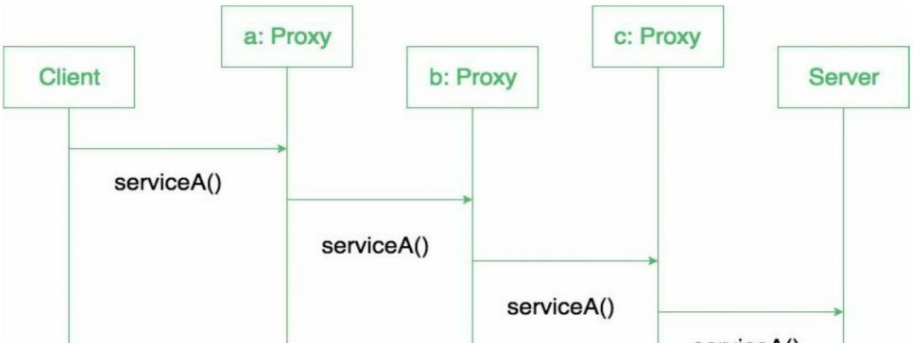
Proxy Design Pattern

Proxy means ‘in place of’, representing’ or ‘in place of’ or ‘on behalf of’ are literal meanings of proxy and that directly explains **Proxy Design Pattern**. Proxies are also called surrogates, handles, and wrappers. They are closely related in structure, but not purpose, to Adapters and Decorators.

A real world example can be a cheque or credit card is a proxy for what is in our bank account. It can be used in place of cash, and provides a means of accessing that cash when required. And that’s exactly what the Proxy pattern does – “**Controls and manage access to the object they are protecting**“.

Behavior

As in the decorator pattern, proxies can be chained together. The client, and each proxy, believes it is delegating messages to the real server:



When to use this pattern?

Proxy pattern is used when we need to create a wrapper to cover the main object's complexity from the client.

Types of proxies

Remote proxy:
They are responsible for representing the object located remotely. Talking to the real object might involve marshalling and unmarshalling of data and talking to the remote object. All that logic is encapsulated in these proxies and the client application need not worry about them.

Virtual proxy:

These proxies will provide some default and instant results if the real object is supposed to take some time to produce results. These proxies initiate the operation on real objects and provide a default result to the application. Once the real object is done, these proxies push the actual data to the client where it has provided dummy data earlier.

Protection proxy:

If an application does not have access to some resource then such proxies will talk to the objects in applications that have access to that resource and then get the result back.

Smart Proxy:

A smart proxy provides additional layer of security by interposing specific actions when the object is accessed. An example can be to check if the real object is locked before it is accessed to ensure that no other object can change it.

Some Examples

restricted site list, then it connects to the real internet. This example is based on Protection proxies.

Lets see how it works :

Interface of Internet

```
package com.saket.demo.proxy;

public interface Internet
{
    public void connectTo(String serverhost) throws Exception;
}
```

RealInternet.java

```
package com.saket.demo.proxy;

public class RealInternet implements Internet
{
    @Override
    public void connectTo(String serverhost)
    {
        System.out.println("Connecting to "+ serverhost);
    }
}
```

ProxyInternet.java

```
package com.saket.demo.proxy;

import java.util.ArrayList;
import java.util.List;


public class ProxyInternet implements Internet
{
```

```
            bannedSites = new ArrayList<String>();
            bannedSites.add("abc.com");
            bannedSites.add("def.com");
            bannedSites.add("ijk.com");
            bannedSites.add("lnm.com");
        }

        @Override
        public void connectTo(String serverhost) throws Exception
        {
            if(bannedSites.contains(serverhost.toLowerCase()))
            {
                throw new Exception("Access Denied");
            }

            internet.connectTo(serverhost);
        }

    }
```

Client.java

```
    package com.saket.demo.proxy;

    public class Client
    {
        public static void main (String[] args)
        {
            Internet internet = new ProxyInternet();
            try
            {
                internet.connectTo("geeksforgeeks.org");
                internet.connectTo("abc.com");
            }
            catch (Exception e)
            {
                System.out.println(e.getMessage());
            }
        }
    }
```

```
Connecting to geeksforgeeks.org
Access Denied
```

Benefits:

- One of the advantages of Proxy pattern is security.
- This pattern avoids duplication of objects which might be huge size and memory intensive. This in turn increases the performance of the application.
- The remote proxy also ensures about security by installing the local code proxy (stub) in the client machine and then accessing the server with help of the remote code.

