WebSphere® eXtreme Scale Versão 7.0
Visão Geral do Produto

Visão Geral do Produto WebSphere eXtreme Scale

WebSphere® eXtreme Scale Versão 7.0
Visão Geral do Produto

Visão Geral do Produto WebSphere eXtreme Scale

Essa edição aplica-se à versão 7, release 0, do WebSphere eXtreme Scale e a todos os releases e modificações subsequentes, até que seja indicado de outra forma em novas edições.

Índice

Figuras

Tabelas

Sobre o *Visão Geral do Produto*

O conjunto da documentação do WebSphere eXtreme Scale inclui três volumes que fornecem as informações necessárias para utilizar, programar e administrar o produto WebSphere eXtreme Scale.

Biblioteca do WebSphere eXtreme Scale

A biblioteca do WebSphere eXtreme Scale contém os seguintes livros:

- O *Guia de Administração* contém as informações necessárias para os administradores de sistema, incluindo como planejar implementações do aplicativo, planejar capacidade, instalar e configurar o produto, iniciar e parar servidores, monitorar o ambiente e proteger o ambiente.
- O *Guia de Programação* contém informações para desenvolvedores de aplicativos sobre como desenvolver aplicativos para o WebSphere eXtreme Scale utilizando as informações da API incluídas.
- O *Visão Geral do Produto* contém um visualização de alto nível dos conceitos do WebSphere eXtreme Scale, incluindo cenários de caso de uso e tutoriais.

Para fazer download dos manuais, vá para a Página da Biblioteca do WebSphere eXtreme Scale.

Também é possível acessar as mesmas informações nesta biblioteca no Centro de Informações do WebSphere eXtreme Scale.

Quem Deve Utilizar este Manual

Este manual é destinado a qualquer pessoa que esteja interessada em aprender sobre o WebSphere eXtreme Scale.

Como este Manual Está Estruturado

O manual contém informações sobre os seguintes tópicos principais:

- **Capítulo 1** inclui uma visão geral do WebSphere eXtreme Scale
- **Capítulo 2** inclui informações sobre os conceitos de armazenamento em cache no produto.
- **Capítulo 3** inclui informações sobre a integração de cache.
- **Capítulo 4** inclui informações sobre a escalabilidade.
- **Capítulo 5** inclui informações sobre a disponibilidade.
- **Capítulo 6** inclui informações sobre a segurança.
- **Capítulo 7** inclui informações sobre o processamento de transação.
- **Capítulo 8** inclui tutoriais para os conceitos básicos do produto.
- **Capítulo 9** inclui o glossário do produto.

Obtendo Atualizações para este Manual

É possível obter as atualizações para esse manual ao fazer download da versão mais recente da Página da Biblioteca do WebSphere eXtreme Scale.

Como Enviar Seus Comentários

Entre em contato com a equipe de documentação. Você localizou o que precisava? O conteúdo era exato e completo? Envie seus comentários sobre esta documentação por e-mail para wasdoc@us.ibm.com.

Capítulo 1. WebSphere eXtreme ScaleVisão Geral

WebSphere eXtreme Scale é uma grade de dados na memória, elástica e escalável. Ele dinamicamente armazena em cache, particiona, replica e gerencia dados do aplicativo e lógica de negócios em múltiplos servidores. O WebSphere eXtreme Scale executa grandes volumes de processamento de transações com alta eficiência e escalabilidade linear, e fornece qualidades de serviço como integridade transacional, alta disponibilidade e tempos de resposta previsíveis.

A escalabilidade elástica do WebSphere eXtreme Scale é capacitada através do armazenamento em cache do objeto distribuído. O termo elástico significa que a grade monitora e gerencia a si própria, permite escalar no sentido ascendente e descendente, e se recupera automaticamente de falhas. O escalamento no sentido ascendente permite incluir capacidade de memória enquanto a grade estiver em execução, sem exigir uma reinicialização. De maneira oposta, efetuar scale-in remove a capacidade da memória imediata.

O WebSphere eXtreme Scale pode ser usado de diferentes formas. Ele pode ser usado como um cache muito poderoso ou como uma forma de um espaço de processamento de banco de dados de memória para gerenciar o estado de aplicativos ou como uma plataforma para construção de aplicativos poderosos de Extreme Transaction Processing (XTP).

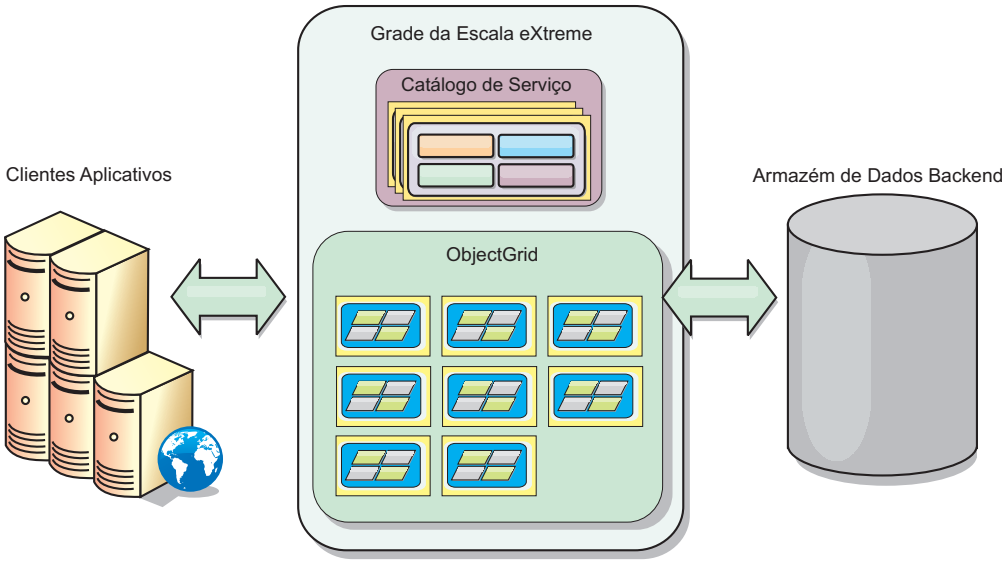
Porém, é importante notar que o eXtreme Scale não deve ser considerado, na realidade, um banco de dados de memória já que, muitas vezes, algumas de suas complexidades são tão simples de serem manipuladas que até mesmo o eXtreme Scale pode gerenciá-las. Você pode ter algumas vantagens iguais com os dois cenários porque cada um deles está na memória, mas se alguma máquina do banco de dados de memória falhar, o problema não será imediatamente resolvido. Esse evento será realmente desastroso se o seu ambiente inteiro estiver nessa máquina.

Para resolver o problema desse tipo de falha, o eXtreme Scale divide o conjunto de dados fornecido em partições (equivalentes aos esquemas de árvore restritos), cada um delas sendo uma cópia primária (shard) e também shards de réplica para fazer backup dos dados. Um banco de dados de memória não fornece esse tipo de funcionalidade porque ele não é estruturado e dinâmico dessa maneira, exigindo que você faça manualmente o que seria feito automaticamente pelo eXtreme Scale. Além disso, como um banco de dados de memória é logicamente um banco de dados, ele pode permitir operações SQL e ter maior desempenho em termos de velocidade de processamento em comparação com os bancos de dados que não estão na memória. O WebSphere eXtreme Scale possui sua própria linguagem de consulta em vez de suporte SQL, mas por ser mais significativamente elástico, ele permite particionamento de dados e fornece recuperação de falha confiável.

O recurso de cache write-behind permite que o WebSphere eXtreme Scale sirva como um cache front-end para um banco de dados para aumentar o rendimento enquanto reduz a carga do banco de dados e a contenção. O WebSphere eXtreme Scale fornece scale-in previsível e scale-out a um custo de processamento também previsível.

A imagem a seguir mostra que em um ambiente de cache distribuído e coerente, os clientes da grade do eXtreme Scale enviam e recebem dados da grade, que podem ser automaticamente sincronizados com um armazém de dados backend. O cache é

coerente pois todos os clientes vêem dados no cache. Cada parte dos dados é armazenada em exatamente um servidor gravável no cache, evitando a cópia desnecessária de registros que podem conter potencialmente diferentes versões dos dados. Um cache coerente mantém mais dados conforme mais servidores são incluídos na grade, e escala linearmente conforme o tamanho da grade cresce. Os dados também podem ser opcionalmente replicados para se obter tolerância a falhas opcional.



O WebSphere eXtreme Scale possui servidores que oferecem sua grade de dados na memória. Esses servidores podem executar dentro do WebSphere Application Server ou em uma Java™ Virtual Machines simples do Java Standard Edition (J2SE), permitindo mais de uma dessas por máquina física. Assim, a grade de dados na memória pode ser muito grande. Ela não é limitada pela (nem causará impacto) memória ou pelo espaço de endereço do aplicativo ou servidor de aplicativos. A memória pode ser a soma da memória de centena, ou milhares, de Java Virtual Machines em execução em muitas máquinas diferentes.

Assim como no espaço de processamento de banco de dados de memória, o WebSphere eXtreme Scale ainda pode ser apoiado pelo disco, banco de dados, ou ambos.

Como eXtreme Scale fornece diversas APIs Java, muitos usos não necessitam de programação do usuário mas apenas de configuração e implementação na infraestrutura do seu WebSphere.

Paradigma Básico

O paradigma fundamental da grade é um par de chave-valor, no qual a grade armazena valores (objetos Java), com uma chave associada (outro objeto Java), através do qual o valor é subsequentemente recuperado. No eXtreme Scale, um mapa é um contêiner para estes pares de chave-valor.

O WebSphere eXtreme Scale oferece uma quantidade de configurações de grade, desde um único cache local simples até um cache grande distribuído, usando múltiplas JVMs e/ou servidores.

Além do armazenamento de objetos Java simples, os objetos com relacionamentos podem ser armazenados e uma linguagem de consulta do tipo SQL (SELECT ... FROM ... WHERE) pode ser usada para recuperá-los. Por exemplo, um objeto de ordem pode ter um objeto cliente e múltiplos objetos de item associados a ele. O WebSphere eXtreme Scale suporta relacionamentos de um para um, um para muitos, muitos para um e muitos para muitos.

O WebSphere eXtreme Scale também suporta uma interface de programação EntityManager para armazenamento de entidades no cache, como entidades do Java Enterprise Edition. Os relacionamentos de entidades podem ser automaticamente descobertos a partir do arquivo XML do descritor ou de anotações nas classes Java. Assim, uma entidade pode ser recuperada do cache pela chave primária utilizando o método de localização EntityManager. As entidades podem ser persistidas para a grade, assim como removidas dela, tudo dentro de um limite de transação.

O WebSphere eXtreme Scale fornece capacidades extreme transaction processing (XTP) que assegurem uma infraestrutura de aplicativo mais inteligente para suportar seus aplicativos críticos de negócios de maior demanda. Você pode vencer as limitações de desempenho de TI tradicional para gerar os níveis de escala global, as eficiências do processo e inteligência de negócios necessárias para resultados mais inteligentes e para vantagens sustentáveis de negócios sobre os concorrentes.

Com o suporte para WebSphere Real Time, a oferta Java em tempo real líder de setor, o WebSphere eXtreme Scale permite que os aplicativos XTP tenham tempos de resposta consistentes e previsíveis. Consulte as informações sobre o Suporte em Tempo Real no *Guia de Administração*.

Antes da implementação do eXtreme Scale em um ambiente de produção, há diversas opções a serem consideradas, incluindo a quantidade de servidores a serem usados, a quantidade de armazenamento em cada servidor e a replicação síncrona ou assíncrona.

Considere um exemplo distribuído em que a chave é um nome alfabético simples. O cache pode ser dividido em 4 partições por chave: a partição 1 para chaves que começam com A-E, partição 2 para chaves que começam com F-L, e assim por diante. Para disponibilidade, uma partição possui (está armazenada em) um shard primário e um shard de réplica. As alterações nos dados do cache são feitas no shard primário, e replicadas para o shard secundário. Para um cache distribuído, (grade ou ObjectGrid no vocabulário do eXtreme Scale), você configura a quantidade de servidores eXtreme Scale que irão conter os dados da grade, e o eXtreme Scale distribui os dados em shards por estas instâncias do servidor. Para disponibilidade, os shards de réplica são colocados em máquinas separadas a partir de shards primários.

O WebSphere eXtreme Scale usa um serviço de catálogo para localizar o shard primário para cada chave. Ele manipula a movimentação de shards entre servidores do eXtreme Scale ou as máquinas físicas que os contêm falham e são subsequentemente recuperadas. Por exemplo, se o servidor que contém um shard de réplica falhar, o eXtreme Scale será alocado a um novo shard de réplica. Se um servidor que contém um shard primário falhar, o shard de réplica é promovido como shard primário e, como antes, um novo shard de réplica é construído.

A interface de programação mais simples do eXtreme Scale é o ObjectMap, que é uma interface de mapas simples: map.put(key,value) para colocar um valor no cache, e map.get(key) para recuperar subsequentemente o valor.

Além de armazenar em cache objetos Java simples, os objetos com relacionamentos podem ser armazenados em cache, e uma linguagem de consulta do tipo SQL (SELECT ... FROM ... WHERE) pode ser usado para recuperá-los. Por exemplo, um objeto de ordem pode ter um objeto cliente e múltiplos objetos de item associados a ele. O WebSphere eXtreme Scale suporta relacionamentos de um para um, um para muitos, muitos para um e muitos para muitos. Uma interface de programação EntityManager também é suportada no eXtreme Scale, para armazenamento de entidades no cache, como entidades do Java Enterprise Edition. Os relacionamentos de entidades podem ser automaticamente descobertos a partir de um arquivo XML do descritor de entidades ou de anotações nas classes Java. Assim, uma entidade pode ser recuperada do cache através da chave primária usando o método de localização EntityManager. As entidades podem ser persistidas para a grade, assim como removidas dela, tudo dentro de um limite de transação.

Recursos Novos e Reprovados neste Release

O WebSphere eXtreme Scale inclui vários novos recursos na Versão 7.0, incluindo integração com o cache dinâmico, mapas de matriz de bytes, etc.

O Quê Há de Novo no WebSphere eXtreme Scale Versão 7.0

Tabela 1. Novos Recursos no WebSphere eXtreme Scale Versão 7.0

Recurso	Descrição
Integração do Cache Dinâmico	O provedor de cache dinâmico permite que os aplicativos que usam o cache dinâmico do WebSphere Application Server usem com eficiência os recursos avançados e os aprimoramentos de desempenho do WebSphere eXtreme Scale. Esse recurso resulta em uma melhor qualidade de serviço, escalabilidade linear e alta disponibilidade em relação a uma variedade de aplicativos de negócios com alterações mínimas invasivas. Consulte as informações sobre o cache dinâmico no *Visão Geral do Produto*.
Mapas de Matriz de Byte	Com os mapas de matriz de byte, o aplicativo pode armazenar o valor de um par valor-chave em uma matriz de byte em vez de em forma de objeto, que reduz a área de cobertura de memória que um gráfico grande de objetos pode consumir. Consulte informações sobre os mapas de matriz de bytes no *Guia de Programação*.
WebSphere Real Time	Com o suporte para o WebSphere Real Time, a oferta líder de mercado em tempo real Java, O WebSphere eXtreme Scale permite que os aplicativos XTP tenham tempos de resposta mais consistentes e previsíveis. Consulte as informações sobre o Suporte em Tempo Real no *Guia de Administração*.
Ativação Métrica	O WebSphere eXtreme Scale inclui implementações de adaptadores de acesso métrico para melhorar a integração com o IBM® Tivoli Monitoring (ITM) e Hyperic HQ, permitindo um insight abrangente sobre o comportamento operacional das soluções de negócios. Consulte as informações sobre as ferramentas de monitoramento do fornecedor para ativação métrica no*Guia de Administração*..
Mapas Dinâmicos	A construção de aplicativos multitenant foi amplamente simplificada. Os modelos de mapas permitem que os aplicativos criem novos mapas on demand, evitando a necessidade de usar discriminadores de aplicativos nas chaves ou de criar mapas extras que nunca poderão ser usados. Consulte informações sobre os mapas dinâmicos no *Guia de Programação*.
Tempo Limite do Pedido	O WebSphere eXtreme Scale foi aprimorado para manipular várias tarefas de nova tentativa e de lógica de exceção comuns dentro o middleware de grade. O tempo limite de pedido para os clientes tira a dificuldade dos desenvolvedores para elaborar a lógica de nova tentativa para a maioria das operações de interação de mapa. A maioria das condições que podem ser tentadas novamente são agora manipuladas automaticamente, para que você possa focar nos aspectos lógicos de negócios da implementação de aplicativo. Consulte informações sobre o tempo limite de pedido no *Guia de Administração*.
Índice Composto	Esse recurso simplifica o uso de índice ao consultar vários atributos e reduz a sobrecarga de possuir vários índices definidos. A consulta também foi otimizada para obter vantagem de índices compostos. Consulte as informações sobre índice composto no *Guia de Programação*..

Recursos Reprovados

Tabela 2. Recursos Reprovados

Reprovação	Ação de migração recomendada
Recurso de Particionamento (WPF): O recurso de particionamento é um conjunto de APIs de programação que permitem que os aplicativos Java EE suportem armazenamento em cluster assimétrico.	Os recursos do WPF podem ser executados alternativamente no WebSphere eXtreme Scale.
StreamQuery: Uma consulta contínua dos dados transferidos armazenados nos mapas do ObjectGrid.	Nenhum(a)
Configuração de Grade Estática: Uma topologia estática baseada em cluster que usa o arquivo XML de implementação de cluster.	Substituído com a topologia de implementação dinâmica melhorada para gerenciar grades de dados grandes.
Propriedades do Sistema Reprovadas: As propriedades do sistema para especificar os arquivos de propriedades do servidor e do cliente foram reprovadas.	Ainda é possível usar esses argumentos, mas altere as propriedades do sistema para os novos valores. Consulte as informações sobre os arquivos de propriedades no *Guia de Administração* para obter mais informações.

Teste Gratuito

Para começar a usar o WebSphere eXtreme Scale, faça download de uma versão gratuita para testar. É possível desenvolver aplicativos inovadores de alto desempenho ao estender o conceito de armazenamento em cache de dados usando os recursos avançados.

Download de Teste

Para fazer download de uma versão de teste gratuita do eXtreme Scale, vá para Download de Teste do eXtreme Scale.

Depois de efetuar o download e a descompactação da versão de teste do eXtreme Scale, navegue até o diretório gettingstarted, e leia GETTINGSTARTED_README.txt. Este tutorial permite que você comece a usar o eXtreme Scale, crie uma grade em diversos servidores e execute alguns aplicativos simples para armazenar e recuperar dados em uma grade. Antes da implementação do eXtreme Scale em um ambiente de produção, há diversas opções a serem consideradas, incluindo a quantidade de servidores a serem usados, a quantidade de armazenamento em cada servidor e a replicação síncrona ou assíncrona.

Trabalhando com o WebSphere eXtreme Scale

WebSphere eXtreme Scale é uma grade de dados na memória, elástica e escalável. Ele dinamicamente armazena em cache, particiona, replica e gerencia dados do aplicativo e lógica de negócios em múltiplos servidores.

Como isso não é um banco de dados de memória, é necessário considerar os requisitos de configuração específicos para o eXtreme Scale. A primeira etapa para implementar uma grade de dados do eXtreme Scale é iniciar um grupo de núcleos e o serviço de catálogo, que atuará como coordenador para todas as outras Java

Virtual Machines participando da grade e gerenciando as informações de configuração. Os processos do WebSphere eXtreme Scale são iniciados com chamadas de script de comando simples a partir da linha de comandos.

A próxima etapa é iniciar os processos do servidor do WebSphere eXtreme Scale para a grade para armazenar e recuperar dados. Conforme os servidores são iniciados, eles automaticamente se registram no grupo de núcleos e o serviço de catálogo permitido que eles cooperem no fornecimento de serviços de grade. Uma quantidade maior de servidores aumenta tanto a capacidade quanto a confiabilidade da grade.

Uma grade local é uma grade simples de uma única instância em que todos os dados estão em uma grade. Para usar eficientemente o eXtreme Scale como um espaço de processamento de banco de dados de memória, poderá configurar e implementar uma grade distribuída. Os dados da grade distribuída são espalhados por vários servidores do eXtreme Scale que a contêm, sendo que cada servidor contém somente parte dos dados, chamada de partição.

Um parâmetro-chave de configuração de grade distribuída é a quantidade de partições na grade. Os dados da grade são particionados nesta quantidades de subconjuntos, cada um dos quais chamados de partição. O serviço de catálogo localiza a partição para um determinado datum com base em sua chave. A quantidade de partições afeta diretamente a capacidade e escalabilidade da grade. Um servidor pode conter uma ou mais partições da grade. Dessa forma, o espaço de memória do servidor limita o tamanho de uma partição. De maneira contrária, aumentar a quantidade de partições aumenta a capacidade da grade. A capacidade máxima de uma grade é a quantidade de partições vezes o tamanho da memória utilizável de um servidor, que pode ser uma JVM.

Os dados da partição são armazenados em um shard. Para disponibilidade, uma grade pode ser configurada com réplicas, que pode ser síncrona ou assíncrona. Alterações nos dados da grade são feitas no shard primário, e replicadas nos shards de réplica. A memória total consumida/necessária por uma grade é, dessa forma, o tamanho da grade vezes (1 (para o primário) + a quantidade de réplicas).

O WebSphere eXtreme Scale distribui shards de grade sobre a quantidade de servidores que contêm a grade. Estes servidores podem estar nas mesmas máquinas físicas e/ou em máquinas separadas. Para disponibilidade, os shards de réplica são colocados em máquinas separadas a partir de shards primários.

O WebSphere eXtreme Scale monitora o status de seis servidores e as movimentações dos shards entre eles e/ou suas máquinas físicas que os contêm falham e subsequentemente recuperam. Por exemplo, se o servidor que contém um shard de réplica falhar, o eXtreme Scale alocará um novo shard de réplica, e replicará os dados do primário na nova réplica. Se um servidor contendo um shard primário falhar, o shard de réplica é promovido como shard primário e, como antes, um novo shard de réplica é construído. Se você iniciar um servidor adicional para a grade, os shards serão distribuídos entre todos os servidores para que a carga em cada um seja o mais equilibrada possível. Isto é chamado de expansão. De maneira semelhante, para escalar no sentido descendente, você pode interromper um dos servidores para reduzir os recursos consumidos por uma grade, e novamente os shards serão equilibrados entre os servidores restantes, como em uma situação de falha.

Mudanças de Nomes de Produtos

Esteja ciente de que o WebSphere eXtreme Scale era anteriormente conhecido por outros nomes.

Mudanças de Nomes de Produtos

Ao se referenciar a outra documentação, materiais de marketing ou apresentações, mantenha em mente que o eXtreme Scale era anteriormente conhecido pelos seguintes nomes.

- ObjectGrid
- WebSphere Extended Deployment Data Grid

Apesar do próprio produto agora ser conhecido como WebSphere eXtreme Scale, o ObjectGrid aparece na documentação e em outros lugares porque ele é o nome do artefato que ativa a tecnologia de grade.

Planejando Implementação do Aplicativo

Antes de implementar o WebSphere eXtreme Scale em um ambiente de produção, considere as seguintes opções.

Planejando Implementação do Aplicativo

A lista a seguir inclui itens a serem considerados:

- Número de sistemas e processadores: Quantas máquinas físicas e processadores são necessários no ambiente?
- Número de servidores: Quantos servidores do eXtreme Scale para hospedar mapas do eXtreme Scale?
- Número de partições: A quantidade de dados armazenados nos mapas é um fator na determinação do número de partições necessárias.
- Número de réplicas: Quantas réplicas são necessárias para cada primário no domínio?
- Replicação síncrona ou assíncrona: Os dados são vitais, portanto, tal replicação síncrona é necessária? Ou o desempenho é a maior prioridade, tornando a replicação assíncrona a escolha correta?
- Tamanhos de heap: Qual é quantidade de dados a ser armazenada em cada servidor?

Guia de Programação e Administração

O *Visão Geral do Produto*descreve os conceitos fundamentais para entendimento do WebSphere eXtreme Scale. Há dois guias adicionais que expandem sobre os conceitos descritos neste guia.

Use o *Guia de Administração* para tarefas de configuração e de administração geral, e o *Guia de Programação* para descrições das APIs Java para acesso e configuração da grade do eXtreme Scale.

Integração com Outros Produtos WebSphere Application Server

Você pode integrar o WebSphere eXtreme Scale com outros produtos servidores como WebSphere Application Server e WebSphere Application Server Community Edition.

Configuração do Gerenciador de Sessões HTTP para Funcionar com WebSphere Application Server Community Edition

O WebSphere Application Server Community Edition pode compartilhar estado de sessão, mas não de uma maneira eficiente e escalável. O WebSphere eXtreme Scale fornece uma camada de persistência distribuída e de alto desempenho, que pode ser utilizada para replicar o estado, mas não se integra prontamente a qualquer servidor de aplicativos fora do WebSphere Application Server. É possível integrar estes dois produtos para fornecer uma solução de gerenciamento de sessões escalável. Consulte o *Guia de Administração* para obter detalhes adicionais.

Configurando o Gerenciador de Sessões do WebSphere eXtreme Scale para Trabalhar com o WebSphere Application Server

O gerenciador de sessões HTTP primeiro era enviado com o WebSphere Extended Deployment DataGrid Versão 6.1.0.0. As versões subsequentes até a Versão 6.1.0.5 não mudaram os métodos de utilização, pois elas atendem à especificação J2EE (Java 2 Enterprise Edition) para integração e recuperação de sessão, mas há melhorias de desempenho e QoS em cada release. Para assegurar que esteja obtendo a melhor qualidade de serviço, a recomendação é aplicar o Fix Pack do WebSphere eXtreme Scale versão 6.1.0.5.

Consulte o *Guia de Administração* para obter detalhes.

Capítulo 2. Conceitos de Armazenamento em Cache

O WebSphere eXtreme Scale pode operar como um espaço de processamento de banco de dados de memória, que pode ser utilizado para fornecer armazenamento em cache sequencial para um backend de banco de dados ou atuar como um cache secundário. O armazenamento em cache sequencial utiliza o eXtreme Scale como o meio principal de interação com os dados. Quando o eXtreme Scale é utilizado como um cache secundário, o backend é utilizado em conjunto com o eXtreme Scale. Esta seção descreve vários conceitos e cenários de armazenamento em cache e descreve as topologias disponíveis para implementação de uma grade do eXtreme Scale.

Arquitetura e Topologia

Com o WebSphere eXtreme Scale, é possível configurar armazenamento em cache de dados de memória local e armazenamento em cache de dados cliente/servidor.

O WebSphere eXtreme Scale requer infraestrutura adicional mínima para operar. A infraestrutura consiste em scripts para instalar, iniciar e parar um aplicativo Java Platform, Enterprise Edition em um servidor. Os dados armazenados em cache são armazenados no servidor eXtreme Scale e os cliente conectam-se remotamente ao servidor.

Caches distribuídos oferecem desempenho, disponibilidade e escalabilidade melhorados e podem ser configurados usando topologias dinâmicas, nas quais os servidores são equilibrados automaticamente. Também é possível incluir servidores adicionais sem restaurar seus servidores eXtreme Scale existentes. É possível criar implementações simples ou grandes a nível de terabytes, em que milhares de servidores são necessários.

Mapas

Um mapa é um contêiner para pares chave/valor, que permite que um aplicativo armazene um valor indexado por uma chave. Os mapas suportam índices que podem ser incluídos nos atributos de índice na chave ou no valor. Esses índices são automaticamente usados pelo tempo de execução de consulta para determinar a maneira mais eficiente de executar uma consulta.

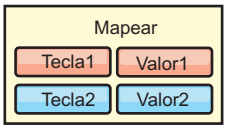


Figura 1. Mapa

Um conjunto de mapas é uma coleta de mapas com um algoritmo de particionamento comum. Os dados nos mapas são replicados com base na política definida no conjunto de mapas. Um conjunto de mapas é utilizado apenas para topologias distribuídas e não é necessário para topologias locais.

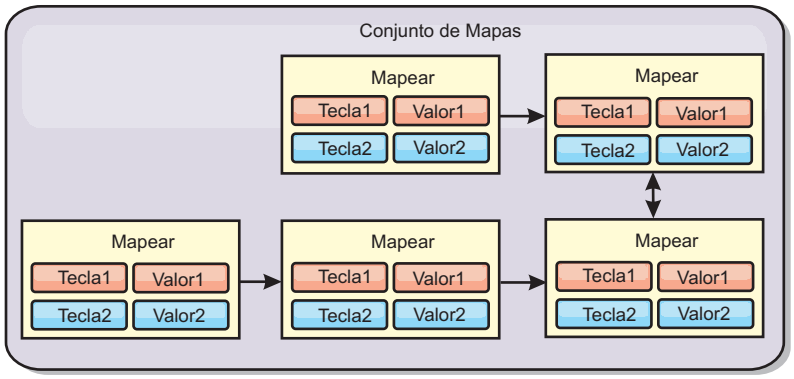


Figura 2. Conjuntos de Mapas

Um conjunto de mapas pode ter um esquema associado a ele. Um esquema são os metadados que descrevem os relacionamentos entre cada mapa ao utilizar tipos ou entidades de Objeto homogêneos.

O eXtreme Scale pode armazenar objetos Java serializáveis em cada um dos mapas usando a API ObjectMap. Um esquema pode ser definido nos mapas para identificar o relacionamento entre os objetos nos mapas, em que cada mapa suspende objetos de um único tipo. A definição de um esquema para mapas é necessária para consultar o conteúdo dos objetos de mapa. O eXtreme Scale pode ter vários esquemas de mapa definidos. Consulte as informações da API do ObjectMap API no *Guia de Programação* para obter mais detalhes.

O eXtreme Scale também pode armazenar entidades utilizando a API do EntityManager. Cada entidade está associada a um mapa. O esquema para um conjunto de mapas de entidade é automaticamente descoberto usando um arquivo XML do descritor de entidade ou classes Java anotadas. Cada entidade tem um conjunto de atributos-chave e um conjunto de atributos não-chave. Uma entidade também pode ter relacionamentos com outras entidades. O eXtreme Scale suporta relacionamentos um para um, um para muitos, muitos para um e muitos para muitos. Cada entidade é mapeada fisicamente para um único mapa no conjunto de mapas. As entidades permitem que os aplicativos tenham facilmente gráficos de objeto complexos que expandem vários Mapas. Uma topologia distribuída pode ter vários esquemas de entidade.Consulte as informações da API do EntityManager no *Guia de Programação* para obter mais detalhes.

Contêineres, Partições e Shards

O contêiner é um serviço que armazena dados do aplicativo para a grade. Estes dados geralmente são divididos em partes, que são chamadas de partições e hospedadas em vários contêineres. Cada contêiner, por sua vez, hospeda um subconjunto de dados completos. Uma JVM pode hospedar um ou mais contêineres e cada contêiner pode hospedar vários shards.

Lembre-se: Planeje o tamanho do heap para os contêineres, que hospedam todos os dados. Configure as configurações de heap apropriadamente.

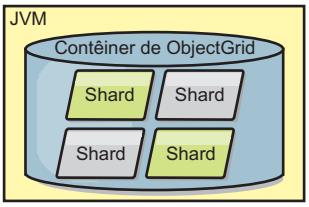


Figura 3. Contêiner

Partições hospedam um subconjunto de dados na grade. O eXtreme Scale automaticamente coloca várias partições em um único contêiner e propaga as partições à medida que mais contêineres se tornam disponíveis.

Importante: Escolha o número de partições cuidadosamente antes da implementação final já que o número de partições não pode ser alterado dinamicamente. Um mecanismo hash é usado para localizar partições na rede e é impossível o ObjectGrid re-hash o conjunto de dados inteiro após ele ter sido implementado. Superestime o número de partições.

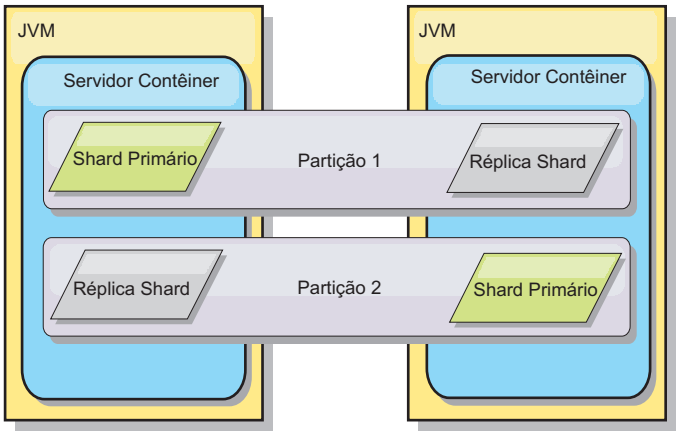


Figura 4. Partição

Os shards são instâncias de partições e possuem uma das duas funções: primário ou de réplica. O fragmento primário e suas réplicas constituem a manifestação física da partição. Cada partição tem vários shards que hospedam, cada um deles, todos os dados contidos nessa partição. Um shard é o primário e os outros são réplicas, que são cópias redundantes dos dados no primeiro shard. Um fragmento primário é a única instância da partição que permite que as transações sejam gravadas no cache. Um fragmento de réplica é uma instância ″espelhada″ da partição. Ele recebe atualizações de forma síncrona e assíncrona do fragmento primário. O fragmento de réplica permite apenas a leitura das transações do cache. As réplicas nunca são hospedadas no mesmo contêiner que as primárias e normalmente não são hospedadas na mesma máquina que as primárias.

