Nhóm 6:Tìm hiểu về Proxy

**1. Giới thiệu**

* Proxy (hay còn gọi là Surrogate) là một mẫu thiết kế thuộc nhóm cấu trúc (Structural Pattern).
* Điều khiển gián tiếp việc truy xuất đối tượng thông qua một đối tượng được ủy nhiệm
* Cung cấp 1 class đại diện để quản lí sự truy xuất đến thành phần của 1 class khác
* Giải quyết vấn đề security, perfomance, validation,…
* Tần suất sử dụng: khá cao

## 2. Mục đích ra đời

### Problem

Giả sử ta có một bài toán truy cập vào 1 object lớn. Object này chiếm nhiều tài nguyên hệ thống. Ta cần nó thường xuyên, nhưng không phải luôn luôn. Ví dụ như khi ta truy vấn cơ sở dữ liệu.

Ta có thể implement lazy initialization, tức là chỉ tạo khi cần. Khi đó client muốn truy cập đều phải chạy qua đoạn code này, tuy nhiên vấn đề phát sinh là sẽ khiến code duplicateText

Description automatically generated

Điều hay nhất là có thể là đưa dòng code này vào chính đối tượng đó. Nhưng nếu lớp này là 3rd party thì không thể.

Một vấn đề khác về mặt security, hoặc ta muốn validate nó mà không cần đến client, như khi upload 1 file nào đó.

### Solution

Proxy nói rằng ta cần tạo 1 lớp mới đại diện cho lớp service đang có với cùng 1 interface, lớp này gọi là proxy. Sau đó khi update ứng dụng thì nó sẽ truyền đối tượng proxy cho tất cả client phía đối tượng gốc. Khi nhận 1 yêu cầu từ phía client, proxy tạo 1 service thật và delegate tất cả nhiệm vụ đến nó.

A picture containing text

Description automatically generated

Nếu phải chạy thứ gì đó trước hay sau logic chính của lớp, proxy cho phép làm điều đó mà không làm thay đổi lớp đó. Bởi vì Proxy implement cùng interface với lớp chính, nó có thể kết nối với bất cứ client nào đang chờ liên lạc từ server thật.

Trả lời hai câu này để quyết định cài proxy hay không:

* Có 1 “expensive resource” nhưng chỉ khởi tạo khi cần đến nó?
* Có 1 resource cần kiểm tra security, availability hay validation mà không muốn làm ở phía Client?

=> Wrap 1 object và cung cấp các phương thức để truy cập vào object đó

Proxy giải quyết các vấn đề:

* Có những tình huống mà khách hàng không hoặc không thể tham chiếu trực tiếp đến một Đối tượng, nhưng vẫn muốn tương tác với đối tượng.
* Đối tượng proxy có thể hoạt động như trung gian giữa máy khách và đối tượng đích.
* Đối tượng proxy có cùng giao diện với đối tượng đích.
* Proxy giữ một tham chiếu đến đối tượng mục tiêu và có thể chuyển tiếp các yêu cầu đến mục tiêu theo yêu cầu
* Proxy hữu ích ở bất cứ nơi nào có nhu cầu tham chiếu phức tạp hơn đến một đối tượng hơn là con trỏ đơn giản hoặc tham chiếu đơn giản có thể cungười cấp

