代理模式是**常用的结构型设计模式之一**，当无法直接访问某个对象或访问某个对象存在困难时可以通**过一个代理对象来间接访问**，为了保证客户端使用的透明性，**所访问的真实对象与代理对象需要实现相同的接口**。根据代理模式的使用目的不同，代理模式又可以分为多种类型，例如**远程代理、保护代理、虚拟代理、缓冲代理、智能引用代理**等，它们应用于不同的场合，满足用户的不同需求。



图15-1 商品代购示意图

W

在软件开发中，也有一种设计模式可以提供与代购网站类似的功能。由于某些原因，客户端不想或不能直接访问一个对象，此时可以通过一个称之为“代理”的第三者来实现间接访问，该方案对应的设计模式被称为代理模式。

       代理模式是一种应用很广泛的结构型设计模式，而且变化形式非常多，常见的代理形式包括远程代理、保护代理、虚拟代理、缓冲代理、智能引用代理等，后面将学习这些不同的代理形式。

代理模式定义如下：

**代理模式（Proxy Pattern）**：给某一个对象提供一个代理或占位符，并由代理对象来控制对原对象的访问。

代理模式是**一种对象结构型模式**。在代理模式中引入了一个新的代理对象，代理对象在客户端对象和目标对象之间起到中介的作用，它去掉客户不能看到的内容和服务或者增添客户需要的额外的新服务。

代理模式结构与实现

模式结构

代理模式的结构比较简单，其核心是代理类，为了让客户端能够一致性地对待真实对象和代理对象，在代理模式中引入了抽象层，代理模式结构如图15-2所示：

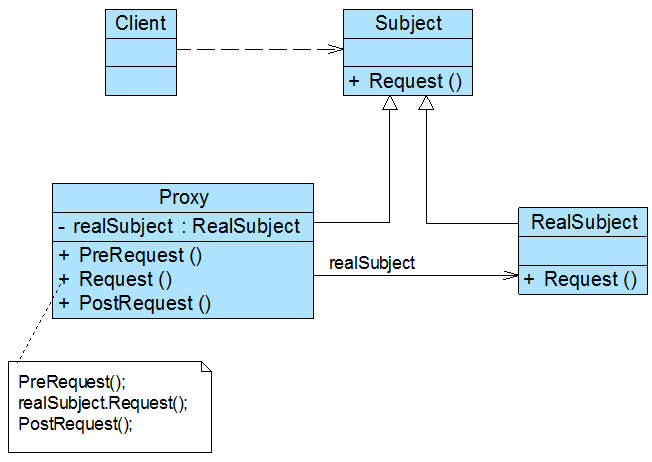


图15-2 代理模式结构图

由图15-2可知，代理模式包含如下三个角色：

       (1) **Subject（抽象主题角色）**：它声明了真实主题和代理主题的共同接口，这样一来在任何使用真实主题的地方都可以使用代理主题，客户端通常需要针对抽象主题角色进行编程。

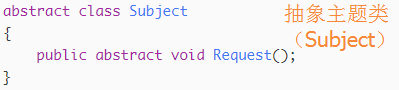
       (2) **Proxy（代理主题角色）**：它包含了对真实主题的引用，从而可以在任何时候操作真实主题对象；在代理主题角色中提供一个与真实主题角色相同的接口，以便在任何时候都可以替代真实主题；代理主题角色还可以控制对真实主题的使用，负责在需要的时候创建和删除真实主题对象，并对真实主题对象的使用加以约束。通常，在代理主题角色中，客户端在调用所引用的真实主题操作之前或之后还需要执行其他操作，而不仅仅是单纯调用真实主题对象中的操作。

       (3) **RealSubject（真实主题角色）**：它定义了代理角色所代表的真实对象，在真实主题角色中实现了真实的业务操作，客户端可以通过代理主题角色间接调用真实主题角色中定义的操作。

模式实现

代理模式的结构图比较简单，但是在**真实的使用和实现过程中要复杂很多，特别是代理类的设计和实现**。

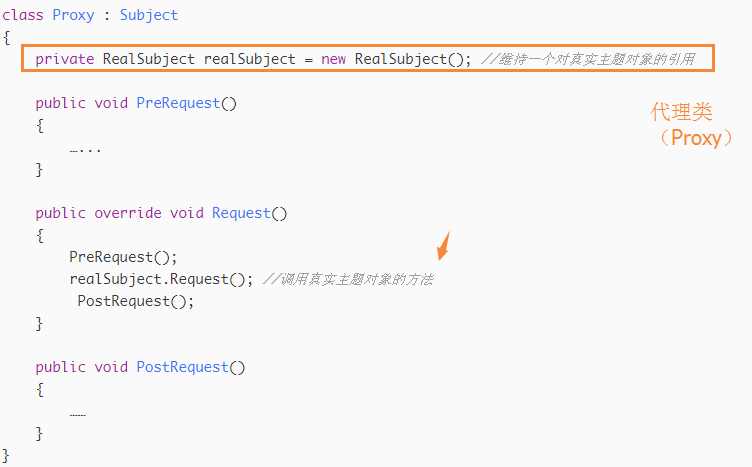
       抽象主题类声明了真实主题类和代理类的公共方法，它可以是接口、抽象类或具体类，客户端针对抽象主题类编程，一致性地对待真实主题和代理主题，典型的**抽象主题类**代码如下：



