## 1. A respeito de sistemas distribuídos heterogêneos, faça o que se pede a seguir.

**a) Apresente exemplos das diferenças nas formas de representação dos dados que podem causar problemas em sistemas distribuídos.**

**R:**

Em sistemas distribuídos heterogêneos, diferentes formas de representação de dados podem causar problemas devido à incompatibilidade entre os formatos. Aqui estão alguns exemplos de diferenças nas formas de representação de dados que podem gerar problemas:

- Representação de números: Diferentes sistemas podem usar formatos de representação numérica diferentes, como ponto flutuante de precisão simples ou dupla, inteiro com ou sem sinal, codificação de ponto fixo, etc. Se os sistemas não concordarem sobre o formato numérico utilizado, a comunicação e o processamento de dados numéricos podem ser inconsistentes.

- Representação de caracteres: Diferentes sistemas podem usar conjuntos de caracteres diferentes, como ASCII, Unicode ou EBCDIC. Se não houver um acordo sobre a codificação de caracteres utilizada, a interpretação correta dos caracteres pode ser comprometida, resultando em erros de leitura e exibição incorreta dos dados.

- Tamanho e ordem dos bytes: A ordem dos bytes (endianness) pode variar entre os sistemas, podendo ser big-endian ou little-endian. Além disso, sistemas diferentes podem ter diferentes tamanhos para tipos de dados primitivos, como inteiros de 32 bits ou 64 bits. Essas diferenças podem causar problemas ao transmitir dados binários entre sistemas.

- Formato de dados estruturados: Os sistemas podem usar formatos diferentes para representar dados estruturados, como JSON, XML, CSV, entre outros. Se não houver um acordo sobre o formato de dados estruturados utilizado, a interpretação correta dos dados pode ser comprometida e dificultar a interoperabilidade entre os sistemas.

Essas diferenças nas formas de representação dos dados podem levar a erros de interpretação, corrupção de dados e comportamentos inesperados nos sistemas distribuídos heterogêneos.

**b) Explique o que é eXternal Data Representation (XDR) e como uma biblioteca XDR pode ser usada em chamadas a procedimentos remotos.**

**R:**

A eXternal Data Representation (XDR) é uma forma de representar dados de forma independente da arquitetura específica de um sistema. É um padrão desenvolvido pelo IETF (Internet Engineering Task Force) que permite que os dados sejam codificados e decodificados de maneira consistente, independentemente das diferenças nas arquiteturas de hardware e sistemas operacionais.

Uma biblioteca XDR contém funções que facilitam a codificação e decodificação de dados no formato XDR. Ao usar uma biblioteca XDR em chamadas a procedimentos remotos, os dados são codificados no formato XDR antes de serem enviados através da rede. Na extremidade receptora, os dados são decodificados de volta para o formato nativo do sistema.

Essa abordagem garante que os dados sejam transmitidos e interpretados corretamente, independentemente das diferenças nas representações de dados entre sistemas distribuídos heterogêneos. A biblioteca XDR fornece uma interface consistente para a codificação e decodificação de dados, tornando a comunicação entre sistemas mais confiável e interoperável.

**2. A figura a seguir apresenta o modelo de implementação RPC. Explique o papel de cada um dos seus componentes e explique de que forma o modelo request-reply funciona neste modelo.**

**R:**

**Módulo de comunicação (Cliente):** envia as solicitações RPC para o servidor e recebe as respostas correspondentes, garantindo a transmissão confiável das mensagens.

**Stub Procedure (Cliente):** é uma representação local do procedimento remoto no cliente. Ele encapsula os parâmetros da chamada RPC, realiza o marshalling e envia a mensagem de solicitação ao servidor.

**Client Program:** refere-se a um programa de software que é executado em uma máquina cliente e interage com um servidor ou serviço remoto. O cliente programa geralmente é responsável por enviar solicitações de serviço para o servidor e processar as respostas recebidas.

**Client Process:** refere-se a uma instância em execução de um programa cliente. Em um ambiente de computação distribuída, várias instâncias do programa cliente podem ser iniciadas em diferentes máquinas ou processos para interagir com o servidor.

