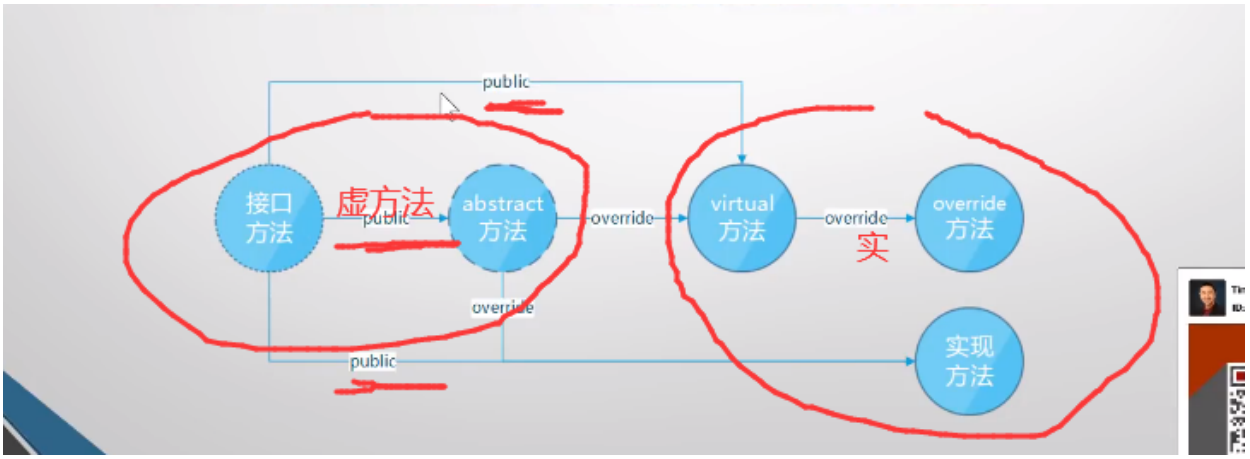
、单元测试是依赖倒置的开发中的直接应用



2 六大设计原则

H3 单一职责原则（SRP：Single Responsibility Principle）

单一职责原则，SRP又称单一功能原则，面向对象五个基本原则（SOLID）之一。它规

定一个类应该只有一个发生变化的原因。该原则由罗伯特·C·马丁（Robert C. Martin）

于《敏捷软件开发：原则、模式和实践》一书中给出的。

定义

一个模块只负责一件事。

一个类 只负责一件事 。

一个方法只负责一件事。

常见违背单一职责场景

在方法中出现多个分支，分别去执行各自的逻辑，功能虽然可以实现。

但如果需求变更，就会非常的不稳定。

如何遵守单一职责原则

定义抽象父类or父类虚方法

子类继承后重写，不同的实现

实例：不同动物的叫声处理，喊叫方法中多个分支，还是不同类不同喊叫方法。

单一职责的优缺点

优点

每个类相对简单，只负责自己的事情。

需求变更时，只修改变更类，其他类不受影响。

缺点

代码量会有所增加

解读代码成本增加

何时遵循？何时不遵守？

类型逻辑足够简单，方法足够少，可以不遵守

类型复杂，方法很多，一定要遵循单一职责原则

单一职责的不同层面以方法为单位

一个方法只负责一件事

方法中可以封装成一个新方法的，就封装成一个新方法

以类为单位

一个类只负责一件事

不属于该类的内容，创建新的类去封装

以模块/项目为单位

一个模块/项目只负责一件事

不属于该模块的内容，交由属于的模块去负责

H3 里氏替换原则（LSP：Liskov Substitution Principle）

里氏替换原则，LSP作为OO的高层原则，主张使用“抽象(Abstraction)”和“多态(Polymorphism)”将设计中的静态结构改为动态结构，维持设计的封闭性。“抽象”是语言

提供的功能。“多态”由继承语义实现。

定义

任何使用基类的地方，都可以安全的去使用其子类。

父类有的内容，子类必须有【类的强继承】

如果父类出现了子类不应该有的内容，那么就应该断开两个类的继承关系

然后，重新创建新的父类，包含子类该拥有的内容

子类必须有自己的行为和特征

父类已经实现的内容，子类不要再写【不要使用new关键词隐藏父类方法】

如果子类希望可以重写父类方法，父类方法用abstruct或virtual修饰

实例：游戏角色中的攻击方法，不能被不能攻击的子类继承。

H3 迪米特法则（LKP：Least Knowledge Principle ）

迪米特法则（Law of Demeter）又叫作最少知识原则（Least Knowledge Principle 简写
LKP），一个类对于其他类知道的越少越好，就是说一个对象应当对其他对象有尽可能
少的了解，只和朋友通信，不和陌生人说话。

