设计模式

设计模式的五项原则

1. 单一职责原则

   一个是避免相同的职责分散到不同的类中，另一个是避免一个类承担太多职责。减少类的耦合，提高类的复用性。

2. 接口隔离原则

   表明客户端不应该被强迫实现一些他们不会使用的接口，应该把大接口中的方法分组，然后用多个接口代替它，每个接口服务于一个子模块。

   简单说，就是使用多个专门的接口比使用单个接口好很多。

   该原则观点如下：
   1）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

   2）客户端程序不应该依赖它不需要的接口方法。

3. 开放封闭原则

   对扩展开放，对更改封闭

   不应该影响或大规模影响已有的程序模块。一句话概括：一个模块在扩展性方面应该是开放的而在更改性方面应该是封闭的。

   核心思想就是对抽象编程，而不对具体编程。

4. 替换原则

   子类型必须能够替换掉他们的父类型、并出现在父类能够出现的任何地方。

   主要针对继承的设计原则

   1）父类的方法都要在子类中实现或者重写，并且派生类只实现其抽象类中生命的方法，而不应当给出多余的,方法定义或实现。

   2）在客户端程序中只应该使用父类对象而不应当直接使用子类对象，这样可以实现运行期间绑定。

5. 依赖倒置原则

   高层模块（稳定）不应该依赖于底层模块（变化）---- 高层和底层都应该依赖于抽象

   抽象（稳定）不应该依赖于实现（变化）-- 实现应该依赖于抽象

   父类不能依赖子类，他们都要依赖抽象类。

1. Liskov替换

   子类必须能够替换他们的基类

2. 优先使用对象组合，而不是类继承
3. 使用封装创建对象之间的分解层

   一侧变化、一侧稳定
4. 针对接口编程，并非针对实现编程

设计模式

模式

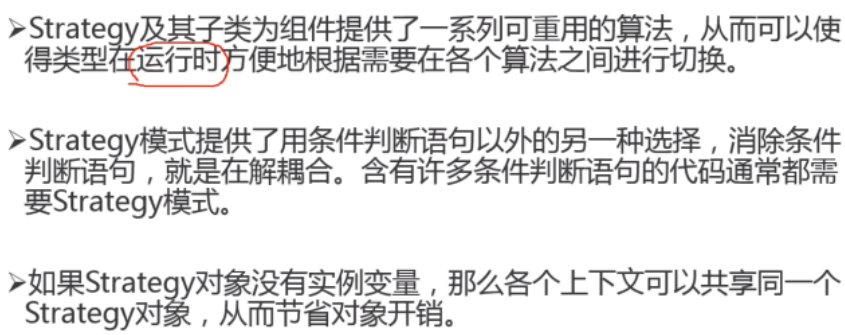
定义一系列算法，将其一个个独立封装，并且是他们可以互相替换，可以让算法独立于使用它的客户程序

1. ## 策略模式

   为子类提供可重用算法，可以让其在运行时能够自由切换（虚函数多态）

   提供了条件判断语句的另一种算法，满足开放封闭原则

   （大量的if else会产生很大的判断运行负担）

2. ## 工厂方法：

   避免具体依赖，解决接口选择问题，定义一个创建对象的接口，让其子类决定实例化哪一个工厂类

   通过对象创建绕开new，避免new产生的紧耦合

   将每个具体类创建一个单独的工厂类，继承基工程类（虚函数），调用函数返回new 工厂具体类。

   在主类中只创建一个工厂类，即使用多态new

3. ## 抽象工厂：

   一系列相互依赖的对象的创建工作

4. ## 单件模式

   在系统中只存在一个实例，用于解决一个全局类的频繁创建和销毁的问题

   拷贝构造函数、默认构造函数放置在private之中

   懒汉式（线程非安全版本：在public中使用static函数和static对象）

```
class singleton {
private:
    singleton() {}
    static singleton *p;
public:
    static singleton *instance();
};

singleton *singleton::p = nullptr;

singleton* singleton::instance() {
    if (p == nullptr)
        p = new singleton();
    return p;
```

```
}
```

饿汉式（线程安全）

```
class singleton {
private:
    singleton() {}
    static singleton *p;
public:
    static singleton *instance();
};

singleton *singleton::p = new singleton();
singleton* singleton::instance() {
    return p;
}
```

线程安全版本：加锁--开销大

```
//线程安全版本，但锁的代价过高
Singleton* Singleton::getInstance() {
    Lock lock;
    if (m_instance == nullptr) {
        m_instance = new Singleton();
    }
    return m_instance;
}
```

双检查锁

```
//双检查锁，但由于内存读写reorder不安全
Singleton* Singleton::getInstance() {

    if(m_instance==nullptr){
        Lock lock;
        if (m_instance == nullptr) {
            m_instance = new Singleton();
        }
    }
    return m_instance;
}
```

new的构造函数执行顺序（有可能被优化为）从：分配内存、构造函数、返回指针 改变为 分配内存、返回指针、构造函数。有可能访问到还没有调用构造函数的实例对象（加volatile）

5. 观察者模式

定义对象间一对多的依赖关系，当对象状态发生变化时，所有依赖于他的对象都会被通知并自动更新

6. 装饰器模式

对已经存在的某些类进行装饰，以此来扩展一些功能，从而动态的为一个对象增加新的功能。装饰器模式是一种用于代替继承的技术，无需通过继承增加子类就能扩展对象的新功能。使用对象的关联关系代替继承关系，更加灵活，同时避免类型体系的快速膨胀。