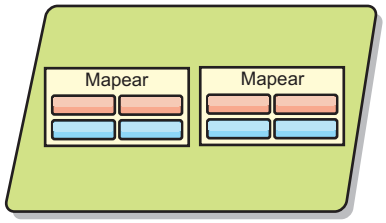


Figura 5. Shard

Para aumentar a disponibilidade dos dados, ou aumentar garantias de persistência, replique os dados. Entretanto, a replicação inclui custo na transação e troca desempenho em retorno para a disponibilidade. Com o eXtreme Scale, é possível controlar o custo já que ambas as replicações síncrona e assíncrona são suportadas, bem como modelos de replicação híbrida utilizando os modos de replicação síncrona e assíncrona. O shard da réplica síncrona recebe atualizações como parte da transação do shard primário para garantir consistência de dados. Uma réplica síncrona pode duplicar o tempo de resposta porque a transação precisa executar commit na réplica primária e na réplica síncrona antes da transação ser concluída. Um shard de réplica assíncrono recebe atualizações após o commit da transação para limitar o impacto no desempenho, mas introduz a possibilidade de perda de dados enquanto que a réplica assíncrona pode ser diversas transações atrás do shard primário.

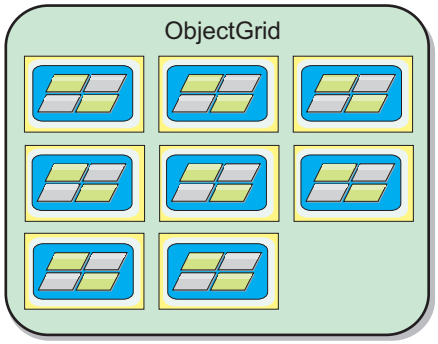


Figura 6. ObjectGrid

Clientes

Os clientes se conectam a um serviço de catálogo, recuperam uma descrição da topologia do servidor e se comunicam diretamente com cada servidor, conforme necessário. Quando a topologia do servidor é alterada porque novos servidores são incluídos ou porque os servidores existentes falharam, o cliente é automaticamente redirecionado para o servidor apropriado que está hospedando os dados. Os clientes devem examinar as chaves dos dados do aplicativo para determinar para qual partição o pedido deve ser roteado. Os Clientes podem ler dados de várias partições em uma única transação. Entretanto, os clientes podem atualizar apenas uma única partição em uma transação. Após o cliente atualizar algumas entradas, a transação do cliente deve utilizar tal partição para atualizações.

As possíveis combinações de implementação são incluídas na lista a seguir:

- Um serviço de catálogo existe em sua própria grade de Java Virtual Machines. Um único serviço de catálogo pode ser utilizado para gerenciar várias instâncias do eXtreme Scale.

- Um contêiner pode ser iniciado em uma JVM por si ou pode ser carregado em uma JVM arbitrária com os outros contêineres para instâncias diferentes do ObjectGrid.
- Um cliente pode existir em algum JVM e comunicar-se com uma ou mais instâncias do ObjectGrid. Também pode existir um cliente na mesma JVM que um contêiner.

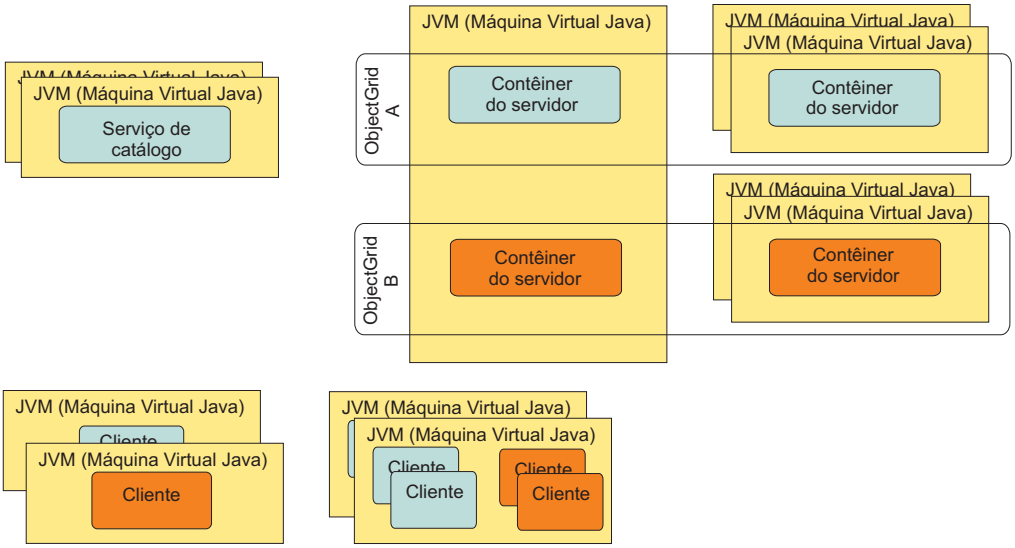


Figura 7. Possíveis Topologias

Serviço de Catálogo

O serviço de catálogo hospeda lógica que deve estar inativa durante um estado estável e tem pouca influência na escalabilidade. O serviço de catálogo é construído para atender a centenas de contêineres que se tornam disponíveis simultaneamente e executa serviços para gerenciar os contêiners.

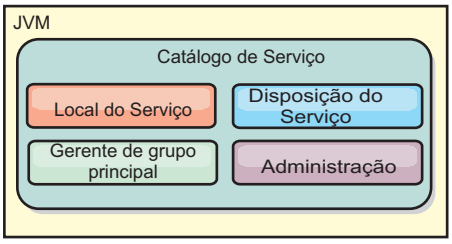


Figura 8. Serviço de Catálogo

As responsabilidades de catálogo consistem nos seguintes serviços:

Serviço de local
O serviço de local fornece localidade para clientes que estão procurando aplicativos de hosting de contêineres e para contêineres que estão procurando registrar aplicativos hospedados com o serviço de disposição. O serviço de local é executado em todos os membros da grade para efetuar o scale out desta função.

Serviço de disposição
O serviço de disposição é o sistema nervoso central para a grade e é responsável por alocar shards individuais para seu contêiner de host. O

serviço de disposição é executado como um serviço eleito ″um-entre-N″ no cluster de forma que sempre exista exatamente uma instância do serviço de disposição em execução. Se tal instância deve ser interrompida, outro processo assume. Todos os estados do serviço de catálogo são replicados em todos os servidores hospedando o serviço de catálogo para redundância.

Gerenciador de grupo principal
O gerenciador de grupo principal gerencia agrupamento peer para monitoramento de funcionamento, organiza contêineres em grupos menores de servidores e automaticamente federa os grupos de servidores. Quando um contêiner entra em contato com o serviço de catálogo pela primeira vez, ele aguarda para ser designado a um grupo novo ou existente de várias Java virtual machines (JVM). Cada grupo das Java virtual machines monitora a disponibilidade de cada um dos membros através da pulsação. Um destes membros do grupo retransmite as informações de disponibilidade ao serviço de catálogo para permitir reação a falhas por meio de realocação e encaminhamento de rotas.

Administração
Os quatro estágios de administração do seu ambiente do WebSphere eXtreme Scale são planejamento, implementação, gerenciamento e monitoramento. Consulte o *Guia de Administração* para obter mais informações sobre cada estágio.

Para disponibilidade, configure uma grade de serviço de catálogo. Uma grade de serviço de catálogo consiste em múltiplas Java virtual machines, incluindo uma JVM principal e um número de Java virtual machines de backup.

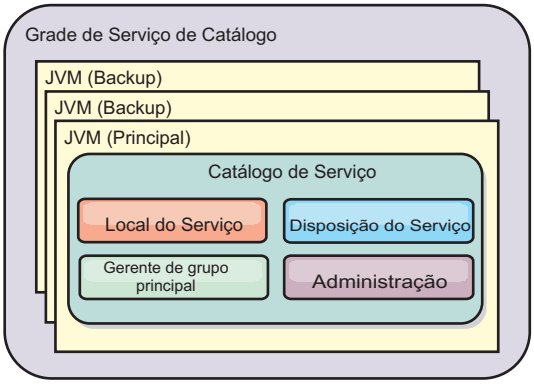


Figura 9. Grade de Serviço de Catálogo

Cache de Memória Local

No caso mais simples, o eXtreme Scale pode ser usado como um cache de grade de dados de memória local. Isso pode beneficiar especificamente os aplicativos de alta simultaneidade em que vários encadeamentos precisam acessar e modificar os dados transientes. Os dados mantidos em uma grade do eXtreme Scale local podem ser indexados e recuperados usando o suporte de consulta do WebSphere eXtreme Scale. A habilidade de consultar os dados pode ajudar muito os desenvolvedores ao trabalharem com grandes conjuntos de dados de memória em relação o suporte limitado da estrutura de dados oferecido pelo Java Virtual Machine (JVM), que está pronto para ser usado como está.

A topologia de cache em memória local para eXtreme Scale é usado para oferecer acesso transacional e consistente aos dados temporários dentro de uma única Java virtual machine.

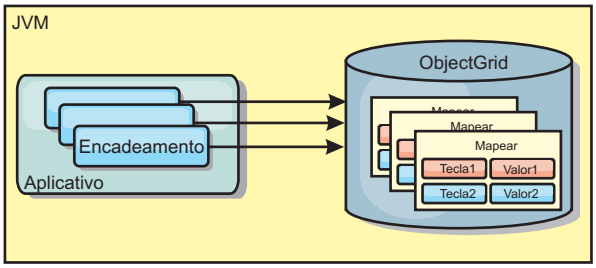


Figura 10. Cenário de Cache em Memória Local

Vantagens

- Configuração simples: Um ObjectGrid pode ser criado programaticamente ou declarativamente com o arquivo XML descritor de implementação ObjectGrid ou com outras estruturas como Spring
- Rápido: Cada BackingMap pode ser ajustado de maneira independente para utilização de memória e simultaneidade ideais.
- Ideal para topologias de uma única Java virtual machine com pequenos conjuntos de dados ou para armazenamento em cache de dados frequentemente acessados.
- Transacional. As atualizações BackingMap podem ser agrupadas em uma única unidade de trabalho e podem ser integradas como um último participante nas transações de duas fases como transações JTA (Java Transaction Architecture).

Desvantagens

- Não tolerante a falhas.
- Os dados não são replicados. Caches em memória são melhores para dados de referência somente para leitura.
- Não escalável. A quantidade de memória necessária pelo banco de dados pode ultrapassar a capacidade da Java virtual machine.
- Ocorrem problemas na inclusão de Java virtual machines:
  - Os dados não podem ser facilmente particionados
  - Você deve replicar manualmente o estado entre as Java virtual machines ou cada instância do cache poderá ter diferentes versões dos mesmos dados.
  - A invalidação é custosa.
  - Cada cache deve ser aquecido de maneira independente. O aquecimento é o período de carregamento de um conjunto de dados para que o cache seja preenchido com dados válidos.

Quando Utilizar

A topologia de implementação de cache em memória local deve ser usada somente quando a quantidade de dados a serem armazenados em cache for pequena (puder ser colocada em uma única Java virtual machine) e for relativamente estável. Dados antigos devem ser tolerados com esta abordagem. A utilização de evictors para manter os dados mais frequentemente ou recentemente usados no cache pode ajudar a diminuir o tamanho do cache e a aumentar a relevância dos dados.

Cache em Memória Local Replicado pelo Peer

Uma das limitações de um cache local do WebSphere eXtreme Scale é que há vários processos com instâncias de cache independentes, é difícil manter o cache sincronizado.

O eXtreme Scale inclui dois plug-ins que automaticamente propagam alterações de transação entre instâncias do eXtreme Scale peer. O plug-in JMSObjectGridEventListener automaticamente propaga alterações do eXtreme Scale usando o JMS (Java Messaging Service).

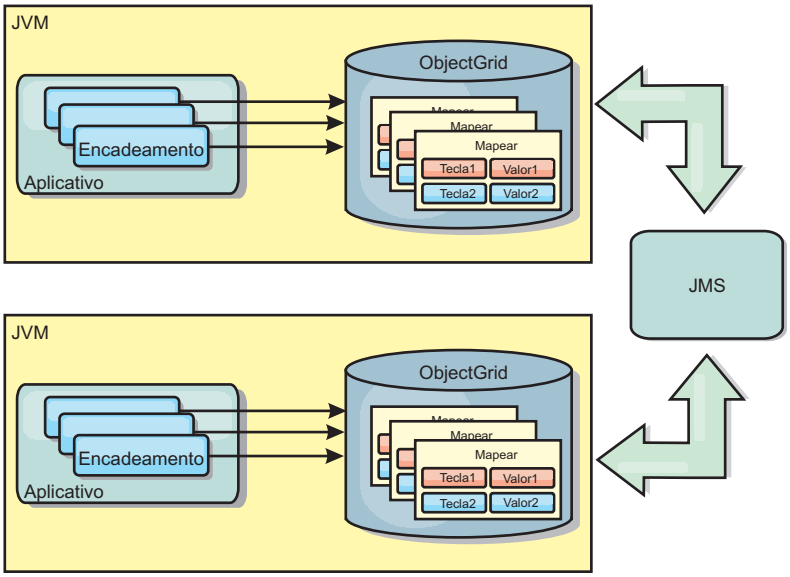


Figura 11. Cache Replicado pelo Peer com Alterações que são Propagadas com JMS

Se você estiver executando em um ambiente WebSphere Application Server, o plug-in TranPropListener também está disponível. O plug-in TranPropListener usa o gerenciador de alta disponibilidade (HA) para propagar as alterações em cada instância do cache do eXtreme Scale de peer.

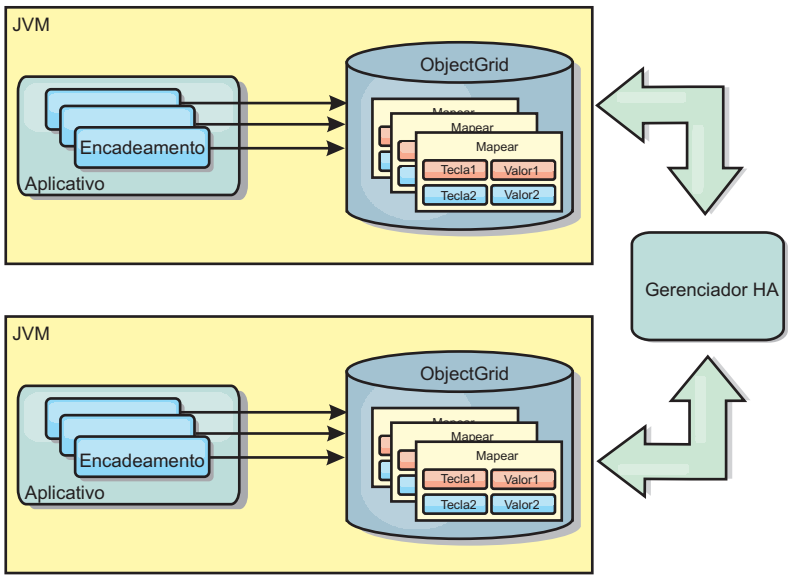


Figura 12. Cache Replicado pelo Peer com Alterações que são Propagadas com o Gerenciador de Alta Disponibilidade

Vantagens

- Os dados são mais válidos porque os dados são atualizados mais frequentemente.
- Com o plug-in TranPropListener, como no ambiente local, o eXtreme Scale pode ser criado programaticamente ou declarativamente com o arquivo XML descritor de implementação do eXtreme Scale ou com outras estruturas como Spring. A integração com o gerenciador de alta disponibilidade é feita automaticamente.
- Cada BackingMap pode ser independentemente ajustado para melhor utilização da memória e concorrência.
- As atualizações BackingMap podem ser agrupadas em uma única unidade de trabalho e podem ser integradas como um último participante nas transações de duas fases como transações JTA (Java Transaction Architecture).
- Ideal para poucas topologias JVM com um conjunto de dados razoavelmente pequeno ou para armazenamento em cache de dados frequentemente acessados.
- As atualizações em cada eXtreme Scale são replicadas para todas as instâncias do eXtreme Scale do peer. As alterações são consistentes desde que uma assinatura durável seja utilizada.

Desvantagens

- A configuração e a manutenção para o JMSObjectGridEventListener podem ser complexas. O eXtreme Scale pode ser criado programaticamente ou declarativamente com o arquivo XML descritor de implementação do eXtreme Scale ou com outras estruturas tais como Spring.
- Não escalável: A quantidade de memória necessária para que o banco de dados possa dominar a JVM.
- Funciona inadequadamente ao incluir Java Virtual Machines:
  - Os dados não podem ser facilmente particionados
  - A invalidação é custosa.
  - Cada cache deve ser aquecido de maneira independente

Quando Utilizar

Esta topologia de implementação deve ser utilizada apenas quando a quantidade de dados a ser armazenada em cache for pequena (podendo ajustar-se a uma única JVM) e for relativamente estável.

Cache Distribuído

O WebSphere eXtreme Scale é mais frequentemente usado como um cache compartilhado, para fornecer acesso transacional a dados para múltiplos componentes onde, caso contrário, um banco de dados tradicional seria usado. Isto pode aprimorar a facilidade de desenvolvimento e implementação do aplicativo eliminando a necessidade de configurar um banco de dados.

O cache é coerente porque todos os clientes vêem os mesmos dados no cache. Cada pedaço de dado é armazenado em exatamente um servidor no cache, evitando cópias de registros desperdiçadas que poderiam potencialmente conter diferentes versões dos dados. Um cache coerente também pode conter mais dados à medida que mais servidores são incluídos à grade, e escalar de forma linear à medida que a grade cresce em tamanho. Porque os clientes acessam dados desta grade com chamadas procedurais remotas, ela também é conhecida como um cache remoto (ou cache distante). Através do particionamento de dados, cada processo contém um subconjunto exclusivo do conjunto de dados total. Grades maiores podem conter mais dados e atender mais pedidos para tais dados. A coerência também elimina a necessidade de inserir dados de invalidação ao redor da grade porque não há dados antigos. O cache coerente retém somente a cópia mais recente de cada pedaço de dados.

Se você estiver executando um ambiente doWebSphere Application Server, o plug-in TranPropListener também estará disponível. O plug-in TranPropListener usa o componente de alta disponibilidade (Gerenciador HA) do WebSphere Application Server para propagar as alterações para cada instância de cache ObjectGrid de peer.

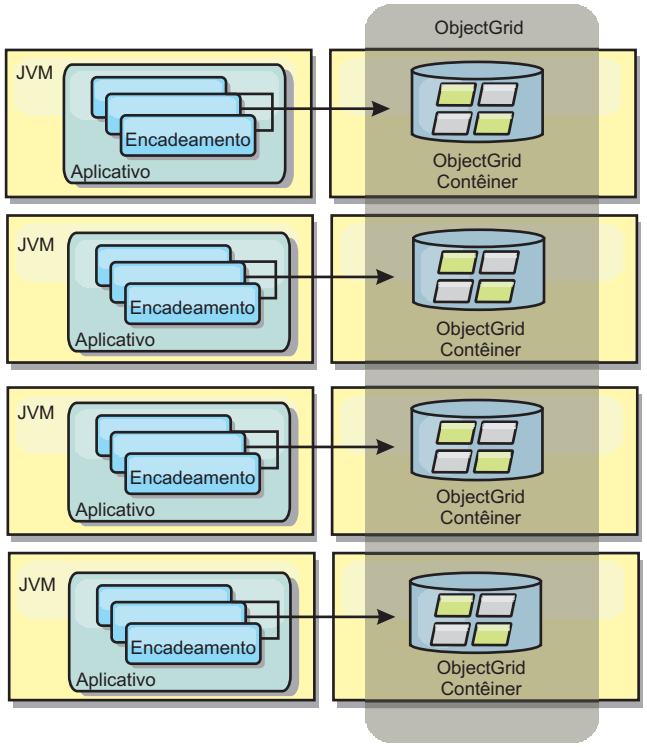


Figura 13. Cache Distribuído

Cache Local

Opcionalmente, os clientes têm um cache local, sequencial quando o eXtreme Scale é usado em uma topologia distribuída. Este cache opcional é chamado de cache local, um ObjectGrid independente em cada cliente, servindo como um cache para o cache remoto, do lado do cliente. O cache local é ativado por padrão quando o bloqueio é configurado como otimista ou nenhum e não pode ser utilizado quando é configurado como pessimista.

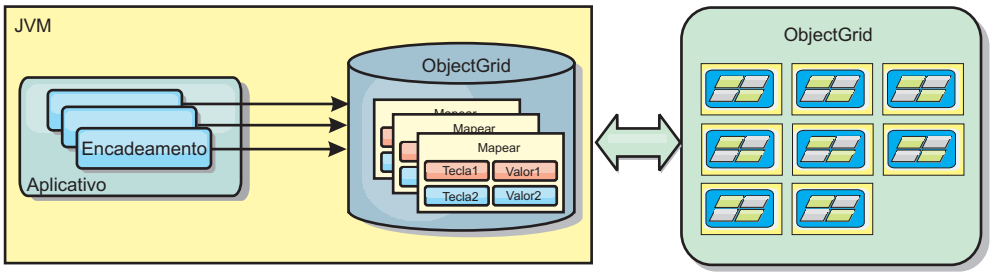


Figura 14. Cache Local

Um cache local é muito rápido porque fornece acesso em memória a um subconjunto do conjunto inteiro de dados em cache que é armazenado remotamente nos servidores do eXtreme Scale. O cache local não é particionado e contém dados de qualquer uma das partições eXtreme Scale remotas. O WebSphere eXtreme Scale pode ter até três camadas de cache, como a seguir.

1. O cache da camada da transação contém todas as alterações para uma única transação. O cache da transação contém uma cópia de trabalho dos dados até que a transação seja confirmada. Quando uma transação do cliente solicita dados de um ObjectMap, a transação é verificada primeiro

2. O cache local na camada do cliente contém um subconjunto de dados da camada do servidor. Quando a camada da transação não possui os dados, eles são buscados em um cache local, se disponíveis, e inseridos no cache da transação.
3. A grade na camada do servidor contém a maioria dos dados e é compartilhada entre todos os clientes. A camada do servidor pode ser particionada, o que permite que uma grande quantidade de dados seja armazenada em cache. Quando o cache local do cliente não possui os dados, eles são buscados na camada do servidor e inseridos no cache cliente. A camada do servidor também pode ter um plug-in do Utilitário de Carga. Quando a grade não tem os dados necessários, o Utilitário de Carga é chamado e os dados resultantes são inseridos do armazém de dados de backend para a grade.

Para desativar o cache local, configure o atributo numberOfBuckets como 0 no arquivo descritor do ObjectGrid de substituição do cliente. Consulte Bloqueio da Entrada de Mapa para obter detalhes sobre as estratégias de bloqueio do eXtreme Scale. O cache local também pode ser configurado para ter uma política de despejo configurada e diferentes plug-ins usando uma configuração do descritor do eXtreme Scale de substituição.

Vantagem

- Tempo de resposta rápido porque todos os acessos aos dados é local.

Desvantagens

- Aumenta a duração dos dados antigos.
- Deve utilizar um evictor para invalidar dados para evitar a falta de memória.

Quando Utilizar

Utilize quando o tempo de resposta for importante e dados antigos puderem ser tolerados.

Cache integrado

As grades do eXtreme Scale podem ser executadas nos processos existentes como servidores eXtreme Scale integrados ou podem ser gerenciadas como processos externos. As grades integradas são úteis quando você está executando em um servidor de aplicativos, como o WebSphere Application Server. É possível iniciar servidores eXtreme Scale que não são integrados usando scripts da linha de comandos e executar em um processo Java.

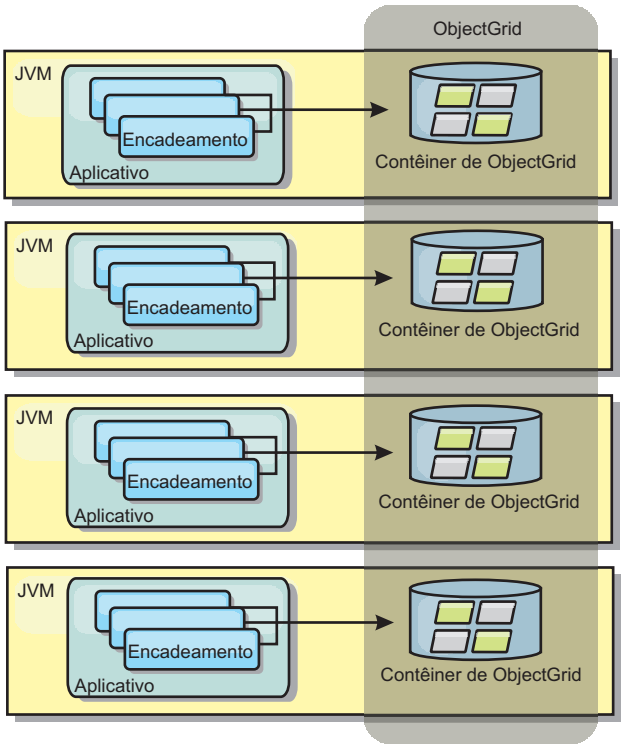


Figura 15. Cache Integrado

Vantagens

- Administração simplificada já que há menos processos para gerenciar.
- Implementação do aplicativo simplificada, já que a grade está utilizando o carregador de classe do aplicativo cliente.
- Particionamento de suporte e alta disponibilidade.

Desvantagens

- Aumento da área de cobertura da memória no processo do cliente já que todos os dados são colocados no processo.
- Aumento da utilização da CPU para atender pedidos de clientes.
- Mais difícil para manipular atualizações de aplicativo, pois os clientes estão usando os mesmos arquivos archive de Java do aplicativo que os servidores.
- Menos flexível. A escala dos clientes e servidores de grade não pode aumentar na mesma proporção. Quando os servidores são externamente definidos, é possível ter mais flexibilidade no gerenciamento do número de processos.

Quando Utilizar

Utilize grades integradas quando há grande quantidade de memória livre no processo do cliente para dados da grade e dados de failover potenciais.

Para obter mais informações, consulte o tópico sobre a ativação do mecanismo de invalidação do cliente no *Guia de Administração*.

Integração com o Banco de Dados

O WebSphere eXtreme Scale é usado para colocar um banco de dados tradicional na frente e eliminar a atividade de leitura que normalmente é armazenada no banco de dados. Um cache coerente pode ser utilizado com um aplicativo direta ou indiretamente, utilizando um mapeador relacional de objeto. O cache coerente pode transferir o banco de dados ou o backend a partir das leituras. Em um cenário levemente mais complexo, tal como o acesso transacional a um conjunto de dados no qual apenas parte dos dados requer garantias de persistência tradicional, a filtragem pode ser utilizada para transferir até mesmo transações de gravação.

É possível configurar o eXtreme Scale para funcionar como um espaço de processamento de banco de dados de memória altamente flexível. Entretanto, o eXtreme Scale não é um object relational mapper (ORM). Ele não tem conhecimento de onde vieram os dados no eXtreme Scale. Um aplicativo ou um ORM pode colocar dados em um servidor eXtreme Scale. É responsabilidade da origem dos dados certificar-se de que eles permaneçam consistentes com o banco de dados no qual os dados se originaram. Isto significa que o eXtreme Scale não pode invalidar dados que são extraídos de um banco de dados automaticamente. O aplicativo ou mapeador deve fornecer esta função e gerenciar os dados armazenados no eXtreme Scale.

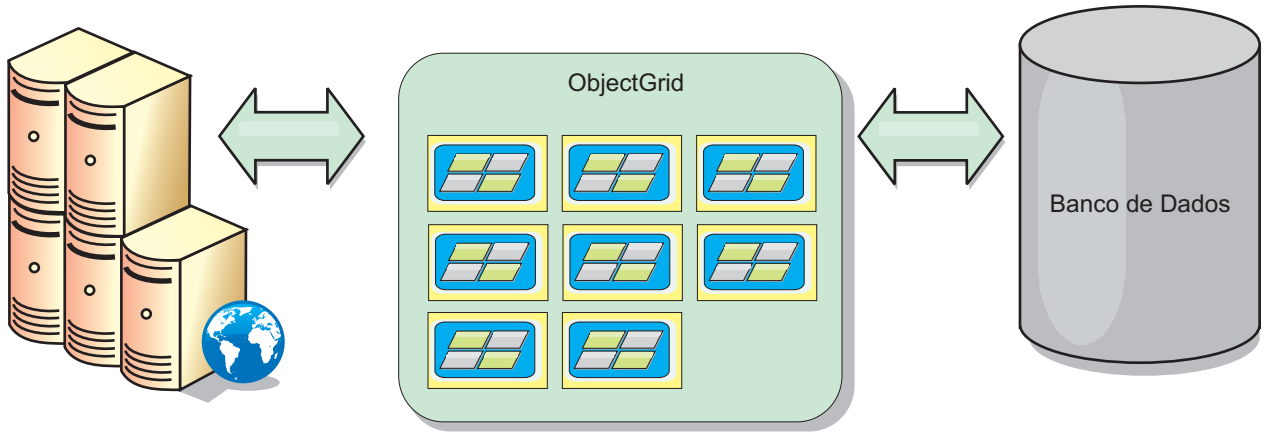


Figura 16. ObjectGrid como um Buffer de Banco de Dados

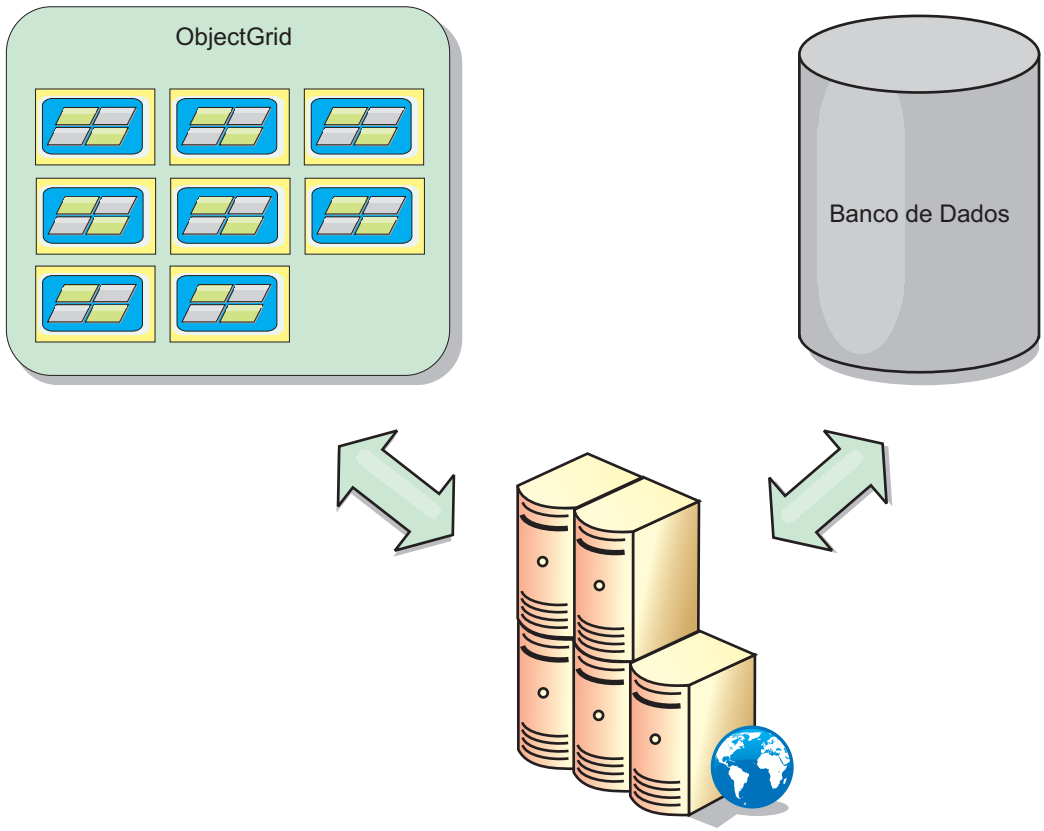


Figura 17. ObjectGrid como um Cache Secundário

Cache Esparso e Completo

O WebSphere eXtreme Scale pode ser utilizado como um cache esparso ou um cache completo. Um cache escasso somente mantém um subconjunto dos dados totais, enquanto que um cache completo mantém todos os dados e pode ser preenchido lentamente, On Demand. Caches esparsos normalmente são acessados utilizando chaves (ao invés de índices ou consultas) já que os dados estão disponíveis apenas parcialmente.

Quando uma chave não está presente (uma ausência de cache), a próxima camada é chamada e os dados são buscados e inseridos na camada respectiva do cache. Se você estiver utilizando uma consulta ou índice, apenas os valores atualmente carregados são acessados e os pedidos não são encaminhados para as outras camadas. Um cache completo contém todos os dados necessários e pode ser acessado usando atributos não-chaves com índices ou consultas.

