Proxies

Sistemas Distribuídos e Mobile

Prof. Me. Gustavo Torres Custódio
gustavo.custodio@ulife.com.br

Roteiro

Design Patterns

O Proxy

Estrutura Estática

Aplicabilidade

Prós e Contras

Prática

Design Patterns

Design Patterns / Padrões de Projeto

- Padrões de projeto são soluções típicas para problemas comuns em projeto de software.
  - Eles são como plantas de obra pré fabricadas que você pode customizar para resolver um problema de projeto recorrente em seu código.
- Você não pode apenas encontrar um padrão e copiá-lo para dentro do seu programa, como você faz com funções e bibliotecas.
- O padrão não é um pedaço de código específico, mas um conceito geral para resolver um problema em particular.

Design Patterns / Padrões de Projeto

- Os padrões são frequentemente confundidos com algoritmos, porque ambos os conceitos descrevem soluções típicas para alguns problemas conhecidos.
- Enquanto um algoritmo sempre define um conjunto claro de ações para atingir uma meta, um padrão é mais uma descrição de alto nível de uma solução.
  - O código do mesmo padrão aplicado para dois programas distintos pode ser bem diferente.

Motivação

- Um design pattern descreve a solução para um problema recorrente no âmbito do desenvolvimento de software.
- Documenta uma comprovada solução utilizada recorrentemente em diferentes contextos,
  - captura o conhecimento adquirido pelos desenvolvedores de software em anos de experiência profissional
  - detalhando inclusive aspectos positivos e negativos da solução abordada e diferentes estratégias de implementação.

Motivação

- Contribuem para uma fluência na comunicação entre desenvolvedores
  - criam um vocabulário único que representa as soluções para problemas comumente encontrados em um espectro bastante abstraído.

Classificações de padrões

- *Design Patterns* tradicionais, ou originais (GoF) são divididos em 3 grandes tipos:

Padrões Criacionais

- Fornecem vários **mecanismos de criação de objetos**, que aumentam a flexibilidade e reutilização de código já existente.
- *Factory, Abstract Factory, Builder, Prototype e Singleton.*

Padrões Comportamentais

- São voltados aos algoritmos e a **designação de responsabilidades entre objetos**.
- *Chain of Responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, TemplateMethod, Visitor.*

Padrões Estruturais

- Explicam como **montar objetos e classes em estruturas maiores** mas ainda mantendo essas estruturas flexíveis e eficientes.
- *Adapter, Bridge, Composite, Decorator, Façade, FlyWeight, Proxy*.

O Proxy

Proxy

- Proxy é um padrão de projeto estrutural que permite que você forneça um substituto ou um espaço reservado para outro objeto.
- Um proxy **controla o acesso ao objeto original**, permitindo que você faça algo ou antes ou depois da requisição chegar ao objeto original.

Proxy - O Problema

- Por que controlar o acesso a um objeto?
- Você tem um **objeto grande que consome muitos recursos do sistema**.
  - Você precisa dele de tempos em tempos, mas não sempre.
- Você quer controlar de maneira transparente em termos de API o consumo de algum recurso dispendioso (RPC, Sockets, SOAP).

Proxy - O Problema

- Você precisa prover um **substituto** (ou *placeholder*) para um outro objeto controlar seu acesso;
- Você precisa usar um nível extra de direção para fornecer acesso distribuído, controlado ou inteligente;
- Você precisa adicionar um agregador e delegador para **proteger o componente real de complexidade indevida** (maioria das vezes).

Proxy - O Problema

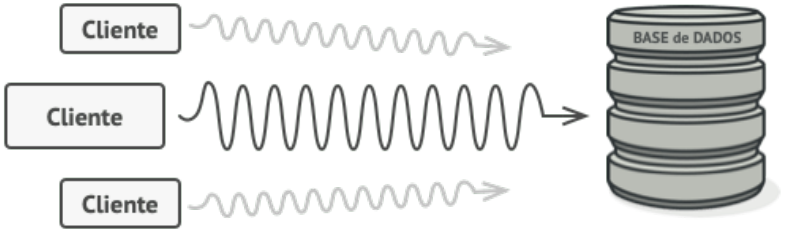


Figura: Conexões com Bancos de dados

Proxy - O Problema

- O processo de conexão com o banco de dados **pode ser bem lento** (principalmente Oracle);
- A conexão não precisa ser feita a todo momento.
  - *pool* de requisições.
  - a cada requisição, basta utilizar uma conexão diferente.
- Porém tanto o código que efetua a conexão quanto o código que solicita e gerencia o acesso ao *pool* de conexões deve ser de responsabilidade do cliente.