**Stub Procedure (Servidor):** é uma representação local do procedimento remoto no servidor. Ele recebe as mensagens de solicitação, realiza o unmarshalling dos parâmetros, encaminha a chamada para a função correspondente e encapsula a resposta em uma mensagem de resposta.

**Módulo de comunicação (Servidor):** recebe as solicitações RPC do cliente e envia as respostas correspondentes, garantindo a transmissão confiável das mensagens.

**Dispatcher (Servidor):** recebe as mensagens de solicitação e encaminha para a função apropriada no server stub procedure.

**Services Procedures (Servidor):** são as funções reais que implementam a lógica dos procedimentos remotos no servidor. Elas são invocadas pelo server stub procedure para executar as operações solicitadas.

**Server Process:** é uma instância em execução de um programa de servidor em um sistema distribuído. Esse processo é responsável por receber solicitações dos clientes, processá-las e fornecer as respostas correspondentes.

O modelo request-reply funciona da seguinte maneira em um sistema com esses componentes:

1. O cliente envia uma solicitação RPC ao servidor. A solicitação inclui o nome do procedimento a ser executado e os parâmetros necessários.

2. O cliente stub procedure no cliente recebe a solicitação RPC. Ele empacota os parâmetros em uma mensagem de solicitação e a envia ao servidor por meio do módulo de comunicação do cliente.

3. A mensagem de solicitação é transmitida do cliente para o servidor por meio de protocolos de comunicação, como TCP ou UDP.

4. O módulo de comunicação do servidor recebe a mensagem de solicitação e a encaminha para o server stub procedure no servidor.

5. O server stub procedure no servidor recebe a mensagem de solicitação. Ele realiza o unmarshalling, extraindo os parâmetros da mensagem.

6. O server stub procedure encaminha a chamada para a função apropriada, conhecida como services procedure, com base no nome do procedimento recebido na solicitação.

7. A services procedure executa a lógica do procedimento remoto com os parâmetros recebidos. Ela pode acessar recursos, executar cálculos ou qualquer outra tarefa necessária.

8. Após a conclusão da services procedure, o servidor encapsula o resultado em uma mensagem de resposta.

9. A mensagem de resposta é enviada de volta ao cliente por meio do módulo de comunicação do servidor.

10. O módulo de comunicação do cliente recebe a mensagem de resposta e a entrega ao cliente stub procedure no cliente.

11. O cliente stub procedure realiza o unmarshalling da mensagem de resposta, extraindo o resultado retornado pela services procedure.

12. O cliente recebe o resultado da solicitação RPC e pode continuar sua execução, processando a resposta conforme necessário.

**3. A figura a seguir apresenta o modelo de implementação RMI. Explique o papel de cada um dos seus componentes e explique de que forma o modelo request-reply funciona neste modelo. Além disso, compare este modelo com o modelo RPC.**

**R:**

**Lado do cliente**

**Remote reference module:** É responsável por fornecer uma referência para o objeto remoto que será chamado pelo cliente. Essa referência contém informações necessárias para localizar o objeto remoto, como seu endereço de rede.

**Communication module:** Gerencia a comunicação entre o cliente e o servidor. Ele é responsável por estabelecer a conexão com o servidor, enviar a solicitação e receber a resposta.

**Object A:** É um objeto local no cliente que deseja invocar um procedimento remoto no objeto B no servidor.

**Proxy for B:** É um objeto local no cliente que atua como um representante do objeto remoto B no servidor. O proxy recebe as chamadas de método do objeto A e as transforma em solicitações RPC para o objeto B no servidor.

**Lado do servidor:**

**Skeleton & dispatcher for B's class:** O skeleton é responsável por receber as solicitações RPC enviadas pelo cliente e direcioná-las para a classe apropriada no servidor. Ele lida com o unmarshalling dos parâmetros da solicitação RPC. O dispatcher é responsável por encaminhar a solicitação recebida para o método apropriado dentro da classe B.

**Remote object B:** É o objeto remoto no servidor que contém os métodos que podem ser invocados pelo cliente. O objeto B contém a implementação real dos métodos que serão executados quando chamados pelo cliente.

