

架构

业界先进的开源框架 JdonFramework

基于内存领域模型 事件驱动架构 WHAT HOW分离

◀ 上一主题



▶ 下一主题

Go 共有 8 回复 / 1 页

发新帖子 发表回复

banq



: 10787
: 2002 08 03

17:08



SOLID原则

2010 02 22 11:28

设计模式 对象责任职责协作 接口实现 DDD领域驱动设计

由 Robert Martin提出的S.O.L.I.D 原则，用来更好编写面向对象程序，更灵活应对变化。

4

顶一下

S - Single Responsibility Principle 单一职责，简称SRP

这个我前面几篇文章刚刚写：对象的责任与职责
如何从职责和协作中发现丰富对象？

比如：报表的内容和报表的格式都会变化改变，但是这两种变化的性质不同，一个是实质内在，一个是表面上的，SRP认为这是问题的两个方面，其实代表不同的职责，应该将它们分离放入不同的类或模块中，而不应该放在一起，否则的话，因为不同原因发生变化，导致对方变动，比如报表格式变新的样式，这个变化是不应该涉及到内容的。

这个模式和GoF模式中职责链模式Chain-of-responsibility pattern类似，体现了职责分离，分散关注Separation of concerns等OO思想。当然，也只有认识到事物高凝聚Cohesion本质，才能发现耦合。

O - Open/Closed Principle 开闭原则

我一直使用变化和不变来说明，封装不变部分，开放变化部分，一般使用接口继承实现方式来实现“开放”应对变化，说大白话就是：你不是要变化吗？，那么我就让你继承实现一个对象，用一个接口来抽象你的职责，你变化越多，继承实现的子类就越多。

L - Liskov Substitution Principle 里氏替换原则 简称LSP

这是一个针对行为职责可替代的原则，如果S是T的子类型，那么S对象就应该在不改变任何抽象属性情况下替换所有T对象。这里的抽象属性是指对象的字段属性。

我们使用接口时经常碰到一个问题，需要使用接口子类中的方法，而接口中没有这个方法，那么只能要么修改接口，要么将接口downcast为具体子类。为什么会出现这个尴尬现象？有几种情况导致，其中一种情况是是将当前的类重构到接口时，没有将类中所有方法extract到接口中，可能因为这些被你漏掉的方法不属于当前接口，那么，它又违背了单一职责原理，说明你当前这个类的方法设计得又不合理。

所以，如果单一职责设计的足够好，那么LSP原则则是检验的方法。LSP原则是对对象职责和协作的一种检验约束方法，此外还有DBC(design by contract)原则，为了保证实现接口的子类职责行为的约束，DBC三要素都必须被重视满足：

- 1.Preconditions前置条件不能在子类中被强化。
- 2.Postconditions后置条件不能在子类中被弱化。
- 3.子类自身不变性Invariants必须在子类自己中封装满足。这也是前面“不改变任何抽象属性”的意思。

I - Interface Segregation Principle 接口分离 简称ISP原则

这类似General Responsibility Assignment Software Patterns中高凝聚原则 High Cohesion Principle，这是解决胖接口，接口很大很丰富，就要进行解耦切分，把一个接口切分为多个接口，把一个大的职责切分为小职责以及这些职责之间的协作交互。

切分时必须依据高凝聚原则，单一职责进行切分。

这个原则起源于施乐公司，他们需要建立了一个新的打印机系统，可以执行诸如装订的印刷品一套，传真多种任务。此系统软件创建从底层开始编制，并实现了这些任务功能，但是不断增长的软件功能却使软件本身越来越难适应变化和维护。每一次改变，即使是最小的变化，有人可能需要近一个小时的重新编译和重新部署。这是几乎不可能再继续发展，所以他们聘请罗伯特Robert帮助他们。

他们首先设计了一个主要类Job,几乎能够用于实现所有任务功能。只要调用Job类的一个方法就可以实现一个功能，Job类就变动非常大，是一个胖模型啊，对于客户端如果只需要一个打印功能，但是其他无关打印的方法功能也和其耦合，ISP原则建议在客户端和Job类之间增加一个接口层，对于不同功能有不同接口，比如打印功能就是Print接口，然后将大的Job类切分为继承不同接口的子类，这样有一个Print Job类，等等。

