Прокси — это шаблон проектирования. Мы создаем и используем его для добавления и изменения функционала уже существующих классов. В таком случае, прокси-объект применяется вместо исходного. Обычно он использует тот же метод, что и оригинальный, и в Java прокси-классы расширяют исходные. Прокси может вызвать метод исходного объекта, так как у него есть дескриптор оригинала.

**-----------**

**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.

--------------

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.

Но в чём же здесь польза? Вы могли бы поместить в класс заместителя какую-то промежуточную логику, которая выполнялась бы до (или после) вызовов этих же методов в настоящем объекте. А благодаря одинаковому интерфейсу, объект-заместитель можно передать в любой код, ожидающий сервисный объект.

---------------

<https://habr.com/ru/post/88722/>

#### **Проблема**

Необходимо контролировать доступ к объекту, не изменяя при этом поведение клиента.

**Применимость**

**Ленивая инициализация (виртуальный прокси). Когда у вас есть тяжёлый объект, грузящий данные из файловой системы или базы данных.**

 Вместо того, чтобы грузить данные сразу после старта программы, можно сэкономить ресурсы и создать объект тогда, когда он действительно понадобится.

**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).**

 Прокси может проверять доступ при каждом вызове и передавать выполнение служебному объекту, если доступ разрешён.

**Локальный запуск сервиса (удалённый прокси). Когда настоящий сервисный объект находится на удалённом сервере.**

 В этом случае заместитель транслирует запросы клиента в вызовы по сети в протоколе, понятном удалённому сервису.

**Логирование запросов (логирующий прокси). Когда требуется хранить историю обращений к сервисному объекту.**

 Заместитель может сохранять историю обращения клиента к сервисному объекту.

**Кеширование объектов («умная» ссылка). Когда нужно кешировать результаты запросов клиентов и управлять их жизненным циклом.**

 Заместитель может подсчитывать количество ссылок на сервисный объект, которые были отданы клиенту и остаются активными. Когда все ссылки освобождаются, можно будет освободить и сам сервисный объект (например, закрыть подключение к базе данных).

Кроме того, Заместитель может отслеживать, не менял ли клиент сервисный объект. Это позволит использовать объекты повторно и здóрово экономить ресурсы, особенно если речь идёт о больших прожорливых сервисах.

-----------

## Преимущества и недостатки

* Позволяет контролировать сервисный объект незаметно для клиента.
* Может работать, даже если сервисный объект ещё не создан.
* Может контролировать жизненный цикл служебного объекта.
* Усложняет код программы из-за введения дополнительных классов.
* Увеличивает время отклика от сервиса.

-----------------

Таким образом, прокси-классы удобно реализуют многие вещи:

* логирование старта и остановки метода;
* дополнительную проверку аргументов;
* имитацию поведения исходного класса;
* реализацию отложенной инициализации затратных ресурсов;

<https://leodev.ru/blog/java/java-dynamic-proxy/#.XkZJqGgzY-U>

<https://www.codeflow.site/ru/article/java-dynamic-proxies>

<https://habr.com/ru/company/otus/blog/434214/>

<https://www.ibm.com/developerworks/ru/library/j-jtp08305/index.html>

Implementation  
<https://sourcemaking.com/design_patterns/proxy/java/1>

Compare performance

<https://www.jrebel.com/blog/java-code-generation-libraries-comparison>

<https://martinsdeveloperworld.wordpress.com/tag/javassist/>

English theory

<https://howtodoinjava.com/design-patterns/structural/proxy-design-pattern/>

<https://www.baeldung.com/java-dynamic-proxies>

<https://dzone.com/articles/java-dynamic-proxy>