College of Computer Science, Chongqing University

Software Engineering

A Practitioner's Approach Seventh Edition

14 构件级设计

zmqmail@cqu.edu.cn 13708390417

重庆大学 计算机学院 COLLEGE OF COMPUTER SCIENCE

面向对象编程

14.1 什么是构件?

- OMG unified modeling language specification [OMG01] defines
- A component as
  - "... a modular, deployable, and replaceable part of a system that encapsulates **implementation** and exposes a set of **interfaces**."
    系统中模块化的、可部署的和可替换的**部件**，该部件封装了实现并暴露一组**接口**

Component

implementation ⊕ **interfaces**

14.1 什么是构件?

- 面向对象的观点 （OO view）：
  - 类

    A component contains a set of **collaborating classes**

- 传统观点（Conventional view）：
  - 数据结构+处理逻辑

    A component contains **processing logic**, the internal **data structures** that are required to implement the processing logic, and an interface that enables the component to be invoked and data to be passed to it.

14.2.1 Basic Design Principles(7条、和其它有些资料的描述6条方式略有不同)

- **OCP**：开闭原则 The Open-Closed Principle.
  - A module [component] should be open for extension but closed for modification.
- **LSP**：Liskov替换原则 The Liskov Substitution Principle.
  - Subclasses should be substitutable for their base classes.
- **DIP**：依赖倒置原则 Dependency Inversion Principle.
  - Depend on abstractions. Do not depend on concretions
- **ISP**：接口分离原则 The Interface Segregation Principle.
  - Many client-specific interfaces are better than one general purpose interface.
- **REP**：发布复用等价性原则 The Release Reuse Equivalency Principle.
  - The granule of reuse is the granule of release.
- **CCP**：共同封装原则 The Common Closure Principle.
  - Classes that change together belong together.
- **CRP**：共同复用原则 The Common Reuse Principle.
  - Classes that aren’ t reused together should not be grouped together.

Design Pattern，DP

设计模式

6大设计原理

设计模式

六大设计原则

- Single Responsibility Principle（单一职责原则）
- Liskov Substitution Principle （里氏替换原则）
- Dependence Inversion Principle （依赖倒置原则）
- Interface Segregation Principle （接口隔离原则）
- Low Of Demeter （迪米特法则）
- Open Close Principle （开闭原则）

单一职责原则

Single Responsibility Principle

单一职责适用于接口、类、方法。
顾名思义，就是要求一个接口或类只有一个职责，它就负责一件事情。

- 定义
  - 单一职责原则,又称单一功能原则。
  - 它规定**一个类应该只有一个发生变化的原因**。
- 好处
  - 类的复杂性降低，有清晰明确的定义
  - 提高了可读性和可维护性
  - 使得变更引起的风险降低
- 注意
  - 职责没有一个量化的标准，并且受非常多因素的制约，现实中实现起来会有困难。

有且仅有一个原因引起类的变更

This is sometimes hard to see!
这个有时候很难说!

Liskov Substitution Principle

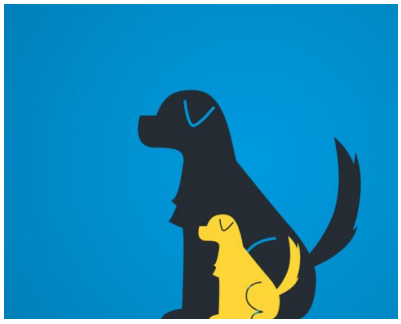
里氏替换原则

通俗点讲，只要父类能出现的地方子类就可以出现，而且替换为子类也不会产生任何错误或异常，使用者可能根本就不需要知道是父类还是子类。

- 定义
  - 所有引用基类的地方必须能**透明地使用其子类的对象**。
- 里氏替换原则的4层含义
  - 子类必须完全实现父类的方法
  - 子类可以有自己的个性
  - 覆盖或实现父类的方法时输入参数可被放大
  - 覆写或实现父类的方法时输出结果可被缩小