Proxy - A Solução

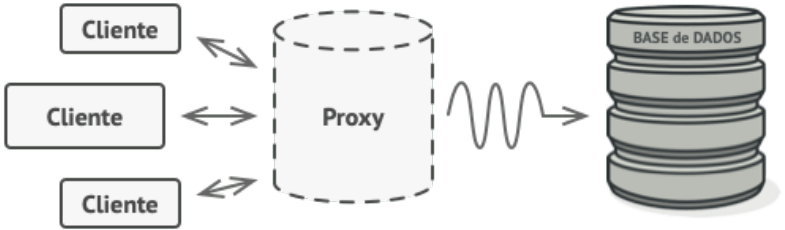


Figura: Conexões com Bancos de dados com proxies

Proxy - A Solução

- Inserir um **objeto mediador que encapsule (esconda)** a responsabilidade de gerenciar conexões com o banco de dados e a seleção/entrega das conexões no pool de conexões.
- Dessa maneira a API esconde do desenvolvedor cliente da API toda a funcionalidade que **não faz parte do negócio**, neste caso, o gerenciamento de acesso às conexões do banco de dados.

Proxy - A Solução

- Esse objeto mediador, o proxy, é também chamado de “substituto” ou Stub.
- Além de substituir e mediar o estado conversacional entre o cliente e o objeto original, pode adicionar comportamentos como permissão e contagem de acesso, ou auditoria SEM QUE O CLIENTE TENHA PLENA CIÊNCIA do mecanismo.

Analogia

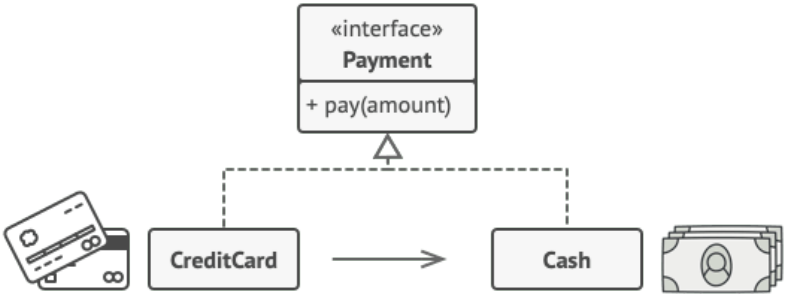


Figura: Cartão de crédito com proxy para utilização da conta bancária

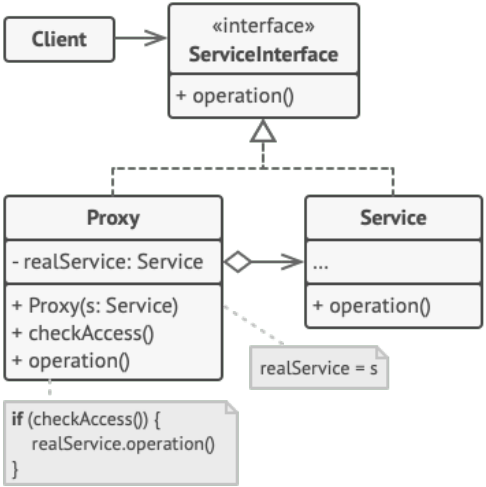
Analogia com o mundo real

- Um cartão de crédito é um proxy para uma conta bancária, que é um proxy para uma porção de dinheiro.
- Ambos implementam a mesma interface.
- Um cliente se sente bem porque não precisa ficar carregando montanhas de dinheiro.
- Um dono de loja também fica feliz uma vez que a renda da transação é adicionada eletronicamente para sua conta sem o risco de perdê-la no depósito ou de ser roubado quando estiver indo ao banco.

Proxies

Estrutura Estática

Estrutura


Client
«interface»
ServiceInterface
+ operation()
Proxy
- realService: Service
+ Proxy(s: Service)
+ checkAccess()
+ operation()
Service
...
+ operation()
realService = s
if (checkAccess()) {
realService.operation()
}

Estrutura estática

1. A **Interface do Serviço** declara o serviço a ser "proxyado". O proxy deve seguir essa interface para ser capaz de se disfarçar como um objeto do serviço (`ServiceInterface`).

2. O Serviço é uma classe que **fornece alguma lógica de negócio útil (`Service`)**.

Estrutura estática

3. A classe `Proxy` tem um campo de referência que aponta para um objeto do serviço. Após o proxy finalizar seu processamento, ele passa a requisição para o objeto do serviço.

4. O **`Cliente` deve trabalhar tanto com os serviços quanto com os proxies** através da mesma interface. Dessa forma é possível passar um proxy para qualquer código que espere um objeto real do serviço, executando a "mágica" da substituição.

