```
对接口编程而不是对实现编程
                                   面向对象设计原则
                                                           优先使用对象组合而不是继承
                                                             1、开闭原则(Open Close Principle)
                                                             开闭原则的意思是:对扩展开放,对修改关闭。在程序需要进行拓展的时候,不能去修改原有的代码,实现一个热插拔的
                                                             效果。简言之,是为了使程序的扩展性好,易于维护和升级。想要达到这样的效果,我们需要使用接口和抽象类
                                                             2、里氏代换原则(Liskov Substitution Principle)
                                                             任何基类可以出现的地方,子类一定可以出现。LSP 是继承复用的基石,只有当派生类可以替换掉基类,且软件单位的功
                                                             能不受到影响时,基类才能真正被复用,而派生类也能够在基类的基础上增加新的行为。实现开闭原则的关键步骤就是抽
                                                             象化,而基类与子类的继承关系就是抽象化的具体实现,所以里氏代换原则是对实现抽象化的具体步骤的规范
                                                             3、依赖倒转原则(Dependence Inversion Principle)
                                                             这个原则是开闭原则的基础,具体内容:针对接口编程,依赖于抽象而不依赖于具体。
                                   设计模式的六大原则
                                                             4、接口隔离原则(Interface Segregation Principle)
                                                             使用多个隔离的接口,比使用单个接口要好。它还有另外一个意思是:降低类之间的耦合度。由此可见,其实设计模式就
                                                             是从大型软件架构出发、便于升级和维护的软件设计思想,它强调降低依赖,降低耦合。
                                                             5、迪米特法则,又称最少知道原则(Demeter Principle)
                                                             最少知道原则是指:一个实体应当尽量少地与其他实体之间发生相互作用,使得系统功能模块相对独立。
                                                             6、合成复用原则(Composite Reuse Principle)
                                                             合成复用原则是指:尽量使用合成/聚合的方式,而不是使用继承
                                                            意图: 主要将数据结构与数据操作分离
                                                            主要解决: 稳定的数据结构和易变的操作耦合问题,解决功能映射
                                                            何时使用:需要对一个对象结构中的对象进行很多不同的并且不相关的操作,而需要避免让这些
                                                            操作"污染"这些对象的类
                                                            如何解决:在被访问的类里面加一个对外提供接待访问者的接口
                                                            关键代码:在数据基础类里面有一个方法接受访问者,将自身引用传入访问者
                                                            dynam_cast
                                   访问者模式(visitor)
                                                                    class B;
                                                                    class C;
                                                                     class A {
                                                                     public:
                                                                                                               class OutputVisitor : public A::IVisitor {
                                                                      class IVisitor {
                                                                                                                 virtual void visit(A *obj) {
                                                                      public:
                                                                                                                  cout << "this is a class A father" << endl;
                                                                        virtual void visit(A *) = 0;
                                                                        virtual void visit(B *) = 0;
                                                                                                                 //dynamic_cast:还原成完整类类型,通过虚函数表地址判断,所以原有基
                                                            code
                                                                                                                类一定是抽象类(virtual)
                                                                      //virtual void Accept(IVisitor *) = 0;
                                                                      virtual void Accept(IVisitor *vis) {
                                                                                                                 //有父类指针到子类指针根据不同类型做不同功能映射
                                                                                                                 //如果少加一个派生类的条件分支if-else就会出错
                                                                         vis->visit(this);
                                                                                                                 //visitor访问者模式,解决功能映射
                                                                      virtual \sim A() {}
                                                                                                                 OutputVisitor vis1;
                                                                                                                 for (int i = 0; i < 10; ++i) {
                                                                                                                  arr[i]->Accept(&vis1);
                                                                    class B : public A {
                                                                      void Accept(IVisitor *vis) {
                                                                        vis->visit(this);
                                                             意图:保证一个类仅有一个实例,并提供一个访问它的全局访问点
                                                             主要解决:一个全局使用的类频繁地创建与销毁
                                                             何时使用: 当您想控制实例数目,节省系统资源的时候
                                                            如何解决: 判断系统是否已经有这个单例,如果有则返回,如果没有则创建
                                                            关键代码:构造函数是私有的
                                                                     class HttpServer {
                                                                     public:
                                   单例模式(singleton)
                                                                       static HttpServer *getInstance() {
                                                                        //多线程有内存泄漏,加锁
                                                                         //双重验证,可以允许读,解决效率问题
                                                                        if (instance == nullptr) { //防止无意义的重复加锁
Design&pattern
                                                                          std::unique_lock<std::mutex> lock(m_mutex);
                                                                          if (instance == nullptr) { //防止内存泄露
                                                                                                                                          int main() {
                                                                            instance = new HttpServer();
                                                                                                                                            HttpServer h1;
                                                                                                                                            HttpServer h2(h1);
                                                                        return instance;
                                                            code
                                                                                                                                            HttpServer h3 = h1;
