设计模式汇总表

编号	设计模式名称（英文）	涉及的文件	何处功能/函数体现	备注说明
1				
2	**Command Pattern**	Commodity.h Commodity.cpp	CommoditySale::execute() CommodityDisplay::execute()	CommoditySale::execute()，商品售卖命令。 CommodityDisplay::execute()，商品展示命令（复合商品会依次展示其中包含的单件商品）。
3	**Flyweight Pattern**	CommodityFactory.h CommodityFactory.cpp	GetSingleCommodity(int ID, string name, int price, int shopID, int amount)	构造新商品
4	**Iterator Pattern**	Commodity.h Commodity.cpp	CompositeCommodityIterator::opteraor=(CompositeCommodityIterator&) CompositeCommodityIterator::opteraor==(CompositeCommodityIterator&) CompositeCommodityIterator::opteraor!(CompositeCommodityIterator&) CompositeCommodityIterator::operator++() CompositeCommodityIterator::operator*() CompositeCommodity::begin() CompositeCommodity::end()	CompositeCommodityIterator::opteraor=(CompositeCommodityIterator&)，迭代器赋值。 CompositeCommodityIterator::opteraor==(CompositeCommodityIterator&)，迭代器等于判断。 CompositeCommodityIterator::opteraor!=(CompositeCommodityIterator&)，迭代器不等于判断。 CompositeCommodityIterator::operator++()，迭代器自加。 CompositeCommodityIterator::operator*()，迭代器取值。

				CompositeCommodity::begin()，获取复合商品的第一个元素的迭代器。 CompositeCommodity::end()，获取复合商品的最后一个元素的迭代器。
5	**Decorator Pattern**	shop.h shop.cpp	class shopDecorator : public Shop {} class AddStarShopDecorator : public shopDecorator {} shopDecorator::shopDecorator():Shop(),decoratedShop(){} void shopDecorator::accept(){} AddStarShopDecorator::AddStarShopDecorator():shopDecorator(){} void AddStarShopDecorator::accept(){}	基类 shopDecorator 属于抽象装饰器类； 派生类 AddStarShopDecorator 属于具体装饰器类；
6	**Prototype Pattern**	shopInterface.h shopInterface.cpp ShoppingCart.h ShoppingCart.cpp Preferences.h Preferences.cpp	class shopCache{} void shopCache::loadCache() {} Shop *shopCache::cloneShop() {} class Preferences{} class ShoppingCart{}	实现商店缓存的类 shopCache； loadCache创建了多个不同类型的商店实例； cloneShop用于寻找商店进行复制； 实现用户偏好设置复制 Preferences 实现购物车内容复制 ShoppingCart
7	**Abstract Factory Pattern**	shopInterface.h	class AbstractFactory {} virtual Shop *createShop(string type) = 0;	定义抽象工厂类 AbstractFactory，用于创建Shop对象
8	**Factory Pattern**	shopInterface.h shopInterface.cpp PaymentMethod.h	class ShopFactory : public AbstractFactory {} Shop *createShop(string type); class PaymentMethod{} PaymentMethod* createPaymentMethod()	ShopFactory继承自 AbstractFactory createShop根据用户提供type类型生产对应店铺对象

		PaymentMethod.cpp		PaymentMethod支付抽象基类 createPaymentMethod产生具体的支付方式

9	**Null Object Pattern**	shop.h
		shop.cpp
		shopInterface.cpp
			class nullShop : public Shop{}
			void accept(Visitor &v);
			nullShop(...);
				nullShop继承自Shop类
				内部有accept函数以及nullShop构造函数

10	**Template Method**	shop.h	class restaurantShop : public Shop {}
			void accept(Visitor &v);
			restaurantShop(...);
				以restaurantShop为例
				继承自Shop类
				内部有accept函数以及restaurantShop构造函数