Um cache completo é pré-carregado com dados antes de os aplicativos o utilizarem, e pode efetivamente funcionar como um substituto do banco de dados. Depois de os dados serem carregados, ele pode ser tratado de forma semelhante a de um banco de dados. Como todos os dados estão disponíveis, consultas e índices podem ser usados para localizar e agregar os dados.

Cache Secundário e Cache Sequencial

O WebSphere eXtreme Scale é utilizado para fornecer armazenamento em cache sequencial para um backend de banco de dados ou como um cache secundário

para um banco de dados. O armazenamento em cache sequencial utiliza o eXtreme Scale como o meio principal de interação com os dados. Quando o eXtreme Scale é utilizado como um cache secundário, o backend é utilizado em conjunto com o eXtreme Scale.

Cache Secundário

O eXtreme Scale pode ser utilizado com um cache secundário para uma camada de acesso a dados do aplicativo. Neste cenário, o eXtreme Scale é utilizado para armazenar temporariamente objetos que normalmente poderiam ser recuperados de um banco de dados de backend. Os aplicativos verificam se o eXtreme Scale contém os dados desejados. Se houver dados, eles são retornados para o responsável pela chamada. Se não houver dados, eles são recuperados do backend e inseridos no eXtreme Scale para que o próximo pedido possa utilizar a cópia armazenada em cache. O diagrama a seguir ilustra como o eXtreme Scale pode ser utilizado como um cache secundário utilizando uma camada de acesso a dados arbitrária como OpenJPA ou Hibernate.

Plug-ins do cache secundário para Hibernate e OpenJPA

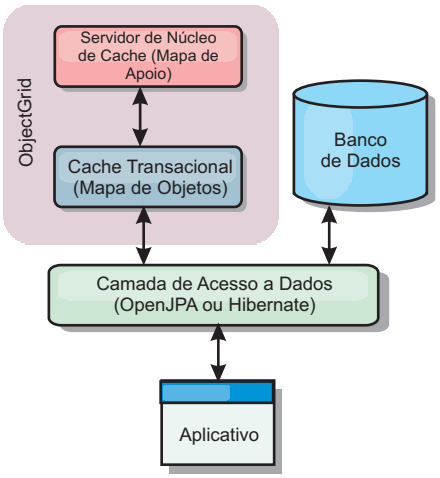


Figura 18. Cache Secundário

Os plug-ins de cache para ambos, OpenJPA e Hibernate, são incluídos noeXtreme Scale , o que permite que você use o eXtreme Scale como um cache secundário automático. Usar o eXtreme Scale como um provedor de cache aumenta o desempenho ao ler e enfileirar dados e reduz a carga para o banco de dados. Existem vantagens que o eXtreme Scale tem sobre as implementações de cache integrado porque o cache é automaticamente replicado entre todos os processos. Quando um cliente armazena um valor em cache, todos os outros clientes estarão aptos a utilizar o valor armazenado em cache.

Cache Sequencial

Quando utilizado como um cache sequencial, o eXtreme Scale interage com o backend utilizando um plug-in do Utilitário de Carga. Este cenário pode simplificar o acesso a dados possibilitando que aplicativos acessem as APIs do eXtreme Scale diretamente. Vários cenários diferentes de armazenamento em cache são suportados no eXtreme Scale para garantir que os dados no cache e os dados no backend sejam sincronizados. O diagrama a seguir ilustra como um cache

sequencial interage com o aplicativo e o back end.

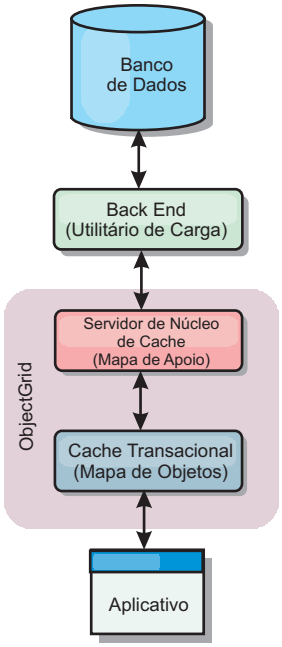


Figura 19. Cache Sequencial

Técnicas de Sincronização de Banco de Dados

Quando o WebSphere eXtreme Scale é utilizado como um cache, os aplicativos devem ser criados para tolerar dados antigos se o banco de dados puder ser atualizado de maneira independente de uma transação do eXtreme Scale. Para atuar como um espaço de processamento de banco de dados de memória sincronizado, o eXtreme Scale fornece várias maneiras de manter o cache atualizado.

Técnicas de Sincronização de Banco de Dados

Atualização periódica

O cache pode ser automaticamente invalidado ou atualizado automaticamente usando o atualizador de banco de dados baseado em tempo JPA (Java Persistence API).O atualizador periodicamente consulta o banco de dados usando um provedor JPA para todas as atualizações ou inserções que ocorreram desde a atualização anterior. Quaisquer alterações identificadas são automaticamente invalidadas ou atualizadas quando utilizadas com um cache esparso. Se utilizadas com um cache completo, as entradas podem ser descobertas e inseridas no cache. As entradas nunca são removidas do cache.

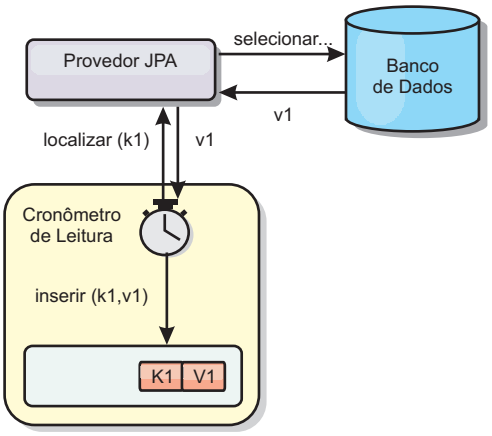


Figura 20. Atualização Periódica

Despejo

Os caches esparsos podem utilizar políticas de despejo para automaticamente remover dados do cache sem afetar o banco de dados. Há três políticas integradas incluídas no eXtreme Scale: time-to-live, least-recently-used e least-frequently-used. Todas as três políticas podem opcionalmente despejar dados mais agressivamente à medida que a memória torna-se restrita ao ativar a opção de despejo baseada em memória. Consulte o tópico "Despejo" na página 27 para obter mais detalhes.

Invalidação Baseada em Eventos

Caches escassos e completos podem ser invalidados ou atualizados usando um gerador de eventos como JMS (Java Message Service). A invalidação utilizando JMS pode ser manualmente vinculada a qualquer processo que atualiza o backend utilizando um acionador do banco de dados. Um plug-in ObjectGridEventListener do JMS é fornecido no eXtreme Scale que pode notificar quando o cache do servidor tiver qualquer alteração. Isto pode diminuir a quantidade de tempo que o cliente pode visualizar dados antigos.

Invalidação programática

As APIs do eXtreme Scale permitem interação manual do cache local e do servidor usando os métodos de API Session.beginNoWriteThrough(), ObjectMap.invalidate() e EntityManager.invalidate(). Se um processo do cliente ou servidor não precisar mais de uma parte dos dados, os métodos de invalidação podem ser utilizados para remover dados do cache local ou do servidor. O método beginNoWriteThrough aplica qualquer operação ObjectMap ou EntityManager para o cache local sem chamar o utilitário de carga. Se chamada a partir de um cliente, a operação é aplicável apenas para o cache local (o utilitário de carga remoto não é chamado). Se chamada no servidor, a operação é aplicável apenas ao cache principal do servidor sem chamar o utilitário de carga.

Atualização Periódica

Quando o recurso de espaço de processamento do banco de dados de memória doWebSphere eXtreme Scale é utilizado como um cache, os aplicativos deverão ser criados para tolerar dados antigos se o banco de dados puder ser atualizado de maneira independente de uma transação do eXtreme Scale. O uso de uma atualização periódica permite que o eXtreme Scale mantenha o cache atualizado.

O cache pode ser automaticamente invalidado ou atualizado automaticamente usando o atualizador de banco de dados baseado em tempo JPA (Java Persistence API). O atualizador periodicamente consulta o banco de dados usando um provedor JPA para todas as atualizações ou inserções que ocorreram desde a atualização anterior. Quaisquer alterações identificadas são automaticamente invalidadas ou atualizadas quando utilizadas com um cache esparso. Se utilizadas com um cache completo, as entradas podem ser descobertas e inseridas no cache. As entradas nunca são removidas do cache.

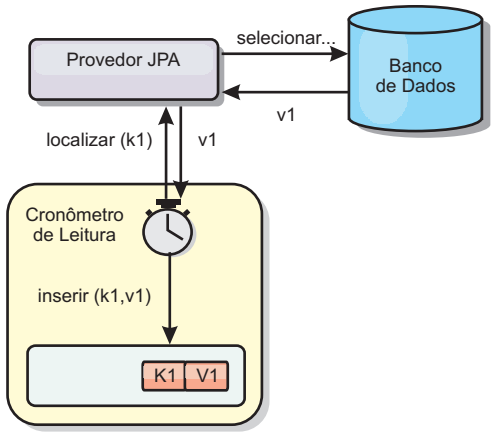


Figura 21. Atualização Periódica

Os caches esparsos podem utilizar políticas de despejo para automaticamente remover dados do cache sem afetar o banco de dados. Há três políticas integradas incluídas no eXtreme Scale: time-to-live, least-recently-used e least-frequently-used. Todas as três políticas podem opcionalmente despejar dados mais agressivamente à medida que a memória torna-se restrita ao ativar a opção de despejo baseada em memória. Para obter informações adicionais sobre Evictors, consulte “Despejo”.

Despejo

O WebSphere eXtreme Scale fornece um mecanismo padrão para despejar entradas do cache e um plug-in para criar evictors customizados. Um evictor controla a associação de entradas em cada BackingMap. O evictor padrão utiliza uma política de despejo de TTL (Time-to-Live) para cada BackingMap. Se você fornecer um mecanismo de evictor conectável, geralmente ele utilizará uma política de evicção baseada no número de entradas em vez de baseada no tempo.

Evictor Time to Live Padrão

O WebSphere eXtreme Scale oferece um evictor de TTL (Time To Live) para cada BackingMap. O evictor TTL mantém um tempo de expiração para cada entrada criada. Ao atingir o tempo de expiração de uma entrada, o evictor remove a entrada do BackingMap. A fim de minimizar o impacto que a remoção de uma entrada causa no desempenho, o evictor de TTL pode aguardar o tempo de expiração antes de despejar uma entrada. Ele não removerá nenhuma entrada até que ela expire.

O BackingMap possui atributos utilizados para controlar como o evictor time to live calcula o tempo de expiração para cada entrada. Os aplicativos configuram o atributo ttlType para especificar como o evictor TTL deve calcular o tempo de expiração. O atributo ttlType pode ser configurado como um dos seguintes valores:

1. None: Indica que uma entrada no BackingMap nunca expira. O evictor TTL não retira estas entradas.
2. Creation time : Indica que o horário em que uma entrada é criada é utilizada no cálculo de prazo de expiração.
3. Last access time : Indica que o horário em que uma entrada foi acessada pela última vez é utilizado no cálculo do prazo de expiração.

Se o atributo ttlType não estiver configurado em um BackingMap, o tipo padrão de None é utilizado de forma que o evictor TTL não despeje entradas. Se o atributo ttlType estiver configurado com o horário da criação ou com o horário do último acesso, o valor do atributo time to live no BackingMap é incluído no horário da criação ou no horário do último acesso para calcular o prazo de expiração. A precisão de tempo do atributo de mapa time to live está em segundos. Um valor de 0 para o atributo time to live é um valor especial que é utilizado para indicar que a entrada do mapa pode existir para sempre, ou seja, a entrada permanece no mapa até que o aplicativo explicitamente remova ou invalide a entrada do mapa.

Evictors Opcionais

O evictor TTL padrão utiliza uma política de evicção baseada no tempo e o número de entradas no BackingMap não tem efeito sobre o tempo de expiração de uma entrada. É possível utilizar um evictor conectável opcional para despejar entradas com base no número de entradas existentes em vez de no tempo.

Os seguintes evictors conectáveis opcionais fornecem alguns algoritmos comumente utilizados para decidir quais entradas liberar quando um BackingMap crescer além de algum limite de tamanho. *

- O evictor LRUEvictor utiliza um algoritmo LRU (Least Recently Used) para decidir quais entradas serão despejadas quando o BackingMap exceder um número máximo de entradas.
- O evictor LFUEvictor utiliza um algoritmo LFU (Least Frequently Used) para decidir quais entradas serão despejadas quando o BackingMap exceder um número máximo de entradas.

O BackingMap informa um evictor conforme as entradas são criadas, modificadas ou removidas de uma transação. O BackingMap acompanha estas entradas e escolhe quando liberar uma ou mais entradas do BackingMap.

Um BackingMap não possui informações de configuração para um tamanho máximo. Em vez disso, as propriedades do evictor são configuradas para controlar o comportamento do evictor. O LRUEvictor e o LFUEvictor possuem uma propriedade de tamanho máximo utilizada para fazer o evictor começar a liberar entradas quando o tamanho máximo for excedido. Assim como o evictor TTL, os evictores LRU e LFU podem não liberar imediatamente uma entrada quando o número máximo de entradas for atingido para minimizar o impacto no desempenho.

Se o algoritmo LRU ou LFU não for adequado para um determinado aplicativo, você poderá redigir seus próprios evictors para criar a sua estratégia de despejo.

Despejo Baseado em Memória

Importante: O despejo baseado em memória é suportado somente no Java Platform, Enterprise Edition Versão 5 ou posterior.

Todos os evictors integrados suportam despejo baseado em memória que pode ser ativado na interface BackingMap através da definição do atributo evictionTriggers de BackingMap como `MEMORY_USAGE_THRESHOLD`. Para obter informações adicionais sobre como configurar o atributo evictionTriggers na BackingMap, consulte as informações sobre a interface BackingMap e o arquivo XML do descritor do ObjectGrid no *Guia de Administração*.

O despejo baseado em memória é baseado no limite de uso do heap. Quando o despejo baseado em memória é ativado no BackingMap e o BackingMap possui algum evictor integrado, o limite de uso é configurado com uma porcentagem padrão de memória total se o limite ainda não tiver sido configurado anteriormente.

Ao usar o despejo baseado em memória, você deveria configurar o limite de coleta de lixo para o mesmo valor que a utilização de heap de destino. Por exemplo, se o limite de despejo baseado em memória for configurado em 50 por cento e o limite de coleta de lixo estiver configurado no nível padrão de 70 por cento, então a utilização de heap pode chegar no máximo a 70 por cento. Esta aumento da utilização de heap ocorre proque o despejo baseado em memória é acionado somente depois de um ciclo de coleta de lixo.

O algoritmo de despejo baseado em memória usado pelo WebSphere eXtreme Scale é sensível ao comportamento do algoritmo de coleta de lixo em uso. O melhor algoritmo para despejo baseado em memória é o padrão da IBM através do coletor. A geração de algoritmos de coleta de lixo podem provocar comportamento indesejado e, dessa forma, você não deve usar estes algoritmos com o despejo baseado em memória.

Para alterar a porcentagem de limite de uso, configure a propriedade memoryThresholdPercentage no contêiner e os arquivos de propriedades do servidor para os processos do servidor do eXtreme Scale.

Durante o tempo de execução, se o uso da memória exceder o limite de uso destinado, os evictors baseados em memória iniciam o despejo de entradas e tentam manter o uso da memória abaixo do limite de uso destinado. Porém, não há garantia de que a velocidade do despejo seja rápida o suficiente para evitar um possível erro de falta de memória se o tempo de execução do sistema continuar consumindo memória rapidamente.

Boas Práticas do Evictor Padrão:

É possível modificar o comportamento do evictor time to live (TTL) padrão configurando atributos e propriedades.

Além dos evictors de plug-in que são descritos no tópico Boas Práticas de Desempenho do Evictor de Plug-in, um evictor TTL padrão é criado com cada mapa de apoio. O evictor padrão remove entradas com base em um conceito de time-to-live. Este comportamento é definido pelo atributo ttlType. Existem três atributos ttlTypes:

- Nenhum: especifica que as entradas nunca expiram e, portanto, nunca são removidas do mapa.
- Hora de criação: especifica que a entradas são despejadas dependendo de quando foram criadas.
- Hora do último acesso: especifica que as entradas são despejadas dependendo de quando foram acessadas pela última vez.

Propriedade TimeToLive

Esta propriedade, junto com a propriedade ttlType, é a mais importante, de uma perspectiva de desempenho. Se estiver utilizando CREATION_TIME ttlType, o evictor liberará uma entrada quando sua hora de criação for igual a seu valor de atributo TimeToLive. Se você configurar o valor de atributo TimeToLive como 10 segundos, tudo o que estiver em todo o mapa será liberado após dez segundos. É importante ter atenção ao configurar este valor para o CREATION_TIME ttlType. Este evictor é melhor utilizado quando existem quantidades de inclusões no cache razoavelmente altas que são utilizadas apenas durante uma quantidade de tempo configurada. Com esta estratégia, tudo o que é criado é removido após a quantidade de tempo configurada.

A seguir está um exemplo de onde um tipo TTL de CREATION_TIME é útil. Você está utilizando um aplicativo da Web que obtém preços de ações e a obtenção dos preços mais recentes não é crítica. Neste caso, os preços de ações são armazenados em cache em um ObjectGrid por 20 minutos. Após 20 minutos, as entradas do mapa do ObjectGrid expiram e são liberadas. Aproximadamente a cada vinte minutos, o mapa do ObjectGrid utiliza o plug-in do Utilitário de Carga para atualizar os dados do mapa com dados atuais do banco de dados. O banco de dados é atualizado a cada 20 minutos com os preços de ações mais recentes. Portanto, para este aplicativo, é recomendável a utilização de um valor TimeToLive de 20 minutos.

Se estiver utilizando o atributo LAST_ACCESSED_TIME ttlType, configure o TimeToLive como um número inferior do que se estiver utilizando CREATION_TIME ttlType, porque o atributo TimeToLive de entradas é reconfigurado sempre que é acessado. Em outras palavras, se o atributo TimeToLive for igual a 15 e existir uma entrada por 14 segundos, mas for acessada, ela não expirará novamente durante outros 15 segundos. Se você configurar o TimeToLive como um número relativamente alto, muitas entradas poderão nunca ser liberadas. No entanto, se você configurar o valor como algo semelhante a 15 segundos, as entradas poderão ser removidas quando não forem acessadas com freqüência.

A seguir está um exemplo de onde um tipo TTL de LAST_ACCESSED_TIME é útil. Um mapa do ObjectGrid é utilizado para reter dados de sessão de um cliente. Os dados de sessão devem ser destruídos se o cliente não utilizá-los por algum período de tempo. Por exemplo, é excedido o tempo limite dos dados de sessão após 30 minutos sem atividade do cliente. Neste caso, a utilização de um tipo TTL de LAST_ACCESSED_TIME com o atributo TimeToLive configurado como 30 minutos é exatamente o que é necessário para este aplicativo.

O exemplo a seguir cria um mapa de suporte, configura seu atributo ttlType do evictor padrão e configura sua propriedade TimeToLive.

```
ObjectGrid objGrid = new ObjectGrid;
BackingMap bMap = objGrid.defineMap("SomeMap");
bMap.setTtlEvictorType(TTLType.LAST_ACCESSED_TIME);
bMap.setTimeToLive(1800);
```

A maior parte das configurações de evictor deve ser feita antes de inicializar o ObjectGrid.

Também é possível gravar seus próprios evictors: Para obter informações adicionais, consulte as informações sobre como gravar um evictor customizado no *Guia de Programação*.

Cenários de Armazenamento em Cache Sequencial

O armazenamento em cache sequencial utiliza o eXtreme Scale como o meio principal de interação com os dados. Quando o eXtreme Scale é usado como um cache em linha, o aplicativo interage com o backend usando o plug-in do utilitário de carga.

A opção de armazenamento em cache em linha simplifica o acesso aos dados pois ela permite que os aplicativos acessem diretamente as APIs do eXtreme Scale. O WebSphere eXtreme Scale suporta diversos cenários de armazenamento em cache em linha, como os seguintes:

- Read-through
- Write-through
- Write-behind

Cenário de Armazenamento em Cache Read-through

Um cache read-through é um cache esparso que lentamente carrega entradas de dados por chave à medida que elas são solicitadas. Isto é feito sem exigir que o responsável pela chamada saiba quais entradas estão preenchidas. Se os dados não puderem ser localizados no cache do eXtreme Scale, o eXtreme Scale irá recuperar os dados ausentes do plug-in do utilitário de carga, que carrega os dados do banco de dados backend e insere os dados no cache. Pedidos subsequentes para a mesma chave de dados serão localizados no cache até que ele possa ser removido, invalidado ou despejado.

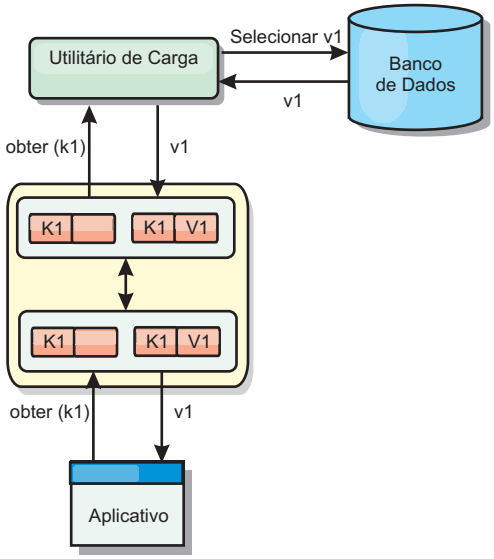


Figura 22. Armazenamento em Cache Read-through

Cenário de Armazenamento em Cache Write-through

Em um cache write-through, cada gravação no cache é gravada de maneira síncrona no banco de dados utilizando o Utilitário de Carga. Este método fornece consistência com o backend, mas diminui o desempenho de gravação pois a operação do banco de dados é síncrona. Como o cache e o banco de dados são ambos atualizados, as leituras subsequentes para os mesmos dados serão localizadas no cache, evitando a chamada do banco de dados. Um cache

write-through sempre é utilizado em conjunto com um cache read-through.

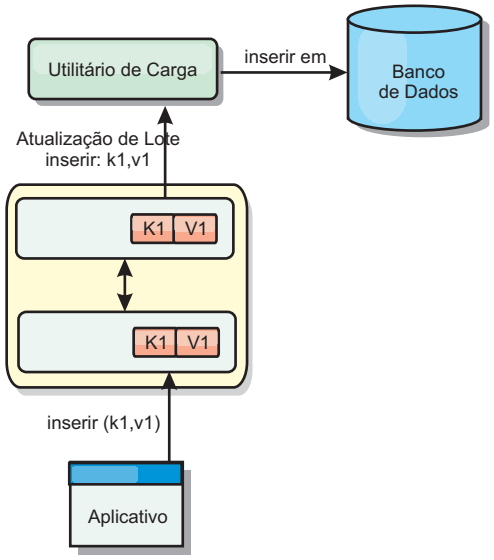


Figura 23. Armazenamento em Cache Write-through

Cenário de Armazenamento em Cache Write-behind

A sincronização do banco de dados pode ser aprimorada pela gravação de alterações de maneira assíncrona. Isto é conhecido como um cache write-behind ou write-back. Alterações que normalmente poderiam ser gravadas de maneira síncrona no utilitário de carga são, ao invés disso, armazenadas em buffer no eXtreme Scale e gravadas no banco de dados utilizando um encadeamento secundário. O desempenho de gravação é significativamente aumentado pois a operação do banco de dados é removida da transação do cliente e as gravações do banco de dados podem ser compactadas. Consulte “Armazenamento em Cache Write-behind” na página 34 para obter mais informações.

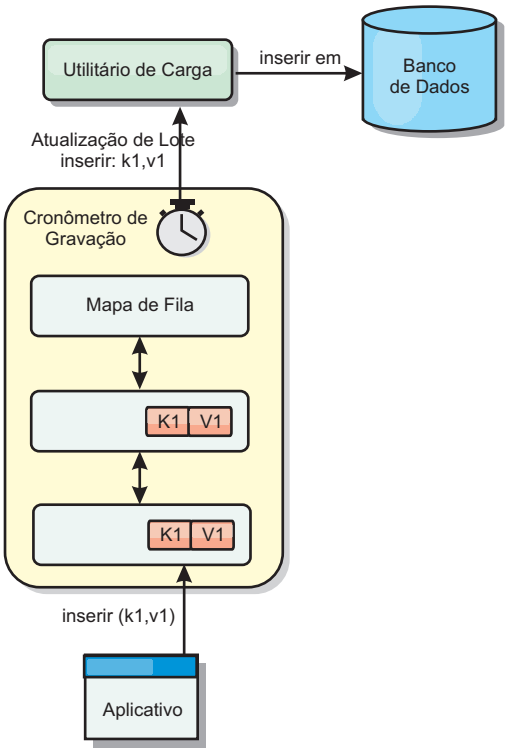


Figura 24. Armazenamento em Cache Write-behind

Consulte “Armazenamento em Cache Write-behind” na página 34 para obter informações adicionais.

Utilitários de Carga

Um utilitário de carga é um plug-in que funciona como um link entre um BackingMap e um backend como um banco de dados.

O utilitário de carga é chamado quando o cache não pode satisfazer uma solicitação para uma chave, fornecendo capacidade read-through e lazy-population do cache. Um utilitário de carga também permite atualizações no banco de dados quando os valores do cache mudam. Todas as alterações sem um transação são agrupadas juntas para permitir que a quantidade de interações do banco de dados seja reduzida. Um plug-in TransactionCallback é usado em conjunto com o utilitário de carga para acionar a demarcação da transação backend. O uso deste plug-in é importante quando múltiplos mapas são incluídos em uma única transação ou quando os dados da transação forem enviados para o cache sem consolidação.

O utilitário de carga também pode usar atualizações super qualificadas para evitar manter bloqueios do banco de dados. Armazenar um atributo de versão no valor do cache, permite ao utilitário de carga ver a imagem antes e depois do valor quando ele for atualizado no cache. Este valor pode assim ser usado ao atualizar o banco de dados ou backend para verificar se os dados foram atualizados. Um utilitário de carga também pode ser configurado para pré-carregar a grade quando é iniciado. Quando particionado, uma instância do utilitário de carga é associada a cada partição. Se o Mapa ″Company″ tiver dez partições, haverá dez instâncias do utilitário de carga, uma por partição primária. Quando shard primário para o Mapa é ativado, o método preloadMap para o utilitário de carga é chamado síncrona ou assincronamente, o qual permite o carregamento da partição do mapa com dados a partir do backend ocorra automaticamente. Quando chamado

sincronamente, todas as transações do cliente são bloqueadas, evitando o acesso inconsistente à grade. Alternativamente, um pré-utilitário de carga pode ser usado para carregar a grade toda.

Para obter mais informações sobre os utilitários de carga, consulte as informações sobre os utilitários de carga no *Guia de Administração*.

Armazenamento em Cache Write-behind

É possível usar o armazenamento em cache write-behind para reduzir o gasto adicional que ocorre na atualização de um banco de dados backend.

Apresentação

O armazenamento em cache write-behind enfileira assincronamente as atualizações no plug-in do Utilitário de Carga. É possível melhorar o desempenho desconectando atualizações, inserções e remoções para um mapa, a sobrecarga de atualização do banco de dados de backend. A atualização assíncrona é executada após um atraso baseado em tempo (por exemplo, cinco minutos) ou um atraso baseado em entradas (1000 entradas).

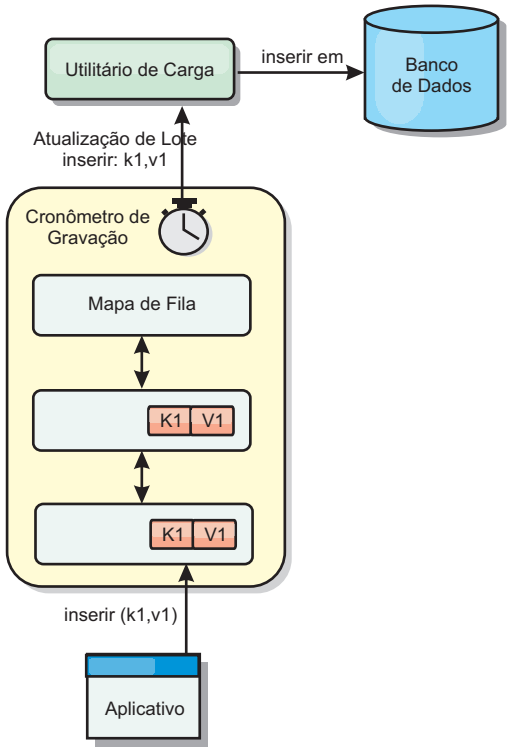


Figura 25. Armazenamento em Cache Write-behind

A configuração write-behind em um BackingMap cria um encadeamento entre o utilitário de carga e o mapa. O utilitário de carga então delega pedidos de dados através do encadeamento de acordo com as definições da configuração no método BackingMap.setWriteBehind. Quando uma transação do eXtreme Scale insere, atualiza ou remove uma entrada de um mapa, um objeto LogElement é criado para cada um destes registros. Estes elementos são enviados para o utilitário de carga write-behind e enfileirados em um ObjectMap especial denominado mapa de fila. Cada mapa de apoio com a configuração write-behind ativada possui seus

próprios mapas de fila. Um encadeamento write-behind remove periodicamente os dados enfileirados dos mapas de fila e executa o push deles para o utilitário de carga de backend real.

O utilitário de carga write-behind enviará apenas os tipos insert, update e delete dos objetos LogElement para o utilitário de carga real. Todos os outros tipos de objetos LogElement, por exemplo, o tipo EVICT, são ignorados.

Benefícios

Ativar o suporte write-behind possui os seguintes benefícios:

- **Isolamento de falha de backend:** O armazenamento em cache write-behind fornece uma camada de isolamento das falhas de backend. Quando o banco de dados de backend falha, as atualizações são enfileiradas no mapa de fila. Os aplicativos podem continuar a conduzir transações para o eXtreme Scale. Quando o backend se recupera, os dados no mapa de fila são enviados para o backend.
- **Carga de backend reduzida:** O utilitário de carga write-behind mescla as atualizações em uma base de chave, portanto, apenas uma atualização mesclada por chave existe no mapa de fila. Esta mesclagem diminui o número de atualizações no backend.
- **Desempenho de transação aprimorado:** Tempos de transação do eXtreme Scale individuais são reduzidos porque a transação não precisa aguardar até que os dados sejam sincronizados com o backend.

Considerações de Design do Aplicativo

Ativar o suporte write-behind é simples, mas o design de um aplicativo para trabalhar com o suporte write-behind precisa de consideração cuidadosa. Sem o suporte de write-behind, a transação de ObjectGrid engloba a transação de backend. A transação do ObjectGrid inicia antes da transação de backend iniciar e termina após a transação de backend terminar.

Com suporte write-behind ativado, a transação do ObjectGrid é concluída antes que a transação de backend inicie. A transação do ObjectGrid e a transação de backend não estão acopladas.

Limitadores de Integridade Referencial

Cada mapa de apoio que é configurado com suporte write-behind possui seu próprio encadeamento write-behind para enviar os dados para o backend. Portanto, os dados que são atualizados em diferentes mapas em uma transação do ObjectGrid são atualizadas no backend em diferentes transações de backend. Por exemplo, a transação T1 atualiza a chave key1 no mapa Map1 e a chave key2 no mapa Map2. A atualização da key1 para o mapa Map1 é atualizada no backend em uma transação de backend e a key2 atualizada para o mapa Map2 é atualizada no backend em outra transação de backend por encadeamentos write-behind diferentes. Se os dados armazenados no Map1 e Map2 possuírem relações, tais como limitadores de chave estrangeira no backend, as atualizações podem falhar.

Ao projetar os limitadores de integridade referencial em seu banco de dados de backend, certifique-se de que atualizações fora de ordem sejam permitidas.

Comportamento do Bloqueio de Mapa de Fila

Outra grande diferença de comportamento da transação é o comportamento do bloqueio. O ObjectGrid suporta três diferentes estratégias de bloqueio: PESSIMISTIC, OPTIMISITIC e NONE. Os mapas de fila write-behind utilizam a estratégia de bloqueio pessimista não importando qual estratégia de bloqueio está configurada para seu mapa de apoio. Existem dois diferentes tipos de operações que adquirem um bloqueio no mapa de fila:

- Quando uma transação do ObjectGrid é confirmada ou um flush (flush de mapa ou flush de sessão) acontece, a transação lê a chave no mapa de fila e coloca um bloqueio S na chave.
- Quando uma transação do ObjectGrid é confirmada, a transação tenta atualizar o bloqueio S para o bloqueio X na chave.

Devido a este comportamento do mapa de fila extra, é possível visualizar algumas diferenças de comportamento de bloqueio.