**Communication module:** Assim como no lado do cliente, o módulo de comunicação no lado do servidor é responsável por gerenciar a comunicação entre o servidor e o cliente. Ele recebe as solicitações do cliente, envia as respostas correspondentes e garante a entrega confiável das mensagens.

**Remote reference module:** No lado do servidor, o remote reference module permite que o servidor forneça uma referência remota para objetos que podem ser acessados por clientes remotos. Essa referência contém informações como o endereço de rede do servidor e o identificador do objeto remoto.

No modelo request-reply, o cliente faz uma chamada de método no objeto A localmente. O proxy para B intercepta essa chamada, empacota-a em uma solicitação RPC e a envia para o servidor através do módulo de comunicação no cliente. O servidor recebe a solicitação através do módulo de comunicação no servidor, o skeleton a recebe e a encaminha para o dispatcher, que redireciona para o método apropriado na classe B. O método é executado e retorna um resultado. O resultado é então encapsulado em uma resposta RPC, que é enviada de volta para o cliente. O cliente recebe a resposta através do módulo de comunicação no cliente, o proxy para B a recebe, realiza o unmarshalling e retorna o resultado para o objeto A local.

Comparando com o modelo RPC em geral, o modelo request-reply é um exemplo específico de implementação do modelo RPC. No modelo RPC, os componentes principais permanecem os mesmos: cliente, servidor, comunicação entre eles e stubs. No entanto, o modelo request-reply é uma forma de implementação em que a comunicação ocorre em um padrão de solicitação e resposta, em que o cliente envia uma solicitação e espera uma resposta do servidor. Isso permite uma interação síncrona entre o cliente e o servidor, onde o cliente aguarda a resposta antes de prosseguir. Outros modelos RPC, como o modelo publish-subscribe, permitem uma comunicação assíncrona, em que o cliente não espera uma resposta imediata do servidor.

## 4. Sobre invocação remota de método:

**1. Por que o desenvolvedor deve descrever a interface remota usando uma IDL?**

**R:**

O desenvolvedor deve descrever a interface remota usando uma IDL (Interface Description Language) para garantir a interoperabilidade entre diferentes sistemas e linguagens de programação. A IDL permite que os desenvolvedores definam a estrutura da interface, especificando os tipos de dados, os parâmetros dos métodos e as operações suportadas. Essa descrição padronizada permite que o código do cliente e do servidor seja gerado

automaticamente a partir da IDL em diferentes linguagens de programação. Dessa forma, a IDL facilita a comunicação e a integração entre componentes remotos.

**2. Qual a relação entre a descrição da interface remota e o stub?**

**R:**

A descrição da interface remota e o stub estão intimamente relacionados. O stub é o componente responsável por representar a interface remota no lado do cliente. Ele atua como um intermediário entre o cliente e o servidor, traduzindo as chamadas de método feitas pelo cliente em solicitações de rede e transmitindo-as para o servidor. Para realizar essa tradução corretamente, o stub precisa ter acesso à descrição da interface remota. A descrição da interface contida na IDL é usada para gerar o código do stub, fornecendo informações sobre os métodos disponíveis, seus parâmetros e tipos de retorno. O stub utiliza essas informações para serializar e desserializar os dados e garantir a conformidade na invocação remota.

**3. Por que o stub tem a sua implementação baseada no padrão de projeto proxy?**

**R:**

O stub é implementado com base no padrão de projeto proxy porque compartilha muitos dos mesmos princípios e funcionalidades. Um proxy é um objeto que atua como um substituto ou representante de outro objeto. No contexto da invocação remota, o stub é o proxy que representa o objeto remoto no cliente. Ele fornece uma interface semelhante à do objeto remoto e lida com os detalhes da comunicação de rede subjacente. Assim como um proxy permite que um cliente acesse um objeto de forma transparente, o stub permite que o cliente invoque métodos remotos como se estivesse chamando métodos locais. Portanto, o uso do padrão de projeto proxy para implementar o stub é uma escolha natural para simplificar a interação cliente-servidor em um ambiente distribuído.