里氏替换原则的4层含义

- **子类必须完全实现父类的方法**
- 子类可以有自己的个性
- 覆盖或实现父类的方法时输入参数可被放大
- 覆写或实现父类的方法时输出结果可被缩小



3

Dependence Inversion Principle

依赖倒置原则

解耦

- 定义
  - 高层模块不应该依赖低层模块，两者都应该依赖其抽象
  - 抽象不应该依赖细节
  - 细节应该依赖抽象
- 好处
  - 采用依赖倒置原则可以减少类间的耦合性，提高系统的稳定性，降低并行开发引起的风险，提高代码的可读性和可维护性。

依赖倒置核心思想:面向接口编程

具体实践

- 实践方法
  - 每个类尽量都有接口或抽象类，或者抽象类和接口两者都具备
  - 变量的表面类型尽量是接口或者是抽象类（有些不必，如xxxUtil类等）
  - 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
  - 接口负责定义public属性和方法，并且声明与其他对象的依赖关系，抽象类负责公共构造部分的实现，实现类准确的实现业务逻辑，同时在适当的时候对父类进行细化。
  - “**面向接口编程**”就基本上抓住了依赖倒置原则的核心。
- 在Java语言中的表现
  - 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；
  - 接口或抽象类不依赖于实现类；
  - 实现类依赖接口或抽象类。

Interface Segregation Principle

接口隔离原则

接口隔离原则与单一职责的审视角度是不相同的，单一职责要求的是类和接口职责单一，注重的是职责，这是业务逻辑上的划分，而接口隔离原则要求接口的方法尽量少。

- 定义
  - 客户端不应该依赖它不需要的接口
  - 类间的依赖关系应该建立在**最小的接口**上
- 结构隔离原则的4层原则
  - 接口要尽量小
  - 接口要高内聚
  - 定制服务（单独为一个个体提供优良的服务）
  - 接口设计是有限度的（灵活设计接口粒度大小）

接口隔离原则的4层原则

- 接口要尽量小
- 接口要高内聚
- 定制服务（单独为一个个体提供优良的服务）
- 接口设计是有限度的（灵活设计接口粒度大小）

所有接口设计一定要注意适度

适度的“度”怎么来判断的呢？根据经验和常识判断！

5

Low Of Demeter

迪米特法则

一个类应该对自己需要耦合或调用的类知道得最少，你（被耦合或调用的类）的内部是如何复杂都和我没关系，那是你的事情，我就知道你提供的这么多public方法，我就调用这么多，其他的我一概不关心。

- 定义
  - 也称为最少知识原则（Least Knowledge Principle，LKP）：一个对象应该对其他对象有最少的了解。
- 迪米特法则的几层含义
  - 只和朋友交流
  - 朋友间也是有距离的
  - 是自己的就是自己的

6

Open Close Principle

开闭原则

开闭原则告诉我们应尽量通过**扩展软件实体的行为来实现变化**，而不是通过修改已有的代码来完成变化，它是为软件实体的未来事件而制定的对现行开发设计进行约束的一个原则。

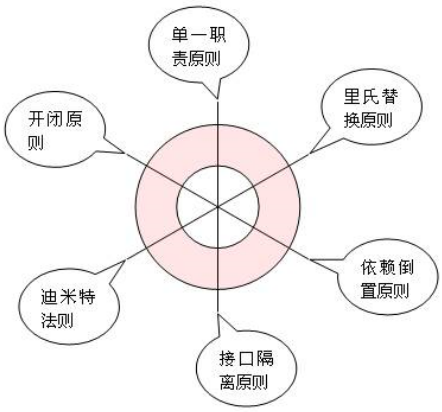
- 定义
  - 一个软件实体如类、模块和函数应该对扩展开放，对修改关闭。
- 开闭原则是最基础的一个原则。有以下好处：
  - 方便测试
  - 提高复用性
  - 提高可维护性
  - 符合面向对象开发技术