                                                                                                                                            HttpServer *t1 = HttpServer::getInstance();
                                                                     private:
                                                                                                                                            HttpServer *t2 = HttpServer::getInstance();
                                                                       static HttpServer *instance;
                                                                                                                                            cout << t1 << " " << t2 << endl;
                                                                       static std::mutex m_mutex;
                                                                                                                                            return 0;
                                                                       HttpServer() {}
                                                                       HttpServer(const HttpServer &) = delete;
                                                                       ~HttpServer() {}
                                                                     //HttpServer *HttpServer::instance = new HttpServer(); //恶汉模式,直接初始化
                                                                     HttpServer *HttpServer::instance = nullptr; //懒汉模式,加锁,判断,需要时在初始化
                                                                     std::mutex HttpServer::m_mutex;
                                                          意图:定义一个创建对象的接口,让其子类自己决定实例化哪一个工厂类,工厂模式使其创建过程延迟到子类进行
                                                          主要解决: 主要解决接口选择的问题
                                                          何时使用:我们明确地计划不同条件下创建不同实例时
                                   工厂模式(factory)
                                                          如何解决:让其子类实现工厂接口,返回的也是一个抽象的产品
                                                           关键代码: 创建过程在其子类执行
                                                                围绕一个超级工厂创建其他工厂。该超级工厂又称为其他工厂的工厂
                                                                意图:提供一个创建一系列相关或相互依赖对象的接口,而无需指定它们具体的类
                                                                主要解决: 主要解决接口选择的问题
                                                                何时使用:系统的产品有多于一个的产品族,而系统只消费其中某一族的产品。
                                                                如何解决:在一个产品族里面,定义多个产品。
                                                                                                           class BmwCar: public ICar {
                                                                         //交给工厂,能且只能有工厂创建对象
                                                                                                            BmwCar() {}
                                                                        class ICar {
                                   抽象工厂模式(abstract)
                                                                         public:
                                                                                                            class Factory : public ICar::IFactory {
                                                                          class IFactory {
                                                                          public:
                                                                                                                                                     ICar::IFactory *fac[3] = {new BenzCar::Factory(),
                                                                                                              virtual ICar *create() {
                                                                            virtual ICar *create() = 0;
                                                                                                               return new BmwCar();
                                                                                                                                                     new BmwCar::Factory(), new BmwCar::
                                                                                                                                                     Factory());
                                                                          virtual void runc() = 0;
                                                                         protected:
                                                                                                            virtual void runc() {
                                                                                                                                                     int main() {
                                                                          ICar() {}
                                                                                                              cout << "BmwCar run" << endl;</pre>
                                                                                                                                                       srand(time(0));
                                                                          ~ICar() {}
                                                                                                                                                       ICar *cars[10];
                                                                code
                                                                                                                                                       for (int i = 0; i < 10; ++i) {
                                                                        class BenzCar : public ICar {
                                                                                                           class AudiCar : public ICar {
                                                                                                                                                        cars[i] = fac[rand() % 3]->create();
                                                                          BenzCar() {}
                                                                                                            AudiCar() {}
                                                                         public:
                                                                                                                                                       for (int i = 0; i < 10; ++i) {
                                                                                                           public:
                                                                          class Factory : public ICar::IFactory {
                                                                                                            class Factory : public ICar::IFactory {
                                                                                                                                                        cars[i]->runc();
                                                                          public:
                                                                                                            public:
                                                                            virtual ICar *create() {
                                                                                                              virtual ICar *create() {
                                                                                                                                                       return 0;
