Java Design Patterns

Proxy

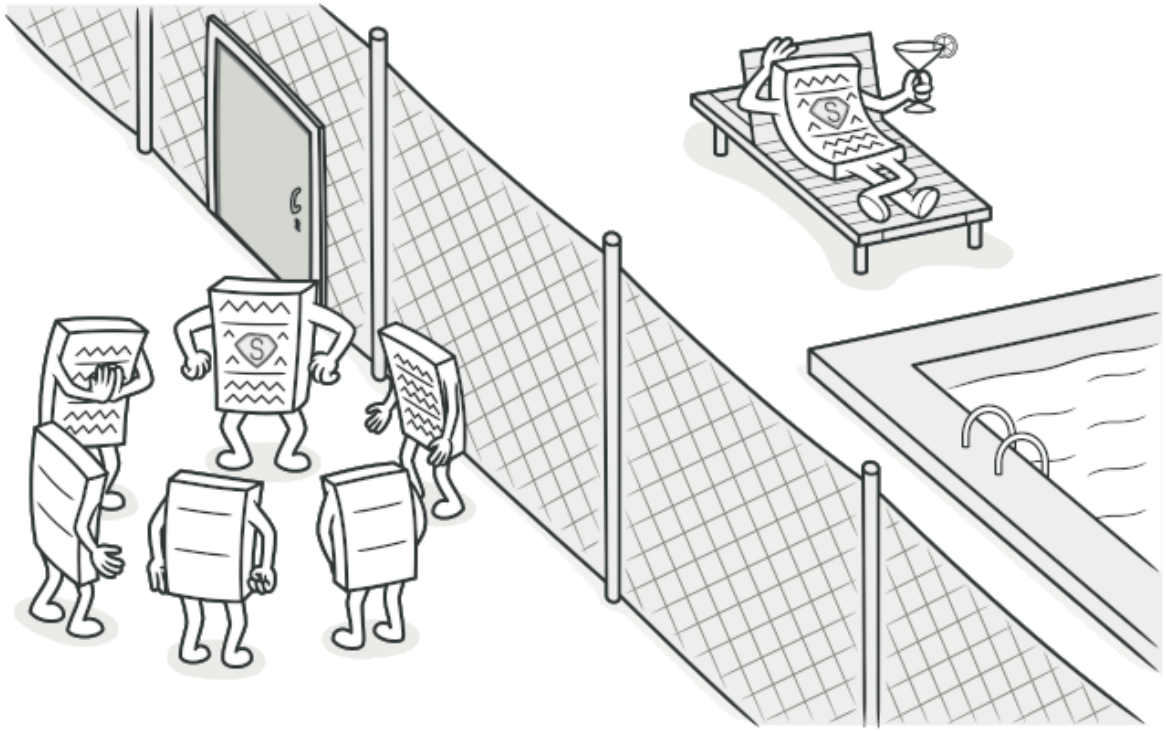
Тема

Proxy

Суть паттерна

Заместитель

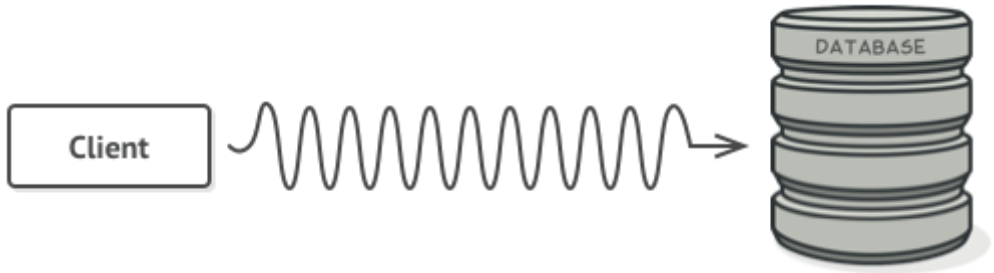
Заместитель — это структурный паттерн проектирования, который позволяет подставлять вместо реальных объектов специальные объекты-заменители. Эти объекты перехватывают вызовы к оригинальному объекту, позволяя сделать что-то до или после передачи вызова оригиналу.



Проблема

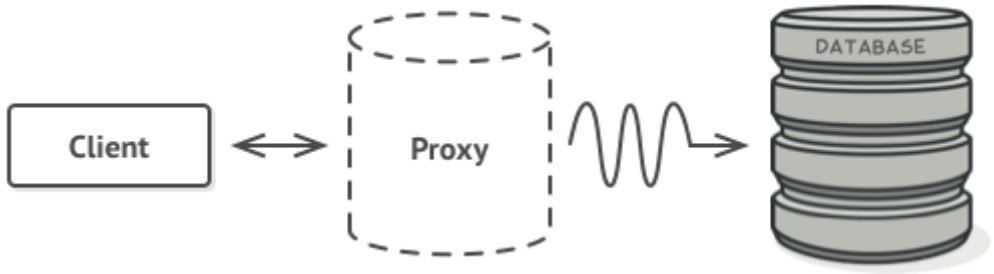
Постановка задачи

Для чего вообще контролировать доступ к объектам? Рассмотрим такой пример: у вас есть внешний ресурсоёмкий объект, который нужен не все время, а изредка.