**4. Explique o funcionamento do serviço de nomes e o porquê de seu emprego na invocação remota.**

**R:**

O serviço de nomes desempenha um papel importante na invocação remota, fornecendo um mecanismo para localizar objetos remotos em um sistema distribuído. Ele atua como um diretório centralizado ou distribuído que associa nomes significativos a objetos remotos registrados. Quando um cliente deseja invocar um método em um objeto remoto, ele precisa conhecer a localização desse objeto na rede. O serviço de nomes fornece um mecanismo de pesquisa, onde o cliente pode consultar o nome do objeto desejado e obter informações sobre sua localização, como o endereço IP e a porta. Isso permite que o cliente estabeleça uma conexão com o objeto remoto correto e envie a solicitação RPC para o destino apropriado. O serviço de nomes é importante para facilitar a descoberta e o acesso a

objetos remotos em um ambiente distribuído, tornando a invocação remota mais eficiente e escalável.

## 5. Na construção de uma aplicação com objetos distribuídos:

**1. Qual a função do stub e do skeleton? Por que também são conhecidos como proxy?**

**R:**

O stub e o skeleton são componentes-chave na construção de uma aplicação com objetos distribuídos e são conhecidos como proxy devido à sua função semelhante ao padrão de projeto proxy.

O stub é responsável por representar a interface remota no lado do cliente. Ele age como um substituto do objeto remoto e fornece uma interface local para o cliente interagir com o objeto remoto. O stub recebe as chamadas de método do cliente e as transforma em solicitações de rede, que são enviadas para o servidor. Ele é responsável por serializar os parâmetros do método, transmiti-los por meio da rede para o skeleton no servidor e aguardar a resposta.

O skeleton é o equivalente do stub no lado do servidor. Ele recebe as solicitações de rede enviadas pelo stub, desserializa os parâmetros, invoca o método correspondente no objeto remoto e retorna o resultado. O skeleton é responsável por receber as solicitações do cliente, tratar a comunicação de rede subjacente e garantir que o objeto remoto seja invocado corretamente.

**2. Qual a função do módulo de referência remota?**

**R:**

O módulo de referência remota tem a função de gerenciar as referências aos objetos distribuídos. Ele fornece um mecanismo para que os clientes obtenham uma referência ao objeto remoto desejado. Essa referência contém informações necessárias para localizar e acessar o objeto remoto, como o endereço de rede (como o IP e a porta) do servidor onde o objeto reside. O módulo de referência remota também pode fornecer recursos adicionais, como a resolução de nomes de objetos por meio de um serviço de nomes. Ele é responsável por encapsular os detalhes da localização e comunicação com o objeto remoto, permitindo que o cliente interaja com o objeto como se estivesse localmente.

**3. Qual o papel da referência ao objeto? Explique cada uma das informações que a compõem.**

**R:**

A referência ao objeto é um objeto que representa uma conexão com um objeto remoto específico em um ambiente distribuído. Ela contém informações importantes para identificar e acessar o objeto remoto. As principais informações que compõem uma referência ao objeto são:

- Endereço de rede: Essa informação inclui o endereço IP e a porta onde o objeto remoto está hospedado. Permite que o cliente estabeleça uma conexão com o servidor correto que contém o objeto remoto.

- Identificador do objeto: É um valor único que identifica o objeto remoto no contexto do servidor. Permite ao servidor identificar o objeto correto e encaminhar as chamadas de método para ele.

- Informações adicionais: Além das informações básicas de localização, a referência ao objeto pode conter informações adicionais, como protocolos de comunicação suportados, configurações de segurança ou qualquer outra informação necessária para a interação correta com o objeto remoto.

Essas informações são encapsuladas na referência ao objeto e permitem que o cliente estabeleça uma conexão com o objeto remoto e invoque seus métodos de forma transparente.

**6. As semânticas de invocação do RPC são principalmente três. Explique quais são e como funciona cada uma delas. Inclua na sua discussão os modelos de falhas que empregam, explicando suas características e a relação destes com as respectivas semânticas.**

**R:**

As semânticas de invocação do RPC são síncrona, assíncrona e at-most-once.

- Na semântica síncrona, o cliente faz uma chamada remota e aguarda pela resposta antes de continuar a execução. Se ocorrer uma falha de comunicação, o cliente pode ficar bloqueado ou receber um erro indicando a falha. O modelo de falha comum é lidar com timeouts para tratar falhas de comunicação.