D - Dependency Inversion Principle 依赖反转原则 DIP

a.高级别模块不应依赖于低层次的模块。双方应依赖于抽象。

	<p>b.抽象不应当取决于细节。细节应该依赖抽象。</p> <p>Dependency Injection 依赖注入模式也是属于这种类型变种，GoF设计模式中适配器模式中，高层次类只依赖于adapter接口，而被适配者低层次也只依赖adapter接口。</p> <p>SOLID原则如今在DCI架构中能够得到真正实现和发展。 [该贴被banq于2010-02-22 11:28修改过]</p>
<div>flycoder</div> <div><div>17:04</div><div>10 20</div><div>23人关注</div></div>	<div>SOLID原则2010 02 22 16:48</div> <div>DCI架构感觉总是有点难真正理解</div> <div>顶一下</div>
<div>banq</div> <div><div>17:08</div><div>03</div><div>23人关注</div></div>	<div>回复:SOLID原则2010 02 22 17:58</div> <div><div>2010年02月22日 16:48 "flycoder"的 言论</div><div>DCI架构感觉总是有点难真正理解</div><div>1</div><div>顶一下</div></div> <div>如果说SOLID原则指导了切分原则，那么DCI就实现这些细分碎片在运行时刻的组装。</div>
<div>weidagang2046</div> <div><div>09:04</div><div>01</div><div>23人关注</div></div>	<div>回复:SOLID原则2010 02 23 11:15</div> <div>本文对SRP的理解不太符合这条原则的本意；对LSP的理解部分是对的，部分有偏差。个人理解：SRP的要点在于类的抽象层次；LSP的要点在于行为抽象，文中讲的是“对象的字段属性”，而DbC“子类自身不变性Invariants必须在子类自己中封装满足”应该是“基类的Invaraints必须被子类所继承”。</div> <div>顶一下</div> <div>请参考这两篇文章： http://www.cnblogs.com/weidagang2046/archive/2010/01/31/1660482.html http://www.cnblogs.com/happyhippy/archive/2007/05/06/737040.html</div> <div>[该贴被weidagang2046于2010-02-23 11:17修改过]</div>
<div>banq</div> <div><div>17:08</div><div>03</div><div>23人关注</div></div>	<div>回复: 回复:SOLID原则2010 02 23 13:55</div> <div>多谢讨论，两篇文章推荐得很好。我认为SOLID原则属于对象职责范围的原则，也就是谈对象行为原则，也就是谈接口方面的事情，宁可用接口实现，不用或少用基类继承。</div> <div>顶一下</div> <div>对于SRP单一职责说起来很简单，真正展开很多很多。我文章中推荐的"对象设计：角色、责任和协作"(Object Design: Roles, Responsibilities, and Collaborations)"一书籍就是专门谈职责，如何实现单一职责的划分，所以，参看我的前两篇文章就可以，没有在SRP中说明。</div> <div>个人体会：如果真正掌握GOF设计模式，对于SOLID原则已经不言而喻了，23种设计模式处处体现SOLID原则，学习GOF设计模式是体会SOLID原则的最好体验方法。</div> <div>[该贴被banq于2010-02-23 13:56修改过]</div>
<div>ilove2009</div> <div><div>17:23</div><div>18</div><div>23人关注</div></div>	<div>SOLID原则2010 02 23 16:32</div> <div>这篇文章看起来不是很通俗易懂，有些地方感觉有点拗口。</div> <div>顶一下</div> <div>个人觉得理解SOLID并运用的好的话，自然会和23种模式靠拢，所以可以不用管23种模式。当然23种模式也是前人总结出来的精华，通过这些理解SOLID也不是不可以，但不是唯一的方式吧。主要的是时刻认识我们设计的目标是什么</div>

T

回复:SOLID原则

2010 02 23 17:24

2

2010 02 23 13:37

设计原则相比设计模式更加根本。设计原则强调设计价值，即遵循原则带来哪些好处；设计模式则可以看作是实现这些价值的惯用方法。

顶一下

xmuzyu



376

2007 03 26 12:16

7人关注

SOLID原则

2010 02 23 19:12

归根结底：高内聚，松耦合。

顶一下

T

回复:SOLID原则

2010 02 24 12:03

2

2010 02 23 13:37

2010年02月23日 19:12 "xmuzyu"的言论

归根结底：高内聚，松耦合。

这都还不够根本，因为"高内聚，松耦合"不是价值，而是方法。它背后的价值（之一）是降低维护成本。
[该贴被T于2010-02-24 12:03修改过]

设计模式(197)

对象责任职责协作(4)

接口实现(28)

DDD领域驱动设计(196)

共有 8 回复 / 1 页 Go


返回本主题 返回主题列表 返回面首 上一主题 下一主题


Demo Version - Winnovative Software Components


查询本论坛内 近一天 回复超过 10 的热门帖子 查询


标题

B I U









内容

提交时自动拷贝以上内容到剪贴板 Ctrl-V可取出；提问前请先查询[标签列表](#)

发 表

[使用帮助](#) [RSS](#)   [Google](#)  [MY YAHOO!](#)

解惑之道在J道，打造中国最具影响力的的软件架构社区 推荐FireFox或Chrome快速浏览本站
OpenSource **JIVEJDON** Powered by **JdonFramework** Code © 2002-09 **jdon.com**

Demo Version - Winnovative Software Components