## 3. Kiến trúc

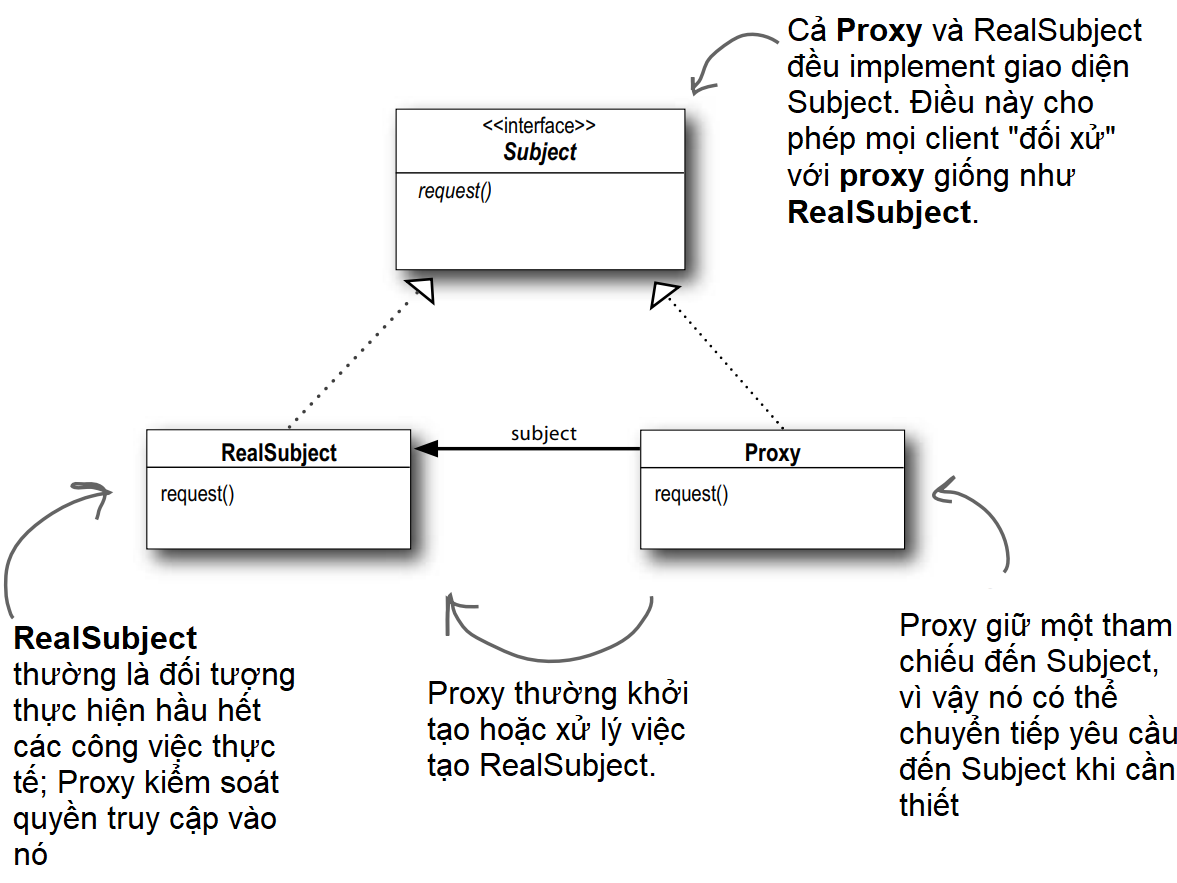
Diagram

Description automatically generated

Các thành phần trong mô hình:

* ServiceInterface: Định nghĩa giao diện chung cho Service và Proxy để Proxy có thể được sử dụng bất kỳ nơi nào mà Service được mong đợi.
* Service: Định nghĩa ServiceInterface mà Proxy đại diện.
* Proxy:
  + Duy trì một tham chiếu cho phép Proxy truy cập vào Service . Proxy có thể tham chiếu đến ServiceInterface nếu Service và ServiceInterface giống nhau.
  + Cung cấp interfaces giống với ServiceInterface để Proxy có thể thay thế cho Service .
  + Kiểm soát quyền truy cập vào Service và chịu trách nhiệm cho việc tạo và xóa nó.
  + Một vài trách nhiệm khác phụ thuộc vào loại Proxy:
  + Remote proxies chịu trách nhiệm mã hóa một yêu cầu và các đối số của nó và gửi yêu cầu được mã hóa đến Service trong một không gian địa chỉ khác.
  + Virtual proxies có thể cache thông tin bổ xung về Service để trì hoãn việc truy cập nó.
  + Protection proxies kiểm tra xem có quyền truy cập cần thiết để thực hiện yêu cầu hay không.

Để chi tiết hơn, ta có sơ đồ Proxy Pattern



Đầu tiên chúng ta có một Subject, cung cấp giao diện cho RealSubject và Proxy.

Bằng cách triển khai cùng một giao diện, Proxy có thể được thay thế cho RealSubject ở bất cứ nơi nào nó xuất hiện.

RealSubject là đối tượng thực hiện công việc thực sự. Nó yêu thích đối tượng mà Proxy đại diện và kiểm soát quyền truy cập.