定义

一个对象应该对其他对象保持 最少的了解 ，只与直接朋友进行通信。

类与类之间的关系：

纵向：继承关系

横向：聚合、组合、关联、 依赖 「出现在方法内部」

高内聚、低耦合

降低耦合度的方法

1. 少使用类的继承，多用接口隐藏实现的细节。
2. 模块的功能化分尽可能的单一，道理也很简单，功能单一的模块供其它模块调
用的机会就少。（其实这是高内聚的一种说法，高内聚低耦合一般同时出
现）。3. 遵循一个定义只在一个地方出现。
4. 少使用全局变量。

5. 类属性和方法的声明少用public，多用private关键字。
6. 多用设计模式，比如采用MVC的设计模式就可以降低界面与业务逻辑的耦合
度。
7. 尽量不用“硬编码”的方式写程序，同时也尽量避免直接用SQL语句操作数据
库。
8. 最后当然就是避免直接操作或调用其它模块或类（内容耦合）；如果模块间必
须存在耦合，原则上尽量使用数据耦合，少用控制耦合，限制公共耦合的范
围，避免使用内容耦合。
增强内聚度方法
1. 模块只对外暴露最小限度的接口，形成最低的依赖关系。
2. 只要对外接口不变，模块内部的修改，就不得影响其他模块。
3. 删除一个模块，应当只影响有依赖关系的其他模块，而不应该影响其他无关部
分。
通过降低访问修饰符权限，减少联系，减少耦合

实例：学校介绍->班级介绍->学生介绍，两两联系，学校不要与学生直接联系

H3 依赖倒置原则（DIP：Dependence Inversion Principle）

依赖倒置原则（Dependence Inversion Principle）是程序要依赖于抽象接口，不要依赖
于具体实现。简单的说就是要求对抽象进行编程，不要对实现进行编程，这样就降低了
客户与实现模块间的耦合。

定义

高层模块不应该依赖于低层模块，两者应该 依赖抽象 ，而不是依赖细节。

高层：方法调用方

底层：被调用方

面向抽象编程

属性、字段、方法参数、返回值，一切都尽量使用抽象【类/接口】

抽象不变，高层就不变

抽象一般是稳定的，低层的扩展变化不会影响到高层，低层就可以横向的自由扩
展，架构稳定

80%的设计模式跟抽象有关

抽象的好处

一个方法可以满足不同类型的参数传入

支持动态扩展，只要是实现了这个抽象，不需要修改上层

实例：学生类实现不同手机的使用方法，学生依赖手机，

新手机出现时，学生类也要更

新新方法。

不同的手机应该依赖抽象（手机），学生类也应该依赖于抽象（手机用户）

H3 接口隔离原则（ISP：Interface Segregation Principle）

客户端不应该依赖它不需要的接口。一个类对另一个类的依赖应该建立在最小的接口

上。定义：

使用多个专门的接口比使用单一的总接口要好，但也不建议一个接口只对应一个方

法。

一个类对另外一个类的依赖性应当是建立在 最小的接口上的。

一个接口代表一个角色，不应当将不同的角色都交给一个接口。没有关系的接口合

并在一起，形成一个臃肿的大接口，这是对角色和接口的污染。

接口的正确定义

既不能大而全，也建议不能一个接口一个方法

应该按照功能的密不可分来定义接口

应该是动态的，随业务变化而变化，设计的时候要留好提前量，避免抽象的变化

实例：手机的核心功能就是打电话和发短信，拍照、上网等其他功能的接口，不要被手

机所依赖。

参看.Net类中的接口的设计与实现

H3 开闭原则（OCP：Open Closed Principle）【总则】

在面向对象编程领域中，开闭原则规定“软件中的对象（类，模块，函数等等）应该对

于扩展是开放的，但是对于修改是封闭的”，这意味着一个实体是允许在不改变它的源

代码的前提下变更它的行为。该特性在产品化的环境中是特别有价值的，在这种环境

中，改变源代码需要代码审查，单元测试以及诸如此类的用以确保产品使用质量的过

程。遵循这种原则的代码在扩展时并不发生改变，因此无需上述的过程。

定义

对扩展开放，对修改关闭

扩展：添加新代码（类）

修改：修改原代码（类）

开闭原则是一个目标，没有任何手段，又被称为总则

为什么要遵循开闭原则

面向对象语言是静态语言，最害怕变化，因为会波及很多东西。

最理想的就是新增类，对原代码没有改动，原有代码才是可信的

遇到需求变更该怎么办呢？

直接修改现有方法 （最不可取）

增加方法 （稍好一些）

增加类 （那更好啦）

增加类库/框架 （那最好啦）