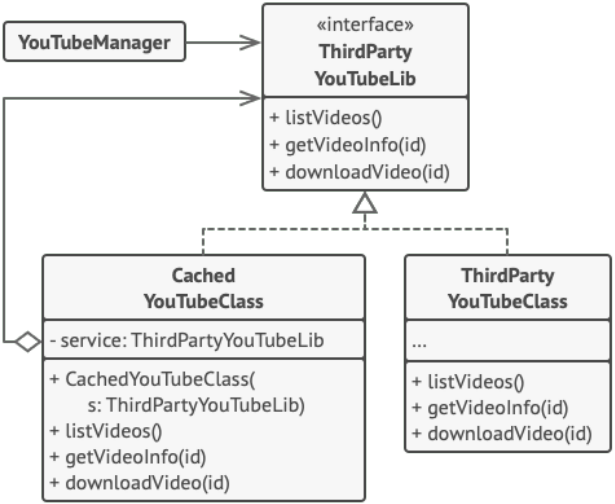
Exemplo - Download de Vídeos do YouTube

- O exemplo ilustra como o padrão Proxy pode ajudar a introduzir uma inicialização tardia (lazy-load) e cache para um biblioteca de integração terceirizada do YouTube.
- A biblioteca de terceiros nos fornece uma classe de download de vídeo. Contudo, ela é muito ineficiente.

Exemplo - Download de Vídeos do YouTube

- Se a aplicação cliente pedir o mesmo vídeo múltiplas vezes, a biblioteca apenas baixa de novo e de novo, ao invés de colocar o vídeo em cache e reutilizar o primeiro arquivo de download.
- A classe proxy implementa a mesma interface que a classe original do terceiro e a delega todo o trabalho. Contudo, ela mantém um registro dos arquivos baixados e retorna o resultado em cache quando a aplicação pede o mesmo vídeo múltiplas vezes.

Exemplo - Download de Vídeos do YouTube



Proxies

Aplicabilidade

Aplicabilidade

- Problema: Inicialização tardia (proxy virtual). É quando você tem um objeto do serviço peso-pesado que gasta recursos do sistema por estar sempre rodando, mesmo quando você precisa dele de tempos em tempos.
- Aplicaçã: Ao invés de criar um objeto quando a aplicação inicializa, você pode atrasar a inicialização do objeto para um momento que ele é realmente necessário, diminuindo consumo de recursos e inicializando a aplicação de maneira mais rápida.

Aplicabilidade

- Problema: Controle de acesso (proxy de proteção). Este é quando você quer que apenas clientes específicos usem o objeto do serviço; por exemplo, quando seus objetos são partes cruciais de um sistema operacional e os clientes são várias aplicações iniciadas (incluindo algumas maliciosas).
- Aplicação: O proxy pode passar o pedido para o objeto de serviço somente se as credenciais do cliente coincidem com certos critérios.

Aplicabilidade

- Problema: Execução local de um serviço remoto (proxy remoto). Este é quando o objeto do serviço está localizado em um servidor remoto.
- Aplicação: Neste caso, o proxy passa o pedido do cliente pela rede, lidando com todos os detalhes complexos pertinentes a se trabalhar com redes (RPC).

Aplicabilidade

- Problema: Cache de resultados de pedidos (proxy de cache). Usado quando é necessário colocar em cache os resultados de pedidos do cliente e gerenciar o ciclo de vida deste cache, especialmente se os resultados são muito grandes.
- Aplicação: O proxy pode implementar o armazenamento em cache para pedidos recorrentes que sempre acabam nos mesmos resultados. O proxy pode usar como parâmetros dos pedidos as chaves de cache.

Aplicabilidade

- Problema: Referência inteligente. Quando é necessário eliminar um objeto peso-pesado assim que não há mais clientes que o utilizam
- Aplicação: O proxy pode manter um registro de clientes que obtiveram uma referência ao objeto serviço ou seus resultados. De tempos em tempos, o proxy pode verificar com os clientes se eles ainda estão ativos. Se a lista cliente ficar vazia, o proxy pode remover o objeto serviço e liberar os recursos de sistema que ficaram até então utilizados desnecessariamente.

Proxies

Prós e Contras

Pontos Positivos

- É possível controlar o objeto do serviço sem o conhecimento dos clientes.
- É possível gerenciar o ciclo de vida de um objeto do serviço quando os clientes não se importam mais com ele.
- O proxy trabalha até mesmo se o objeto do serviço ainda não estiver pronto ou disponível (rede).
- Princípio aberto/fechado (*open / closed principle*). Você pode introduzir novos proxies sem mudar o serviço ou clientes.

Pontos Negativos

- O código pode ficar mais complicado uma vez que é necessário introduzir uma série de novas classes que compõe o padrão proxy;
- A resposta de um serviço pode ter atrasos. O padrão proxy por si só não controla estes possíveis atrasos, porém é possível construir estratégias dentro dos proxies que o façam.

Proxies

Prática

Exemplo Prático

- Vamos criar um proxy que bloqueia uma lista de sites.

Referências


Cooper, J. W. (2000).
Java design patterns: a tutorial.


Gamma, E., Helm, R., Johnson, R., Vlissides, J., and Patterns, D. (1995).
Elements of reusable object-oriented software.
Design Patterns.


Guerra, E. (2014).
Design Patterns com Java: Projeto orientado a objetos guiado por padrões.

Editora Casa do Código.

Conteúdo

https://gustavotcustodio.github.io/sdmobile.html

Obrigado

gustavo.custodio@ulife.com.br