- Se o mapa do usuário for configurado como a estratégia de bloqueio PESSIMISTIC, não há muita diferença no comportamento de bloqueio. Sempre que um flush ou commit é chamado, um bloqueio S é colocado na mesma chave no mapa de fila. Durante o momento do commit, um bloqueio X não é adquirido apenas para a chave no mapa do usuário, ele também é adquirido para a chave no mapa de fila.
- Se o mapa do usuário for configurado com a estratégia de bloqueio OPTIMISTIC ou NONE, a transação do usuário seguirá o padrão de estratégia de bloqueio PESSIMISTIC. Sempre que um flush ou commit é chamado, um bloqueio S é adquirido para a mesma chave no mapa de fila. Durante o momento do commit, um bloqueio X é adquirido para a chave no mapa de fila utilizando a mesma transação.

Novas Tentativas de Transações do Utilitário de Carga

O ObjectGrid não suporta transações 2-phase ou XA. O encadeamento write-behind remove registros do mapa de fila e atualiza os registros no backend. Se o servidor falhar no meio da transação, algumas atualizações de backend podem ser perdidas.

O utilitário de carga write-behind automaticamente tentará gravar novamente transações falhas e enviará uma LogSequence duvidosa para o backend para evitar a perda de dados. Esta ação requer que o utilitário de carga seja idempotente, o que significa que o Loader.batchUpdate(TxId, LogSequence) é chamado duas vezes com o mesmo valor, ele fornece o mesmo resultado como se tivesse sido aplicado uma vez. As implementações do utilitário de carga devem implementar a interface RetryableLoader para ativar este recurso. Consulte a documentação da API para obter mais detalhes.

Falha do Utilitário de Carga

O plug-in do utilitário de carga pode falhar quando não consegue se comunicar com o back end do banco de dados. Isto pode acontecer se o servidor de banco de dados ou a conexão de rede estiver inativa. O utilitário de carga write-behind irá enfileirar as atualizações e tentará executar o push das alterações de dados para o utilitário de carga periodicamente. O utilitário de carga deve notificar o tempo de execução do ObjectGrid que há um problema de conectividade do banco de dados lançando uma exceção LoaderNotAvailableException.

Portanto, a implementação do Utilitário de Carga deve poder distinguir uma falha de dados ou uma falha de utilitário de carga físico. A falha de dados deve ser lançada ou relançada como uma LoaderException ou uma OptimisticCollisionException, mas uma falha de utilitário de carga físico deve ser lançada ou relançada como uma LoaderNotAvailableException. O ObjectGrid manipula estas exceções de maneira diferente:

- Se uma LoaderException for capturada pelo utilitário de carga write-behind, o utilitário de carga write-behind a considerará falha devido a alguma falha de dados, tal como uma falha de chave duplicada. O utilitário de carga write-behind irá remover a atualização do lote e tentará atualizar um registro em um momento para isolar a falha de dados. Se uma {{LoaderException}} for capturada durante uma atualização de registro, um registro de atualização falho é criado e registrado no mapa de atualização falho.
- Se uma LoaderNotAvailableException for capturada pelo utilitário de carga write-behind, o utilitário de carga write-behind a considerará falha porque não pode se conectar ao final do banco de dados, por exemplo, porque o backend do banco de dados estiver inativo, uma conexão com o banco de dados não estiver disponível ou a rede estiver inativa. O utilitário de carga write-behind aguardará por 15 segundo e, em seguida, tentará novamente executar uma atualização de lote no banco de dados.

O erro comum é lançar uma LoaderException enquanto uma LoaderNotAvailableException deve ser lançada. Todos os registros enfileirados no utilitário de carga write-behind se tornarão atualizações de registro falhas, o que frustra o propósito do isolamento de falha do backend.

Considerações sobre Desempenho

O suporte ao armazenamento em cache write-behind aumenta o tempo de resposta removendo a atualização do utilitário de carga da transação. Ele também aumenta o rendimento do banco de dados já que as atualizações de banco de dados são combinadas. É importante compreender o gasto adicional introduzido pelo encadeamento write-behind, que executa o pull dos dados da mapa de fila e executa o push para o utilitário de carga.

A contagem máxima de atualização ou o tempo máximo de atualização necessário a ser ajustado com base nos padrões de uso e no ambiente esperados. Se o valor da contagem máxima de atualização ou o tempo máximo de atualização for muito pequeno, o gasto adicional do encadeamento write-behind pode exceder os benefícios. Configurar um valor maior para estes dois parâmetros também pode aumentar o uso da memória para enfileirar os dados e aumentar o tempo de envelhecimento dos registros do banco de dados.

Para obter um melhor desempenho, ajuste os parâmetros write-behind com base nos seguintes fatores:

- Proporção de transações de leitura e gravação
- Mesma frequência de atualização de registro
- Latência de atualização de banco de dados.

Pré-carregamento de Dados e Aquecimento

Em muitos cenários, você pode se beneficiar do pré-carregamento da grade com dados.

Quando usado como um cache completo, a grade deve manter todos os dados e deve ser carregada antes de todos os clientes poderem se conectar a eles. Quando um cache escasso é usado, você deve aquecer o cache com dados para que os clientes possam ter acesso imediato aos dados quando eles se conectarem.

Há duas abordagens para o pré-carregamento de dados na grade: Usando um plug-in do Utilitário de Carga ou um utilitário de carga do cliente, conforme descrito nas seguintes seções.

Plug-in do Utilitário de Carga

O plug-in do utilitário de carga está associado a cada mapa e é responsável pela sincronização de um único shard de partição primária com o banco de dados. O método preloadMap do plug-in do utilitário de carga é chamado automaticamente quando um shard é ativado. Assim, se você tiver 100 partições, existem 100 instâncias do utilitário de carga, cada um carregando os dados para sua partição. Quando executado de modo síncrono, todos os clientes serão bloqueados até que o pré-carregamento seja concluído.

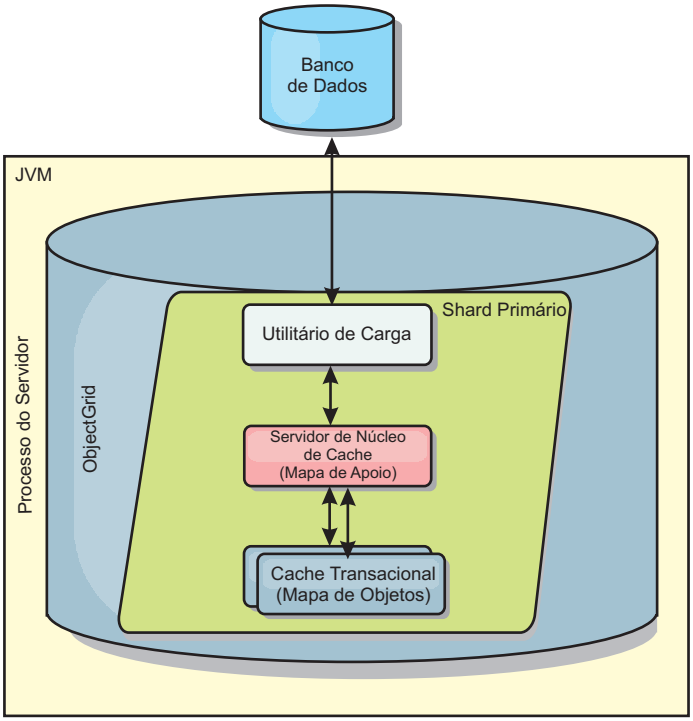


Figura 26. Plug-in do Utilitário de Carga

Utilitário de Carga do Cliente

Um utilitário de carga do cliente é um padrão para uso de um ou mais clientes para carregar a grade com dados. O uso de múltiplos clientes para carregamento de dados da grade pode ser efetivo quando o esquema de partições não está armazenado no banco de dados. Você pode chamar os utilitários de carga manual ou automaticamente quando a grade for iniciada. Os utilitários de carga do cliente podem usar opcionalmente o StateManager para configurar o estado da grade no modo de pré-carregamento, para que os clientes possam acessar a grade enquanto ela estiver pré-carregando os dados. O WebSphere eXtreme Scale inclui um utilitário de carga baseado em JPA (Java Persistence API) que você pode usar para

carregar automaticamente a grade com os provedores JPA OpenJPA ou Hibernate.

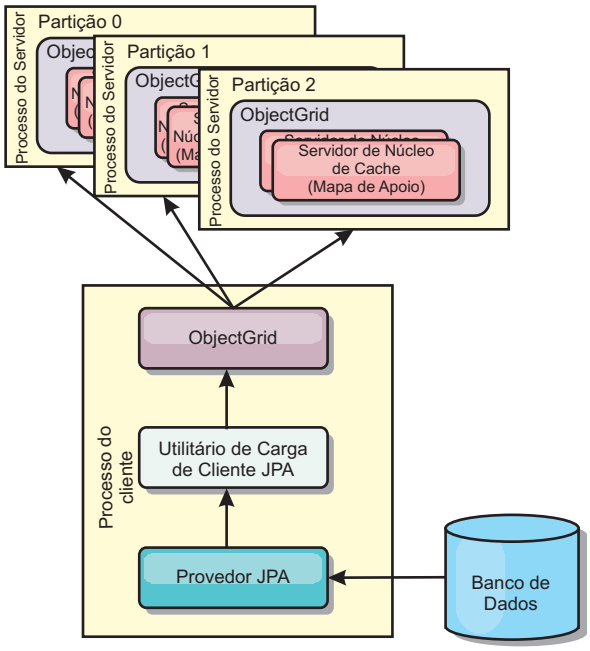


Figura 27. Utilitário de Carga do Cliente

Conceitos de Cache de Objeto Java

O WebSphere eXtreme Scale é primariamente usado como uma grade de dados e cache para objetos Java. Existem várias APIs que podem ser usadas para interagir com a grade do eXtreme Scale para acessar e armazenar esses objetos.

Este tópico descreve algumas das APIs comuns e alguns dos conceitos que você deve conhecer sobre quando escolher uma API e topologia de implementação. Consulte o tópico"Arquitetura e Topologia" na página 9 para obter uma descrição dos vários serviços e topologias que o eXtreme Scale fornece.

O componente central do WebSphere eXtreme Scale é o ObjectGrid. O ObjectGrid é o espaço de nomes que armazena dados relacionados e contém conjuntos de mapas hash, cada um deles retendo pares chave-valor. Esses mapas podem ser agrupados e particionados e transformados em altamente disponíveis e escaláveis.

Porque a grade retém objetos Java por natureza, existem algumas considerações importantes quanto ao projeto de um aplicativo para que a grade possa armazenar e acessar dados eficientemente. Os fatores que podem afetar a escalabilidade, desempenho e utilização da memória incluem o seguinte.

Considerações do Carregador de Classes e do Caminho de Classe

Como o eXtreme Scale armazena objetos Java no cache por padrão, as definições de classe devem existir no caminho de classe sempre que os dados forem acessados.

Especificamente, o cliente eXtreme Scale e os processos do contêiner devem incluir as classes ou JARs no caminho de classe ao iniciar o processo. Ao projetar um aplicativo para uso com o eXtreme Scale, separe qualquer lógica de negócios dos objetos de dados persistentes.

Consulte Carregamento de classe no WebSphere Application Server Centro de Informações de Implementação de Rede para obter mais informações.

Para obter as considerações dentro de uma configuração de Estrutura Spring, consulte a seção de pacotes no tópico sobre a integração com a estrutura Spring no *Guia de Programação*.

Para obter as configurações relacionadas ao uso do agente de instrumentação do WebSphere eXtreme Scale, consulte o tópico do agente de instrumentação no *Guia de Programação*.

Gerenciamento de Relacionamentos

Linguagens orientadas a objetos como Java, e relacionamentos ou associações de suporte a bancos de dados relacionais. Os relacionamentos diminuem a quantidade de armazenamento através do uso de referências de objetos ou chaves estrangeiras.

Ao usar relacionamentos em uma grade, os dados devem ser organizados em uma árvore limitada. Deve existir um tipo root na árvore e todos os filhos devem estar associados a somente um root. Por exemplo: Departamento pode ter muitos Funcionários e um Funcionário pode ter muitos Projetos. Porém um Projeto não pode ter muitos Funcionários pertencentes a diferentes departamentos. Depois de uma raiz ser definida, todo o acesso a este objeto raiz e seus descendentes será gerenciado através da raiz. O WebSphere eXtreme Scale usa o código hash da chave do objeto raiz para escolher uma partição.

Por exemplo: partition = (hashCode MOD numPartitions).

Quando todos os dados para um relacionamento estiverem ligados a um única instância do objeto, toda a árvore pode ser co-localizada em uma única partição e pode ser acessada muito eficientemente usando uma transação. Se os dados englobarem múltiplos relacionamentos, então múltiplas partições devem estar envolvidas que envolvem chamadas remotas adicionais, o que pode levar a gargalos no desempenho.

Dados de Referência

Alguns relacionamentos incluem dados de busca ou referência como: CountryName. Este é um caso especial em que os dados devem existir em cada partição. Aqui, os dados podem ser acessados por qualquer chave root e o mesmo resultado será retornado. Os dados de referência como estes devem ser usados somente em casos onde os dados forem razoavelmente estáticos sendo que a atualização deles pode ser cara, pois necessitam de atualização em cada partição. A API DataGrid é uma técnica comum para manter os dados de referência atualizados.

Custos e Benefícios de Normalização

A normalização dos dados usando os relacionamentos pode ajudar a reduzir a quantidade de memória usada pela grade pois a duplicação de dados é diminuída. Porém, em geral, quanto mais dados relacionais forem incluídos, menos eles irão

expandir. Quando os dados são agrupados juntos, torna-se mais caro manter os relacionamentos e manter os tamanhos gerenciáveis. Como os dados das partições da grade baseiam-se na chave da raiz da árvore, o tamanho da árvore não é levado em consideração. Assim, se você tiver uma grande quantidade de relacionamentos para uma instância da árvore, a grade pode ficar desequilibrada, fazendo com que uma partição mantenha mais dados do que as outras.

Quando os dados são não-normalizados ou sequenciais, os dados que seriam normalmente compartilhados entre dois objetos são, em vez disso, duplicados e cada tabela pode ser independentemente particionada, oferecendo uma grade mais balanceada. Apesar disto aumentar a quantidade de memória usada, permite que o aplicativo escale pois uma única linha de dados pode ser acessada que pode ter todos os dados necessários. Isto é ideal para grades com maior quantidade de leituras pois a manutenção dos dados se torna mais cara.

Para obter informações adicionais, consulte Classificação de sistemas XTP e escalamento.

Gerenciamento de Relacionamentos Usando as APIs de Acesso a Dados

A API ObjectMap é a mais rápida, mais flexível e granular das APIs de acesso a dados, oferecendo uma abordagem transacional baseada em sessão no acesso aos dados na grade de mapas. A API ObjectMap permite aos clientes usarem operações CRUD comuns (create, read, update e delete) para gerenciar pares chave-valor de objetos na grade distribuída.

Ao usar a API ObjectMap, os relacionamentos de objetos devem ser expressos pela incorporação da chave estrangeira para todos os relacionamentos no objeto-pai.

A seguir, está um exemplo.

```
public class Department {
 Collection<String> employeeIds;
}
```

A API EntityManager simplifica o gerenciamento de relacionamentos através da extração de dados persistentes a partir de objetos incluindo as chaves estrangeiras. Quando o objeto é posteriormente recuperado da grade, o gráfico de relacionamentos é reconstruído, como no exemplo a seguir.

```
@Entity
public class Department {
 Collection<String> employees;
}
```

A API EntityManager é muito semelhante a outras tecnologias de persistência do objeto Java como JPA e Hibernate na qual ela sincroniza um gráfico de instâncias de objetos Java gerenciados com o armazenamento persistente. Neste caso, o armazenamento persistente é uma grade eXtreme Scale, em que cada entidade é representada como um mapa e o mapa contém os dados da entidade em vez das instâncias do objeto.

Considerações-Chave sobre Cache

O WebSphere eXtreme Scale usa mapas hash para armazenar dados na grade, na qual um objeto Java é usado para a chave.

Diretrizes

Ao escolher uma chave, considere os seguintes requisitos.

- As chaves nunca podem ser alteradas. Se uma porção da chave precisar ser alterada, a entrada do cache deverá ser removida e reinserida.
- As chaves devem ser pequenas. Como as chaves são usadas em toda operação de acesso a dados, é recomendável manter a chave pequena para que ela possa ser serializada eficientemente e usar menos memória.
- Implementa um bom hash e algoritmo de igualdade. Os métodos hashCode() e equals(Object o) devem sempre ser substituídos para cada objeto-chave.
- Armazenar o hashCode da chave. Se possível, armazene em cache o código hash na instância do objeto-chave para acelerar os cálculos de hashCode(). Como a chave é imutável, o hashCode deve ser armazenável em cache.
- Evite duplicar a chave no valor. Ao usar a API ObjectMap, é conveniente armazenar a chave dentro do objeto do valor. Quando isso é feito, os dados-chave são duplicados na memória.

Desempenho de Serialização

O WebSphere eXtreme Scale usa múltiplos processos Java para reter os dados. Os dados, que estão na forma de instâncias de objeto Java, é convertido para bytes e de volta aos objetos novamente conforme necessário para mover os dados entre os processos do cliente e do servidor. Delegar os dados é a operação mais dispendiosa e deve ser endereçada pelo desenvolvedor de aplicativos quando o esquema é projetado, configurando a grade e interagindo com as APIs de acesso a dados.

As rotinas de cópia e serialização Java são relativamente lentas e podem consumir de 60% a 70% do processador em uma configuração típica. As seções a seguir são escolhas para melhorar o desempenho da serialização.

Gravação em um ObjectTransformer para cada BackingMap

Um ObjectTransformer pode ser associado com um BackingMap. O aplicativo pode ter uma classe que implementa a interface ObjectTransformer e fornece implementações para as seguintes operações:

- Copiando valores
- Serializando e aumentando chaves para e de fluxos
- Serializando e aumentando valores para e de fluxos

O aplicativo não precisa copiar chaves, porque elas são consideradas imutáveis.

Nota: O ObjectTransformer é chamado apenas quando o ObjectGrid conhece os dados que estão sendo transformados. Por exemplo, quando agentes de API do DataGrid são utilizados, os próprios agentes bem como os dados da instância do agente ou dados retornados do agente devem ser otimizados utilizando técnicas de serialização customizadas. O ObjectTransformer não é chamado para os agentes de API do DataGrid.

Usando Entidades

Ao utilizar a API do EntityManager com entidades, o ObjectGrid não armazena os objetos de entidade diretamente nos BackingMaps. A API do EntityManager converte o objeto de entidade em objetos de Tupla. Consulte Para obter mais informações, consulte o tópico sobre o uso de um utilitário de carga com os mapas

e tuplas de entidade no *Guia de Programação*. Os mapas de entidade são automaticamente associados com um ObjectTransformer altamente otimizado. Sempre que a API do ObjectMap ou a API do EntityManager for utilizada para interagir com mapas de entidade, o ObjectTransformer da entidade será chamado.

Serialização Customizada

Há alguns casos nos quais os objetos devem ser modificados para utilizar a serialização customizada, tais como a implementação da interface java.io.Externalizable ou pela implementação dos métodos writeObject e readObject para classes implementando a interface {[java.io.Serializable}}. As técnicas de serialização customizadas devem ser empregadas quando os objetos são serializados utilizando mecanismos que não os métodos da API do ObjectGrid ou da API do EntityManager.

Por exemplo, quando objetos ou entidades são armazenados como dados da instância em um agente da API do DataGrid ou o agente retorna objetos ou entidades, tais objetos não são transformados utilizando um ObjectTransformer. O agente, entretanto, utilizará automaticamente o ObjectTransformer ao utilizar a interface `EntityMixin`. Consulte Agentes do DataGrid e Mapas Baseados em Entidade para obter mais detalhes.

Matrizes de Byte

Ao usar as APIs ObjectMap ou DataGrid, os objetos de valor e chave são serializados sempre que os clientes interagem com a grade e quando os objetos são replicados. Para evitar o gasto adicional de serialização, use matrizes de byte em vez de objetos Java. As matrizes de byte são muito mais baratas para armazenar em memória porque o JDK tem menos objetos para buscar durante a coleta de lixo e elas podem ser aumentadas somente quando necessário. As matrizes de byte somente devem ser usadas se você não precisar acessar os objetos usando consultas ou índices. Como os dados são armazenados como bytes, os dados somente podem ser acessados através de sua chave.

O WebSphere eXtreme Scale pode armazenar dados automaticamente como matrizes de bytes usando a opção de configuração de mapa CopyMode.COPY_TO_BYTES, ou ele pode ser manipulado manualmente pelo cliente. Esta opção armazenará os dados de maneira eficiente na memória e também pode aumentar automaticamente os objetos dentro da matriz de bytes para uso por consulta e índices sob demanda.

Capítulo 3. Integração de Cache

O WebSphere eXtreme Scale pode integrar-se com outros produtos relacionados ao armazenamento em cache. O JPA pode ser usado entre o WebSphere eXtreme Scale e o banco de dados para integrar as alterações como um utilitário de carga. Também é possível usar o provedor de cache dinâmico do WebSphere eXtreme Scale para plugar o WebSphere eXtreme Scale no componente de cache dinâmico no WebSphere Application Server. Outra extensão para o WebSphere Application Server é o gerenciador de sessões HTTP do WebSphere eXtreme Scale, que pode ajudar a armazenar em cache as sessões HTTP.

Usando o eXtreme Scale com JPA

O Java Persistence API (JPA) é uma especificação que permite o mapeamento de objetos Java para bancos de dados relacionais. O JPA contém uma especificação completa de object-relational mapping (ORM) usando anotações de metadados da linguagem Java, descritores XML, ou ambos para definir o mapeamento entre objetos Java e um banco de dados relacional. Há um número de implementações de software livre e comerciais.

Para usar o JPA, é necessário ter um provedor JPA suportado, como OpenJPA ou Hibernate, arquivos JAR e um arquivo`META-INF/persistence.xml` no seu caminho da classe.

Visão Geral dos Utilitários de Carga JPA

Um Utilitário de Carga do Java Persistence API (JPA) é uma implementação do plug-in do utilitário de carga que usa o JPA para interagir com o banco de dados.

Os plug-ins JPALoader com.ibm.websphere.objectgrid.jpa.JPALoader e JPAEntityLoader com.ibm.websphere.objectgrid.jpa.JPAEntityLoader são dois plug-ins do utilitário de carga do JPA integrados que são usados para sincronizar os mapas do ObjectGrid com um banco de dados. É necessário ter uma implementação do JPA, como Hibernate ou OpenJPA, para usar este recurso. O banco de dados pode ser qualquer back end que seja suportado pelo provedor JPA escolhido.

É possível usar o plug-in do JPALoader ao armazenar dados usando a API ObjectMap. Use o plug-in do JPAEntityLoader ao armazenar dados usando a API EntityManager.

Arquitetura do Utilitário de Carga do JPA

O Utilitário de Carga do JPA é usado para mapas do eXtreme Scale que armazenam objetos Java antigos simples (POJO).

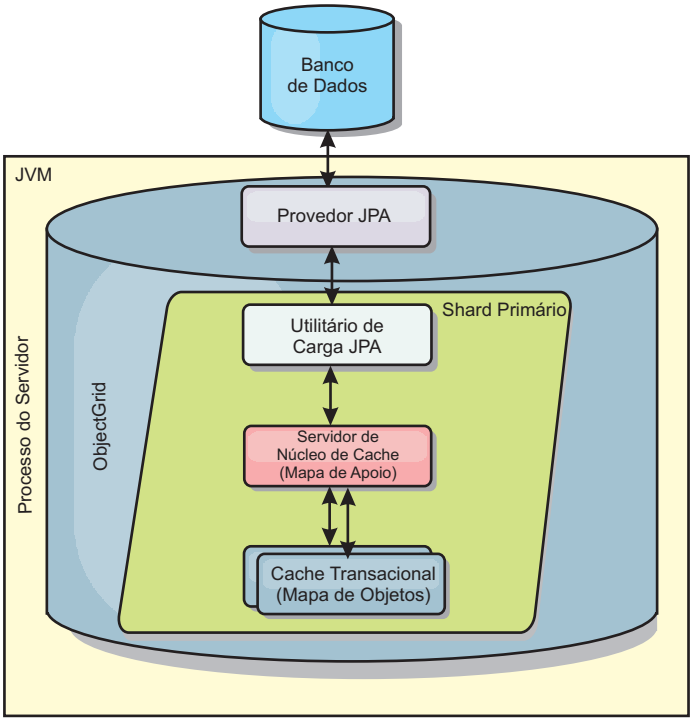


Figura 28. Arquitetura do Utilitário de Carga do JPA

Quando um método ObjectMap.get(Object key) é chamado, o eXtreme Scale executa as primeiras verificações se a entrada está contida na camada do ObjectMap. Se não, o tempo de execução delega a solicitação ao Utilitário de Carga do JPA. Sob solicitação de carregamento da chave, o JPALoader chama o método EntityManager.find(Object key) do JPA para localizar os dados de uma camada do JPA. Se os dados estiverem contidos no gerenciador de entidades JPA, ele serão retornados; caso contrário, o provedor JPA interage com o banco de dados para obter o valor.

Quando uma atualização para o ObjectMap ocorre, por exemplo, usando o método ObjectMap.update(Objectkey, Object value), o tempo de execução do eXtreme Scale cria um LogElement para esta atualização e a envia para o JPALoader. O JPALoader chama o método EntityManager.merge(Object value) do JPA para atualizar o valor no banco de dados.

Para o JPAEntityLoader, as mesmas quatro camadas estão envolvidas. Porém, como o plug-in JPAEntityLoader é usado para mapas que armazenam entidades do eXtreme Scale, as relações entre as entidades poderiam complicar o cenário de uso. Uma entidade do eXtreme Scale é diferenciada de uma entidade do JPA. Para obter mais informações, consulte Plug-in JPAEntityLoader .

Métodos

Utilitários de Carga Fornecem Três Métodos Principais:

1. get: Retorna uma lista de valores que correponde à lista de chaves que são passadas por meio da recuperação de dados usando o JPA. O método usa o JPA para localizar as entidades no banco de dados. Para o plug-inJPALoader, a lista retornada contém uma lista de entidades JPA diretamente a partir da operação find. Para o plug-in JPAEntityLoader, a lista retornada contém tuplas do valor da entidade de eXtreme Scale convertidas de entidades do JPA.

2. batchUpdate: grava os dados dos mapas do ObjectGrid para o banco de dados. Dependendo dos diferentes tipos de operação (inserir, atualizar ou excluir), o utilitário de carga usa as operações de persistir, mesclar ou remover para atualizar os dados para o banco de dados. Para o JPALoader, os objetos no mapa são utilizados diretamente como entidades JPA. Para o JPAEntityLoader, as tuplas de entidade no mapa são convertidas nos objetos que são utilizados como entidades JPA.
3. preloadMap: Pré-carrega o mapa usando o método do utilitário de carga do ClientLoader.load. Para mapas particionados, o método preloadMap é chamado apenas em uma partição. A partição é especificada na propriedade preloadPartition da classe JPALoader ou JPAEntityLoader. Se o valor de preloadPartition for configurado para menor que zero ou maior que (*total_number_of_partitions* - 1), o pré-carregamento será desativado.

Ambos os plug-ins JPALoader e JPAEntityLoader funcionam com a classe JPATxCallback para coordenar as transações do eXtreme Scale e as transações do JPA. O JPATxCallback precisa ser configurado na instância do ObjectGrid para utilizar estes dois utilitários de carga.

Plug-in do Cache JPA

O WebSphere eXtreme Scale inclui plug-ins de cache de nível 2 (L2) para ambos os provedores OpenJPA e Hibernate Java Persistence API (JPA).

Usar o eXtreme Scale como um provedor de cache L2 aumenta o desempenho na leitura e consulta de dados e reduz a carga para o banco de dados. O WebSphere eXtreme Scale tem vantagens sobre as implementações de cache integradas porque o cache é automaticamente replicado entre todos os processos. Quando um cliente armazena em cache um valor, todos os outros clientes conseguem usar o valor armazenado em cache que está localmente na memória.

Com os plug-ins de cache do ObjectGrid OpenJPA e Hibernate, é possível criar três tipos de topologia: integrada, integrada-particionada e remota.

Topologia Integrada

Uma topologia integrada cria um servidor eXtreme Scale dentro do espaço do processo de cada aplicativo. O OpenJPA e o Hibernate lêem a cópia na memória do cache diretamente e gravam para todas as outras cópias. É possível melhorar o desempenho de gravação usando replicação assíncrona. Esta topologia padrão executa melhor quando a quantidade de dados armazenados em cache é pequena o suficiente para ajustar-se a um único processo.

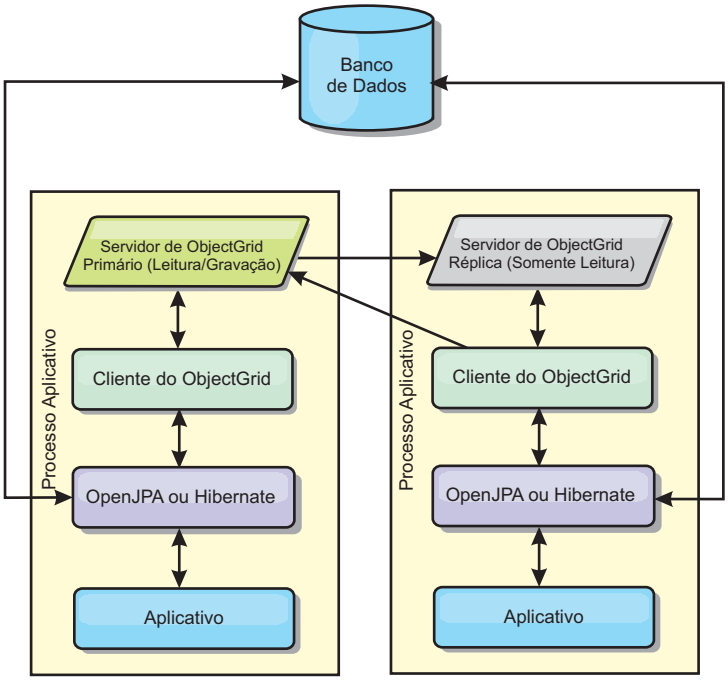


Figura 29. Topologia Integrado do JPA

Vantagens:

- Todas as leituras de cache são muito rápidas, com acessos locais.
- Simples para configurar.

Limitações:

- A quantidade de dados é limitada ao tamanho do processo.
- Todas as atualizações de cache são enviadas para um processo.

Topologia Integrada e Particionada

Quando os dados armazenados em cache são muito grandes para ajustar-se em um único processo, a topologia integrada e particionada utiliza partições do ObjectGrid para dividir os dados sobre vários processos. O desempenho não é tão alto quanto o da topologia integrada porque a maioria dos caches é remota. Porém, ainda é possível usar esta opção quando a latência do banco de dados é alta.

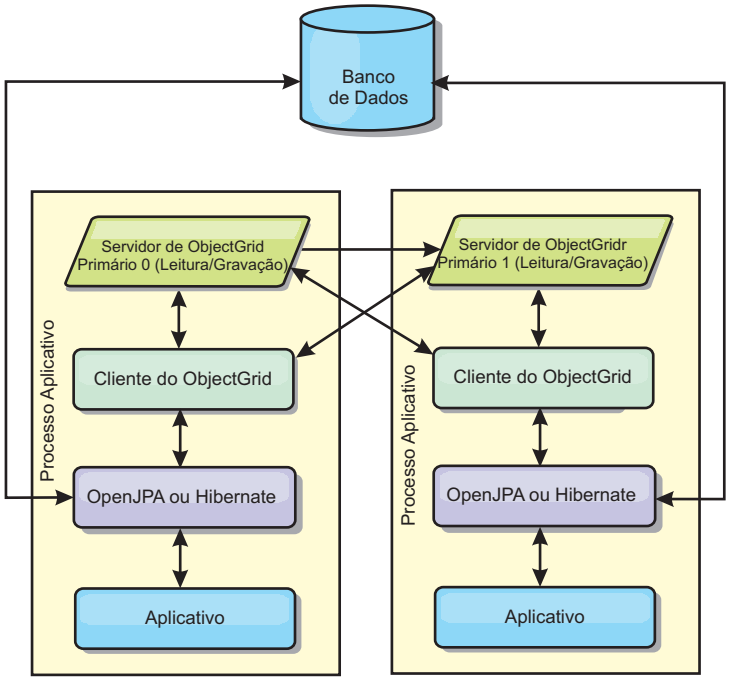


Figura 30. Topologia Particionada Integrada do JPA

Vantagens:

- Armazena grandes quantidades de dados.
- Simples para configurar
- As atualizações de cache são propagadas para vários processos.

Limitação:

- A maioria das leituras e atualizações de cache é remota.

Por exemplo, para armazenar em cache 10 GB de dados com no máximo 1 GB por JVM, dez Java Virtual Machines serão necessários. Portanto, o número de partições deve ser configurado para 10 ou mais. De forma ideal, o número de partições deve ser definido para um número primo no qual cada shard armazena uma quantidade razoável de memória. Geralmente, a configuração de numberOfPartitions é igual ao número de Java Virtual Machines. Com esta configuração, cada JVM armazena uma partição. Se você ativar a replicação, você deve aumentar o número de Java Virtual Machines no sistema; caso contrário, cada JVM também armazena uma partição de réplica, que consome tanta memória quando uma partição primária.