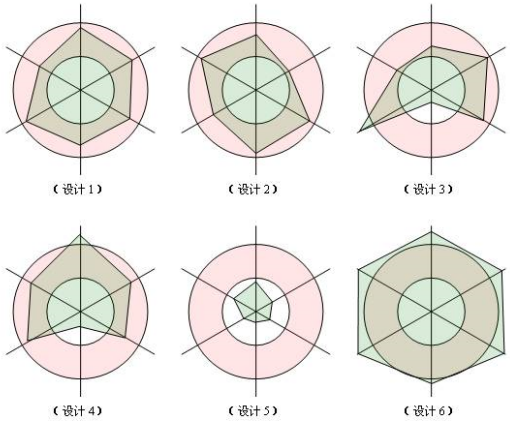
开闭原则并不意味不做任何的修改

开闭原则说是对扩展开放，对修改关闭，并不意味着不做任何的修改。

六大设计原则

- Single Responsibility Principle（单一职责原则）
- Liskov Substitution Principle （里氏替换原则）
- Dependence Inversion Principle （依赖倒置原则）
- Interface Segregation Principle （接口隔离原则）
- Low Of Demeter （迪米特法则）
- Open Close Principle （开闭原则）





回顾与实践

设计原则与设计模式

2000年，Robert C.Martin 在《设计原则和设计模式》论文中首次提出 SOLID 概念。Michael Feathers 对这些内容进行提炼并提出 SOLID 缩写。

《设计原则和设计模式》论文
fi.ort.edu.uy/innovaportal/file/2032/1/design_principles.pdf

1.单一职责（SRP）

“一个类应该有且只赋予一个职责”

每个类都应该赋予单一职责，而且这个职责应该完全封装在类中。引起类变化的因素永远不要超出一个。单一职责表示 SOLID 五项原则中的 “S”，在面向对象编程实践中意味着编写设计良好、可读性强、易于维护、易于升级和修改的代码。

实践总结

- 如果无法给 class 指定一个有意义类名，那么可能该类被赋予的职责过多。
- 每个 Web 应用中的对象都应赋予一个职责，所有对象 service 都应专注于该职责（SRP）。
- 如果某个 Java 类提供了多个功能，那么两个功能之间可能会产生耦合。即使只更改一个功能，也可能破坏耦合的关联功能。因此需要额外测试，以免对生产环境造成损害。

参阅《单一职责Java实例》了解更多内容。

《单一职责Java实例》
javaguides.net/2018/02/single-responsibility-principle.html

2. 开闭原则（OCP）

这是应用程序设计应当牢记的第二个重要原则。开闭原则指出：

“软件应该对扩展开放，但对修改关闭”。

简单地说，类、模块和函数等软件实体应该对扩展开放，对修改关闭。开闭原则表示 SOLID 五项原则中的“O”，在面向对象编程实践中意味着编写设计良好、可读性强、易于维护、易于升级和修改的代码。

实践总结

- 对扩展开放：class 设计在应对新需求时可以方便地加入新功能。
- 对修改关闭：除了修改 bug，否则不要改动已经设计好的类。
- 加入新功能时，设计和编码应该尽可能避免修改已有代码，最好不修改。
- 扩展已有功能时，应避免紧耦合：不要使用 if-else 或者 switch-case，请根据需要重构代码。
- 实现技术：继承、多态、泛型。
- 适用模式：策略模式、模板方法。

了解更多开闭原则相关内容，请参阅《开闭原则Java实例》中的类图、代码与最佳实践。

《开闭原则Java实例》
javaguides.net/2018/02/open-closed-principle.html

3. 里氏替换原则（LSP）

里式替换原则：

“派生类型必须可以完全替换其基类型。”

