# Proxy Pattern

**Motivation**

Sometimes we need the ability to control the access to an object. For example if we need to use only a few methods of some costly objects we'll initialize those objects when we need them entirely. Until that point we can use some light objects exposing the same interface as the heavy objects. These light objects are called proxies and they will instantiate those heavy objects when they are really need and by then we'll use some light objects instead.

This ability to control the access to an object can be required for a variety of reasons: controlling when a costly object needs to be instantiated and initialized, giving different access rights to an object, as well as providing a sophisticated means of accessing and referencing objects running in other processes, on other machines.

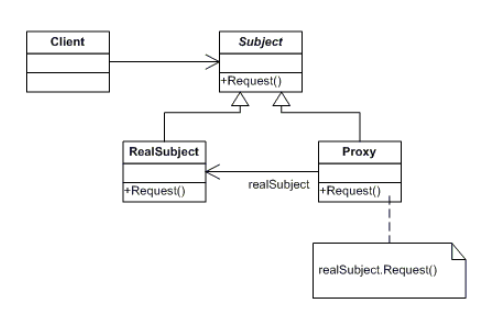
Consider for example an image viewer program. An image viewer program must be able to list and display high resolution photo objects that are in a folder, but how often do someone open a folder and view all the images inside. Sometimes you will be looking for a particular photo, sometimes you will only want to see an image name. The image viewer must be able to list all photo objects, but the photo objects must not be loaded into memory until they are required to be rendered.

**Intent**

The Proxy design pattern provides a surrogate or placeholder for another object to control access to it.

**代理模式**是一种结构型设计模式， 让你能够提供对象的替代品或其占位符。 代理控制着对于原对象的访问， 并允许在将请求提交给对象前后进行一些处理。

代理是一种结构型设计模式， 让你能提供真实服务对象的替代品给客户端使用。 代理接收客户端的请求并进行一些处理 （访问控制和缓存等）， 然后再将请求传递给服务对象。



* Proxy
  + maintains a reference that lets the proxy access the real subject. Proxy may refer to a Subject if the RealSubject and Subject interfaces are the same.
  + provides an interface identical to Subject's so that a proxy can be substituted for for the real subject.
  + controls access to the real subject and may be responsible for creating and deleting it.
  + other responsibilites depend on the kind of proxy:
    - *remote proxies* are responsible for encoding a request and its arguments and for sending the encoded request to the real subject in a different address space.
    - *virtual proxies* may cache additional information about the real subject so that they can postpone accessing it. For example, the ImageProxy from the Motivation caches the real images's extent.
    - *protection proxies* check that the caller has the access permissions required to perform a request.
* Subject
  + defines the common interface for RealSubject and Proxy so that a Proxy can be used anywhere a RealSubject is expected.
* RealSubject
  + defines the real object that the proxy represents.

**场景：**

在需要用比较通用和复杂的对象指针代替简单的指针的时候，使用 Proxy 模式。

下面是一些可以使用 Proxy 模式常见情况：

* 远程代理（Remote Proxy ）为一个对象在不同的地址空间提供局部代表。

**本地执行远程服务 （远程代理）。 适用于服务对象位于远程服务器上的情形。**

 在这种情形中， 代理通过网络传递客户端请求， 负责处理所有与网络相关的复杂细节。

* 虚代理（Virtual Proxy ）根据需要创建开销很大的对象。

**延迟初始化 （虚拟代理）。 如果你有一个偶尔使用的重量级服务对象， 一直保持该对象运行会消耗系统资源时， 可使用代理模式。**

你无需在程序启动时就创建该对象， 可将对象的初始化延迟到真正有需要的时候。

* 保护代理（Protection Proxy ）控制对原始对象的访问。保护代理用于对象应该有不同的访问权限的时候。

**访问控制 （保护代理）。 如果你只希望特定客户端使用服务对象， 这里的对象可以是操作系统中非常重要的部分， 而客户端则是各种已启动的程序 （包括恶意程序）， 此时可使用代理模式。**

代理可仅在客户端凭据满足要求时将请求传递给服务对象。

* 智能指引（Smart Reference ）取代了简单的指针，它在访问对象时执行一些附加操作。

**智能引用。  可在没有客户端使用某个重量级对象时立即销毁该对象。**

代理会将所有获取了指向服务对象或其结果的客户端记录在案。 代理会时不时地遍历各个客户端， 检查它们是否仍在运行。 如果相应的客户端列表为空， 代理就会销毁该服务对象， 释放底层系统资源。

代理还可以记录客户端是否修改了服务对象。 其他客户端还可以复用未修改的对象。

* **记录日志请求 （日志记录代理）。 适用于当你需要保存对于服务对象的请求历史记录时。 代理可以在向服务传递请求前进行记录。**

缓存请求结果 （缓存代理）。 适用于需要缓存客户请求结果并对缓存生命周期进行管理时， 特别是当返回结果的体积非常大时。

代理可对重复请求所需的相同结果进行缓存， 还可使用请求参数作为索引缓存的键值。

**它的典型用途包括：**

*  对指向实际对象的引用计数，这样当该对象没有引用时，可以自动释放它。
*  当第一次引用一个持久对象时，将它装入内存。
*  在访问一个实际对象前，检查是否已经锁定了它，以确保其他对象不能改变它。

**与其他模式的关系**

* [**适配器模式**](https://refactoringguru.cn/design-patterns/adapter)能为被封装对象提供不同的接口， [**代理模式**](https://refactoringguru.cn/design-patterns/proxy)能为对象提供相同的接口， [**装饰模式**](https://refactoringguru.cn/design-patterns/decorator)则能为对象提供加强的接口。
* [**外观模式**](https://refactoringguru.cn/design-patterns/facade)与[**代理**](https://refactoringguru.cn/design-patterns/proxy)的相似之处在于它们都缓存了一个复杂实体并自行对其进行初始化。 代理与其服务对象遵循同一接口， 使得自己和服务对象可以互换， 在这一点上它与外观不同。
* [**装饰**](https://refactoringguru.cn/design-patterns/decorator)和[**代理**](https://refactoringguru.cn/design-patterns/proxy)有着相似的结构， 但是其意图却非常不同。 这两个模式的构建都基于组合原则， 也就是说一个对象应该将部分工作委派给另一个对象。 两者之间的不同之处在于代理通常自行管理其服务对象的生命周期， 而装饰的生成则总是由客户端进行控制。