Por exemplo, em um sistema com 4 Java Virtual Machines, e o valor da configuração de numberOfPartitions de 4, cada JVM hospeda uma partição primária. Uma operação de leitura tem 25% mais chance de buscar dados de uma partição disponível localmente, o que é muito mais rápido comparado ao obter dados de uma JVM remota. Se uma operação de leitura, como a execução de uma consulta, precisar buscar uma coleta de dados que envolva 4 partições igualmente, 75% das chamadas são remotas e 25% das chamadas são locais. Se a configuração de ReplicaMode for definida para `SYNC` ou `ASYNC` e a configuração de ReplicaReadEnabled for definida para `true`, as quatro partições de réplica são criadas e espalhadas pelos quatro Java Virtual Machines. Cada JVM hospeda uma partição primária e uma partição de réplica. A chance de que a operação de leitura execute localmente aumenta em 50%. A operação de leitura que busca uma coleta de dados que envolve quatro partições igualmente tem somente 50% de chamadas

remotas e 50% de chamadas locais. Chamadas locais são muito mais rápidas do que chamadas remotas. Sempre que ocorrem chamada remotas, o desempenho cai.

Topologia Remota

Uma topologia remota armazena todos os dados armazenados em cache em um ou mais processos separados, reduzindo o uso da memória dos processos do aplicativo. É possível configurar o eXtreme Scale para ser particionado e replicado. Uma configuração remota é gerenciada independentemente a partir do aplicativo e do provedor JPA.

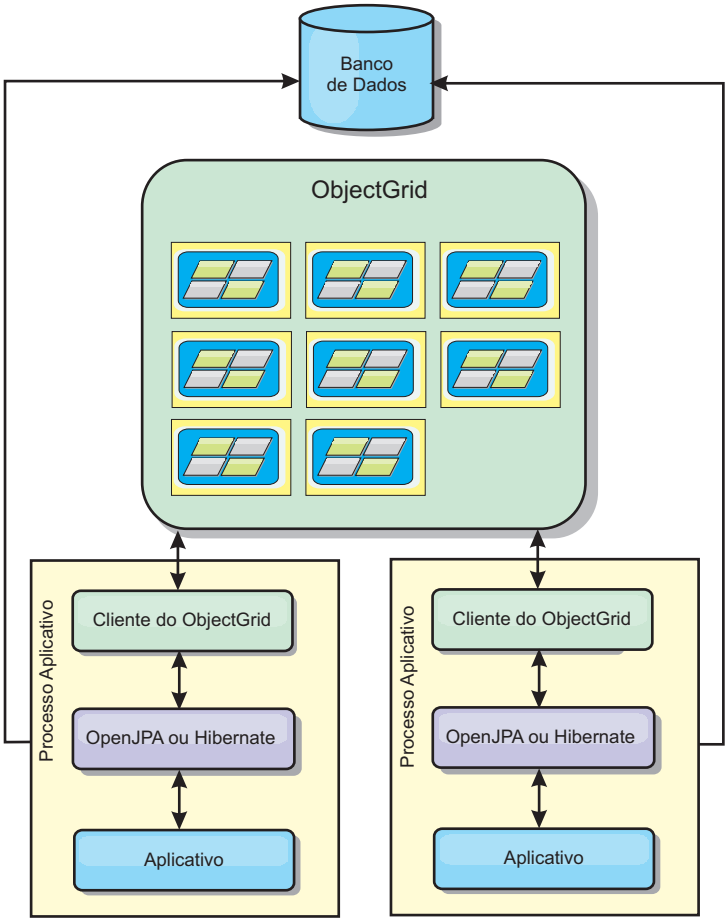


Figura 31. Topologia Remota do JPA

Vantagens:

- Armazena grandes quantidades de dados.
- O processo aplicativo é livre de dados em cache.
- As atualizações de cache são propagadas para vários processos.
- Opções de configuração muito flexíveis.

Limitação:

- Todas as leituras e atualizações de cache são remotas.

Gerenciando de Sessões HTTP

Em um ambiente WebSphere Application Server Versão 5 ou posterior, o gerenciador de sessões HTTP que é fornecido com o WebSphere eXtreme Scale pode substituir o gerenciador de sessões padrão no servidor de aplicativos.

O gerenciador de sessões HTTP do WebSphere eXtreme Scale também pode ser executado no WebSphere Application Server Versão 6.0.2 ou posterior em um ambiente que não está executando o WebSphere Application Server, tal comoWebSphere Application Server Community Edition ou Apache Tomcat.

Características

O gerenciador de sessões foi projetado de forma que possa ser executado em qualquer contêiner do Java Platform, Enterprise Edition Versão 1.4. O gerenciador de sessões não possui nenhuma dependência nas APIs do WebSphere, assim é capaz de suportar diversas versões do WebSphere Application Server, bem como ambiente do servidor de aplicativos do fornecedor.

O gerenciador de sessões HTTP fornece recursos de gerenciamento de sessões para um aplicativo associado. O gerenciador de sessões cria sessões HTTP e gerencia os ciclos de vida das sessões HTTP que estão associadas com o aplicativo. Estas atividades de gerenciamento de ciclo de vida incluem: a invalidação de sessões baseada em um tempo limite ou um servlet explícito ou chamada JavaServer Pages (JSP) e a chamada de listeners de sessão que estão associados com a sessão ou ao aplicativo da Web. O gerenciador de sessões persiste suas sessões em uma instância do ObjectGrid. Este instância pode ser uma instância local e de memória ou uma instância totalmente replicada, armazenada em cluster e particionada. O uso da topologia mais recente permite que o gerenciador de sessões forneça suporte de failover da sessão HTTP quando servidores de aplicativos são encerrados ou terminam inesperadamente. O gerenciador de sessões também pode trabalhar em ambientes que não suportam afinidade, quando a afinidade não é forçada por uma camada do balanceador de carga que pulveriza pedidos para a camada do servidor de aplicativos.

Cenários de Uso

O gerenciador de sessões pode ser usado nos seguintes cenários:

- Em ambientes que utilizam servidores de aplicativos em versões diferentes do WebSphere Application Server, tal como em um cenário de migração clássica.
- Em implementações que utilizam servidores de aplicativos de diferentes fornecedores. Por exemplo, um aplicativo que está sendo desenvolvido em servidores de aplicativos de software livre e que estão hospedados no WebSphere Application Server. Outro exemplo é um aplicativo que está sendo promovido do servidor intermediário para produção. A migração contínua destas versões do servidor de aplicativos é possível enquanto todas as sessões HTTP estão ativas e recebendo manutenção.
- Em ambientes que requerem que o usuário persista sessões com níveis de quality of service (QoS) mais altos e melhores garantias de disponibilidade de sessão durante failover de servidor do que os níveis padrão de QoS do WebSphere Application Server.
- Em um ambiente no qual a afinidade de sessão não pode ser garantida, ou ambientes nos quais a afinidade é mantida por um balanceador de carga de fornecedor e o mecanismo de afinidade precisa ser customizado para este balanceador de carga.

- Em um ambiente para transferir o gasto adicional do gerenciamento de sessões e armazenamento para um processo Java externo.
- Em várias células para ativar failover de sessão entre células.

Como o Gerenciador de Sessões Funciona

O gerenciador de sessões se introduz no caminho da solicitação na forma de um filtro de servlet. É possível incluir esse filtro de servlet em cada módulo da Web no seu aplicativo com ferramentas que são enviadas com o WebSphere eXtreme Scale. Também é possível incluir estes filtros manualmente no descritor de implementação da Web do aplicativo. Este filtro recebe o pedido antes do servlet ou dos arquivos JSP no aplicativo de destino. Neste momento, o filtro intercepta os objetos HTTPRequest e HTTPResponse e cria um objeto do wrapper com sua própria implementação.

Esta implementação de filtro intercepta todas as chamadas relacionadas de sessão HTTP que são feitas nos objetos HTTPRequest e HTTPResponse. Estas chamadas são atendidas pelo gerenciador de sessões e não são passados através do gerenciador de sessões base na implementação do servidor Java Platform, Enterprise Edition subjacentes. O gerenciador de sessões cria e mantém sessões e seus ciclos de vida, incluindo tempo limite baseado em invalidações e a ativação de listeners em invalidações de sessão e outros eventos de ciclo de vida.

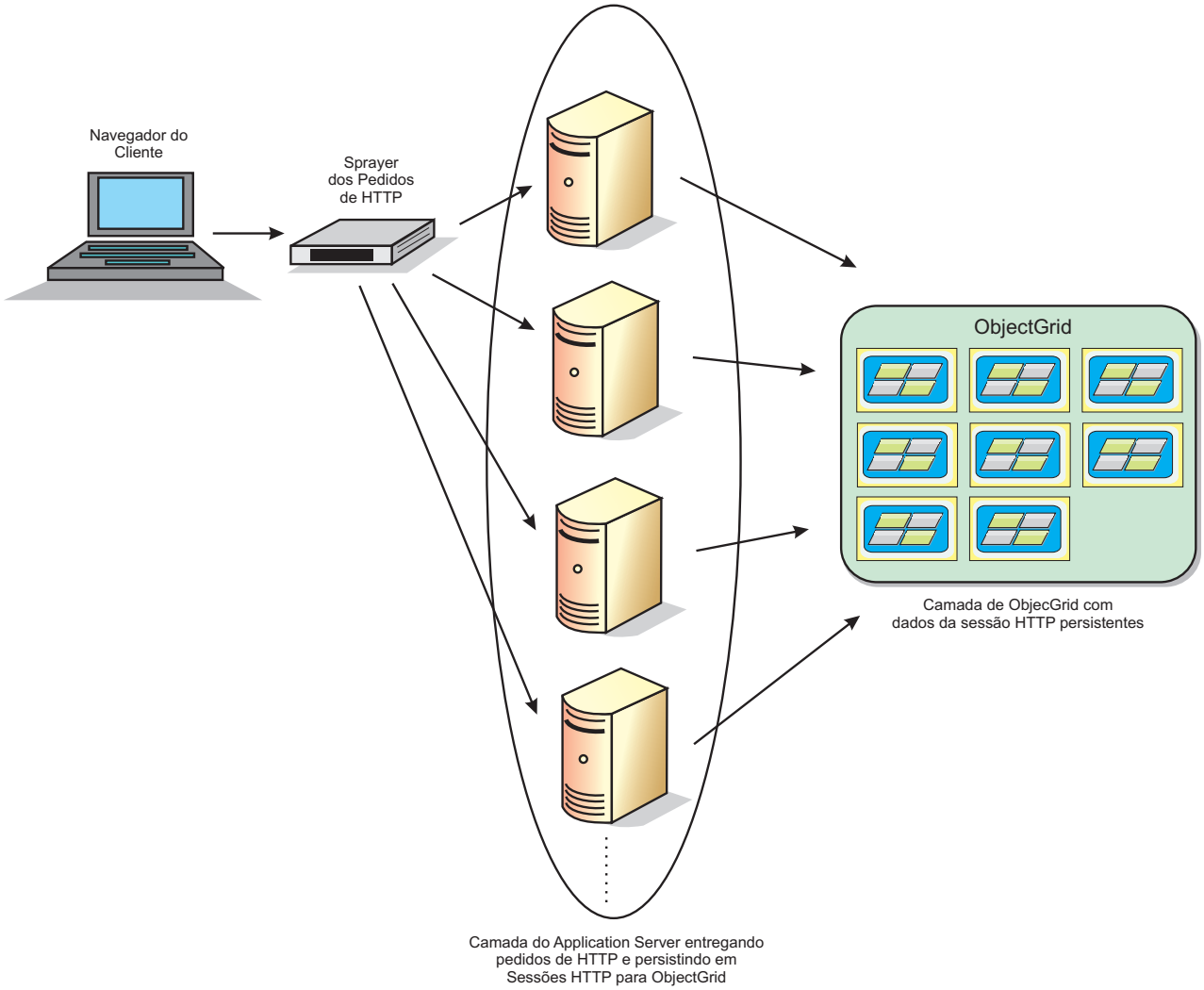


Figura 32. Topologia do Gerenciamento de Sessões HTTP com uma Configuração de Contêiner Remoto

Topologias de Implementação

O gerenciador de sessões pode ser configurado utilizando dois diferentes cenários de implementação dinâmicos:

- **Contêineres do eXtreme Scale conectados por rede integrados**

  Neste cenário, os servidores eXtreme Scale são colocados nos mesmos processos que os servlets. O gerenciador de sessões pode ser comunicar diretamente com a instância do local, evitando atrasos de rede caros.
- **Contêineres do eXtreme Scale conectados por rede remotos**

  Neste cenário, os servidores eXtreme Scale executam em processos externos ao processo no qual os servlets são executados. O gerenciador de sessões comunica-se com um eXtreme Scale remoto.

Recuperando uma Sessão do eXtreme Scale em um Aplicativo Gerenciador de Sessões

```
public void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {

        HttpSession session = req.getSession(true);

        System.out.println("calling getSession");
```

```
    // call getAttribute("com.ibm.websphere.objectgrid.session")
// to get the ObjectGrid session instance
    Session ogSession = (Session)session.getAttribute
  ("com.ibm.websphere.objectgrid.session");

    System.out.println("ogSession = "+ogSession);
}
```

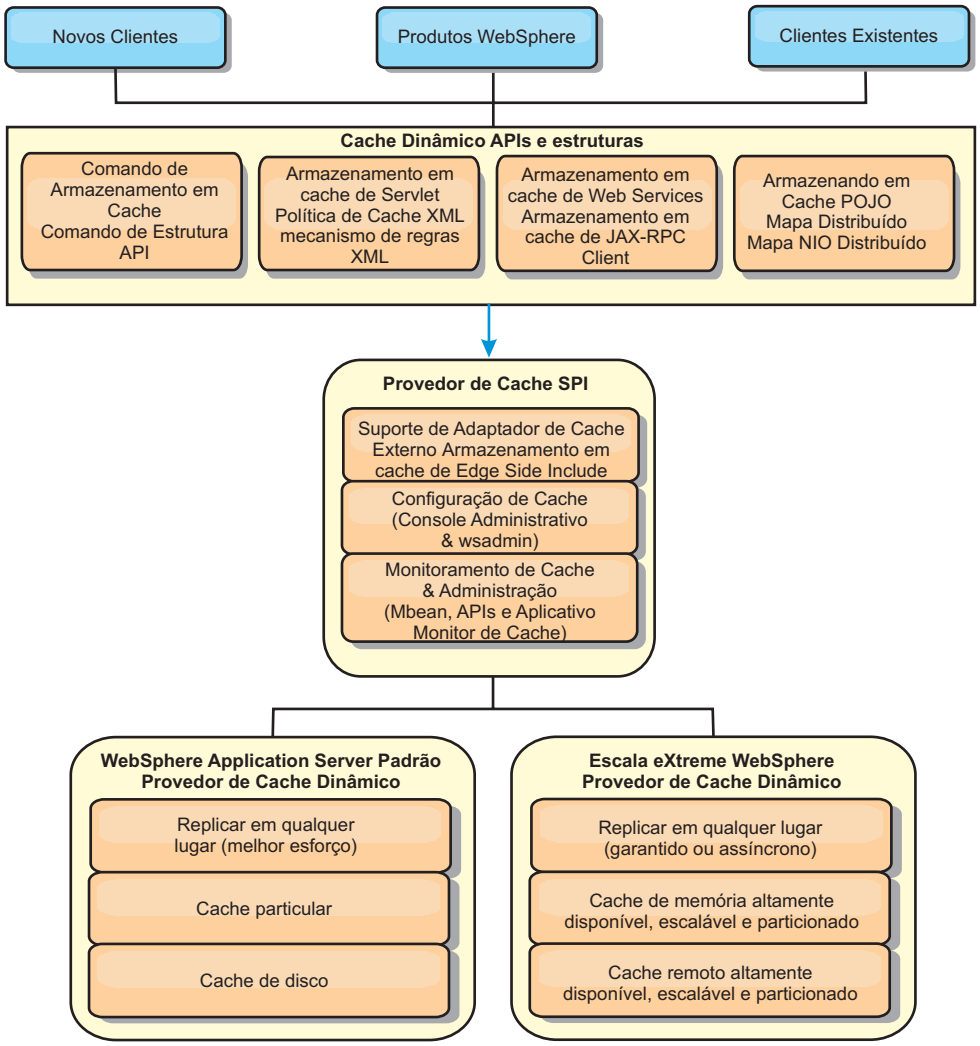
O commit de quaisquer alterações feitas nos mapas com a sessão que são retornadas da chamada do método getAttribute acontece quando ocorre o commit da sessão subjacente.

WebSphere eXtreme ScaleProvedor de Cache Dinâmico

A API de Cache Dinâmico está disponível para aplicativos Java EE implementados no WebSphere Application Server. O cache dinâmico pode ser potencializado para dados de negócios em cache, HTML gerado ou para sincronizar os dados em cache na célula usando o data replication service (DRS).

Visão Geral

Previamente, o único provedor de serviços para a API de Cache Dinâmico era o mecanismo de cache dinâmico padrão construído dentro doWebSphere Application Server. Os clientes podem usar a interface do provedor de serviço de cache dinâmico no WebSphere Application Server para plugar o eXtreme Scale no cache dinâmico. Ao configurar essa capacidade, é possível ativar aplicativos que foram gravados com a API de Cache Dinâmico ou os aplicativos usando o armazenamento em cache de nível do contêiner (como servlets) para alavancar os recursos e as capacidades de desempenho do WebSphere eXtreme Scale.



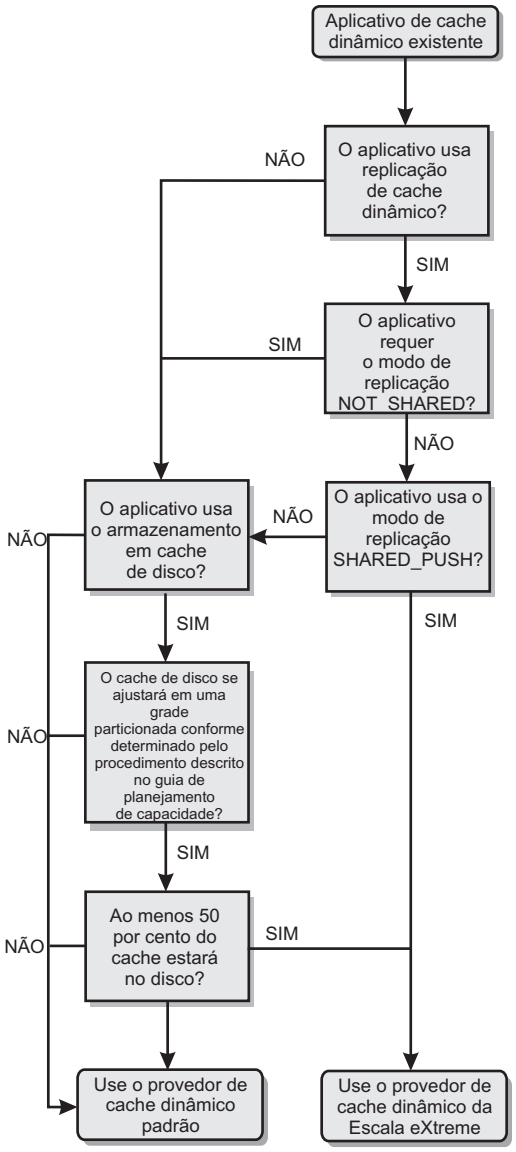
É possível instalar e configurar o provedor de cache dinâmico como descrito em "Configurando o Provedor de Cache Dinâmico para o WebSphere eXtreme Scale" na página 66

Decidindo como Potencializar o WebSphere eXtreme Scale

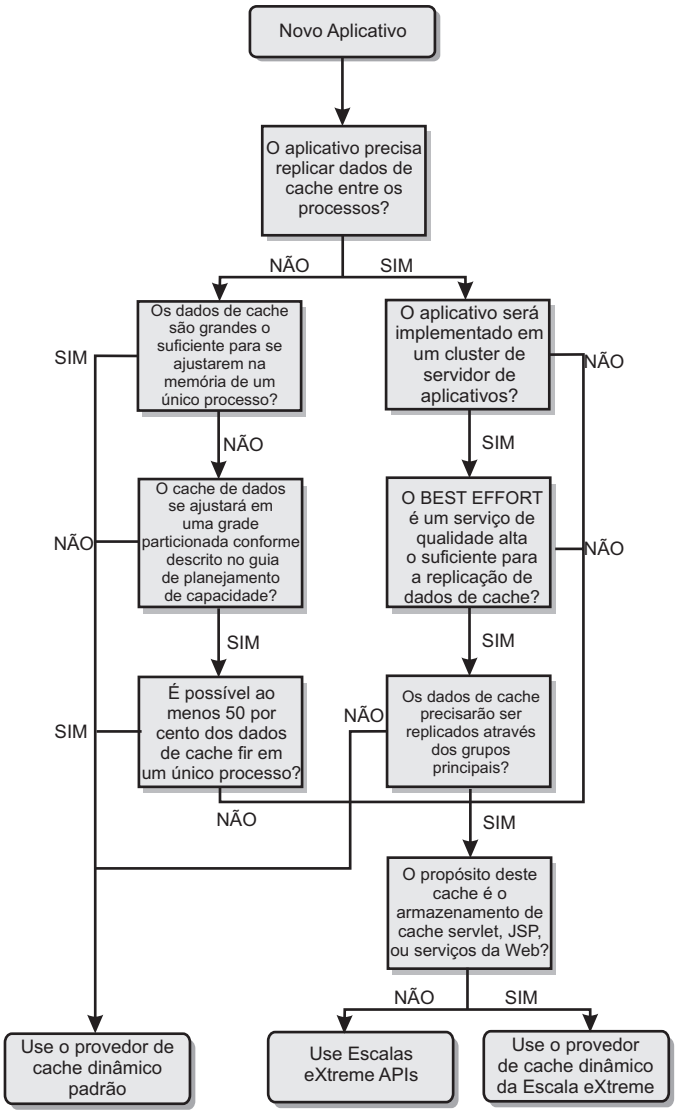
Os recursos disponíveis noWebSphere eXtreme Scale aumentam significantemente os recursos distribuídos da API do cache dinâmico além do que é oferecido pelo mecanismo de cache dinâmico e serviço de replicação de dados. Com o eXtreme Scale, é possível criar caches que são verdadeiramente distribuídos entre múltiplos servidores, em em de simplesmente replicados e sincronizados entre os servidores. Também, os caches do eXtreme Scale são transacionais e altamente disponíveis, garantindo que cada servidor veja o mesmo conteúdo para o serviço de cache dinâmico. O WebSphere eXtreme Scale oferece um qualidade de serviço mais alta para replicação de cache do que o DRS.

Porém, essas vantagens não significam que o provedor de cache dinâmico doeXtreme Scale seja a escolha certa para cada aplicativo. Use as árvores de decisão e matriz e comparação de recursos abaixo para determinar qual tecnologia se encaixa melhor no seu aplicativo.

Árvore de Decisão para Migrar Aplicativos de Cache Dinâmico Existente



Árvore de Decisão para Escolher o Provedor de Cache para Novos Aplicativos.



Comparação de Recursos

Tabela 3. Comparação de Recursos

Recursos do cache	Provedor padrão	Provedor do eXtreme Scale	eXtreme ScaleAPI
Cache na memória, local	x	x	x
Armazenamento em cache distribuído	Integrado	Integrado, particionado integrado e particionado remoto	Múltiplo
Linearmente escalável		x	x
Replicação confiável (síncrona)		ORB	ORB

Tabela 3. Comparação de Recursos (continuação)

Recursos do cache	Provedor padrão	Provedor do eXtreme Scale	eXtreme ScaleAPI
Estouro de disco	x		
Despejo	LRU/TTL/baseado em heap	LRU/TTL (por partição)	Múltiplo
Invalidação	x	x	x
Relacionamentos	IDs de dependência, modelos	IDs de dependência, modelos	x
Consultas sem chave			Consulta e índice
Integração de backend			Utilitários de Carga
Transacional		Implícito	x
Armazenamento baseado em chave	x	x	x
Eventos e listeners	x	x	x
Integração do WebSphere Application Server	Somente célula única	Célula múltipla	Célula independente
Suporte à Java Standard Edition		x	x
Monitoramento e estatística	x	x	x
Segurança	x	x	x

Tabela 4. Integração de Tecnologia Transparente

Recursos do cache	Provedor padrão	Provedor do **eXtreme Scale**	eXtreme Scale API
Armazenamento em cache dos resultados do servlet/JSP do WebSphere Application Server	V5.1+	V6.1.0.25+	
Armazenamento em cache do resultado do WebSphere Application Server Web Services (JAX-RPC)	V5.1+	V6.1.0.25+	
Armazenamento em cache de sessão HTTP			x
Provedor de cache para OpenJPA e Hibernate			x
Sincronização de banco de dados usando OpenJPA e Hibernate			x

Tabela 5. Interfaces de Programação

Recursos do cache	Provedor padrão	Provedor do eXtreme Scale	eXtreme ScaleAPI
API baseada em comandos	API da estrutura de comandos	API da estrutura de comandos	API de DataGrid
API baseada em mapa	API de DistributedMap	API de DistributedMap	API do ObjectMap
API do EntityManager			x

Para obter uma descrição mais detalhada sobre como os caches distribuídos do eXtreme Scale funcionam, consulte "Configurações de implementação para eXtreme Scale" no *Guia de Programação*.

Nota: Um cache distribuído do eXtreme Scale somente pode armazenar entradas nas quais ambos, a chave e o valor, implementam a interface java.io.Serializable.

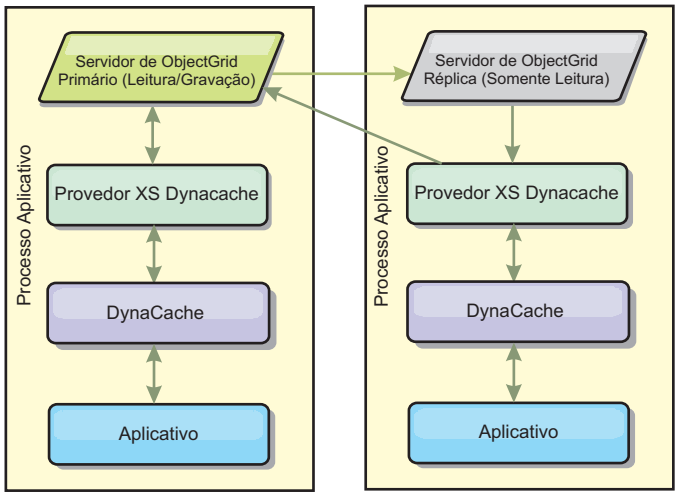
Tipos de topologia

Um serviço de cache dinâmico criado com o provedor doeXtreme Scale pode ser implementado em qualquer uma de três topologias disponíveis, permitindo que você padronize o cache especificamente para desempenho, recurso e necessidades administrativas. Essas topologias são integradas, particionadas integradas e remotas.

Topologia Integrada

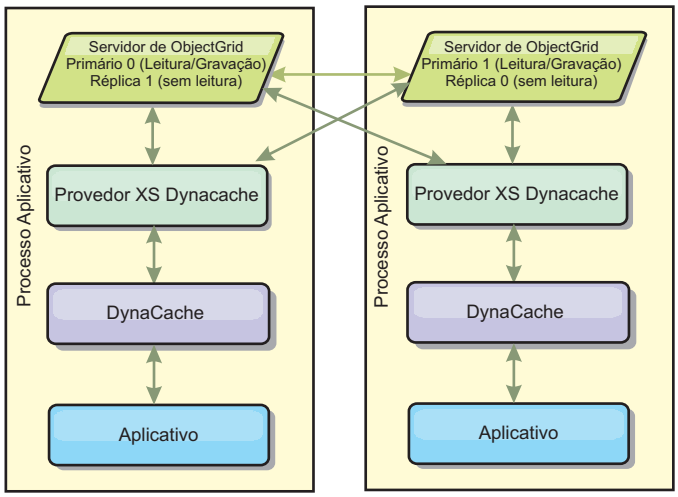
A topologia integrada é similar ao cache dinâmico padrão e ao provedor DRS. Instâncias de cache distribuído criados com a topologia integrada mantêm uma cópia integral do cache dentro de cada processo doeXtreme Scale que acessa o serviço de cache dinâmico, permitindo que todas as operações de leitura ocorram localmente. Todas as operações de gravação passam por um processo de servidor único, no qual os bloqueios transacionais são gerenciados, antes de serem replicados para o restante dos servidores. Consequentemente, esta topologia é melhor para cargas de trabalho nas quais as operações de leitura de cache excedem grandemente as operações de gravação em cache.

Com a topologia integrada, entradas de cache novas e atualizadas não são imediatamente visíveis em cada processo de servidor único. Uma entrada de cache não será visível, mesmo para o servidor que a gerou, até que ela se propague através de serviços de replicação assíncronos do WebSphere eXtreme Scale. Esses serviços operam tão rapidamente quanto o hardware permitir, mas ainda há um pequeno atraso. A topologia integrada é mostrada na seguinte imagem:



Topologia particionada integrada

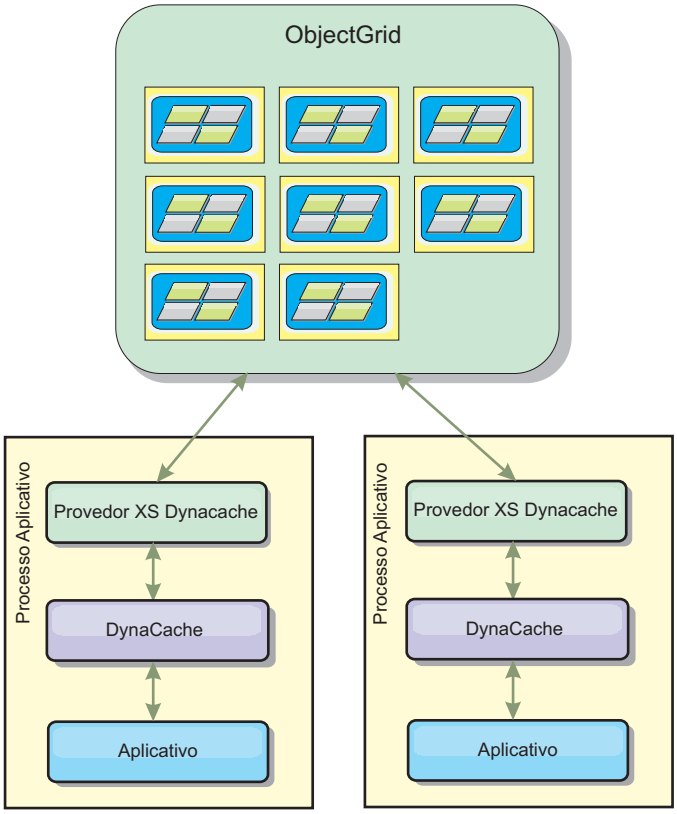
Para cargas de trabalho nas quais as gravações em cache ocorrem tão frequentemente quanto ou mais frequentemente que as leituras, as topologias remotas ou particionadas integradas são recomendadas. A topologia particionada integrada mantém todos os dados do cache dentro dos processos do WebSphere Application Server que acessam o cache. Porém, cada processo somente armazena uma parte dos dados do cache. Todas as leituras e gravações para os dados localizados nesta "partição" passam pelo processo, o que significa que a maioria dos pedidos para o cache será preenchido com uma chamada de procedimento remoto. Isto resulta em uma latência mais alta para operações de leitura do que a topologia integrada, mas a capacidade do cache distribuído de manipular operações de leitura e gravação escalarão linearmente com o número de processos doWebSphere Application Server que acessa o cache. Também, com esta topologia, o tamanho máximo do cache não é limitado pelo tamanho de um único processo doWebSphere. Porque cada processo somente retém uma parte do cache, o tamanho de cache máximo se torna o tamanho agregado de todos os processos, menos o gasto adicional do processo. A topologia particionada integrada é mostrada na seguinte imagem:



Por exemplo, suponha que você tem uma grade de processos do servidor com 256 megabytes de heap livre em cada um deles para hospedar um serviço de cache dinâmico. O provedor de cache dinâmico padrão e o provedor do eXtreme Scale que usa a topologia integrada seriam, ambos, limitados a um tamanho de cache de memória de 256 megabytes menos o gasto adicional. Consulte a seção Planejamento de Capacidade e Alta Disponibilidade, posteriormente neste documento. O provedor doeXtreme Scale que usa a topologia particionada integrada seria limitado a um tamanho de cache de um gigabyte menos o gasto adicional. Desta maneira, o provedor do WebSphere eXtreme Scale possibilita ter serviços de cache dinâmico na memória maiores que o tamanho de um único processo do servidor. O provedor de cache dinâmico padrão conta com o uso de um cache de disco para permitir que instâcias de cache cresçam além do tamanho de um processo único. Em muitas situações, o provedor doWebSphere eXtreme Scale pode eliminar a necessidade de um cache de disco e os dispendiosos sistemas de armazenamento em disco necessários para fazê-los executar.