**真实主题类**继承了抽象主题类，提供了业务方法的具体实现，其典型代码如下：



**代理类**也是抽象主题类的子类，它维持一个对真实主题对象的引用，调用在真实主题中实现的业务方法，在调用时可以在原有业务方法的基础上附加一些新的方法来对功能进行扩充或约束，最简单的代理类实现代码如下：



在实际开发过程中，代理类的实现比上述代码要复杂很多，代理模式**根据其目的和实现方式不同可分为很多种类**，其中常用的几种代理模式简要说明如下：

(1) **远程代理(Remote Proxy)**：为一个位于不同的地址空间的对象提供一个本地的代理对象，这个不同的地址空间可以是在同一台主机中，也可是在另一台主机中，远程代理又称为大使(Ambassador)。

       (2) **虚拟代理(Virtual Proxy)**：如果需要创建一个资源消耗较大的对象，先创建一个消耗相对较小的对象来表示，真实对象只在需要时才会被真正创建。

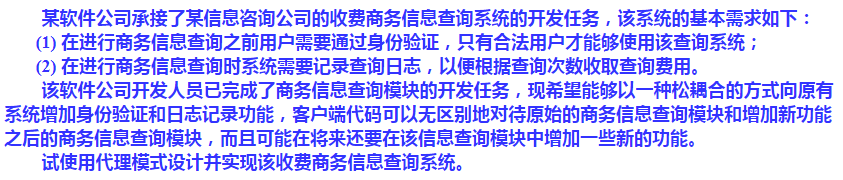
       (3) **保护代理(Protect Proxy)**：控制对一个对象的访问，可以给不同的用户提供不同级别的使用权限。

       (4) **缓冲代理(Cache Proxy)**：为某一个目标操作的结果提供临时的存储空间，以便多个客户端可以共享这些结果。

       (5) **智能引用代理(Smart Reference Proxy)**：当一个对象被引用时，提供一些额外的操作，例如将对象被调用的次数记录下来等。

在这些常用的代理模式中，有些代理类的设计非常复杂，例如远程代理类，它封装了底层网络通信和对远程对象的调用，其实现较为复杂。

**实例说明**



**实例分析及类图**

通过分析，可以采用一种间接访问的方式来实现该商务信息查询系统的设计，在**客户端对象**和**信息查询对象**之间增加一个代理对象，让代理对象来实现身份验证和日志记录等功能，而无须直接对原有的商务信息查询对象进行修改，如图15-3所示：

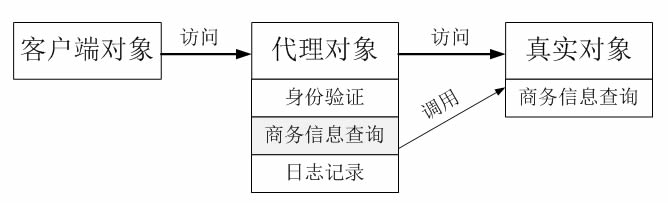


图15-3 商务信息查询系统设计方案示意图

在图15-3中，客户端对象通过代理对象间接访问具有商务信息查询功能的真实对象，在代理对象中除了调用真实对象的商务信息查询功能外，**还增加了**身份验证和日志记录等功能。使用代理模式设计该商务信息查询系统，结构图如图15-4所示。