- Na semântica assíncrona, o cliente faz uma chamada remota e continua sua execução imediatamente, sem aguardar pela resposta. O servidor processa a chamada em segundo plano e envia a resposta posteriormente. Se ocorrer uma falha de comunicação, o cliente não é bloqueado imediatamente. O modelo de falha

pode envolver callbacks ou mecanismos de polling para tratar falhas de comunicação.

- Na semântica at-most-once, é garantido que uma chamada remota seja executada no máximo uma vez no servidor. Isso implica em retransmitir a chamada em caso de falhas de comunicação. O modelo de falha envolve mecanismos para detectar e evitar chamadas duplicadas tanto no cliente quanto no servidor.

Essas semânticas permitem diferentes abordagens para lidar com falhas de comunicação em chamadas remotas, oferecendo diferentes níveis de garantia na execução dos procedimentos remotos.

**7. Uma aplicação distribuída desenvolvida com base na distribuição de processos é diferente de uma construída com base na distribuição de objetos? A primeira forma de distribuição tem vantagens sobre a segunda, ou vice-versa? Explique as diferenças, vantagens e desvantagens.**

**R:**

A distribuição de processos e a distribuição de objetos são abordagens diferentes para o desenvolvimento de aplicativos distribuídos.

- Na distribuição de processos, os componentes da aplicação são divididos em processos separados, com comunicação entre eles por meio de chamadas de procedimento remoto (RPC) ou troca de mensagens assíncronas. Essa abordagem oferece flexibilidade e tolerância a falhas, mas pode ter um overhead de comunicação e dificuldades no gerenciamento de estado compartilhado.

- Na distribuição de objetos, os componentes são encapsulados em objetos distribuídos, com comunicação orientada a objetos por meio de chamadas de métodos. Isso facilita o encapsulamento e a reutilização de código, com comunicação mais simplificada. No entanto, pode ser mais complexo escalar granularmente e gerenciar o estado distribuído.

Ambas as abordagens têm vantagens e desvantagens específicas. A distribuição de processos oferece flexibilidade e tolerância a falhas, mas com maior overhead de comunicação. A distribuição de objetos simplifica a comunicação e o encapsulamento, mas pode ser desafiadora para escala granular e gerenciamento de estado distribuído. A escolha entre as abordagens depende das necessidades e características específicas da aplicação distribuída.

## 8. Um dos mecanismos de comunicação mais usados em sistemas distribuídos é o de chamada remota de procedimento (RPC).

**1. Que vantagens o uso de chamada remota de procedimentos pode oferecer ao programador, se comparado com o uso direto de uma API de mensagens como a de sockets? Há desvantagens? Se sim, quais?**

**R:**

O uso de chamada remota de procedimentos (RPC) pode oferecer várias vantagens em comparação com o uso direto de uma API de mensagens, como a de sockets:

- Abstração de comunicação: O RPC oferece uma abstração de mais alto nível para comunicação entre processos distribuídos. Ele permite que o programador trabalhe com chamadas de procedimentos remotos, sem se preocupar com detalhes de codificação e decodificação de mensagens ou gerenciamento de conexões de rede.

- Simplificação do desenvolvimento: O RPC simplifica o desenvolvimento de aplicativos distribuídos, pois permite que os programadores trabalhem com uma interface mais familiar, semelhante a chamadas de procedimentos locais. Isso facilita a escrita de código, reduzindo a complexidade e melhorando a produtividade.

- Reuso de código: Com o RPC, é possível reutilizar código existente, uma vez que as chamadas remotas podem ser mapeadas para funções ou métodos já implementados. Isso permite que os programadores aproveitem a lógica de negócios já existente em diferentes partes do sistema distribuído.

No entanto, também existem algumas desvantagens no uso de RPC:

- Dependência de infraestrutura: O uso de RPC geralmente requer uma infraestrutura específica, como um middleware de RPC, que pode adicionar complexidade ao sistema distribuído. É necessário configurar e gerenciar essa infraestrutura para garantir a comunicação correta entre os componentes.

