight-o">= new <span class="highlight
n">BossA(staffA</s
an>);
Boss bossB <span cl
ss="highlight-o">= new <span class="highlight
n">BossB(staffB</sp
n>);

/** A 老板向 B 老板求支援 /

bossA.askHelp(bossB); // 打出：B 员工提供服务

/* B 员工向 A 老板求支援 */

staffB.askHelp(bossA); // 打印出：A 员工提供服务

</code></pre>

好像看起来实现了要求了，但是其实这段代码没有按照上面的 3 点规则编写，破坏了第 3 点规则，老板们的员工没有用员工的抽象类，破坏了细节依赖抽象这一点。设想一下，假如现在 A 老板把 A 员工辞退了，重新招了个 C 员工，那么怎么实现呢？是不是需要再新增一个 StaffC 类，然后再修改 BossA 类代码，把 StaffA 换成 StaffC。这样超级麻烦，在平时写项目中要时刻考虑这一点：在具体实现类使用其他类，是不是可以用其抽象类？
代码：
DIPErrorTest.java
<h3 id="正确的示范">正确的示范</h3>
看了上面那个憋屈的代码，再来看下面简洁的代码，才会发现依赖倒置原则是多么强大。
<h4 id="老板和员工抽象类">老板和员工抽象类</h4>
<pre><code class="language-java highlight-chroma">abstract class Boss2 {

}

abstract class Staff2 {

}

</code></pre>

<h4 id="老板类">老板类</h4>
<pre><code class="language-java highlight-chroma">class BossImpl extends Boss2 {

}

</code></pre>

<h4 id="员工类">员工类</h4>
<pre><code class="language-java highlight-chroma">class StaffImpl extends Staff2{

原文链接：依赖倒置原则

```
}
```

```
<h4 id="测试类">测试类</h4>
<pre><code class="language-java highlight-chroma"><span class="highlight-cm">/** 正确范 */</span>
<span class="highlight-n">Staff2</span> <span class="highlight-n">staffA2</span> <span class="highlight-o">=</span> <span class="highlight-k">new</span> <span class="highlight-n">StaffImpl</span><span class="highlight-o">(</span><span class="highlight-s">"A 工"</span><span class="highlight-o">);</span>
<span class="highlight-n">Staff2</span> <span class="highlight-n">staffB2</span> <span class="highlight-o">=</span> <span class="highlight-k">new</span> <span class="highlight-n">StaffImpl</span><span class="highlight-o">(</span><span class="highlight-s">"B 员 "</span><span class="highlight-o">);</span>
<span class="highlight-n">Boss2</span> <span class="highlight-n">bossA2</span> <span class="highlight-o">=</span> <span class="highlight-k">new</span> <span class="highlight-n">BossImpl</span><span class="highlight-o">(</span><span class="highlight-n">staffA2</span><span class="highlight-o">);</span>
<span class="highlight-n">Boss2</span> <span class="highlight-n">bossB2</span> <span class="highlight-o">=</span> <span class="highlight-k">new</span> <span class="highlight-n">BossImpl</span><span class="highlight-o">(</span><span class="highlight-n">staffB2</span><span class="highlight-o">);</span>

<span class="highlight-cm">/** A 老板向 B 老板求支援 /</span>
<span class="highlight-n">bossA2</span><span class="highlight-o">.</span><span class="highlight-na">askHelp</span><span class="highlight-o">(</span><span class="highlight-n">bossB2</span><span class="highlight-o">);</span> <span class="highlight-c1">// 打印出：B 员工提供服务
</span><span class="highlight-c1"></span>
<span class="highlight-cm">/* B 员工向 A 老板求支援 /</span>
<span class="highlight-n">staffB2</span><span class="highlight-o">.</span><span class="highlight-na">askHelp</span><span class="highlight-o">(</span><span class="highlight-n">bossA2</span><span class="highlight-o">);</span> <span class="highlight-c1">// 打印出：A 员工提供服务
</span><span class="highlight-c1"></span>
<span class="highlight-cm">/* A 老板辞退了 A 员工，换成了 C 员工 */</span>
<span class="highlight-n">Staff2</span> <span class="highlight-n">staffC2</span> <span class="highlight-o">=</span> <span class="highlight-k">new</span> <span class="highlight-n">StaffImpl</span><span class="highlight-o">(</span><span class="highlight-s">"C 员工"</span><span class="highlight-o">);</span>
<span class="highlight-n">bossA2</span><span class="highlight-o">.</span><span class="highlight-na">setStaff</span><span class="highlight-o">(</span><span class="highlight-n">staffC2</span><span class="highlight-o">);</span>

<span class="highlight-cm">/** B 员工向 A 老板求支援 */</span>
<span class="highlight-n">staffB2</span><span class="highlight-o">.</span><span class="highlight-na">askHelp</span><span class="highlight-o">(</span><span class="highlight-n">bossA2</span><span class="highlight-o">);</span> <span class="highlight-c1">// 打印出：C 员工提供服务
```

</code></pre>

这代码相比上面错误的示范，简洁了很多，实现的功能却更灵活，这就是依赖倒置原则强大的地，它可以将类的耦合性降低，提供灵活的处理。
代码：
DIPRightTest.java
<h3 id="最佳实践">最佳实践</h3>

变量的表面类型尽量是接口或者是抽象类
任何类都不应该从具体类派生
尽量不要覆写基类的方法
结合里氏替换原则使用
（来自《设计模式之禅》）

<h3 id="总结">总结</h3>
总的来说，要实现依赖倒置原则，要有『面向接口编程』这个思维，掌握好这个思维后，就可以好的运用依赖倒置原则。
参考资料：《大话设计模式》、《Java 设计模式》、《设计模式之禅》、《研磨设计模式》、《ead First 设计模式》
欢迎大家关注公众号 LieBrother，一起学习、进步！