Решение задачи

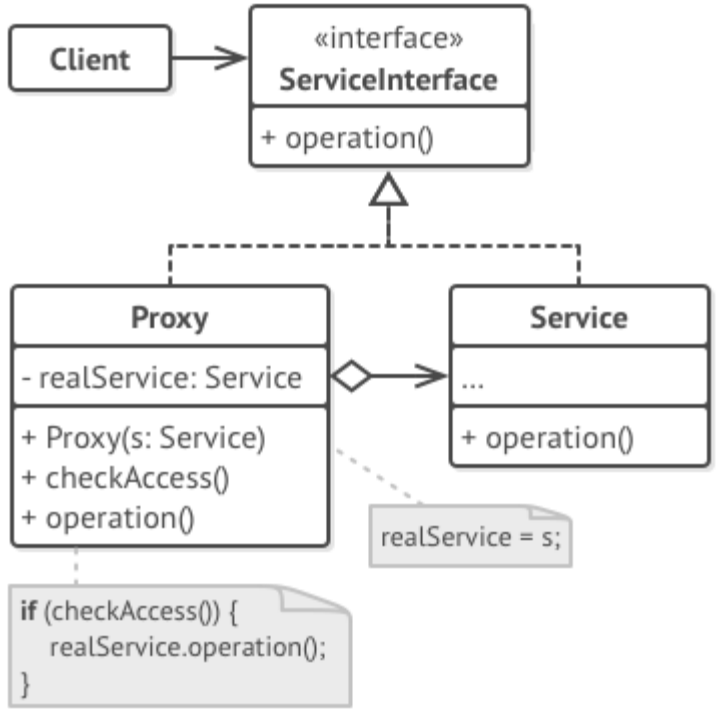
Паттерн заместитель предлагает создать новый класс-дублёр, имеющий тот же интерфейс, что и оригинальный служебный объект. При получении запроса от клиента, объект-заместитель сам бы создавал экземпляр служебного объекта и переадресовывал бы ему всю реальную работу.



Структура

Структура паттерна

1. Интерфейс сервиса определяет общий интерфейс для сервиса и заместителя. Благодаря этому, объект заместителя можно использовать там, где ожидается объект сервиса.
2. Сервис содержит полезную бизнес-логику.
3. Заместитель хранит ссылку на объект сервиса. После того как заместитель заканчивает свою работу (например, инициализацию, логирование, защиту или другое), он передаёт вызовы вложенному сервису.
4. Клиент работает с объектами через интерфейс сервиса. Благодаря этому, его можно «одурачить», подменив объект сервиса объектом заместителя.



Применимость

Применение паттерна

1. Ленивая инициализация (виртуальный прокси). Когда у вас есть тяжёлый объект, грузящий данные из файловой системы или базы данных.
2. Защита доступа (защищающий прокси). Когда в программе есть разные типы пользователей и вам хочется защищать объект от неавторизованного доступа. Например, если ваши объекты — это важная часть операционной системы, а пользователи — сторонние программы (хорошие или вредоносные).
3. Локальный запуск сервиса (удалённый прокси). Когда настоящий сервисный объект находится на удалённом сервере.
4. Логирование запросов (логирующий прокси). Когда требуется хранить историю обращений к сервисному объекту.
5. Кеширование объектов («умная» ссылка). Когда нужно кешировать результаты запросов клиентов и управлять их жизненным циклом.

Шаги реализации

Алгоритм реализации паттерна

1. Определите интерфейс, который бы сделал заместитель и оригинальный объект взаимозаменяемыми.
2. Создайте класс заместителя. Он должен содержать ссылку на сервисный объект. Чаще всего, сервисный объект создаётся самим заместителем. В редких случаях, заместитель получает готовый сервисный объект от клиента через конструктор.
3. Реализуйте методы заместителя в зависимости от его предназначения. В большинстве случаев, проделав какую-то полезную работу, методы заместителя должны передать запрос сервисному объекту.
4. Подумайте о введении фабрики, которая решала бы какой из объектов создавать — заместитель или реальный сервисный объект. Но с другой стороны, эта логика может быть помещена в создающий метод самого заместителя.
5. Подумайте, не реализовать ли вам ленивую инициализацию сервисного объекта при первом обращении клиента к методам заместителя.

Преимущества и недостатки

Плюсы и недостатки

Плюсы:

- Позволяет контролировать сервисный объект незаметно для клиента.
- Может работать, даже если сервисный объект ещё не создан.
- Может контролировать жизненный цикл служебного объекта.

Минусы:

- Усложняет программу за счёт дополнительных классов.
- Увеличивает время отклика от сервиса.

Отношение с другими паттернами

- Адаптер предоставляет классу альтернативный интерфейс. Декоратор предоставляет расширенный интерфейс. Заместитель предоставляет тот же интерфейс.
- Фасад похож на Заместитель тем, что замещает сложную подсистему и может сам её инициализировать. Но в отличие от *Фасада*, *Заместитель* имеет тот же интерфейс, что его служебный объект, благодаря чему их можно взаимозаменять.
- Декоратор и Заместитель имеют похожие структуры, но разные назначения. Они похожи тем, что оба построены на композиции и делегировании работы другому объекту. Паттерны отличаются тем, что *Заместитель* сам управляет жизнью сервисного объекта, а обёртывание *Декораторов* контролируется клиентом.

Информационный видеосервис для разработчиков программного обеспечения

ITVDN

IT VIDEO DEVELOPERS NETWORK

TestProvider PROMETRIC TEST CENTER CyberBionic systematics Microsoft Partner Silver Learning Windows Azure Cloud Partner EBA European Business Association