里式替换原则表示 SOLID 五项原则中的 “L”，在面向对象编程实践中意味着编写设计良好、可读性强、易于维护、易于升级和修改的代码。

实践总结

- 该原则适用于继承层次结构，是开闭原则的一种扩展。
- 这意味着派生出的新类必须保证不会改变基类的行为。基本上，派生类提供的功能只会比基类多。
- 如果派生类行为超出了基类客户端的预期，则违反了里氏替换原则。比如派生类抛出一个基类不会抛出的异常，或者派生类的功能有副作用。必须考虑客户端程序如何使用类继承结构。有时需要重构代码以纠正不符合里氏替换原则的部分。

了解更多里氏替换原则相关内容，请参阅《里氏替换原则Java实例》。

《里氏替换原则Java实例》
javaguides.net/2018/02/liskov-substitution-principle.html

4. 接口隔离原则（ISP）

该原则适用于接口，之前的单一职责原则适用于类。接口隔离原则：

> **“不应强迫客户实现对自身无用的方法”。**

SOLID 五项原则中的“I”表示接口隔离原则，它意味着应将较大的接口拆分为较小的接口。这样可以确保接口的实现类只需要关心它们感兴趣的方法。

了解更多接口隔离原则相关内容，请参阅《接口隔离原则Java实例》。

> 《接口隔离原则Java实例》
> javaguides.net/2018/02/liskov-substitution-principle.html

5. 依赖倒置原则（DIP）

依赖倒置原则：

> **高层模块不应依赖于底层模块。两者都应依赖其抽象。**
> **抽象不应当依赖细节。细节应当依赖于抽象。**

依赖倒置原则表示 SOLID 五项原则中的 “D”，在面向对象编程实践中意味着编写设计良好、可读性强、易于维护、易于升级和修改的代码。

实践总结

- 契约式设计。
- 设计中每个依赖项都应该是接口或抽象类。不应当依赖任何具体类。
- 工厂类与抽象工厂可以用作依赖框架，也有像 Spring IOC（控制反转容器）这样的专门框架。

了解更多依赖倒置原则相关内容，请参阅《依赖倒置原则Java实例》。

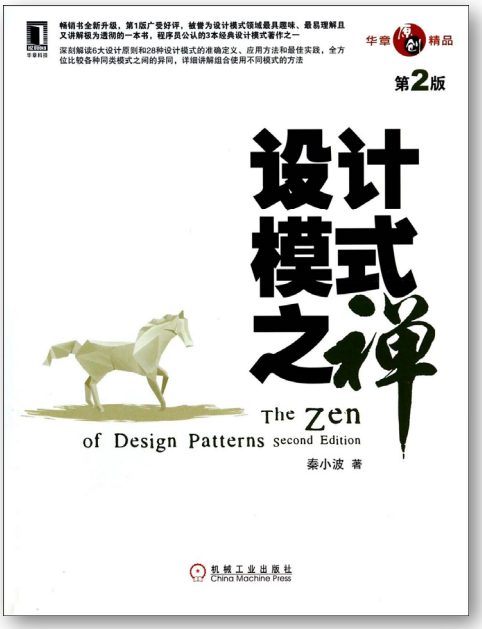
> 《依赖倒置原则Java实例》
> javaguides.net/2018/02/liskov-substitution-principle.html

24种模式与6大设计原则

自行学习

灵活运用，而非刻意遵循

架构设计基础知识整理
http://www.codeceo.com/article/architecture-design-basics.html


畅销书全新升级，第1版广受好评，被誉为设计模式领域最具趣味、最易理解且又讲解极为透彻的一本书，程序员公认的3本经典设计模式著作之一
深剖解读6大设计原则和28种设计模式的准确定义、应用方法和最佳实践，全方位比较各种同类模式之间的异同，详细讲解组合使用不同模式的方法
华章精品
第2版
设计模式之禅
The Zen of Design Patterns second Edition
秦小波 著
机械工业出版社
China Machine Press

Question?

Q&A?

College of Computer Science, Chongqing University

Software Engineering

A Practitioner's Approach Seventh Edition

14 构件级设计

zmqmail@cqu.edu.cn 13708390417

重庆大学 计算机学院 COLLEGE OF COMPUTER SCIENCE