- Restrições de linguagem e plataforma: Alguns sistemas RPC são restritos a determinadas linguagens de programação ou plataformas específicas, limitando a interoperabilidade entre diferentes componentes distribuídos.

**2. Em relação ao RMI, que vantagens este pode oferecer ao programador, se comparado com o uso direto de uma API de mensagens como a de sockets? E quando comparado ao RPC?**

**R:**

O RMI (Remote Method Invocation) é uma forma específica de RPC que é projetada para ser usada em sistemas distribuídos baseados em objetos. Comparado ao uso direto de uma API de mensagens, assim como ao RPC em geral, o RMI oferece vantagens adicionais:

- Orientação a objetos: O RMI permite que os programadores chamem métodos em objetos remotos, abstraindo ainda mais a comunicação entre os componentes distribuídos. Isso facilita o desenvolvimento de sistemas distribuídos baseados em objetos e permite uma maior expressividade no código.

- Serialização transparente de objetos: O RMI lida com a serialização e desserialização automática de objetos durante a comunicação remota. Isso permite que os programadores passem objetos complexos como parâmetros ou retornos de chamadas remotas sem a necessidade de codificar manualmente a serialização e desserialização dos objetos.

- Integração com a plataforma Java: O RMI é uma tecnologia nativa da plataforma Java, o que significa que é facilmente integrada com outros recursos e bibliotecas do ecossistema Java. Isso proporciona uma maior facilidade de desenvolvimento e suporte de ferramentas para os desenvolvedores Java.

Em resumo, o RMI oferece vantagens adicionais em relação ao uso direto de uma API de mensagens ou ao RPC em geral, como a orientação a objetos e a serialização transparente de objetos, além de uma melhor integração com a plataforma Java.

**9. Três padrões arquiteturais importantes para sistemas distribuídos são: “Proxy”, “Layers” e “Broker”. Explique como cada um deles funciona, incluindo diagramas, exemplos de aplicação e pseudocódigos.**

**R:**

**Padrão Proxy:**

O padrão Proxy é usado para fornecer um substituto ou representante de outro objeto para controlar o acesso a esse objeto. Ele age como uma interface para o objeto real, permitindo o controle sobre sua criação, acesso e comportamento.

**Funcionamento:**

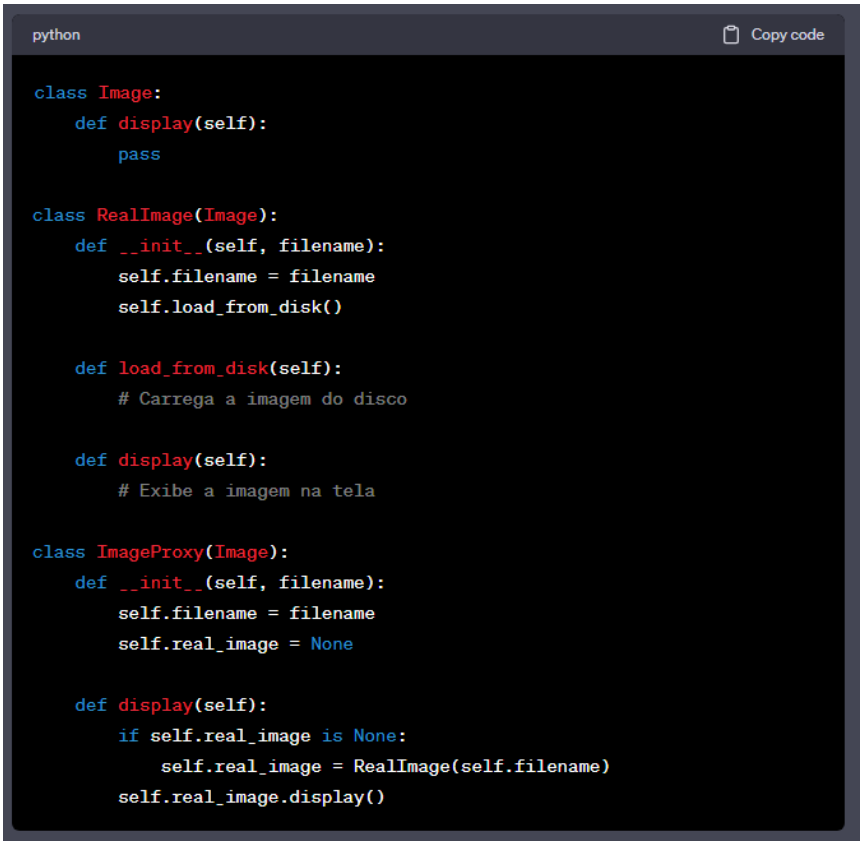
O padrão Proxy envolve a criação de uma classe Proxy que possui a mesma interface que o objeto real. O Proxy recebe as solicitações do cliente e as encaminha para o objeto real. Ele pode realizar ações adicionais antes ou depois de repassar a chamada ao objeto real.