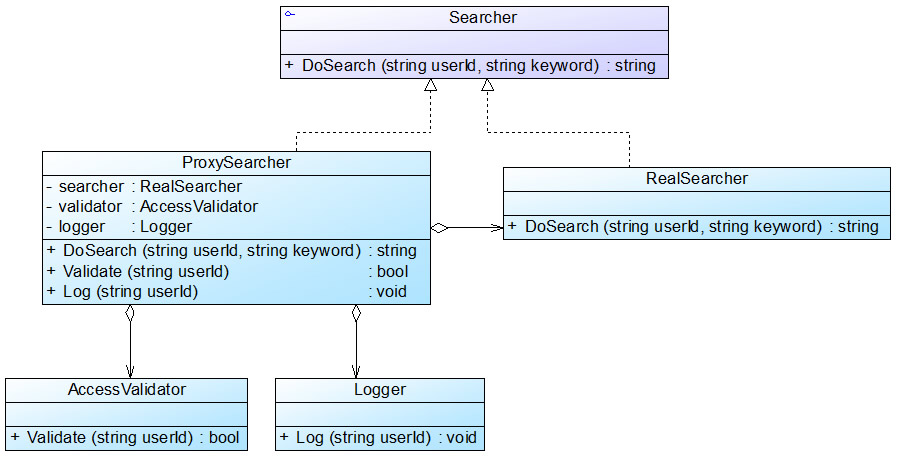


图15-4 商务信息查询系统结构图

Proxy（代理类）：ProxySearcher

Subject（抽象主题类）：Searcher

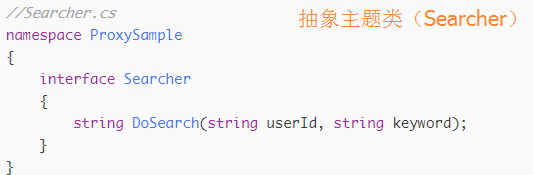
RealSubject（真实主题类）：RealSearcher

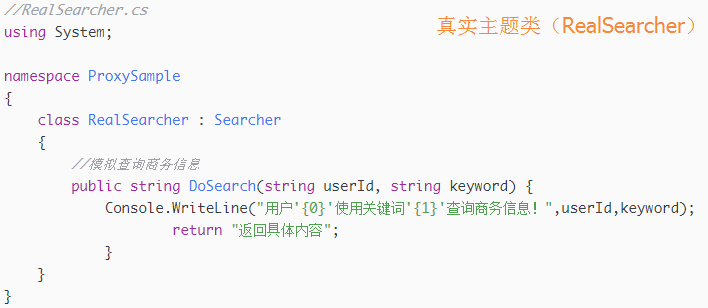
业务类：AccessValidator（验证用户身份）、Logger（记录用户查询日志）

**实例代码**







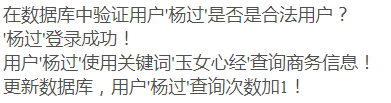






**结果及分析**

编译并运行程序，输出结果如下：



本实例是**保护代理**和**智能引用代理**的应用实例，在代理类ProxySearcher中**实现对真实主题类的权限控制和引用计数**，如果需要在访问真实主题时增加新的访问控制机制和新功能，只需增加一个新的代理类，再修改配置文件，在客户端代码中使用新增代理类即可，源代码无须修改，符合开闭原则。

1.模式优点

共同优点如下：

(1) 能够协调调用者和被调用者，在一定程度上降低了系统的耦合度。

(2) 客户端可以针对抽象主题角色进行编程，增加和更换代理类无须修改源代码，符合开闭原则，系统具有较好的灵活性和可扩展性。

2.模式缺点

(1) 由于在客户端和真实主题之间增加了代理对象，因此有些类型的代理模式可能会造成请求的处理速度变慢，例如保护代理。

       (2) 实现代理模式需要额外的工作，而且有些代理模式的实现过程较为复杂，例如远程代理。

3.模式适用场景

(1) 当客户端对象需要访问远程主机中的对象时可以使用**远程代理**。

       (2) 当需要用一个消耗资源较少的对象来代表一个消耗资源较多的对象，从而降低系统开销、缩短运行时间时可以使用**虚拟代理**，例如一个对象需要很长时间才能完成加载时。

       (3) 当需要为某一个被频繁访问的操作结果提供一个临时存储空间，以供多个客户端共享访问这些结果时可以使用缓冲代理。通过使用**缓冲代理**，系统无须在客户端每一次访问时都重新执行操作，只需直接从临时缓冲区获取操作结果即可。

       (4) 当需要控制对一个对象的访问，为不同用户提供不同级别的访问权限时可以使用**保护代理**。

       (5) 当需要为一个对象的访问（引用）提供一些额外的操作时可以使用**智能引用代理**。

[代理模式](https://blog.csdn.net/LoveLion/article/details/17517213)

[代理模式（一）：代理模式概述，代理模式结构与实现](https://blog.csdn.net/lovelion/article/details/8227953)

[代理模式（二）：代理模式应用实例（收费商务信息查询系统）](https://blog.csdn.net/lovelion/article/details/8228042)

[代理模式（三）：远程代理，虚拟代理，缓冲代理](https://blog.csdn.net/lovelion/article/details/8228132)

[代理模式（四）：代理模式效果与适用场景](https://blog.csdn.net/lovelion/article/details/8228156)