Das Entwurfsmuster: Proxy

Das Proxy-Entwurfsmuster ist ein **GoF-Strukturmuster** und stellt dir ein Stellvertreter-Objekt oder Platzhalter (*Proxy*) für ein anderes Objekt (*Subjekt*) zur Verfügung.

Absicht

Das Proxy-Muster bietet dir Kontrolle über die Objekterzeugung des Subjekts und dessen Zugriff darauf.

Motivation

Langweilig - die Erste: Sicherer Browser dank Zugriffsproxy

Stell dir mal vor, dass du einen sicheren Browser entwickeln möchtest, welcher aus Kostengründen allerdings z. B. bereits vorhandene Browserschnittstellen im Hintergrund verwenden soll. Damit die Nutzer deines Browsers vor Datendiebstahl geschützt sind, soll dein Browser nur sichere HTTPS-Verbindungen erlauben und alle unsicheren HTTP-Verbindungen blockieren. Da der Quelltext deiner Engine allerdings nicht verändert werden kann/darf (z. B. aus Lizenzgründen), kannst du die Zugriffsbeschränkung auf HTTPS nicht einfach so in das Ursprungsobjekt einfügen. Jetzt kommt dein *Zugriffsproxy* ins Spiel: Er kontrolliert den Zugriff auf den relevanten Teil der Browserschnittstelle (hier das Subjekt) und delegiert alle weiteren Methodenaufrufe direkt an das Subjekt weiter. Somit muss der Quelltext deiner geplanten Engine nicht verändert werden und alle sind glücklich!

Motiviert - die Zweite: Lazy-Loading mit virtuellem Proxy

Ein weiteres sehr simples Beispiel, warum du den Zugriff zu einem Objekt kontrollieren möchtest, ist ein *virtueller Proxy* zum verzögerten Laden eines Bildes (Lazy Loading). Es ist nämlich nicht immer notwendig, dass ein Bild bereits beim Instanzzieren eines Objekts in den Speicher geladen werden muss.

Stell dir vor, dass du ganz viele Bilder auf deiner Werbe-Website deines sicheren Browsers hast. Da viele dieser Bilder im unteren unsichtbaren Bereich („below the fold“) sind, ist es beim Laden der Website nicht sinnvoll, noch nicht sichtbare Bilder deiner Website zu laden - es reicht also, dass nur diese Bilder geladen werden, die aktuell tatsächlich im sichtbaren Bereich der Website sind. Die folgende Abbildung zeigt das Lazy-Loading anhand einer kürzlich benutzten horizontalen Scrollbalken:

Abbildung 1: Beispiel des Lazy-Loading anhang einer vertikalen Scrollbar

Das Laden des realen Bildes (also das Subjekt) wird also in diesem Beispiel durch einen Stellvertreter vertreten (Proxy) und der Zugriff auf das tatsächliche Objekt wird soweit verzögert, bis er wirklich erst notwendig ist.

Hochmotiviert - die Dritte: Reporting der Downloads dank Remote-Proxy

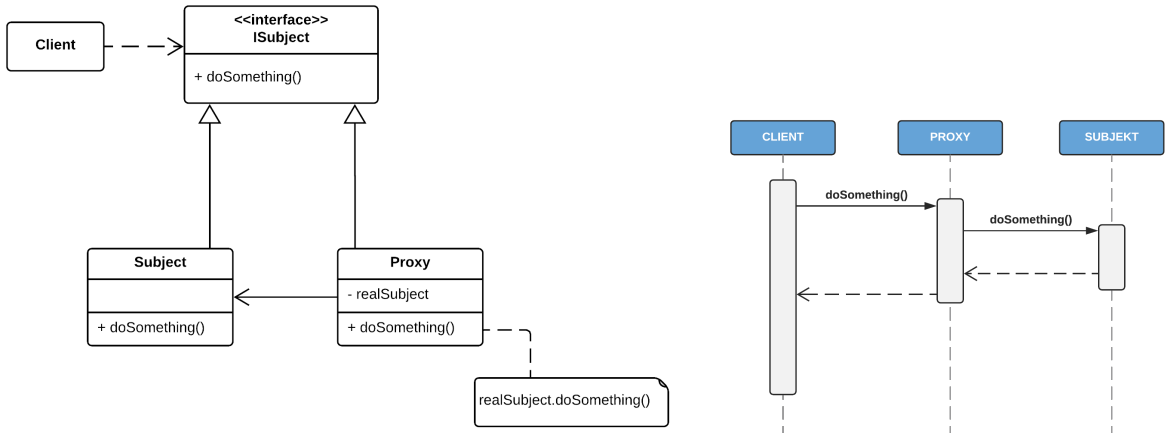
Da du natürlich am Ende des Tages wissen möchtest, ob dein sicherer Browser von den Nutzern tatsächlich heruntergeladen wird, hast du dir bei der Planung der Website bereits einige Download-Statistiken überlegt:

- Datum
- Uhrzeit (in Stunden)
- Endgerätetyp

Diese Statistiken werden über die Downloads über deine Website generiert. Da du den Browser aber noch zusätzlich auf Download-Portalen anbietest, musst du diese

Zahlen natürlich ebenfalls in deine Statistik miteinbeziehen. Alle Download-Portale bieten dir API-Schnittstellen an, welche ebenfalls die oben genannten Daten liefern. Mithilfe des Proxy-Objekts kannst du nun in die ursprüngliche Zugriffkontrolle des Statistik-Objekts eingreifen und z. B. die Eigenschaften des Objekts auf die von der jeweiligen API-Schnittstelle zur Verfügung gestellten Daten setzen. Weitere Methoden vom Proxies werden dann einfach an das eigentliche Reportobjekt (unserem Subjekt) delegiert.

Struktur & Sequenzdiagram



- **ISubject** definiert das Interface für das Subjekt und den Proxy
- **Subject** ist die Klasse, welche die eigentliche Logik implementiert
- **Proxy** beinhaltet eine Eigenschaft, die eine Referenz zum Subjekt speichert. Methodenaufrufe werden in der Regel anschließend an das Subjekt delegiert.
- **Client** ruft Methodenaufrufe über das definierte ISubject auf, sodass das Liskovsche Substitutionsprinzip (LSP) erfüllt bleibt. Er kann also mit beiden Klassen gleichweise kommunizieren.

Verwendbarkeit

Proxies werden häufig in Situationen verwendet, in welcher anstelle eines normalen Zeigers eine komplexere Referenz auf ein Objekt notwendig ist. Es wird dabei meistens unter folgende Arten von Proxies unterschieden:

- **Virtual Proxies** verzögern die Instanziierung eines Objekts (Lazy Loading)
- **Remote Proxies** kontrollieren den Zugriff auf ein entferntes Objekt
- **Protection Proxies** ermöglichen eine Zugriffskontrolle auf das Subjekt
- **Cache Proxies** ermöglichen die Zwischenspeicherung zur schnelleren Auslieferung an den Client
- **Logging Proxies** ermöglichen das Protokollieren eines Aufrufs, bevor dieser an das Subjekt delegiert wird.

Es existieren noch einige weitere Arten des Proxy-Pattern, welche allerdings seltener verwendet werden.

In der realen Welt

Auch in der realen Welt verwenden wir regelmäßig die Prinzipien von Proxies. Nehmen wir mal an, du möchtest beim Aldi nebenan deinen wöchentlichen Einkauf mit deiner Kreditkarte/Girocard bezahlen. Dabei ist deine Karte sozusagen der Proxy für dein Bankkonto. Deine Kreditkarte erlaubt dem Ladeninhaber das Gelf für die Ware zu erhalten, ohne das er direkten Zugriff auf dein gesamtes Bankkonto erhält.

Proxy-Programmierbeispiele in PHP

```
1 cd ~/Sites/
2 git clone https://github.com/it4need/php-proxy-pattern
```

Damit du das Proxy-Pattern von Grund auf verstehst, gibt es im Repository einige Beispiele in PHP zu den jeweiligen Proxy-Varianten. Du findest im Repository ebenfalls

Unit-Tests, damit du den Sinn von den einzelnen Proxies verstehst. Die Tests sind gleichzeitig deine Dokumentation für die Anwendung des Musters.

Vorteile

- Proxies können den Zugang zu Objekten kontrollieren, bevor sie die Methodenaufrufe an das Subjekt delegieren
- Der Client bemerkt die Kontrolle des Objekts nicht
- Proxies funktionieren ebenfalls, wenn das eigentliche Objekt noch nicht im Speicher verfügbar ist (z.B. Lazy Loading)
- Proxies können helfen, die Geschwindigkeit der Applikation zu steigern (z.B. Caching)
- Proxies können die Interprozesskommunikation/Fernkommunikation zwischen Objekten ermöglichen

Nachteile

- Proxies können je nach Implementierung die Antwortzeit verzögern
- Proxies fügen eine weitere Abstraktionsschicht ein, welche zu Umwegen führen kann und die Komplexität steigert
- Es müssen alle von der Schnittstelle definierten Methoden für das Proxy erstellt und an das Subjekt delegiert werden

Beziehungen zu anderen Entwurfsmuster

- Das *Decorator*-Entwurfsmuster sieht ähnlich aus, hat allerdings ein anderes Ziel: Er fügt zusätzliches Verhalten hinzu.
- Das *Facade*-Pattern kontrolliert nicht den Zugriff auf das Objekt; es stellt eine andere Schnittstelle bereit

- Das *Adapter*-Pattern verändert im Gegensatz zum Proxy-Entwurfsmuster die Schnittstellendefinition, aber nicht die Geschäftslogik (sinnbildmäßiger Stromadapter).