Topologia Remota

A topologia remota também pode ser usada para eliminar a necessidade de um cache de disco. A única diferença entre as topologias remotas e particionadas integradas é que todas os dados em cache são armazenados fora dos processos do WebSphere Application Server quando você está usando a topologia remota. O WebSphere eXtreme Scale suporta processos de contêiner independentes para dados do cache. Esses processos de contêiner têm um gasto adicional mais baixo do que um processo do WebSphere Application Server e também não são limitados ao uso de uma Java Virtual Machine (JVM) particular. Por exemplo, os dados para um serviço de cache dinâmico que está sendo acessado por um processo do WebSphere Application Server de 32 bits poderiam ser alocados em um processo de contêiner do eXtreme Scale que estivesse executando em uma JVM de 64 bits. Isso permite aos usuários potencializar a capacidade de memória aumentada de processos de 64 bits para armazenamento em cache, sem incorrer um gasto adicional de 64 bits para processos do servidor de aplicativos. A topologia remota é mostrada na seguinte imagem:



Compactação de dados

Outro recurso de desempenho oferecido pelo provedor de cache dinâmico do WebSphere eXtreme Scale que pode ajudar os usuários a gerenciar o gasto adicional de cache é a compactação. O provedor de cache dinâmico padrão não permite compactação de dados em cache na memória. Com o provedor doeXtreme Scale, isso se torna possível. A compactação de cache que usa o algoritmo de deflação pode ser ativada em qualquer uma das três topologias distribuídas. Ativar a compactação aumentará o gasto adicional para operações de leitura e gravação, mas aumentará drasticamente a densidade do cache para aplicações como armazenamento em cache de servlet e JSP.

Cache de Memória Local

O provedor do WebSphere eXtreme Scale Dynamic Cachetambém pode ser usado para recuperar instâncias do Cache Dinâmico que têm **replicação desativada**. Como o provedor de Cache Dinâmico padrão, esses caches podem armazenar dados não-serializáveis. Eles também podem oferecer melhor desempenho do que o provedor de Cache Dinâmico em grandes servidores corporativos com múltiplos processadores porque o caminho do código doeXtreme Scale é projetado para maximizar a simultaneidade de cache de memória.

Mecanismo de Cache Dinâmico e Diferenças Funcionais do eXtreme Scale

No caso de caches na memória locais nos quais a replicação está desativada, não deve haver nenhuma diferença funcional apreciável entre caches recuperados pelo provedor de cache dinâmico padrão e o WebSphere eXtreme Scale. Os usuários não devem notar uma diferença funcional entre os dois caches, exceto que os caches

recuperados do WebSphere eXtreme Scale não suportam transferência de disco ou estatísticas e operações relacionados ao tamanho do cache de memória.

No caso de caches nos quais a replicação está ativada, não deve haver nenhuma diferença funcional apreciável nos resultados retornados pela maioria das chamadas de API de Cache Dinâmico, independentemente de o cliente estar usando o provedor de cache dinâmico padrão ou o provedor de cache dinâmico do eXtreme Scale. Para algumas operações não é possível emular o comportamento do mecanismo de cache dinâmico usando o eXtreme Scale.

Estatísticas do Cache Dinâmico

As estatísticas do cache dinâmico são reportadas via o aplicativo CacheMonitor ou o Dynamic Cache MBean. Quando o provedor de cache dinâmico do eXtreme Scale for usado, as estatísticas ainda serão reportadas através dessas interfaces, mas o contexto dos valores estatísticos serão diferente.

Se uma instância do cache dinâmico é compartilhada entre três servidores nomeados A, B e C, então o objeto das estatísticas do cache dinâmico somente retorna estatísticas para a cópia do cache no servidor no qual a chamada é feita. Se as estatísticas são recuperadas no servidor A, elas refletem somente a atividade no servidor A.

Com o eXtreme Scale, existe somente um único cache distribuído compartilhado entre todos os servidores, assim, é impossível controlar a maioria das estatísticas em uma base servidor-a-servidor como o provedor de cache dinâmico padrão faz. Uma lista das estatísticas reportadas pela API de Estatísticas de Cache e o que elas representam quando você está usando o provedor de cache dinâmico do WebSphere eXtreme Scale estão a seguir. Como o provedor padrão, essas estatísticas não são sincronizadas e, portanto, podem variar até 10% para cargas de trabalho simultâneas.

- **Acessos ao Cache** : Os acessos ao cache são controlados por servidor. Se o tráfego no Servidor A gerar 10 acessos ao cache e o tráfego no Servidor B gerar 20 acessos ao cache, as estatísticas do cache relatarão 10 acessos ao cache no Servidor A e 20 acessos ao cache no Servidor B.
- **Perdas de Acertos no Cache**: As perdas de acerto no cache são controladas por servidor assim como os acessos ao cache.
- **Entradas do Cache de Memória**: Esta estatística relata o número de entradas de cache no cache distribuído. Cada servidor que acessa o cache relatará o mesmo valor para esta estatística, e esse valor será o número total de entradas do cache de memória sobre todos os servidores.
- **Tamanho do Cache de Memória em MB**: Esta métrica não é atualmente suportada e sempre retornará -1.
- **Remoções do Cache**: Esta estatística relata o número total de entradas removidas do cache por qualquer método, e é um valor agregado para o cache distribuído inteiro. Se o tráfego no Servidor A gerar 10 invalidações e o tráfego no Servidor B gerar 20 invalidações, então o valor em ambos os servidores será 30.
- **Remoções de Cache Menos Usado Recentemente (LRU)**: Esta estatística é agregada, como as remoções de cache. Ele controla o número de entradas que foram removidas para manter o cache sob seu tamanho máximo.
- **Invalidações de Tempo Limite**: Esta também é uma estatística agregada, e ela controla o número de entradas que foram removidas devido a tempo limite.

- **Invalidações Explícitas**: Também uma estatística agregada, ela controla o número de entradas que foram removidas com invalidação direta por chave, ID de dependência ou modelo.
- **Estatísticas Estendidas** : O provedor de cache dinâmico do eXtreme Scale exporta as seguintes cadeias de chave de estatística estendida.
  - **com.ibm.websphere.xs.dynacache.remote_hits**: O número total de acessos ao cache controlados no contêiner do eXtreme Scale. Esta é uma estatística agregada, e seu valor no mapa de estatísticas estendidas é um `longo`.
  - **com.ibm.websphere.xs.dynacache.remote_misses**: O número total de perdas de acerto no cache controladas no contêiner do eXtreme Scale. Uma estatística agregada, seu valor no mapa de estatísticas estendidas é um `longo`.

Relatando Estatísticas de Reconfiguração

O provedor de cache dinâmico permite reconfigurar as estatísticas de cache. Com o provedor padrão a operação de reconfiguração somente limpa as estatísticas no servidor afetado. O provedor de cache dinâmico do eXtreme Scale controla a maioria de seus dados estatísticos nos contêineres de cache remoto. Estes dados não são limpos ou alterados quando as estatísticas são reconfiguradas. Em vez disso, o comportamento do cache dinâmico padrão é simulado no cliente por meio do relato da diferença entre o valor atual de uma determinada estatística e o valor dessa estatística na última vez que a reconfiguração foi chamada nesse servidor.

Por exemplo, se o tráfego no Servidor A gerar 10 remoções de cache, as estatísticas no Servidor A e no Servidor B relatarão 10 remoções. Agora, se as estatísticas no Servidor B são reconfiguradas e o tráfego no Servidor A gerar 10 remoções adicionais, as estatísticas no Servidor A relatarão 20 remoções e as estatísticas no Servidor B relatarão 10 remoções.

Eventos do Cache Dinâmico

A API do Cache Dinâmico permite aos usuários registrar listeners de evento. Quando estiver usando o eXtreme Scale como o provedor de cache dinâmico, os listeners de evento funcionarão como esperado para caches na memória local.

Para caches distribuídos, o comportamento de evento dependerá da topologia que estiver sendo usada. Para caches que usam a topologia integrada, os eventos serão gerados no servidor que manipula as operações de gravação, também conhecidos como o shard primário. Isso significa que somente um servidor receberá notificações de evento, mas ele terá todas as notificações de evento normalmente esperadas do provedor de cache dinâmico. Porque o WebSphere eXtreme Scale escolhe o shard primário no tempo de execução, é impossível garantir que um processo de servidor particular sempre receba esses eventos.

Caches particionados integrados gerarão eventos em qualquer servidor que hospede uma partição do cache. Assim, se um cache tem 11 partições e cada servidor em uma grade do WebSphere Network Deployment de 11 servidores hospedar uma das partições, então cada servidor receberá os eventos de cache dinâmico para as entradas de cache que ele hospeda. Nenhum processo de servidor único veria todos os eventos a menos que todas as 11 partições estivessem hospedadas nesse processo de servidor. Assim como ocorre com a topologia integrada, é impossível garantir que um processo de servidor particular receberá um conjunto particular de eventos ou quaisquer eventos.

Os caches que usam a topologia remota não suportam eventos de cache dinâmico.

Chamadas de MBean

O provedor de cache dinâmico do WebSphere eXtreme Scale não suporta o armazenamento em disco. Quaisquer chamadas de MBean relacionadas ao armazenamento em disco não funcionarão.

Mapeamento da Política de Replicação de Cache Dinâmico

O provedor de cache dinâmico integrado do WebSphere Application Server suporta múltiplas políticas de replicação de cache. Essas políticas podem ser configuradas globalmente ou em cada entrada de cache. Consulte a documentação de cache dinâmico para obter umadescrição dessas políticas de replicação.

O provedor de cache dinâmico do eXtreme Scale não segue essas políticas diretamente. As características de replicação de um cache são determinadas pelo tipo de topologia distribuídas do eXtreme Scale configurado e se aplicam a todos os valores colocados neste cache, independentemente do conjunto de políticas de replicação na entrada pelo serviço de cache dinâmico. A seguir há uma lista de todas as políticas de replicação suportadas pelo serviço de cache dinâmico e a ilustração de qual topologia do eXtreme Scale fornece características de replicação similares.

Note que o provedor de cache dinâmico do eXtreme Scale ignora as configurações da política de replicação DRS em um cache ou entrada de cache. Os usuários devem escolher a topologia que se adequa às suas necessidades de replicação.

- NOT_SHARED – atualmente nenhuma das topologias fornecidas pelo provedor de cache dinâmico do eXtreme Scale consegue se aproximar dessa política. Isso significa que todos os dados armazenados no cache devem ter chaves e valores que implementem java.io.Serializable.
- SHARED_PUSH – A topologia integrada se aproxima dessa política de replicação. Quando uma entrada de cache é criada, ela é replicada para todos os servidores. Os servidores somente procuram entradas de cache localmente. Se uma entrada não é localizada localmente, presume-se que ela não existe e os outros servidores não são consultados quanto a ela.
- SHARED_PULL and SHARED_PUSH_PULL – As topologias remotas e particionadas se aproximam desta política de replicação. O estado distribuído do cache é completamente consistente entre todos os servidores.

Estas informações são fornecidas principalmente para que você possa se certificar de que a topologia satisfaz suas necessidades de consistência distribuída. Por exemplo, se a topologia integrada é uma melhor escolha para as suas necessidades de desempenho e implementação, mas você precisa do nível da consistência de cache fornecido por SHARED_PUSH_PULL, então considere usar a particionada integrada, ainda que o desempenho possa ser ligeiramente mais lento.

Segurança

Você pode proteger as instâncias de cache dinâmico que estão executando em topologias particionadas integradas ou topologias integradas com a funcionalidade de segurança compilada no WebSphere Application Server. Consulte a documentação em Protegendo servidores de aplicativos no WebSphere Application Server Centro de Informações.

Quando um cache está executando em topologia remota, é possível para um cliente eXtreme Scale independente se conectar ao cache e afetar o conteúdo da instância

do cache dinâmico. O provedor de cache dinâmico do eXtreme Scale tem um recurso de criptografia de gasto adicional baixo que pode evitar que os dados do cache sejam lidos ou alterados por clientes não-WebSphere Application Server. Para ativar esse recurso, configure o parâmetro opcional **com.ibm.websphere.xs.dynacache.encryption_password** para o mesmo valor em cada instância do WebSphere Application Server que acesse o provedor de cache dinâmico. Isso irá criptografar o valor e os metadados do usuário para o CacheEntry usando criptografia AES de 128 bits. É muito importante que o mesmo valor seja configurado em todos os servidores. Os servidores não poderão ler dados colocados em cache por servidores com um valor diferentes para este parâmetro.

Se o provedor do eXtreme Scale detectar que diferentes valores estão configurados para esta variável no mesmo cache, ele gera um aviso no log do processo do contêiner do eXtreme Scale.

Consulte a documentação do eXtreme Scale sobre segurança se autenticação SSL ou do cliente for necessária.

Informações adicionais

- Redbook do Cache Dinâmico
- Documentação do Cache Dinâmico
  - WebSphere Application Server 7.0
  - WebSphere Application Server 6.1
- Documentação do DRS
  - WebSphere Application Server 7.0
  - WebSphere Application Server 6.1

Configurando o Provedor de Cache Dinâmico para o WebSphere eXtreme Scale

A instalação e configuração do provedor de cache dinâmico para o eXtreme Scale depende de quais são seus requisitos e do ambiente configurado.

Antes de Iniciar

Instale o provedor de cache dinâmico.

Para usar o provedor de cache dinâmico, o WebSphere eXtreme Scale deve ser instalado além das implementações de nó do WebSphere Application Server, incluindo o nó do gerenciador de implementação. O provedor de cache dinâmico do eXtreme Scale é suportado nas seguintes versões do WebSphere Application Server.

- WebSphere Application Server 6.1.0.25 + PK85622 e superior
- WebSphere Application Server 7.0.0.3 + PK85622 e superior

Para obter instruções de instalação, consulte Instalando para 6.1 ou Instalando para 7.0.

Por Que e Quando Desempenhar Esta Tarefa

Siga essas etapas para configurar o provedor de cache dinâmico do eXtreme Scale:

1. Ative o provedor de cache dinâmico do eXtreme Scale.

No WebSphere Application Server 7.0, é possível configurar o serviço de cache dinâmico para usar o provedor de cache dinâmico do eXtreme Scale com o console administrativo ou com uma propriedade customizada.

Depois de instalar o eXtreme Scale, o provedor de cache dinâmico do eXtreme Scale estará imediatamente disponível como uma opção ″Provedor de Cache″ no console administrativo. Para obter mais informações, consulte Selecionando um Provedor de Servido de Cache.

Também é possível configurar o provedor de cache dinâmico do eXtreme Scale para uma instância de cache ao configurar os seguinte pares propriedade customizada e valor na instância. Essas propriedades customizadas são a única maneira de ativar o provedor eXtreme Scale no WebSphere Application Server 6.1 da seguinte forma.

```
com.ibm.ws.cache.CacheConfig.cacheProviderName =
   "com.ibm.ws.objectgrid.dynacache.CacheProviderImpl"
```

Se desejar usar o provedor de cache dinâmico do eXtreme Scale para o JPS, serviços da Web ou armazenamento em cache de comando, será necessário configurar a instância baseCache para usar o provedor de cache dinâmico do eXtreme Scale. As mesmas propriedades de configuração são usadas para configurar a instância baseCache. Lembre-se também de que essas propriedades de configuração precisam ser definidas como propriedades customizadas da JVM. Essa condição se aplica a qualquer propriedade de configuração de cache abordada nessa seção, exceto para armazenamento em cache de servlet. Para usar o eXtreme Scale com o provedor de cache dinâmico para armazenamento em cache de servlet, certifique-se de configurar a ativação nas propriedades do sistema em vez de nas propriedades customizadas.

Se o baseCache for configurado para usar o provedor de cache dinâmico do eXtreme Scale, todas as outras instâncias de cache no servidor usarão o provedor eXtreme Scale por padrão. Para fazer com que a instância de cache use o provedor de cache dinâmico padrão, configure a propriedade do provedor de cache da seguinte forma: com.ibm.ws.cache.CacheConfig.cacheProviderName = ″default″. Quaisquer propriedades de configuração do provedor de cache dinâmico do eXtreme Scale configuradas para baseCache são as propriedades de configuração padrão para todas as instâncias de cache suportadas pelo eXtreme Scale. Os clientes precisam ser muito cautelosos ao depender dos valores padrão para essas propriedades.

2. Defina a configuração de replicação para o cache.

   Com o provedor de cache dinâmico do eXtreme Scale, é possível ter instâncias de cache locais e instâncias de cache replicadas. No caso de instâncias de cache locais, nenhuma outra configuração é necessária e o restante desta seção poderá ser ignorado.

   Os caches replicados podem ser criados de duas formas. A primeira forma é ativar a replicação de cache através do console de administração. Essa ativação pode ser feita a qualquer momento no WebSphere Application Server Versão 7.0, mas será necessário criar um domínio de replicação DRS no WebSphere Application Server Versão 6.1.

   A segunda maneira é usar o seguinte par propriedade customizada e valor para forçar o cache a informar que isso é um cache replicado, mesmo se o domínio de replicação DRS não tiver sido designado para ele.

   ```
   com.ibm.ws.cache.CacheConfig.enableCacheReplication = "true"
   ```

3. Configure a topologia para o serviço de cache dinâmico.

O único parâmetro de configuração necessário para o provedor de cache dinâmico do eXtreme Scale é a topologia do cache. A seguinte variável deve ser configurada como uma Propriedade Customizada no serviço de cache dinâmico.

`com.ibm.websphere.xs.dynacache.topology`

A seguir há os três valores possíveis para essa propriedade.

- integrado
- integrado_particionado
- remoto

É necessário usar um dos valores permitidos.

4. Configure o número de contêineres iniciais para o serviço de cache dinâmico.

   É possível aumentar o desempenho dos caches que estão usado a topologia particionada integrada ao configurar o número de contêineres iniciais. A seguinte variável deve ser configurada como um propriedade de sistema na WebSphere Application Server Java virtual machine.

   `com.ibm.websphere.xs.dynacache.num_initial_containers`

   O valor recomendado dessa propriedade de configuração é um número inteiro que seja igual ou pouco menor que o número total de instâncias do WebSphere Application Server que acessam essa instância de cache distribuída. Por exemplo, se um serviço de cache dinâmico for compartilhado entre os membros da grade, a variável deverá ser configurada para o número de membros da grade.

5. Configurar a grade de serviço de catálogo do eXtreme Scale.

   Quando estiver usando o eXtreme Scale como o provedor de cache dinâmico para uma instância de cache distribuída, é necessário configurar uma grade de serviço de catálogo do eXtreme Scale.

   Uma única grade de serviço de catálogo pode atender vários provedores de serviço de cache dinâmicos suportados pelo eXtreme Scale.

   Um serviço de catálogo pode ser executado dentro ou fora de processos WebSphere Application Server.

   Quando estiver executando uma grade de serviço de catálogo, é necessário configurar a propriedade customizada **catalog.services.cluster** para os terminais de serviço de catálogo.

6. Configurar os objetos de chave customizados.

   Quando estiver usando os objetos customizados como chaves, os objetos deverão implementar a interface Serializável ou Externalizável. Quando estiver usando as topologias integradas ou particionadas integradas, é necessário colocar os objetos no caminho de biblioteca compartilhado do WebSphere, como se eles estivessem sendo usados com o provedor de cache dinâmico padrão. Consulte Usando as interfaces DistributedMap e DistributedObjectCache para o cache dinâmico no centro de informações do WebSphere Application Server Network Deployment para obter mais detalhes.

   Se você estiver usando a topologia remota, você deverá colocar os objetos de chaves customizados no CLASSPATH para os contêineres do eXtreme Scale independentes.

7. Configure os servidores de contêiner do eXtreme Scale.

   Os dados em cache são armazenados nos contêineres do WebSphere eXtreme Scale. Os contêineres podem ser executados dentro ou fora dos processos do WebSphere Application Server. O provedor do eXtreme Scale cria automaticamente contêineres dentro do processo do WebSphere quando estiver

usando topologias integradas ou particionadas integradas para uma instância de cache. Nenhuma configuração adicional é necessária para essas topologias.

Quando estiver usando uma topologia remota, é necessário iniciar os contêineres eXtreme Scale independente antes que as instâncias do WebSphere Application Server que acessam a instância do cache seja inicializadas. Certifique-se de que todos os contêineres para um serviço de cache dinâmico específico apontem para os mesmos terminais de serviço de catálogo.

Os arquivos de configuração XML para os contêineres independentes do provedor eXtreme Scale Dynamic Cache estão localizados no diretório `<install_root>/customLibraries/ObjectGrid/dynacache/etc` para instalações além do WebSphere Application Server ou no diretório `<install_root>/ObjectGrid/dynacache/etc` para instalações independentes. Os arquivos são chamados de `dynacache-remote-objectgrid.xml` e `dynacache-remote-definition.xml`. Faça cópias desses arquivos para editar e usar quando ativar contêineres independentes para o provedor de cache dinâmico do eXtreme Scale. O parâmetro **numIntitialContainers** no arquivo **`dynacache-remote-deployment.xml`** deve ser configurado de acordo com o número de processos de contêineres que estão sendo executados.

Nota: O conjunto de processos de contêineres precisa ter memória livre suficiente para atender a todas as instâncias de cache dinâmico configuradas para usar a topologia remota. Qualquer processo do WebSphere Application Server que compartilhar valores iguais ou equivalentes para **catalog.services.cluster** deve usar o mesmo conjunto de contêineres independentes. O número de contêineres e o número de máquinas nas quais eles residem devem ser redimensionados adequadamente. Consulte "Planejamento de Capacidade e Alta Disponibilidade" para obter detalhes adicionais.

Um exemplo de entrada da linha de comandos UNIX® que ativa um contêiner independente para o provedor de cache dinâmico do eXtreme Scale é mostrado no seguinte código:

`startOgServer.sh` *container1* `-objectGridFile` *../dynacache/etc/dynacache-remote-objectgrid.xml* `-deploymentPolicyFile` *../dynacache/etc/dynacache-remote-deployment.xml* `-catalogServiceEndpoints` *MyServer1.company.com:2809*

Planejamento de Capacidade e Alta Disponibilidade

A API de Cache Dinâmico está disponível para os aplicativos Java EE que são implementados no WebSphere Application Server. O cache dinâmico pode ser potencializado para dados de negócios em cache, HTML gerado ou para sincronizar os dados em cache na célula usando o data replication service (DRS).

Visão Geral

Todas as instâncias de cache dinâmico criadas com o provedor de cache dinâmico do WebSphere eXtreme Scale são altamente disponíveis por padrão. O nível e custo de memória de alta disponibilidade depende da topologia usada.

Ao usar a topologia integrada, o tamanho do cache é limitado à quantidade de memória livre em um único processo do servidor, e cada processo do servidor armazena uma cópia completa do cache. Enquanto um único processo do servidor continua a executar, o cache sobrevive. Os dados do cache somente serão perdidos se todos os servidores que acessam o cache forem desligados.

Para armazenamento em cache que usa topologia particionada integrada, o tamanho do cache é limitado a um agregado do espaço livre disponível em todos os processos do servidor. Por padrão, o provedor de cache dinâmico do eXtreme Scale usa 1 réplica para cada shard primário, assim cada parte dos dados armazenados em cache é armazenada duas vezes.

Use a seguinte fórmula A para determinar a capacidade de um cache particionado integrado.

Fórmula A

F * C / (1 + R) = M

Em que:
- F = Memória livre por processo de contêiner
- C = número de contêineres
- R = número de réplicas
- M = Tamanho total do cache

Para uma grade de Implementação de Rede do WebSphere que tenha 256 MB de espaço disponível em cada processo, com 4 processos do servidor no total, uma instância de cache por todos esses servidores poderia armazenar até 512 megabytes de dados. Deste modo, o cache pode sobreviver a um dano no servidor sem perder dados. Também, até dois servidores poderiam ser desligados sequencialmente sem perda de dado algum. Assim, para o exemplo anterior, a fórmula é a seguinte:

256mb * 4 contêineres/ (1 primário + 1 réplica) = 512mb.

Caches que usam a topologia remota têm características de dimensionamento similares às dos caches que usam topologia particionada integrada, mas eles são limitados pela quantidade de espaço disponível em todos os processos de contêiner do eXtreme Scale.

Em topologias remotas, é possível aumentar o número de réplicas para fornecer um nível mais alto de disponibilidade ao custo de gasto adicional de memória. Na maioria dos aplicativos com cachê dinâmico isso seria desnecessário, mas você pode editar o arquivo dynacache-remote-deployment.xml para aumentar o número de réplicas.

Use as seguintes fórmulas, B e C, para determinar o efeito da inclusão de mais réplicas na Alta Disponibilidade do cache.

Fórmula B

N = Mínimo(T -1, R)

Em que:
- N = o número de processos que podem travar simultaneamente
- T = o número total de contêineres
- R = o número total de réplicas

Fórmula C

Limite(T/ (1+N)) = m

Em que:

- T = Número total de contêineres
- N = Número total de réplicas
- m = número mínimo de contêineres necessários para suportar os dados em cache.

Para ajuste de desempenho com o provedor de cache dinâmico, consulte "Ajustando o Provedor de Cache Dinâmico" na página 72.

Dimensionamento do Cache

Antes que um aplicativo que usa o provedor de Cache Dinâmico do WebSphere eXtreme Scale possa ser implementado, os princípios gerais descritos na seção anterior devem ser combinados com os dados ambientais para os sistemas de produção. O primeiro valor a estabelecer é o número total de processos do contêiner e a quantidade de memória disponível em cada processo para conter os dados do cache. Ao usar a topologia integrada, os contêineres de cache serão colocados dentro dos processos do servidor do WebSphere Application, assim, há um contêiner para cada servidor que estiver compartilhando o cache. Determinar o gasto adicional de memória do aplicativo sem o armazenamento em cache ativado e o WebSphere Application Server é a melhor maneira de descobrir quanto espaço está disponível no processo. Isso pode ser feito por meio de análise detalhada dos dados da coleta de lixo. Ao usar a topologia remota, essas informações podem ser localizadas por meio da verificação da saída detalhada da coleta de lixo de um contêiner independente recentemente iniciado que ainda não foi preenchido com dados do cache. A última coisa a lembrar para descobrir quanto espaço por processo está disponível para dados em cache, é reservar algum espaço de heap para a coleta de lixo. O gasto adicional do contêiner, o WebSphere Application Server ou independente, mais o tamanho reservado para o cache não deve ser maior que 70% do heap total.

Assim que essas informações são coletadas, os valores podem ser inseridos na fórmula A, descrita anteriormente, para determinar o tamanho máximo para o cache particionado. Depois que o tamanho máximo é conhecido, a próxima etapa é determinar o número total de entradas do cache que pode ser suportado, o que requer determinar o tamanho médio por entrada do cache. A maneira mais simples de fazer isso é incluir 10% ao tamanho do objeto do cliente. Consulte o Guia de Ajuste para cache dinâmico e serviço de replicação de dados para obter informações mais detalhadas sobre dimensionamento de entradas do cache ao usar o Cache Dinâmico.

Quando a compactação está ativada, ela afeta o tamanho do objeto do cliente, não o gasto adicional do sistema de armazenamento em cache. Use a fórmula a seguir para determinar o tamanho de um objeto em cache ao usar a compactação:

$S = O * C + O * 0.10$

Em que:

- S = Tamanho médio do objeto em cache
- O = Tamanho médio do objeto do cliente não-compactado
- C = Proporção de compactação expressa como uma fração.

Assim, uma proporção de compactação 2 para 1 é 1/2 = 0.50. Para este valor, quanto menor, melhor. Se o objeto que está sendo armazenado é um POJO normal cheio principalmente tipos primitivos, então assuma uma proporção de

compactação de 0.60 para 0.70. Se o objeto em cache é um objeto Servlet, JSP ou WebServices, o método ideal para determinar a taxa de compactação é compactar uma amostra representativa com um utilitário de compactação ZIP. Se isso não for possível, então uma proporção de compactação de 0.2 para 0.35 é comum para esses tipos de dados.

Depois, use estas informações para determinar o número total de entradas do cache que pode ser suportado. Use a seguinte fórmula D:

Fórmula D

T = S / A

Em que:

- T= Número total de entradas do cache
- S = Tamanho total disponível para dados em cache conforme computados usando a fórmula A
- A = Tamanho médio de cada entrada do cache

Finalmente, você deve configurar o tamanho do cache na instância do cache dinâmico para impor este limite. O provedor de cache dinâmico do WebSphere eXtreme Scale difere do provedor de cache dinâmico padrão neste aspecto. Use a fórmula a seguir para determinar o valor para definir o tamanho do cache na instância do cache dinâmico. Use a seguinte fórmula E:

Fórmula E

Cs = Ts / Np

Em que:

- Ts = Tamanho de destino total para o cache
- Cs = Configuração do Tamanho do Cache para definir na instância do cache dinâmico
- Np = Número de partições. O padrão é 47.

Configure o tamanho da instância do cache dinâmico para um valor calculado pela fórmula E em cada servidor que compartilhar a instância do cache.

Ajustando o Provedor de Cache Dinâmico

O provedor de cache dinâmico do WebSphere eXtreme Scale suporta os seguintes parâmetros de configuração para o ajuste de desempenho.

Por Que e Quando Desempenhar Esta Tarefa

- **com.ibm.websphere.xs.dynacache.ignore_value_in_change_event**: Ao registrar um listener de evento de alteração com o provedor de cache dinâmico e gerar uma instância ChangeEvent, haverá um gasto adicional associado à desserialização da entrada do cache para que o valor possa ser colocado dentro o ChangeEvent. Configurar este parâmetro opcional na instância do cache para `true` ignorará a desserialização da entrada do cache ao gerar o ChangeEvents. O valor retornado será nulo no caso de uma operação de remoção ou de uma matriz de byte que contém a forma serializada do objeto. As instâncias InvalidationEvent possuem uma penalidade de desempenho semelhante que pode ser evitada ao configurar com.ibm.ws.cache.CacheConfig.ignoreValueInInvalidationEvent para `true`.

- **com.ibm.websphere.xs.dynacache.disable_recursive_invalidate**: A invalidação de cache dinâmico é recursiva por padrão. Com o provedor de cache dinâmico do WebSphere eXtreme Scale, o gasto adicional associa-se à realização da invalidação recursiva nas topologias particionadas integradas e remotas. Se um cache não precisar de invalidação recursiva, configure esse parâmetro para `true` para evitar gasto adicional.
- **com.ibm.websphere.xs.dynacache.enable_compression**: O provedor de cache dinâmico do eXtreme Scale pode compactar as entradas de cache na memória para aumentar a densidade do cache. Isso causa um gasto adicional para as operações de leitura e de gravação, mas pode economizar uma quantia de memória significativa para os aplicativos como armazenamento em cache de servlet.

Capítulo 4. Escalabilidade

O WebSphere eXtreme Scale é escalável através do uso de dados particionados e pode ser escalado para milhares de contêineres porque cada contêiner é independente de outros contêineres.

O WebSphere eXtreme Scale divide os conjuntos de dados em partições distintas que podem ser movidas entre os processos ou entre as máquinas no tempo de execução. É possível, por exemplo, iniciar com uma implementação de quatro servidores e, em seguida, expandir para uma implementação com dez servidores, conforme as demandas do cache crescem. Como é possível incluir mais máquinas físicas e unidades de processamento para escalabilidade vertical, é possível estender o recurso de escalação elástica do eXtreme Scale horizontalmente com o particionamento. Essa é outra grande diferença com os bancos de dados de memória (IMDBs) ao contrário do eXtreme Scale (que é uma grade de dados), já que os IMDBs podem ser escalados apenas verticalmente.

Com o WebSphere eXtreme Scale, também é possível usar um conjunto de APIs para obter acesso transacional a esses dados particionados e opcionalmente distribuídos. Em termos de desempenho, as opções feitas para interagir com o cache são tão significativas quanto as funções para gerenciar o cache para disponibilidade.

Nota: A escalabilidade não está disponível quando os contêineres se comunicam entre si. O protocolo de gerenciamento de disponibilidade ou de agrupamento principal é uma pulsação $O(N^2)$ e um algoritmo de manutenção de visualização, mas é reduzido mantendo em 20 o número de membros do grupo principal. Apenas a replicação ponto a ponto existe entre shards.

Particionamento

O particionamento é o mecanismo que o WebSphere eXtreme Scale usa para expandir um aplicativo. O particionamento é a separação do estado do aplicativo em partes em que cada parte contém alguns conjuntos dos dados de instância completos. O particionamento não é semelhante à divisão Redundant Array of Independent Disks (RAID), que fatia cada instância em todas as faixas. Cada partição hospeda os dados completos para entradas individuais. O particionamento é um meio mais efetivo para escalar, mas não é aplicável a todos os aplicativos. Os aplicativos que exigem garantias transacionais através de grandes conjuntos de dados não expandem e não podem ser particionados efetivamente, assim o eXtreme Scale não suporta atualmente consolidação de duas fase através de partições.

Importante: Selecione o número de partições com cuidado. O número de partições definido na política de implementação afeta diretamente o número de contêineres aos quais o aplicativo pode ser escalado. Cada partição é constituída por um fragmento primário e pelo número configurado de fragmentos de réplica. A fórmula `(Number_Partitions*(1 + Number_Replicas))` é o número de contêineres que pode ser utilizado para executar o scale out de um único aplicativo.