                                                                              return new BenzCar();
                                                                                                               return new AudiCar();
                                                                          virtual void runc() {
                                                                                                            virtual void runc() {
                                                                            cout << "BenzCar run" << endl;
                                                                                                              cout << "AudiCar run" << endl;</pre>
                                                            为请求创建了一个接收者对象的链。这种模式给予请求的类型,对请求的发送者和接收者进行解耦
                                                            通常每个接收者都包含对另一个接收者的引用。如果一个对象不能处理该请求,那么它会把相同的
                                                            请求传给下一个接收者,依此类推。
                                                            意图:避免请求发送者与接收者耦合在一起,让多个对象都有可能接收请求,将这些对象连接成一条链,并且沿
                                                            着这条链传递请求,直到有对象处理它为止。
                                                            主要解决:职责链上的处理者负责处理请求,客户只需要将请求发送到职责链上即可,无须关心请求的处理细节和
                                                            请求的传递,所以职责链将请求的发送者和请求的处理者解耦了。
                                                            何时使用: 在处理消息的时候以过滤很多道。
                                                            如何解决: 拦截的类都实现统一接口
                                                                     class ICar {
                                                                                                                class ChainMaster {
                                                                     public:
                                                                                                                public:
                                                                       class IFactory {
                                   责任链模式(handle)
                                                                                                                  static ICar::IHandler *getInstance() {
                                                                                                                    if (head == nullptr) {
                                                                        virtual ICar *create() = 0;
                                                                                                                     buildChain();
                                                                       class IHandler {
                                                                                                                    return head;
                                                                       public:
                                                                        IHandler() : next(nullptr) {}
                                                                                                                private:
                                                                        virtual bool is_valid(int) = 0;
                                                                                                                  static ICar::IHandler *head;
                                                                        virtual ICar *do_it() = 0;
                                                                                                                  static void buildChain() {
                                                                        IHandler *next;
                                                                                                                    head = new BmwCar::Factory();
                                                                                                                    head->next = new AudiCar::Factory();
                                                                      virtual void runc() = 0;
                                                                                                                    head->next->next = new BenzCar::Factory();
                                                                     protected:
                                                                                                                    return;
                                                                       ICar() {}
                                                                       ~ICar() {}
                                                                                                                ICar::IHandler* ChainMaster::head = nullptr;
                                                            code
                                                                     class BmwCar: public ICar {
                                                                      BmwCar() {}
                                                                                                                 int main() {
                                                                                                                  srand(time(0));
                                                                      class Factory: public ICar::IFactory,
                                                                                                                  ICar *cars[10];
                                                                     public ICar::IHandler {
                                                                                                                  for (int i = 0; i < 10; ++i) {
                                                                       public:
                                                                                                                    int request = rand() % 3;
                                                                        virtual ICar *create() override {
                                                                                                                    for (auto p = ChainMaster::getInstance(); p; p = p->next) {
                                                                          return new BmwCar();
                                                                                                                     if (p->is_valid(request)) {
                                                                                                                       cars[i] = p->do_it();
                                                                        virtual bool is_valid(int x) override{
                                                                                                                       break;
                                                                          return x == 1;
                                                                        virtual ICar *do_it() override {
                                                                          return this->create();
                                                                                                                  for (int i = 0; i < 10; ++i) {
                                                                                                                   cars[i]->runc();
                                                                       virtual void runc() override {
                                                                                                                  return 0;
                                                                        cout << "BmwCar run" << endl;</pre>
```