Drawbacks/Consequences:

This pattern introduces another layer of abstraction which sometimes may be an issue if the RealSubject code is accessed by some of the clients directly and some of them might access the Proxy classes. This might cause disparate behaviour.

Interesting points:

- There are few differences between the related patterns. Like Adapter pattern gives a different interface to its subject, while Proxy patterns provides the same interface from the original object but the decorator provides an enhanced interface. Decorator pattern adds additional behaviour at runtime.
- Proxy used in Java API: java.rmi.*;

Further Read: Proxy Method in Python

This article is contributed by Saket Kumar. If you like GeeksforGeeks and would like to contribute, you can also write an article using

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Last Updated : 05 Dec, 2022

33

Similar Reads

1. Difference between Forward Proxy and Reverse Proxy
2. Singleton Design Pattern | Implementation
3. The Decorator Pattern | Set 2 (Introduction and Design)
4. Decorator Pattern | Set 3 (Coding the Design)
5. Flyweight Design Pattern
6. Singleton Design Pattern | Introduction
7. Java Singleton Design Pattern Practices with Examples
8. Composite Design Pattern
9. Prototype Design Pattern
10. Mediator design pattern

Previous Next

Vote for difficulty

Current difficulty : Easy

Easy | Normal | Medium | Hard | Expert

Article Tags : Design Pattern, System Design

Practice Tags : System Design

Improve Article | Report Issue

GeeksforGeeks
A-143, 9th Floor, Sovereign Corporate Tower, Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

Languages

Python

Java

C++

GoLang

SQL

R Language

Android Tutorial

Data Structures

Array

String

Linked List

Stack

Queue

Tree

Graph

Algorithms

Sorting

Searching

Greedy

Dynamic Programming

Pattern Searching

Recursion

Backtracking

Web Development

HTML

CSS

JavaScript

Bootstrap

ReactJS

AngularJS

NodeJS

Computer Science

GATE CS Notes

Operating Systems

Computer Network

Database Management System

Software Engineering

Digital Logic Design

Engineering Maths

Python

Python Programming Examples

Django Tutorial

Python Projects

Python Tkinter

OpenCV Python Tutorial

Python Interview Question

Data Science & ML

DevOps

Maths For Machine Learning

Pandas Tutorial

NumPy Tutorial

NLP Tutorial

Deep Learning Tutorial

Kubernetes

Azure

GCP

Competitive Programming

Top DSA for CP

Top 50 Tree Problems

Top 50 Graph Problems

Top 50 Array Problems

Top 50 String Problems

Top 50 DP Problems

Top 15 Websites for CP

System Design

What is System Design

Monolithic and Distributed SD

Scalability in SD

Databases in SD

High Level Design or HLD

Low Level Design or LLD

Top SD Interview Questions

Interview Corner

Company Preparation

Preparation for SDE

Company Interview Corner

Experienced Interview

Internship Interview

Competitive Programming

Aptitude

GfG School

CBSE Notes for Class 8

CBSE Notes for Class 9

CBSE Notes for Class 10

CBSE Notes for Class 11

CBSE Notes for Class 12

English Grammar

Commerce

Accountancy

Business Studies

Microeconomics

UPSC

Polity Notes

Geography Notes

History Notes

UPSC Previous Year Papers

SSC/ BANKING

SSC CGL Syllabus

SBI PO Syllabus

SBI Clerk Syllabus

IBPS PO Syllabus

IBPS Clerk Syllabus

Aptitude Questions

SSC CGL Practice Papers

Write & Earn

Write an Article

Improve an Article

Pick Topics to Write

Write Interview Experience

Internships

Video Internship