Clientes Distribuídos

O protocolo do cliente do WebSphere eXtreme Scale suporta quantidades muito grandes de clientes. A estratégia de particionamento oferece assistência assumindo que todos os cliente nem sempre estão interessados em todas as partições porque as conexões podem ser propagadas em vários contêineres. Os clientes se conectam diretamente às partições para que a latência seja limitada a uma conexão transferida.

Particionamento

Use o particionamento para armazenar grandes quantidades de dados na Java Virtual Machine (JVM). Para particionar dados, utilize um esquema especificado pelo aplicativo para dividir os dados.

Utilizando Partições

Uma grade pode ter muitas partições, ou milhares, se necessário. Ela pode aumentar para o produto até o número de partições vezes o número de shards por partição. Por exemplo, se você tiver 16 partições e cada uma delas tiver um shard primário e um shard réplica, ou dois shards, então, será possível ampliar para até 32 Java Virtual Machines. Neste caso, um shard é definido para cada JVM. É necessário escolher um número razoável de partições com base no número esperado de Java Virtual Machines que provavelmente serão utilizadas. Cada shard aumenta o uso de processador e de memória do sistema. O sistema foi desenvolvido para que possa expandir-se e manipular essa sobrecarga de acordo com o número de Java Virtual Machines de servidor disponíveis.

Os aplicativos não devem utilizar milhares de partições se o aplicativo executar numa grade de quatro Java Virtual Machines de contêiner. Ele deve ser configurado para ter um número razoável de shards para cada JVM de contêiner. Por exemplo, uma configuração inadequada seria 2.000 partições com dois shards em execução em quatro Java Virtual Machines de contêiner. Essa configuração resultaria em 4.000 shards distribuídos em quatro Java Virtual Machines de contêiner ou em 1.000 shards por JVM de contêiner.

Uma configuração melhor é 10 shards para cada JVM de contêiner esperada. Essa configuração ainda permite expandir 10 vezes a configuração inicial enquanto mantém um número adequado de shards por JVM de contêiner.

Considere este exemplo de expansão: atualmente, há seis computadores com duas Java Virtual Machines de contêiner por computador. Espera-se um crescimento para 20 computadores para os próximos três anos. Com 20 computadores, você tem 40 Java Virtual Machines de contêiner e escolhe 60 para ser pessimista. Você quer 4 shards por JVM de contêiner. Existem 60 contêineres potenciais, ou um total de 240 shards. Se tiver um primário e uma réplica por partição, terá 120 partições. Esse exemplo proporciona 240 shards dividido por 12 Java Virtual Machines de contêiner ou 20 shards por JVM de contêiner para uma implementação inicial, podendo efetuar scale out para 20 computadores posteriormente.

Entidades e Particionamento

As entidades do Entity Manager têm uma otimização que ajuda os clientes a trabalharem com entidades em um servidor. O esquema de entidade no servidor para conjunto de mapas pode especificar uma única entidade-raiz. O cliente deve acessar todas as entidades através da entidade-raiz. O gerenciador de entidades

pode, então, localizar entidades relacionadas a partir dessa raiz na mesma partição, sem exigir que os mapas relacionados tenham uma chave comum. A entidade-raiz estabelece a afinidade com uma única partição. Essa partição será utilizada por todas as buscas de entidades dentro da transação uma vez estabelecida a afinidade. A afinidade pode economizar memória, visto que os mapas relacionados não precisam de uma chave comum. A entidade-raiz deve ser especificada com uma anotação de entidade modificada, conforme mostrado no exemplo a seguir:

```
@Entity(schemaRoot=true)
```

Utilize a entidade para localizar a raiz do gráfico do objeto. Todas as entidades filha são assumidas como estando na mesma partição que a raiz. As entidades filha nesse gráfico estão acessíveis apenas a partir de um cliente da entidade-raiz. Entidades-raiz são sempre necessárias nos ambientes particionados ao usar um cliente do eXtreme Scale para se comunicar com o servidor. Apenas um tipo de entidade-raiz pode ser definido por cliente. As entidades-raiz não são necessárias ao utilizar ObjectGrids do estilo Extreme Transaction Processing (XTP), já que toda a comunicação com a partição é conseguida através de acesso direto e local, e não através do mecanismo de cliente e servidor.

Colocação e Partições

Há duas estratégias de colocação disponíveis no WebSphere eXtreme Scale, partição fixa e por contêiner. A estratégia de colocação afeta como as partições são colocadas e como os dados são distribuídos.

Colocação de Partições Fixas

É possível configurar a estratégia de colocação no arquivo XML de política de implementação. A estratégia de disposição padrão é uma colocação de partição fixa, ativada pela configuração `FIXED_PARTITION`. O número de shards primários que são colocados entre os contêineres disponíveis é igual ao número de partições configurado com o elemento numberOfPartitions. Se você tiver configurado réplicas, o número mínimo total de shards é definido pela seguinte fórmula:`((1 shard primário + mínimo de shards síncrono) * partições definidas)`. O número máximo total de shards colocados é definido pela seguinte fórmula: `((1 shard primário + mínimo de shards síncrono + máximo de shards assíncronos) * partições)`. A implementação do WebSphere eXtreme Scale propaga esses shards para os contêineres disponíveis. As chaves de cada mapa são submetidas a hash nas partições designadas com base no total de partições definido. Eles efetuam hash da chave para a mesma partição, mesmo se a partição mover devido ao failover ou alterações do servidor.

Por exemplo, se o valor de numberPartitions for 6 e o valor minSync for 1 para MapSet1, o total de shards para esse conjunto de mapas será 12 porque cada uma das 6 partições requer uma réplica síncrona. Se três contêineres forem iniciados, o WebSphere eXtreme Scale colocará quatro shards por contêiner para o MapSet1.

Colocação por Contêiner

A estratégia de colocação alternativa é a colocação por contêiner, que é ativada com a configuração `PER_CONTAINER` para placementStrategy no elemento de conjunto de mapas no arquivo XML de implementação. Com essa estratégia, o número de shards primários colocados em cada novo contêiner é igual ao número de partições *P* configuradas. O ambiente de implementaçãoWebSphere eXtreme Scale coloca*P* réplicas de cada partição para cada contêiner restante. A configuração numInitialContainers é ignorada quando estiver usando uma colocação por

contêiner. As partições ficam maiores conforme os contêineres crescem. As chaves para os mapas não são fixas em uma determinada partição nessa estratégia. O cliente é roteado para uma partição e usa um primário aleatório. Se um cliente desejar se conectar novamente a mesma sessão usada para localizar uma chave, um manipulador de sessão deverá ser usado.

Para obter mais informações, consulte o tópico usando uma SessionHandle para rotear no *Guia de Programação*.

Para servidores com failover ou interrompidos, o ambiente WebSphere eXtreme Scale mudará os shards primários na estratégia de colocação por contêiner se ainda contiverem dados. Se os shards estiverem vazios, eles serão descartados. Na estratégia por contêiner, os shards primários antigos não são mantidos porque os novos shards primários são colocados em cada contêiner.

Interface PartitionableKey

Quando a configuração do eXtreme Scale usa a estratégia de colocação de partição fixa, ela dependerá do hash da chave para uma partição inserir, obter, atualizar ou remover o valor. O método hashCode é chamado na chave e ele deverá ser bem definido se uma chave customizada for criada. Porém, outra opção é usar a interface PartitionableKey. Com a interface PartitionableKey, será possível usar um objeto diferente da chave para efetuar hash de uma partição.

É possível usar a interface PartitionableKey em situações em que houver vários mapas e os dados que você consolidar serão relatados e, assim, deverão ser colocados na mesma partição. O WebSphere eXtreme Scale não suporta two-phase commit, portanto, várias transações da mapa não deverão ser consolidadas se elas se estenderem por várias partições. Se o PartitionableKey efetuar hash para a mesma partição para as chaves em mapas diferentes no mesmo conjunto de mapas, eles poderão ser consolidados em conjunto.

Também é possível usar a interface PartitionableKey quando grupos de chaves tiverem que ser colocados na mesma partição, mas não necessariamente durante uma única transação. Se as chaves tiverem que ser submetidas a hash com base no local, departamento, tipo de domínio ou algum outro tipo de identificador, as chaves filha poderão receber um PartitionableKey pai.

Por exemplo, os funcionários devem efetuar hash para a mesma partição do seu departamento. Cada chave de funcionário terá um objeto PartitionableKey que pertence ao mapa do departamento. Então, tanto o funcionário quanto o departamento efetuarão hash para a mesma partição.

A interface PartitionableKey fornece um método, chamado ibmGetPartition. O objeto retornado desse método deve implementar o método hashCode. O resultado retornado do uso de um hashCode alternativo será usado para rotear as chaves para uma partição.

Transações de partição única e de partição de grade cruzada

A maior diferença entre as soluções do WebSphere eXtreme Scale e de armazenamento de dados tradicional, como bancos de dados relacionais ou bancos de dados em memória, é o uso do particionamento, que permite que o cache seja escalado de maneira linear. Os tipos importantes de transações a serem considerados são transações de partição única e de cada partição (grade cruzada).

Em geral, as interações com o cache podem ser categorizadas como transações de partição única ou transações de grade cruzada, conforme discutido a seguir.

Transações de Partição Única

As transações de partição única são o método preferido para interagir com os caches que são hospedados pelo WebSphere eXtreme Scale. Quando uma transação é limitada a uma única partição, por padrão, ela é limitada a uma única Java Virtual Machine e, portanto, a um único computador de servidor. Um servidor pode executar *M* número dessas transações por segundo e, se você tiver *N* computadores, poderá executar `M*N` transações por segundo. Se os negócios aumentarem e você precisar executar o dobro dessas transações por segundo, poderá dobrar *N* ao adquirir mais computadores. Em seguida, é possível atender as demandas de capacidade sem alterar o aplicativo, fazer upgrade de hardware ou até mesmo usar o aplicativo off-line.

Além de permitir que o cache seja escalado de maneira significativa, as transações de partição única também maximizam a disponibilidade do cache. Cada transação depende apenas de um computador. Qualquer um dos outros `(N-1)` computadores podem falhar sem afetar o sucesso ou o tempo de resposta da transação. Assim, se você estiver executando 100 computadores e um deles falhar, apenas 1% das transações em andamento no momento em que esse servidor falhou é recuperado. Depois que o servidor falhar, o WebSphere eXtreme Scale relocará as partições que são hospedadas pelo servidor com falha nos outros 99 computadores. Durante esse breve período, antes de concluir a operação, os outros 99 computadores ainda poderão concluir as transações. Apenas as transações que envolveriam as partições que estão sendo relocadas são bloqueadas. Depois que o processo de failover ser concluído, o cache poderá continuar executando, totalmente operacional, a 99% de sua capacidade de rendimento original. Depois que o servidor com falha ser substituído e retornado para a grade, o cache voltará para 100% da capacidade de rendimento.

Transações de Grade Cruzada

Em termos de desempenho, disponibilidade e escalabilidade, as transações de grade cruzada são o oposto das transações de partição única As transações de grade cruzada acessam cada partição e, portanto, cada computador na configuração. Cada computador na grade é instruído a procurar alguns dados e retornar o resultado. A transação não pode ser concluída até cada computador ter respondido e, dessa forma, o rendimento da grade inteira será limitado pelo computador mais lento. Incluir computadores não agiliza o computador mais lento e, assim, não melhora o rendimento do cache.

As transações de grade cruzada têm um efeito semelhante em disponibilidade. Estendendo o exemplo anterior, se você estiver executando 100 servidores e um deles falhar, então, 100% das transações que estão em andamento no momento em que esse servidor falhou será recuperado. Depois que o servidor falhar, o WebSphere eXtreme Scale relocará as partições que são hospedadas por esse servidor nos outros 99 computadores. Durante esse tempo, antes de o processo de failover ser concluído, a grade não poderá processar nenhuma dessas transações. Depois que o processo de failover ser concluído, o cache poderá continuar executando, porém com capacidade reduzida. Se cada computador na grade atender 10 partições, então 10 dos 99 computadores restantes receberão pelo menos uma partição extra como parte do processo de failover. Incluir uma partição extra aumenta a carga de trabalho desse computador em pelo menos 10%. Como o

rendimento da grade é limitado ao rendimento do computador mais lento em uma transação de grade cruzada, em média, o rendimento será reduzido em 10%.

As transações de partição única são preferidas para as transações de grade cruzada para serem escaladas com um cache de objeto distribuído e altamente disponível, como o WebSphere eXtreme Scale. Aumentar o desempenho desses tipos de sistemas requer o uso de técnicas que são diferentes das metodologias relacionais tradicionais, mas é possível transformar as transações de grade cruzada em transações de partição única escalável.

Boas práticas para criar modelos de dados escaláveis

As boas práticas para criar aplicativos escaláveis com produtos como o WebSphere eXtreme Scale incluem duas categorias: princípios básicos e dicas de implementação. Os princípios básicos são ideias principais que precisam ser capturadas no projeto dos próprios dados. Um aplicativo que não observa esses princípios podem não ser escalados tão bem, mesmo para as transações de linha principal. Por outro lado, as dicas de implementação são aplicadas em transações problemáticas em um aplicativo bem projetado que observa os princípios gerais para modelos de dados escaláveis.

Princípios Básicos

Algumas das maneiras importantes de otimizar a escalabilidade são conceitos ou princípios básicos que devem ser mantidos em mente.

Duplicar em vez de normalizar

O que mais deve-se ter em mente sobre os produtos como o WebSphere eXtreme Scale é que eles são designados para propagar dados entre um grande número de computadores. Se o objetivo é concluir a maioria ou todas as transações em uma única partição, o design do modelo de dados precisa garantir que todos os dados que a transação possa precisar estejam localizados na partição. Na maioria das vezes, a única maneira de fazer isso é duplicar os dados.

Por exemplo, considere um aplicativo, como um quadro de avisos. Duas transações muito importantes para um quadro de mensagens mostram todas as postagens de um determinado usuário e todas as postagens de um determinado tópico. Primeiro considere como essas transações trabalhariam com um modelo de dados normalizado que contenha um registro de usuário, um registro de tópico e um registro de postagem que contenha o texto real. Se as postagens forem particionadas com os registros do usuário, a exibição do tópico torna-se uma transação de grade cruzada e vice-versa. Os tópicos e os usuários não podem ser particionados juntos porque eles possuem um relacionamento muitos-para-muitos.

A melhor maneira de fazer com que esse quadro de avisos seja escalável é duplicar as postagens, armazenar uma cópia com o registro de tópico e uma cópia com o registro do usuário. Em seguida, a exibição das postagens de um usuário é uma transação de partição única, exibir as postagens em um tópico é uma transação de partição única e atualizar ou excluir uma postagem é uma transação de duas partições. Todas essas três transações serão escaladas de maneira linear já que o número de computadores na grade aumenta.

Escalabilidade Em Vez de Recursos

O maior obstáculo a ser superado ao considerar os modelos de dados não-normalizados é o impacto que esse modelos causam nos recursos. Manter duas, três ou mais cópias de alguns dados pode parecer que muitos recursos usados são práticos. Ao se deparar com esse cenário, lembre-se dos seguintes fatos: os recursos de hardware se tornam mais baratos a cada dia. Segundo e o mais importante, o WebSphere eXtreme Scale elimina a maioria dos custos implícitos associados à implementação de mais recursos.

Os recursos devem ser medidos em termos de custo em vez de computador, como megabytes e processadores. Os armazenamentos de dados que trabalham com dados relacionais normalizados geralmente precisam estar localizados no mesmo computador. Essa colocação necessária significa que um único computador corporativo maior precisa ser adquirido em vez de vários computadores menores. Com o hardware corporativo, um computador que executa um milhão de transações por segundo normalmente é bem mais barato que 10 computadores capazes de executar 100 mil transações por segundo cada um.

Incluir recursos também gera custos de negócios. Um negócio em crescimento normalmente pode ficar sem capacidade. Quando não houver capacidade, é necessário encerrar para mudar para um computador maior e mais rápido ou é necessário criar um segundo ambiente de produção para o qual você possa mudar. De uma das formas, custos adicionais serão acarretados na forma de negócios perdidos ou ao manter quase o dobro da capacidade necessária durante o período de transação.

Com o WebSphere eXtreme Scale, o aplicativo não precisa ser encerrado para incluir capacidade. Se seus projetos de negócios requererem 10% de capacidade a mais para o próximo ano, aumente 10% o número de computadores na grade. É possível aumentar essa porcentagem sem ocorrer tempo de inatividade do aplicativo e sem adquirir capacidade em excesso.

Evitar transformações de dados

Quando estiver usando o WebSphere eXtreme Scale, os dados deverão ser armazenados em um formato que possa ser consumado diretamente pela lógica de negócios. Dividir os dados em um formato mais primitivo gera custos. A transformação precisa ser feita quando os dados forem gravados e lidos. Com os bancos de dados relacionais, essa transformação é feita sem necessidade porque os dados são definitivamente persistidos no disco muito frequentemente, mas com o WebSphere eXtreme Scale, essas transformações não precisam ser executadas. Na maioria das vezes os dados são armazenados na memória e podem, portanto, serem armazenados no formato exato em que o aplicativo precisa.

Observar essa regra de amostra ajuda a desnormalizar os dados de acordo com o primeiro princípio. O tipo mais comum de transformação dos dados de negócios é as operações JOIN que são necessárias para tornar os dados normalizados em um conjunto de resultados que se ajusta às necessidades do aplicativo. Armazenar os dados no formato correto implicitamente evita a execução dessas operações JOIN e produz um modelo de dados não-normalizado.

Eliminar consultas ilimitadas

Independente de como os seus dados são estruturados, as consultas ilimitadas não são bem escaladas. Por exemplo, não tenha uma transação que solicite uma lista de todos os itens classificados por valor. Essa

transação pode funcionar inicialmente quando o número total de itens for 1.000, mas quando o número total de itens chegar a 10 milhões, a transação retornará todos os 10 milhões de itens. Se você executar essa transação, os dois resultados mais prováveis são o tempo limite da transação se esgotar ou o cliente receber um erro de falta de memória.

A melhor opção é alterar a lógica de negócios para que apenas os 10 ou 20 itens principais possam ser retornados. Essa mudança de lógica mantém o tamanho da transação gerenciável, independente de quantos itens estão no cache.

Definir esquema

A principal vantagem de normalizar os dados é que o sistema de banco de dados controla a consistência de dados em segundo plano. Quando os dados são desnormalizados para escalabilidade, esse gerenciamento de consistência de dados automático já não existe mais. É necessário implementar um modelo de dados que funcione na camada de aplicativos ou como um plug-in para a grade distribuída para garantir a consistência de dados.

Considere o exemplo do quadro de mensagens. Se uma transação remover uma postagem de um tópico, a postagem duplicada no registro do usuário precisará ser removida. Sem um modelo de dados, um desenvolvedor pode gravar o código do aplicativo para remover a postagem do tópico e se esquecer de remover a postagem do registro do usuário. Entretanto, se o desenvolvedor estiver usando um modelo de dados em vez de interagir diretamente com o cache, o método removePost no modelo de dados poderá extrair o ID do usuário da postagem, procurar pelo registro do usuário e remover a postagem duplicada em segundo plano.

Como alternativa, é possível implementar um listener que é executado na partição real que detecta a alteração no tópico e ajusta automaticamente o registro do usuário. Um listener pode ser benéfico porque o ajuste do registro do usuário pode ocorrer localmente caso a partição possua o registro do usuário ou, mesmo se o registro do usuário estiver em uma partição diferente, a transação ocorrerá entre os servidores em vez de ocorrer entre o cliente e o servidor. A conexão de rede entre os servidores provavelmente é mais rápida do que a conexão de rede entre o cliente e o servidor.

Evitar contenção

Evite cenários, como ter um contador global. A grade não será escalável se um registro único estiver sendo usado por um número de vezes desproporcional, comparado ao restante dos registros. O desempenho da grade será limitado pelo desempenho do computador que mantém o registro fornecido.

Nessas situações, tente dividir o registro para que ele seja gerenciado por partição. Por exemplo, considere uma transação que retorna o número total de entradas no cache distribuído. Em vez de fazer com que cada operação de inserção e remoção acesse um único registro que incrementa, faça com que o listener em cada partição controle as operações de inserção e remoção. Com o controle desse listener, a inserção e a remoção poderá se transformar nas operações de partição única.

A leitura do contador se transformará em uma operação de grade cruzada, mas para a maior parte, isso já ficou ineficiente como uma operação de grade cruzada porque o desempenho dependeu do desempenho do computador que hospeda o registro.

Dicas de Implementação

Também é possível considerar as seguintes dicas para obter a melhor escalabilidade.

Índices de Procura Reversa

Considere um modelo de dados não-normalizado adequadamente em que os registros são particionados com base no número do ID do cliente. Esse método de particionamento é a opção lógica porque quase cada operação de negócios executada com o registro do cliente usa o número de ID do cliente. Entretanto, uma transação importante que não usa o número do ID do cliente é a transação de login. É mais comum ter nomes de usuário ou endereços de e-mail para login em vez de números do ID de cliente.

A abordagem simples com o cenário de login é usar uma transação de grade cruzada para localizar o registro do cliente. Conforme explicado anteriormente, essa abordagem não é escalada.

A próxima opção pode ser uma partição no nome do usuário ou e-mail. Essa opção não é prática porque todas as operações baseadas no ID do cliente se transformam em transações de grade cruzada. Além disso, os clientes do seu site podem alterar o nome do usuário ou o endereço de e-mail. Produtos como o WebSphere eXtreme Scale precisam do valor usado para particionar os dados que permanecerem constantes.

A solução correta é usar um índice de consulta reversa. Com o WebSphere eXtreme Scale, um cache pode ser criado na mesma grade distribuída que o cache que mantém todos os registros do usuário. Esse cache é altamente escalável, particionado e escalável. Esse cache pode ser usado para mapear um nome de usuário ou endereço de e-mail para um ID de cliente. Esse cache transforma o login em uma operação de duas partições em vez de uma operação de grade cruzada. Esse cenário não é tão bom quanto uma transação de partição única, mas o rendimento ainda pode ser escalado linearmente conforme o número de computadores aumenta.

Computar no Momento da Gravação

Valores normalmente calculados, como médias ou totais, podem ser despendiosos para serem produzidos porque essas operações normalmente requerem leitura de um grande número de entradas. Como as leituras são mais comuns do que as gravações na maioria dos aplicativos, é eficiente calcular esses valores no momento da gravação e, em seguida, armazenar o resultado no cache. Essa prática torna as operações mais rápidas e mais escaláveis.

Campos Opcionais

Considere um registro de usuário que mantém um número de negócios, doméstico e de telefone.Um usuário pode ter todos, nenhum ou qualquer combinação desses números definida. Se os dados forem normalizados, uma tabela do usuário e um número de telefone existirão. Os números de telefone para um determinado usuário pode ser localizado usando uma operação JOIN entre as duas tabelas.

Desnormalizar esse registro não requer duplicação de dados, porque a maioria dos usuários não compartilha números de telefone. Em vez disso, slots vazios no registro do usuário devem ser permitidos. Em vez de ter uma tabela de número de telefone, inclua três atributos em cada registro do usuário, um para cada tipo de número de telefone. Essa inclusão de atributos elimina a operação JOIN e torna uma procura por número de telefone de um usuário uma operação de partição única.

Colocação de relacionamentos muitos-para-muitos

Considere um aplicativo que controla os produtos e as lojas nas quais os produtos são vendidos. Um único produto é vendido em muitas lojas e uma única loja vende muitos produtos. Suponha que esse aplicativo controla 50 grandes varejistas. Cada produto é vendido em um máximo de 50 lojas, com cada uma vendendo milhares de produtos.

Mantenha uma lista de lojas dentro da entidade do produto (organização A), em vez de manter uma lista de produtos dentro de cada entidade de loja (organização B). Observar algumas das transações que esse aplicativo teria que executar mostra o porquê a organização A é mais escalável.

Primeiro observe as atualizações. Com a organização A, remover um produto do inventário de uma loja bloqueia a entidade do produto. Se a grade manter 10.000 produtos, apenas 1/10.000 da grade precisará ser bloqueada para executar a atualização.Com a organização B, a grade contém apenas 50 lojas, então, 1/50 da grade deverá ser bloqueada para concluir a atualização. Portanto, embora os dois possam ser considerados operações de partição única, a organização A é escalada mais eficientemente.

Agora, considerando as leituras na organização A, observar as lojas nas quais um produto é vendido é uma transação de partição única que é escalada e é rápido porque a transação transmite apenas uma pequena quantia de dados. Com a organização A, essa transação se torna uma transação de grade cruzada porque cada entidade de loja deve ser acessada para ver se o produto é vendido nessa loja, que revela uma enorme vantagem de desempenho para a organização A.

Escalando com Dados Normalizados

Um uso legítimo de transações de grade cruzada é escalar o processamento de dados. Se uma grade tiver 5 computadores e uma transação de grade cruzada for despachada, que classifica cerca 100.000 registros em cada computador, essa transação classificará 500.000 registros. Se o computador mais lento na grade puder executar 10 dessas transações por segundo, a grade poderá classificar cerca de 5.000.000 de registros por segundo. Se os dados da grade dobrarem, cada computador deverá classificar cerca de 200.000 registros e cada transação classificará cerca de 1.000.000 de registros. Esse aumento de dados diminui o rendimento do computador mais lento para 5 transações por segundo, reduzindo, assim, o rendimento da grade para 5 transações por segundo. Além disso, a grade classifica cerca de 5.000.000 de registros por segundo.

Neste cenário, dobrar o número de computadores permite que cada computador volte para o carregamento anterior de classificação de 100.000 registros, permitindo que o computador mais lento processe 10 dessas transações por segundo. O rendimento da grade permanece o mesmo nos 10 pedidos por segundo, mas agora cada transação processa 1.000.000 de registros dobrando, assim, a capacidade da grade em termos de processamento de registros para 10.000.000 por segundo.

Com os aplicativos, como um mecanismo de procura que precisa ser escalado em termos de processamento de dados para acomodar o tamanho crescente da Internet e de rendimento para acomodar o crescimento de usuários, é necessário criar várias grades, com um round robin dos pedidos entre as grades. Se for preciso escalar o rendimento, inclua computadores e outra grade para atender aos pedidos. Se o processamento de dados precisar ser escalado, inclua mais computadores e mantenha o número de grades constante.

Capítulo 5. Disponibilidade

Com alta disponibilidade, o WebSphere eXtreme Scale fornece redundância e detecção de falhas.

WebSphere eXtreme Scale organiza automaticamente grades do Java Virtual Machines em uma árvore vagamente federada, com o serviço de catálogo na raiz e grupos principais que possuem contêineres nas folhas da árvore. Consulte o tópico "Arquitetura e Topologia" na página 9 para obter informações adicionais.

Cada grupo principal é automaticamente criado pelo serviço de catálogo em grupos de cerca de 20 servidores. Os membros do grupo principal fornecem monitoramento de funcionamento para outros membros do grupo. Também, cada grupo principal elege um membro para ser o líder para comunicar as informações do grupo para o serviço de catálogo. Limitar o tamanho do grupo principal permite o bom monitoramento de funcionamento e um ambiente altamente escalável.

Nota: Em um ambiente do WebSphere Application Server, no qual o tamanho do grupo principal pode ser alterado, oeXtreme Scale não suporta mais de 50 membros por grupo principal.

Falhas

Um processo pode falhar de várias maneiras. o processo poderia falhar porque algum limite de recurso foi atingido, tal como o tamanho máximo do heap ou alguma lógica de controle de processo terminou um processo. O sistema operacional poderia falhar, fazendo com que todos os processos em execução no mesmo sistema fossem perdidos. O hardware poderia falhar, embora com menos freqüência, como a NIC (placa da interface de rede), fazendo com que o sistema operacional fosse desconectado da rede. Muitos outros pontos de falha podem ocorrer, fazendo com que o processo se torne indisponível. Neste contexto, todas estas falhas podem ser categorizadas em um dos dois tipos: falha de processo e perda de conectividade.

Falha de Processo

O WebSphere eXtreme Scale reage a falhas do processo muito rapidamente. Quando um processo falha, o sistema operacional é responsável pela limpeza de qualquer recurso pendente que o processo estava utilizando. Essa limpeza inclui a alocação e conectividade da porta. Quando um processo falha, um sinal é enviado para as conexões que estavam sendo utilizadas por esse processo para fechar cada conexão. Com estes sinais, uma falha de processo pode ser detectada instantaneamente por qualquer outro processo que está conectado ao processo falho.

Perda de Conectividade

A perda de conectividade ocorre quando o sistema operacional se desconecta. Como resultado, o sistema operacional não pode enviar sinais a outros processos. Há diversos motivos pelos quais pode ocorrer uma perda de conectividade, mas eles podem ser divididos em duas categorias: falha de host e insulamento.

Falha de Host

Se a máquina for desconectada da tomada de energia, ela desligará instantaneamente.

Insulamento

Este cenário apresenta a condição de falha mais complicada para o software tratar corretamente porque o processo é presumido como indisponível, embora não esteja. Essencialmente, um servidor ou outro processo aparece para o sistema com falho enquanto, de fato, está executando adequadamente.

Falha de Contêiner do eXtreme Scale

Falhas contêiner geralmente são descobertas por contêineres peer através do mecanismo de grupo principal. Quando um contêiner ou conjunto de contêineres falha, o serviço de catálogo migra os fragmentos que foram hospedados nesse contêiner ou contêineres. O serviço de catálogo procura uma réplica síncrona primeiro, antes de migrar para uma réplica assíncrona. Depois da migração dos fragmentos primários para os novos contêineres de host, o serviço de catálogo procura novos contêineres de host para as réplicas que agora estão faltando.

Nota: Ilhamento de contêiner - O serviço de catálogo migra shards dos contêineres quando descobre-se que o contêiner está indisponível. Se esses contêineres se tornarem disponíveis, o serviço de catálogo considerará os contêineres elegíveis para disposição como no fluxo de inicialização normal.

Latência de detecção de failover de contêiner

As falhas podem ser categorizadas em falhas brandas e falhas graves. Falhas brandas normalmente são causadas quando um processo falha. Tais falhas são detectadas pelo sistema operacional, que pode recuperar recursos utilizados, tais como soquetes de rede, muito rapidamente. A detecção de falha típica para falhas brandas é de menos de um segundo. A falhas graves podem levar até 200 segundos até detectar utilizando o ajuste de pulsação padrão. Tais falhas incluem: falhas de máquina física, desconexões de cabo de rede ou falhas de sistema operacional. Deste modo, o eXtreme Scale deve contar com a pulsação para detectar falhas graves que podem ser configuradas. Consulte “Tipos de Detecção de Failover” na página 115 para obter detalhes sobre a diminuição do tempo necessário para detectar uma falha grave.

Falha de Serviço de Catálogo

Porque a grade de serviço do catálogo é uma grade do eXtreme Scale, ela também usa o mecanismo de agrupamento principal da mesma forma que o processo de falha de contêiner. A diferença primária é que a grade do serviço de catálogo usa um processo de eleição peer para definir o shard primário em vez do algoritmo do serviço de catálogo que é usado para os contêineres.

Note que o serviço de disposição e o serviço de agrupamento principal são serviços um-entre-N, mas o serviço e a administração de local são executados em qualquer lugar. O serviço de disposição e o serviço de agrupamento principal são singletons porque são responsáveis pela configuração do sistema. O serviço de local e a administração são serviços somente leitura e existe em qualquer lugar para fornecer escalabilidade.

O serviço de catálogo usa a replicação para tornar-se tolerante a falhas. Se um processo de serviço de catálogo falhar, o serviço deverá ser reiniciado para restaurar o sistema para o nível de disponibilidade desejado. Se todos os processos que estão hospedando o serviço de catálogo falharem, o eXtreme Scale possui uma perda de dados críticos. Essa falha resulta em um reinício necessário de todos os contêineres. Como o serviço de catálogo pode ser executado em muitos processos, essa falha é um evento improvável. Entretanto, se você estiver executando todos os processos em uma única caixa, dentro de um único chassi de blade, ou de um único comutador de rede, será mais provável que uma falha ocorra. Tente remover os modos de falha comuns das caixas que estão hospedando o serviço de catálogo para reduzir a possibilidade de falha.

Várias Falhas de Contêiner

Uma réplica nunca é colocada no mesmo processo que seu primário porque, se o processo for perdido, isso resultará na perda do primário e da réplica. A política de implantação define um atributo booleano de modo de desenvolvimento que o serviço de catálogo utiliza para determinar se uma réplica pode ser colocada na mesma máquina que o shard primário. Em um ambiente de implantação em uma única máquina, talvez você queira ter dois contêineres e replicar entre eles. Entretanto, em produção, utilizar uma máquina única é suficiente porque a perda de tal host resulta na perda de ambos os contêineres. Para alterar entre o modo de desenvolvimento em uma máquina única e um modo de produção com várias máquinas, desative o modo de desenvolvimento no arquivo de configuração da política de implementação.

Replicação para Disponibilidade

A replicação fornece tolerância a falhas e aumenta o desempenho para uma topologia eXtreme Scale distribuída.