**Exemplo de aplicação:**

Um exemplo comum de aplicação do padrão Proxy é o carregamento preguiçoso (lazy loading) de recursos pesados, como imagens em uma página da web. O Proxy atua como um substituto da imagem real e carrega a imagem somente quando necessário. Isso

melhora o desempenho da página, pois recursos pesados não são carregados imediatamente.

**Pseudocódigo:**




## Padrão Layers:

O padrão Layers (camadas) é usado para dividir um sistema em camadas distintas, cada uma sendo responsável por uma funcionalidade específica. Cada camada oferece serviços para a camada acima dela e se comunica com a camada abaixo.

**Funcionamento:**

O padrão Layers envolve a criação de várias camadas que se comunicam entre si por meio de interfaces bem definidas. Cada camada realiza uma funcionalidade específica e oculta sua implementação das camadas adjacentes. As camadas superiores dependem das camadas inferiores para fornecer serviços.

**Exemplo de aplicação:**

Um exemplo comum de aplicação do padrão Layers é a arquitetura de software em três camadas: apresentação, lógica de negócios e acesso a dados. Cada camada possui uma responsabilidade específica e se comunica com as camadas adjacentes. Isso promove a modularidade, reutilização de código e separação de preocupações.

**Pseudocódigo:**

```python
class PresentationLayer:
    def __init__(self, business_layer):
        self.business_layer = business_layer

    def perform_action(self):
        # Realiza ações da camada de apresentação
        self.business_layer.process_data()

class BusinessLayer:
    def __init__(self, data_access_layer):
        self.data_access_layer = data_access_layer

    def process_data(self):
        # Realiza ações da camada de lógica de negócios
        self.data_access_layer.save_data()

class DataAccessLayer:
    def save_data(self):
        # Realiza ações da camada de acesso a dados
```

## Padrão Broker:

O padrão Broker (corretor) é usado para fornecer um intermediário centralizado que coordena a comunicação entre componentes distribuídos em um sistema. O Broker atua como um ponto central de troca de mensagens, roteamento e gerenciamento de serviços.

**Funcionamento:**

O padrão Broker envolve a criação de um componente central chamado Broker, que recebe solicitações e mensagens de outros componentes distribuídos. O Broker roteia as mensagens para os componentes relevantes e pode aplicar regras de segurança, escalonamento e transformação de dados.

**Exemplo de aplicação:**

Um exemplo comum de aplicação do padrão Broker é um sistema de mensagens em um ambiente distribuído. O Broker atua como um intermediário entre os remetentes e os destinatários das mensagens, gerenciando a entrega, roteamento e segurança das mensagens.

**Pseudocódigo:**

```python
class MessageHandler:
    def handle_message(self, message):
        pass

class ComponentA(MessageHandler):
    def handle_message(self, message):
        # Lida com a mensagem recebida por ComponentA

class ComponentB(MessageHandler):
    def handle_message(self, message):
        # Lida com a mensagem recebida por ComponentB

class Broker:
    def __init__(self):
        self.handlers = {}

    def register_handler(self, message_type, handler):
        self.handlers[message_type] = handler

    def handle_message(self, message_type, message):
        handler = self.handlers.get(message_type)
        if handler is not None:
            handler.handle_message(message)
```