11	**Visitor Pattern**	shop.h	class Visitor {} class infoVisitor : public Visitor {} class filterVisitor : public Visitor {} void visit(Shop *shop)	Visitor是抽象访问者类； infoVisitor和filterVisitor是具体访问者； 两个具体访问者类中的visit()函数是实现访问操作的函数
12	**Filter Pattern**	shopFilter.h shopFilter.cpp	class ShopFilter {} class HighShopFilter : public ShopFilter {} class MidShopFilter : public ShopFilter {} class LowShopFilter : public ShopFilter {} list<Shop *> selectShop(list<Shop *> ShopList)	ShopFilter是抽象过滤器类； HighShopFilter、MidShopFilter和LowShopFilter是具体过滤器类； 三个具体过滤器类中的selectShop()函数是用来实现具体过滤操作的
13	**Proxy Pattern**	shop.h shop.cpp	class proxyShop : public Shop {} void proxyShop::accept(Visitor &v) void proxyShop::display()	proxyShop是代理对象类； accept()函数是用来接受访问者访问的函数； display()函数用于展示店铺信息
14	**State Pattern**	order.h order.cpp	class State{} void OrderInterface::pay() {}	ConcreteStateA和ConcreteStateB为具体

		OrderInterface.cpp OrderInterface.h	void Order::pay() {}	的状态类； OrderInterface::pay()和Order::pay()可以改变状态
15	**Bridge Pattern**	Commodity.h Commodity.cpp	CommodityInformationReader::getPrice() CommodityInformationReader::getName() CommodityInformationReader::getID() CommodityInformationReader::getType() CommodityInformationReader::getShop() CommodityInformationVipReader::getPrice() CommodityInformationVipReader::getName() CommodityInformationVipReader::getID() CommodityInformationVipReader::getType() CommodityInformationVipReader::getShop()	CommodityInformationReader::getPrice():int，获得商品价格。 CommodityInformationReader::getName():string，获得商品名称。 CommodityInformationReader::getID():int，获得商品ID。 CommodityInformationReader::getType():string，获得商品类型。 CommodityInformationReader::getShop():Shop*，获得商品的商铺。 CommodityInformationVipReader::getPrice():int，VIP用户获得商品价格（打八折）。 CommodityInformationVipReader::getName():string，VIP用户获得商品名称。 CommodityInformationVipReader::getID():int，VIP用户获得商品ID。 CommodityInformationVipReader::getType():string，VIP用户获得商品类型。 CommodityInformationVipReader::getShop():Shop*，VIP用户获得商品的商铺。
16	**Composite Pattern**	Commodity.h	CommodityInfomationSetter::setName(string)	CommodityInfomationSetter::setName(string)

		Commodity.cpp	CommodityInfomationSetter::setID(int) CommodityInfomationSetter::setType(string) CommodityInfomationSetter::setPrice(int) CommodityInfomationSetter::addCommodity(CommodityInformation*) CommodityInfomationSetter::removeCommodity(CommodityInformation*)	:void，设置商品名称。 CommodityInfomationSetter::setID(int):void，设置商品ID。 CommodityInfomationSetter::setType(string):void，设置商品类型。 CommodityInfomationSetter::setPrice(int):void，设置商品价格。 CommodityInfomationSetter::addCommodity(CommodityInformation*):bool，向组合商品中添加商品，如果调用者为单一商品类，则返回false。 CommodityInfomationSetter::removeCommodity(CommodityInformation*):bool，删除组合商品中的某件商品，如果调用者为单一商品类，则返回false。
17	**Adapter Pattern**	welcome.h	Class showUserCmdsAdapter	要将几个***Cmds类联系在一起，在设计中使用到了 `vector<String>` 的容器，使得能够存储各个类的调用信息，同时使用display函数能够展示具体的 `concreteCommands`
18	**Module Pattern**	welcome.h	class showCmds 及其子类	`showCmds` 是一个全局的命令提示类，其中的各个派生子类代表针对不同子系统的命令，每个函数用于输出对应的命令提示。