Proxy Pattern

Introduction

Proxy pattern is to provide a surrogate or placeholder for another object to control access to it. It uses an extra level of indirection to support distributed, controlled, or intelligent. Add a wrapper and delegation to protect the read component from undue complexity. The proxy pattern is applicable whenever there is a need for a more versatile or sophisticated reference to an object than just a simple pointer.

Motivation

- ➔ Virtual Proxy: Delay or defer the cost of initializing an object.
  - ◆ Cache some details of the data.
  - ◆ Use a small object to provide an interface to a huge real subject.
- ➔ Remote Proxy: Object in a different address space.
  - ◆ Send the request to the real subject in a different address space.
  - ◆ To hide the fact that the real subject is in a different address space and it might respond faster to user requests.
- ➔ Protection Proxy: To control access based on rights.
  - ◆ Give different users different accessing privileges.
  - ◆ E.g. reverse proxy with access control policy (nginx)
- ➔ Smart Proxy: Interpose additional actions when an object is accessed.
  - ◆ Process some simple tasks that are irrelevant to the real subject. Such as counting how many times the subject is referenced.
  - ◆ Keep multiple copies of real subject and manage user tasks to different real subjects. User tasks only see the manager subject. (E.g. Scheduler)

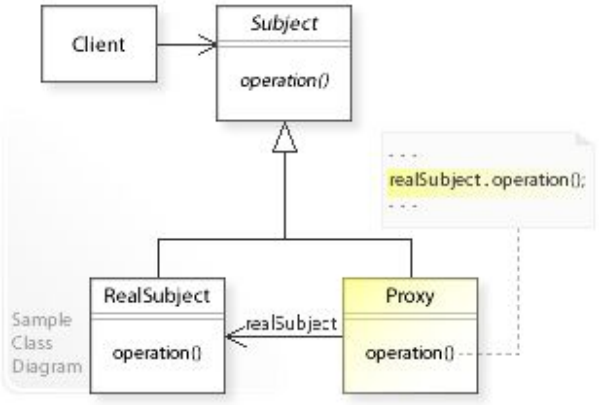
Terminology

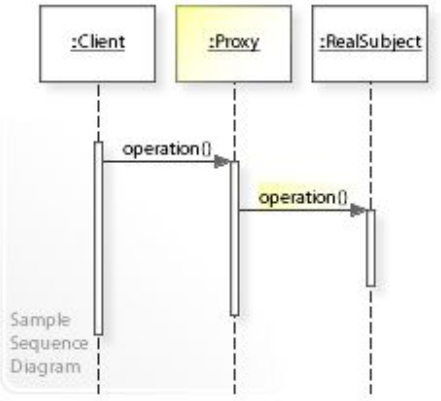
- ➔ Client: object that need allen to a real subject
- ➔ Real object: the actual object/payload to be passed to destination
- ➔ realsub/proxy,so that proxy can be used anywhere that a real subject can.

➔ Proxy: The surrogate object

- Maintain a ref to the actual subject.
- Interface in identical to sub/real sub.
- Control allen to real subject.
- Responsible for creating and deleting it.

Structure





Advantage

➔ A virtual proxy can optimize performance

➔ A remote proxy can hide location

➔ A protect proxy can do additional checks

➔ Reduce cohesion

Disadvantage

➔ when used in other cases may be an overkill.

➔ Implementing a proxy may require extra work, some proxy patterns implementation is very complex.

NFP’s

- ➔ Improve:
  - Performance
  - Security
- ➔ Degrade:
  - Complexity

Similar Patterns

- ➔ Adapter
- ➔ Decorator
  - changes a few methods
  - adds one/more responsibility