**10. A www pode ser considerada um Sistema Distribuído. Neste contexto, considere o serviço de páginas, no qual há dois processos, A e B. Suponha que A é o cliente e B é o servidor e que uma invocação consiste de uma mensagem de requisição de A para B, seguida do processamento da requisição feito pelo B, seguido ainda por uma mensagem de resposta de B para A. Explique, com base neste cenário, as classes de falhas que podem ser apresentadas por uma invocação.**

**R:**

No contexto da World Wide Web (WWW), que é um sistema distribuído, podemos considerar o serviço de páginas, onde dois processos estão envolvidos: o processo A, que atua como cliente, e o processo B, que atua como servidor. Uma invocação nesse cenário ocorre quando o processo A envia uma mensagem de requisição para o processo B, que processa a requisição e envia uma mensagem de resposta de volta para o processo A.

Nesse contexto, algumas classes de falhas que podem ocorrer durante uma invocação incluem:

- **Falhas de comunicação:** Podem ocorrer falhas no envio ou recebimento das mensagens entre os processos A e B. Isso pode ser causado por problemas na rede, como perda de pacotes, latência excessiva ou indisponibilidade temporária da rede.

- **Falhas de tempo limite:** Se a comunicação entre os processos A e B não ocorrer dentro de um tempo limite pré-definido, uma falha de tempo limite pode ocorrer. Isso pode indicar que o processo B está indisponível ou que ocorreu algum problema na comunicação.

- **Falhas no processamento da requisição:** O processo B pode encontrar erros ou problemas ao processar a requisição recebida do processo A. Isso pode ser causado por bugs no código do servidor, erros de lógica ou recursos indisponíveis.

- **Falhas no envio da resposta:** Após processar a requisição, o processo B pode enfrentar problemas ao enviar a resposta de volta para o processo A. Isso pode ocorrer devido a erros de comunicação, falhas na rede ou problemas no código do servidor.

- **Falhas no tratamento de erros:** Se ocorrerem erros durante o processamento da requisição, o processo B deve ser capaz de lidar com essas situações adequadamente. No entanto, se o tratamento de erros não for implementado corretamente, podem ocorrer falhas que levam a comportamentos indesejados ou até mesmo à interrupção do serviço.

**11. O modelo de objetos distribuídos estende o modelo de objetos básico em, principalmente, quatro pontos: localidade dos objetos, chamadas de métodos, referência, interface. Descreva detalhadamente estes quatro pontos**.

**R:**

**Localidade dos objetos:**
No modelo de objetos distribuídos, os objetos podem residir em diferentes máquinas ou nós em uma rede. Isso significa que um objeto pode estar localizado em um processo ou máquina diferente daquele em que o código cliente está sendo executado. A localidade dos objetos distribuídos é uma característica essencial para permitir a comunicação e a interação entre objetos em diferentes nós.

**Chamadas de métodos:**
No modelo de objetos distribuídos, as chamadas de métodos são usadas para interagir com os objetos distribuídos. Quando um cliente deseja invocar um método em um objeto distribuído, ele emite uma solicitação para o objeto remoto. Essa solicitação inclui o nome do método e os parâmetros necessários. O objeto remoto recebe a solicitação, executa o

método e retorna o resultado para o cliente. As chamadas de métodos podem ser síncronas (bloqueantes) ou assíncronas (não bloqueantes), dependendo dos requisitos da aplicação.

**Referência:**
Uma referência a um objeto distribuído é uma estrutura de dados que identifica exclusivamente o objeto remoto e fornece os meios para o cliente se comunicar com o objeto. A referência contém informações, como o endereço de rede do objeto remoto, o identificador do objeto e outros detalhes necessários para estabelecer a conexão e realizar a chamada de método correta. A referência é usada pelo cliente para enviar solicitações de chamada de método ao objeto remoto e receber respostas.

**Interface:**
A interface em um modelo de objetos distribuídos descreve os métodos que podem ser invocados em um objeto remoto. Ela define a assinatura dos métodos, incluindo os tipos de parâmetros e o tipo de retorno. A interface é uma especificação que permite ao cliente conhecer os métodos disponíveis em um objeto remoto e como chamá-los. Ela atua como um contrato entre o cliente e o objeto remoto, garantindo a consistência e a interoperabilidade entre os diferentes componentes distribuídos.