Proxy giữ một tham chiếu đến RealSubject. Trong một số trường hợp, Proxy có thể chịu trách nhiệm tạo và hủy RealSubject. Client tương tác với RealSubject thông qua Proxy. Vì Proxy và RealSubject implements cùng một giao diện (Subject), Proxy có thể được thay thế ở bất kỳ nơi nào có thể sử dụng subject. Proxy cũng kiểm soát quyền truy cập vào RealSubject; điều khiển này có thể cần thiết nếu Subject đang chạy trên một máy từ xa, nếu Subject tốn kém để tạo theo một cách nào đó hoặc nếu quyền truy cập vào Subject cần được bảo vệ theo một cách nào đó.

## 4. Ưu & nhược điểm

#### Ưu điểm

* Open/Closed Principle: Bạn có thể thêm proxy mới mà không cần thay đổi service hoặc clients.
* Cải thiện Performance thông qua lazy loading.
* Nó cung cấp sự bảo vệ cho đối tượng thực từ thế giới bên ngoài.
* Giảm chi phí khi có nhiều truy cập vào đối tượng có chi phí khởi tạo ban đầu lớn.

#### Nhược điểm

* Mã có thể trở nên phức tạp hơn vì bạn cần phải thêm lớp mới.
* Phản hồi từ service có thể bị trì hoãn.

## 5. Khi nào thì sử dụng

Dưới đây chúng ta có thể liệt kê một số trường hợp mà khi gặp sẽ phải cân nhắc sử dụng Proxy pattern:

* Lazy initialization (virtual proxy): Khi bạn có một đối tượng dịch vụ nặng gây lãng phí tài nguyên hệ thống do luôn hoạt động, mặc dù thỉnh thoảng bạn chỉ cần nó.
* Access control (protection proxy): Khi bạn muốn chỉ những khách hàng cụ thể mới có thể sử dụng đối tượng dịch vụ.
* Local execution of a remote service (remote proxy): Đây là khi đối tượng service được đặt trên một máy chủ từ xa.
* Logging requests (logging proxy): Khi bạn muốn giữ lịch sử của các yêu cầu đối với đối tượng service.
* Caching request results (caching proxy): Khi bạn cần lưu trữ kết quả của các yêu cầu máy khách và quản lý vòng đời của bộ nhớ cache này, đặc biệt nếu kết quả khá lớn.
* Smart reference: Khi bạn cần loại bỏ một đối tượng nặng khi không có máy khách nào sử dụng nó.

## 6. Source code minh họa với C#

* B1: Nếu không có service interface, tạo 1 cái để proxy và service có thể thay thế nhau
* B2: Tạo 1 lớp proxy. Nó nên có 1 trường lưu tham chiếu đến service. Thường proxy tạo và quản lí life cycle của service. Đôi khi service được truyền đến proxy thông qua 1 constructor từ client.
* B3: Implement proxy => có thể delegate công việc tới service
* B4: Cân nhắc tạo 1 phương thức static để quyết định xem client gọi proxy hay service.
* B5: Cài đặt thêm lazy initialization (nếu muốn)

public interface Video

{

void display();

}

public class RealVideo : Video

{

private string \_fileName;

public RealVideo(string fileName)

{

\_fileName = fileName;

loadFromDisk(\_fileName);

}

public void display()

{

Console.WriteLine("Displaying " + \_fileName);

}

private void loadFromDisk(string fileName)

{

Console.WriteLine("Loading " + fileName);

}

}

public class ProxyVideo : Video

{

private RealVideo realVideo;

private string fileName;

public ProxyVideo(string fileName)

{

this.fileName = fileName;

}

public void display()

{

if (realVideo == null) // lazy loading

{

realVideo = new RealVideo(fileName);

}

realVideo.display();

}

}

class Client

{

static void Main(string[] args)

{

Video image = new ProxyVideo("Design Pattern.mp4");

//video sẽ được load từ ổ đĩa

image.display();

Console.WriteLine("After loading first time\n");

//video sẽ k tải lại từ ổ đĩa nữa

image.display();

}

}

**7. Design Pattern liên quan**

* *Adapter:* Adapter cung cấp một Interface khác với Interface của object mà chúng adapt, Proxy thì ngược lại, cung cấp cùng một Interface cho subject của nó. Proxy được sử dụng để bảo vệ truy cập và sẽ từ chối thể hiện hành động mà subject thực hiện.
* *Decorator:* Decorator có cách triển khai khá giống với Proxy (dựa trên Composition) nhưng chúng có mục đích sử dụng khác nhau, Decorator thêm trách nhiệm cho đối tượng trong khi Proxy kiểm soát truy cập vào đối tượng đó.