A replicação é ativada pela associação de BackingMaps com um MapSet.

Um MapSet é uma coleta de mapas que estão categorizadas pela partição-chave. Esta partição-chave é derivada da chave do mapa individual pegando seu do módulo de hash a quantidade de partições. Deste modo, se um grupo de mapas no MapSet possui a partição-chave X, estes mapas serão armazenados em uma partição X correspondente na grade; se outro grupo possui a partição-chave Y, todos os mapas serão armazenados na partição Y, e assim por diante. Além disso, os dados nos mapas são replicados com base na política definida no MapSet, que é utilizado apenas para topologias distribuídas do eXtreme Scale (desnecessário para instâncias locais).

Consulte “Particionamento” na página 76 para obter detalhes adicionais.

MapSets recebem designações de qual número de partições receberão e uma política de replicação. A configuração de replicação do MapSet simplesmente identifica o número de shards de réplica síncronas e assíncronas que um MapSet deve ter além do shard primário. Por exemplo, se houver 1 réplica síncrona e 1 réplica assíncrona, todos os BackingMaps designados para o MapSet cada um terá um shard de réplica distribuído automaticamente no conjunto de contêineres disponíveis para o eXtreme Scale. A configuração de replicação também deve ativar clientes para ler dados a partir de servidores replicados de maneira síncrona. Isto pode propagar o carregamento para pedidos de leitura sobre servidores adicionais no eXtreme Scale. A replicação possui apenas um impacto no modelo de programação ao pré-carregar os BackingMaps.

Para obter detalhes sobre as diversas opções de configuração, consulte abaixo:

Pré-carregamento de Mapas

Mapas podem ser associados aos utilitários de carga. Um utilitário de carga é utilizado para buscar objetos quando eles não podem ser localizados no mapa (uma ocorrência de cache) e também para gravar alterações em um backend quando ocorre o commit de uma transação. Os Utilitários de Carga também podem ser utilizados para pré-carregar dados para um mapa. O método preloadMap da interface do Utilitário de Carga é chamado em cada mapa quando sua partição correspondente no MapSet torna-se um primário. O método preloadMap não é chamado nas réplicas. Ele tenta carregar todos os dados referenciados destinados a partir do backend no mapa utilizando a sessão fornecida. O mapa relevante é identificado pelo argumento BackingMap que é passado para o método preloadMap.

```
void preloadMap(Session session, BackingMap backingMap) throws LoaderException;
```

Pré-carregando no MapSet particionado

Os mapas podem ser particionados em N partições. Portanto, os mapas podem ser divididos em vários servidores, com cada entrada identifica por uma chave que é armazenada apenas em um destes servidores. Mapas muitos grandes podem ser mantidos em um eXtreme Scale porque o aplicativo não está mais limitado pelo tamanho de heap de uma JVM única para conter todas as entradas de um Mapa. Aplicativos que desejam pré-carregar com o método preloadMap da interface do Utilitário de Carga deve identificar o subconjunto de dados que ele pré-carrega. Sempre existe um número fixo de partições. É possível determinar este número utilizando o seguinte exemplo de código:

```
int numPartitions = backingMap.getPartitionManager().getNumOfPartitions();
int myPartition = backingMap.getPartitionId();
```

Este exemplo de código mostra como um aplicativo pode identificar o subconjunto de dados para pré-carregar a partir do banco de dados. Os aplicativos sempre devem utilizar estes métodos mesmo quando o mapa não é inicialmente particionado. Estes métodos permitem flexibilidade: Se o mapa for posteriormente particionado pelos administradores, então, o utilitário de carga continua a funcionar corretamente.

O aplicativo deve emitir consultas para recuperar o subconjunto *myPartition* a partir do backend. Se um banco de dados for utilizado, então, pode ser mais fácil ter uma coluna com o identificador de partições para um determinado registro, a menos que haja alguma consulta natural que permita que os dados na tabela sejam particionados facilmente.

Desempenho

A implementação de pré-carregamento copia dados do backend para o mapa, armazenando vários objetos no mapa em uma única transação. O número ideal de registros a serem armazenados por transação depende de vários fatores, incluindo complexidade e tamanho. Por exemplo, depois que a transação incluir blocos com mais de 100 entradas, o desempenho será reduzido conforme você aumenta o número de entradas. Para determinar o número ideal, comece com 100 entradas e, em seguida, aumente o número até que não sejam mais percebidos ganhos de desempenho. Transações maiores resultam em melhor desempenho de replicação.

Lembre-se, apenas o primário executa o código de pré-carregamento. Os dados pré-carregados são replicados do primário para quaisquer réplicas que estão on-line.

Pré-carregando MapSets

Se o aplicativo utiliza um MapSet com vários mapas, então, cada mapa possui seu próprio utilitário de carga. Cada utilitário de carga possui um método preload. Cada mapa é carregado serialmente pelo eXtreme Scale. Pode ser mais eficiente pré-carregar todos os mapas, projetando um único mapa como o mapa de pré-carregamento. Esse processo é uma convenção do aplicativo. Por exemplo, dois mapas, department e employee, podem utilizar o Utilitário de Carga de department para pré-carregar os mapas department e employee. Isto assegura que, transacionalmente, se um aplicativo desejar um departamento, os funcionários desse departamento estarão no cache. Quando o Utilitário de Carga do departamento pré-carregar um departamento do backend, ele também buscará os funcionários para esse departamento. O objeto department e seus objetos employee associados são, então, incluídos no mapa utilizando uma transação única.

Pré-carregamento Recuperável

Alguns clientes têm conjuntos de dados muito grandes que precisam ser armazenados em cache. O pré-carregamento de dados pode consumir muito tempo. Às vezes, o pré-carregamento deve ser concluído antes de o aplicativo ficar on-line. É possível beneficiar-se ao tornar o pré-carregamento recuperável. Supunha que haja um milhão de registros para pré-carregar. O primário está pré-carregando-os e falha no registro de número 800.000. Normalmente, a réplica escolhida para ser o novo primário limpa qualquer estado replicado e começa do início. O eXtreme Scale pode utilizar uma interface ReplicaPreloadController. O utilitário de carga para o aplicativo também precisa implementar a interface ReplicaPreloadController. Este exemplo inclui um método único no Utilitário de Carga: `Status checkPreloadStatus(Session session, BackingMap bmap);`. Este método é chamado pelo tempo de execução do eXtreme Scale antes do método preload da interface do Utilitário de Carga ser chamada normalmente. O eXtreme Scale testa o resultado deste método (Status) para determinar seu comportamento sempre que uma réplica é promovida para um primário.

Tabela 6. Valor de Status e Resposta

Valor do Status Retornado	Resposta do eXtreme Scale
Status.PRELOADED_ALREADY	O eXtreme Scale não chama o método preload porque este valor do status indica que o mapa foi totalmente pré-carregado.
Status.FULL_PRELOAD_NEEDED	O eXtreme Scale limpa o mapa e chama o método preload normalmente.
Status.PARTIAL_PRELOAD_NEEDED	O eXtreme Scale deixa o mapa no estado em que se encontra e chama o pré-carregamento. Essa estratégia permite que o Utilitário de Carga do aplicativo continue o pré-carregamento desse ponto em diante.

Claramente, enquanto um primário está pré-carregando o mapa, ele deve deixar algum estado em um mapa no MapSet que está sendo replicado de forma que a réplica determine qual status retornar. É possível utilizar um mapa extra denominado, por exemplo, RecoveryMap. Este RecoveryMap deve fazer parte do MapSet que está sendo pré-carregado para garantir que o mapa seja replicado consistentemente com os dados que estão sendo pré-carregados. A seguir, está uma implementação sugerida.

À medida que ocorre o commit de cada bloco de registros, o processo também atualiza um contador ou valor no RecoveryMap como parte de tal transação. Os dados pré-carregados e os dados de RecoveryMap são replicados atomicamente para as réplicas. Quando a réplica é promovida para o primário, ela pode verificar o RecoveryMap para saber o que aconteceu.

O RecoveryMap pode conter uma única entrada com a chave de estado. Se não existir nenhum objeto para esta chave, será necessário um pré-carregamento completo (checkPreloadStatus retorna FULL_PRELOAD_NEEDED). Se existir um objeto para esta chave de estado e o valor for COMPLETE, o pré-carregamento será concluído e o método checkPreloadStatus retornará PRELOADED_ALREADY. Caso contrário, o objeto de valor indicará de onde o pré-carregamento deve ser reiniciado e o método checkPreloadStatus retornará PARTIAL_PRELOAD_NEEDED. O utilitário de carga pode armazenar o ponto de recuperação em uma variável de instância para o utilitário de carga para que, quando o pré-carregamento for chamado, ele saiba o ponto de partida. O RecoveryMap também pode conter uma entrada por mapa se cada mapa for pré-carregado de maneira independente.

Manipulando a recuperação no modo de replicação síncrono com um Utilitário de Carga

O tempo de execução doeXtreme Scale é projetado para não perder dados com commit quando o primário falha. A seção a seguir mostra os algoritmos utilizados. Estes algoritmos se aplicam apenas quando um grupo de replicação utiliza a replicação síncrona. Um utilitário de carga é opcional.

O tempo de execução do eXtreme Scale pode ser configurado para replicar todas as alterações a partir de um primário para as réplicas de maneira síncrona. Quando uma réplica síncrona é posicionada ela recebe uma cópia dos dados existentes no shard primário. Durante este período, o primário continua a receber transações e copiá-las assincronamente para a réplica. A réplica não é considerada como estando on-line neste período.

Depois de a réplica capturar o primário, ela entre no modo peer e começa a replicação síncrona. Cada transação consolidada no primário é enviada às réplicas síncronas e o primário aguarda por uma resposta de cada réplica. Uma sequência de consolidação síncrona com um utilitário de carga no primário se parece com o conjunto e etapas a seguir:

Tabela 7. Sequência de Commit no Primário

Etapa com o Utilitário de Carga	Etapa sem o Utilitário de Carga
Obter bloqueios para entradas	igual
Limpar alterações no utilitário de carga	no-op
Salvar alterações no cache	igual
Enviar alterações para réplicas e esperar confirmação	igual
Confirmar para o utilitário de carga por meio do Plug-in TransactionCallback	commit do plug-in chamado, mas não faz nada
Liberar bloqueios para entradas	igual

Observe que as alterações são enviadas para a réplica antes de serem confirmadas para o utilitário de carga. Para determinar quando ocorre o commit das alterações na réplica, revise esta sequência: No momento da inicialização, inicialize as listas tx no primário, conforme abaixo.

```
CommitedTx = {}, RolledBackTx = {}
```

Durante o processamento de confirmação síncrona, utilize a seguinte seqüência:

Tabela 8. Processamento de Commit Síncrono

Etapa com o Utilitário de Carga	**Etapa sem o Utilitário de Carga**
Obter bloqueios para entradas	igual
Limpar alterações no utilitário de carga	no-op
Salvar alterações no cache	igual
Enviar alterações com uma transação confirmada, efetuar rollback da transação para a réplica e esperar confirmação	igual
Limpar lista de transações confirmadas e de transações que receberam rollback	igual
Confirmar o utilitário de carga por meio do plug-in TransactionCallBack	A confirmação do plug-in TransactionCallBack ainda é chamada mas, geralmente, não faz nada
Se a confirmação for bem-sucedida, inclua a transação nas transações confirmadas; caso contrário, inclua nas transações que receberam rollback	no-op
Liberar bloqueios para entradas	igual

Para processamento de réplica, utilize a seguinte seqüência:

1. Receber alterações
2. Confirmar todas as transações recebidas na lista de transações confirmadas
3. Efetuar rollback de todas as transações recebidas na lista de transações que receberam rollback
4. Iniciar uma transação ou sessão
5. Aplicar alterações à transação ou sessão
6. Salvar a transação ou sessão na lista pendente
7. Retornar resposta

Observe que, na réplica, não existem interações do Utilitário de Carga enquanto ele está no modo de réplica. O primário deve enviar todas as alterações por meio do Utilitário de Carga. A réplica não faz nenhuma alteração. Um efeito secundário deste algoritmo é que a réplica sempre tem as transações, mas elas não são confirmadas, até que a próxima transação primária envie o status de confirmação destas transações. Elas são então confirmadas ou recebem rollback na réplica. Mas, até então, as transações não são confirmadas. Podemos incluir um cronômetro no primário que enviará o resultado da transação após um breve período de tempo (alguns segundos). Esse cronômetro limita, mas não elimina, nenhuma deterioração desse espaço de tempo Este staleness é um problemas apenas ao utilizar o modo de leitura de réplica. Do contrário, a deterioração não tem impacto sobre o aplicativo.

Quando o primário falha, é provável que poucos commits ou rollback tenham ocorrido nas transações no primário, mas a mensagem nunca fez isto para a réplica com estas saídas. Quando uma réplica for promovida para o novo primário, uma de suas primeiras ações será manipular esta condição. Cada transação pendente é processada novamente junto ao novo conjunto de mapas do primário. Se houver um Utilitário de Carga, então, cada transação é fornecida para o Utilitário de Carga. Estas transações são aplicadas na ordem FIFO (primeiro a entrar, primeiro a sair) estrita. Se uma transação falhar, ela será ignorada. Se três transações estiverem pendentes, A, B e C, então A poderá confirmar, B poderá efetuar rollback e C também poderá confirmar. Nenhuma transação tem impacto sobre as outras. Suponha que elas sejam independentes.

Um utilitário de carga talvez queira utilizar uma lógica um pouco diferente quando no modo recuperação de failover versus modo normal. O utilitário de carga pode saber facilmente quando está em modo de recuperação de failover, implementando a interface ReplicaPreloadController. O método checkPreloadStatus é chamado apenas quando a recuperação de failover é concluída. Portanto, se o método apply da interface do Utilitário de Carga for chamado antes do checkPreloadStatus, ele será uma transação de recuperação. Depois que o método checkPreloadStatus for chamado, a recuperação de failover estará concluída.

Equilíbrio de Carga entre Réplicas

O eXtreme Scale, a menos que configurado de outra forma, envia todos os pedidos de leitura e gravação para o servidor primário para um determinado grupo de replicação. O primário deve atender todos os pedidos de clientes. Você pode permitir que pedidos de leitura sejam enviados para réplicas do primário. Enviar pedidos de leitura para as réplicas permite que o carregamento dos pedidos de leitura seja compartilhado por várias Java Virtual Machines (JVM). No entanto, utilizar réplicas para pedidos de leitura pode resultar em respostas inconsistentes.

Geralmente, o balanceamento de carga através de réplicas é utilizado apenas quando clientes estão armazenando dados em cache que estão sendo alterados sempre ou quando os clientes estão utilizando bloqueio pessimista.

Se os dados estiverem sendo alterados continuamente e sendo invalidados no cliente, caches locais e o primário deverão ver uma taxa relativamente alta de pedidos get de clientes como resultado. Da mesma forma, no modo de bloqueio pessimista, não existe cache local, portanto, todos os pedidos são enviados para o primário.

Se os dados forem relativamente estáticos ou o modo pessimista não for utilizado, o envio de pedidos de leitura de réplicas não terá um grande impacto no desempenho. A freqüência de pedidos get dos clientes com caches ativos não será alta.

Quando um cliente é iniciado pela primeira vez, seu near cache está vazio. Os pedidos de cache para esse cache vazio são redirecionados para o primário. O cache cliente obtém dados com o tempo, causando a eliminação deste carregamento de pedido. Se houver um grande número de clientes iniciados simultaneamente, este carregamento poderá ser significativo e a leitura de réplicas poderá ser uma opção de desempenho apropriada.

Replicação do Lado do Cliente

Com o eXtreme Scale, é possível replicar um mapa de servidor em um ou mais clientes utilizando a replicação assíncrona. Um cliente pode pedir uma cópia local somente leitura de um mapa de lado do servidor utilizando o método ClientReplicableMap.enableClientReplication.

```
void enableClientReplication(Mode mode, int[] partitions, ReplicationMapListener listener) throws ObjectGridException;
```

O primeiro parâmetro é o modo de replicação. Esse modo pode ser uma replicação contínua ou uma replicação de captura instantânea. O segundo parâmetro é uma matriz de IDs de partição que representa as partições a partir das quais replicar os dados. Se o valor for nulo ou uma matriz vazia, os dados são replicados a partir de todas as partições. O último parâmetro é um listener para receber eventos de replicação de cliente. Consulte ClientReplicableMap e ReplicationMapListener na documentação da API para obter detalhes.

Depois de ativada a replicação, então o servidor começa a replicar o mapa para o cliente. O cliente eventualmente está apenas algumas transações atrás do servidor em questão de tempo.

Arquitetura de Replicação

Com o eXtreme Scale, um banco de dados de memória ou shard pode ser replicado a partir de uma Java Virtual Machine (JVM) para outra. Uma parte representa uma partição que é colocada em um contêiner. Várias partes que representam diferentes partições podem existir em um único contêiner. Para executar a replicação, um shard principal e shards de réplica existem para cada partição. Os shards de réplica são síncronos ou assíncronos. Os tipos e a disposição dos shards de réplica são determinados pelo eXtreme Scale utilizando uma política de implementação, que especifica o número mínimo e máximo de shards síncronos e assíncronos.

Tipos de Shard

A replicação usa três tipos de shards:

- Principal
- Réplica síncrona
- Réplica assíncrona

A parte primária recebe todas as opções insert, update e remove. A parte primária inclui e remove réplicas, replica dados para as réplicas e gerencia confirmações e recuperações de transações.

As réplicas síncronas mantêm o mesmo estado que o primário. Quando um primário replica dados para uma réplica síncrona, a transação não é confirmada, até que seja confirmada na réplica síncrona.

As réplicas assíncronas podem ou não ter o mesmo estado que o primário. Quando um primário replica dados para uma réplica assíncrona, o primário não aguarda a confirmação da réplica assíncrona. Uma réplica assíncrona continua a manter a ordem das transações que são enviadas a partir do primário.

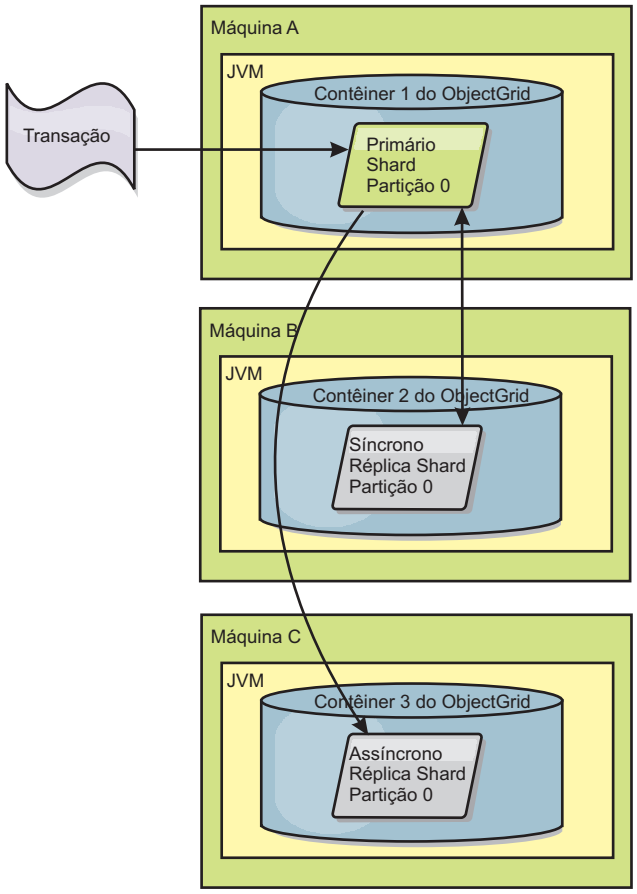


Figura 33. Caminho de Comunicação Entre um Shard Primário e Shards de Réplica

O Custo da Memória da Replicação

Para tornar uma réplica on-line, um mapa de ponto de verificação deve ser criado para copiar dados existentes do primário. No entanto, o custo de memória principal no primário é a memória para cada entrada modificada após ocorrer o ponto de verificação. Se os dados forem muito alterados após o ponto de verificação, essa operação custará uma quantidade de memória que é proporcional ao número de entradas modificadas. Depois que o ponto de verificação for copiado para a réplica, as alterações são fundidas no ponto de verificação. As entradas alteradas não são liberadas, até que tenham sido enviadas para todas as réplicas necessárias.

Mínimo de Shards de Réplica Síncrona

Quando um primário prepara para consolidar dados, ele verifica quantas partes de réplicas síncronas foram aprovadas para confirmar a transação. Se a transação processar normalmente na réplica, ela aprovará a confirmação. Se aconteceu algo errado na réplica síncrona, ela não aprovará a confirmação. Antes de um principal confirmar, o número de shards de réplica síncronas que são aprovadas para confirmação devem corresponder à configuração minSyncReplica da política de implementação. Quando o número de partes de réplicas síncronas sendo aprovado para confirmação for muito baixo, o primário não confirma a transação e resulta em um erro. Essa ação assegura que o número requerido de réplicas síncronas está disponível com os dados corretos. As réplicas síncronas que encontraram erros são

novamente registradas para corrigir seu estado. Para obter informações adicionais sobre novo registro, consulte Recuperação de shard de réplica.

O primário lança um erro ReplicationVotedToRollbackTransactionException, se poucas réplicas síncronas foram aprovadas para confirmação.

Replicação e Utilitários de Carga

Normalmente, uma parte primária grava as alterações de forma síncrona por meio do Utilitário de Carga em um banco de dados. O Utilitário de Carga e o banco de dados sempre estão em sync. Quando o primário falha sobre uma parte da réplica, o banco de dados e o Utilitário de Carga podem não estar em synch. Por exemplo:

- O primário pode enviar a transação para a réplica e, em seguida, falhar antes de confirmar com o banco de dados.
- O primário pode confirmar com o banco de dados e, em seguida, falhar antes de enviar para a réplica.

A abordagem é conduzida para ou a réplica está sendo uma transação em frente de ou atrás do banco de dados. Essa situação não é aceitável. OeXtreme Scale utiliza um protocolo especial e um contrato com a implementação do Utilitário de Carga para resolver este problema sem two phase commit. O protocolo é mostrado a seguir:

Lado primário

- Envie a transação juntamente com os resultados da transação anterior.
- Grave no banco de dados e tente confirmar a transação.
- Se o banco de dados for confirmado, então confirme no eXtreme Scale. Se ele não for confirmado, recupere a transação.
- Registre a saída.

Lado da réplica

- Receba uma transação e armazene-a.
- Para todos os resultados, envie com a transação, confirme todas as transações armazenadas em buffer e descarte todas as transações recuperadas.

Lado da réplica em failover

- Para todas as transações armazenadas em buffer, forneça as transações ao Utilitário de Carga e o Utilitário de Carga tenta confirmá-las.
- O Utilitário de Carga precisa ser gravado para tornar cada transação idempotente.
- Se a transação já estiver no banco de dados, o Utilitário de Carga não realizará nenhuma operação.
- Se a transação não estiver no banco de dados, o Utilitário de Carga aplica a transação.
- Após todas as transações serem processadas, o novo primário pode começar a receber os pedidos.

Esse protocolo assegura que o banco de dados está no mesmo nível que o novo estado primário.

Alocação de Shard: Principal e de Réplica

O serviço de catálogos é responsável pela disposição dos shards. Cada ObjectGrid tem um número de partições. Cada partição tem um shard principal e um conjunto opcional de shards de réplica. O serviço de catálogos não posiciona shards de réplica e primários da mesma partição no mesmo contêiner. Ele também não coloca shards de réplica e principais em contêineres que possuem o mesmo endereço IP a menos que a configuração esteja no modo de desenvolvimento. O serviço de catálogos equilibra os tipos de shards de modo que sejam igualmente distribuídos pelos contêineres disponíveis.

Se um contêiner inicia, então o eXtreme Scale recupera shards de contêineres sobrecarregados para o novo contêiner vazio. Com esse comportamento, o eXtreme Scale estabelece e mantém sua própria elasticidade essencial. A elasticidade é o manifesto da capacidade poderosa de efetuar escalação literalmente, seja para efetuar scale in e scale out.

Efetuando Scaling Out

Efetuar scale out significa que quandoJava Virtual Machines extras, ou contêineres, são incluídos em uma grade do eXtreme Scale, o eXtreme Scale tenta mover shards existentes, primários ou réplicas, do antigo conjunto de JVMs para o novo conjunto. Este movimento permite que a grade se expanda para aproveitar a vantagem do processador, da rede e da memória das JVMs recentemente incluídos. O movimento também equilibra a grade e tenta garantir que cada JVM na grade hospede a mesma quantidade de dados. À medida que a grade se expande, cada servidor hospeda um subconjunto menor da grade total. O eXtreme Scale assume que os dados são distribuídos igualmente entre as partições. Esta expansão possibilita o scaling out.

Efetuando Scaling In

Efetuar scale in significa que se um JVM falhar, o eXtreme Scale tenta reparar o dano. Se a JVM falha tinha uma réplica, então, o eXtreme Scale substitui a réplica perdida criando uma nova réplica em uma JVM sobrevivente. Se a JVM falha tinha um principal, então o eXtreme Scale localiza a melhor réplica nos sobreviventes e promove a réplica para ser o novo principal. O eXtreme Scale, então, substitui a réplica promovida por uma nova réplica que é criada no servidores restantes. Para manter a escalabilidade, o eXtreme Scale preserva a contagem de réplica para partições à medida que os servidores falham.

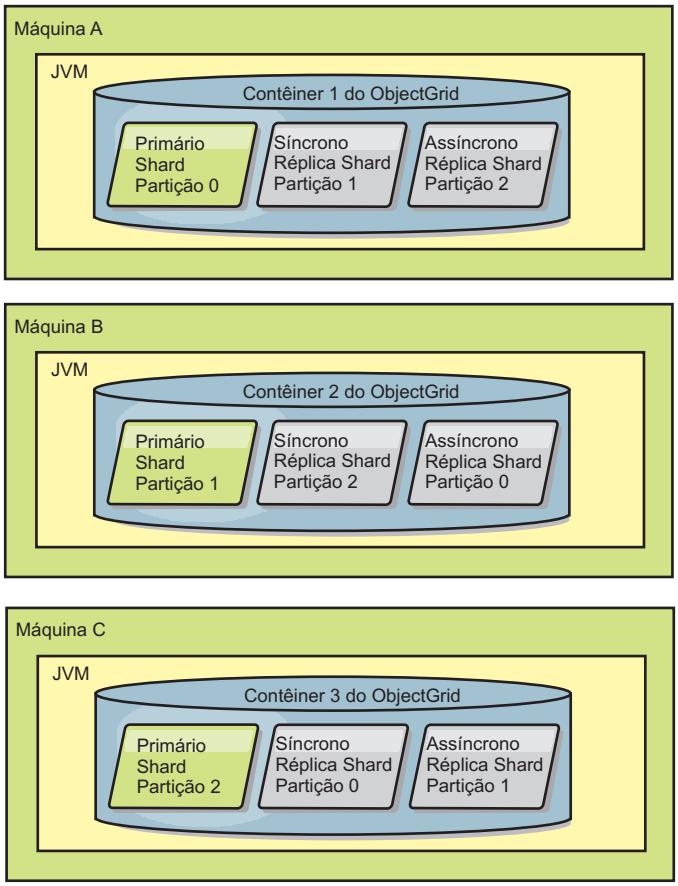


Figura 34. Posicionamento de um Conjunto de Mapas ObjectGrid com um Política de Implementação de 3 Partições com um Valor minSyncReplicas de 1, um Valor maxSyncReplicas de 1, e um Valor maxAsyncReplicas de 1

Eventos de ciclo de Vida, Recuperação e Falha

As partes passam por diferentes estados e eventos para suportarem a replicação. O ciclo de vida de um shard inclui ficar on-line, tempo de execução, encerramento, failover e manipulação de erro. Os shards podem ser promovidos de um shard réplica para um shard primário para manipular alterações do estado do servidor.

Eventos de Ciclo de Vida

Quando as partes primárias e de réplicas são colocadas e iniciadas, elas passam por uma série de eventos para torná-las on-line e deixá-las no modo de escuta.

shard primário

O serviço de catálogo coloca uma parte primária para uma partição. O serviço de catálogo também realiza o trabalho de equilíbrio de locais de partes primárias e o início de failover para partes primárias.

Quando uma parte torna-se uma parte primária, ela recebe uma lista de réplicas do serviço de catálogo. A nova parte primária cria um grupo de réplicas e registra todas as réplicas.

Quando o principal está pronto, uma abertura para a mensagem de negócios é exibida no arquivo `SystemOut.log` para o contêiner no qual está em execução. A

mensagem aberta, ou a mensagem CWOBJ1511I, lista o nome do mapa, nome do conjunto do mapa e número da partição do shard primário que iniciou.

```
CWOBJ1511I: mapName:mapSetName:partitionNumber (primário) é aberto para negócios.
```

Consulte “Alocação de Shard: Principal e de Réplica” na página 98 para obter informações adicionais sobre como o serviço de catálogo coloca shards.

Shard de Réplica

As partes da réplica são principalmente controladas pela parte primária, a não ser que a réplica detecte um problema. Durante um ciclo de vida normal, o shard primário posiciona, registra e cancela o registro de um shard de réplica.

Quando a parte primária inicializa uma parte da réplica, uma mensagem exibe o log que descreve onde a réplica é executada para indicar que está disponível a parte da réplica. A mensagem aberta, ou a mensagem CWOBJ1511I, lista o nome do mapa, nome do conjunto do mapa e número da partição do shard de réplica. Essa mensagem é mostrada a seguir:

```
CWOBJ1511I: mapName:mapSetName:partitionNumber (réplica síncrona) é aberta para negócios.
```

ou

```
CWOBJ1511I: mapName:mapSetName:partitionNumber (réplica assíncrona) é aberta para negócios.
```

Quando a parte da réplica é primeiro iniciada, ela ainda não está no modo de mesmo nível. Quando uma parte da réplica está no modo de mesmo nível, ela recebe dados da primária conforme eles chegam na primária. Antes de digitar o modo de mesmo nível, a parte da réplica precisa de uma cópia de todos os dados existentes na parte primária.

Para obter uma cópia dos dados na parte primária, as partes da réplica síncrona e assíncrona desempenham a mesma seqüência de inicialização. Durante o registro da parte da réplica, a parte primária cria um ponto de verificação dos dados atuais. Os dados na parte primária estão em um estado de cópia na gravação. Os dados atuais que vão para a parte da réplica nunca são modificados, mas uma cópia na gravação é desempenhada, sempre que uma nova transação atualiza registros na primária. A parte da réplica consegue continuar o processamento sem alterar os dados que vão para a parte da réplica. A parte primária mantém uma lista das alterações que foram feitas desde o ponto de verificação.

Os dados do ponto de verificação são enviados para a réplica. Quando o ponto de verificação chegar na réplica, será liberada a memória para o ponto de verificação. São fundidas as alterações desde que o ponto de verificação foi criado. A lista de alterações também é enviada para a parte da réplica. Como as alterações são enviadas, é liberada a memória para a alteração.

Após a réplica concluir a fase do ponto de verificação, ela comuta no modo de mesmo nível e começa a receber dados assim como a primária recebe. Nesse momento, a parte de réplica começa a se comportar como uma parte da réplica síncrona ou assíncrona

Quando um shard de réplica atinge o modo peer, ele imprime uma mensagem para o arquivo `SystemOut.log` para a réplica.

```
CWOBJ1526I: Replica objectGridName:mapsetName:partitionNumber:mapName entering peer mode after X seconds.
```

O tempo faz referência a quantidade de tempo que levou a parte da réplica a obter todos os seus dados iniciais da parte primária, incluindo os dados do ponto de

verificação e quaisquer alterações adicionais feitas durante a cópia do ponto de verificação. O tempo pode ser exibido como zero ou mínimo, se a parte primária não tiver nenhum dado existente para replicar.

Shard de réplica síncrona

Se a nova réplica for uma parte da réplica síncrona, a parte primária iniciará agora um pedido sempre que uma transação é confirmada na primária. A parte primária aguarda até que a parte da réplica responda que obteve os dados. Os dados da parte da réplica síncrona permanecem no mesmo nível que os dados da parte primária.

Shard de réplica assíncrona

Se a nova réplica for uma parte da réplica assíncrona, a parte primária enviará os dados para a réplica, mas não aguardará uma resposta. A réplica assíncrona solicita e aplica os dados enviados da primária. Os dados da parte da réplica assíncrona não garantem a permanência no mesmo nível que os dados da parte primária.

Peer mode for all replica shards

Quando você estiver no modo de mesmo nível, após todas as réplicas receberem uma alteração na transação, a memória para a transação será liberada na parte primária. As partes da réplica apenas recebem dados da parte primária durante as transações que alteram dados tais como inserções, atualizações e remoções. Elas replicam partes que não entram em contato para leitura de dados na parte primária.

Eventos de Recuperação

A replicação é projetada para recuperar a partir da falha e de eventos de erro. Se uma parte primária falhar, outra réplica será transferida. Se os erros estiverem nas partes da réplica, ela tentará a recuperação. O serviço de catálogo controla a disposição e as transações de novos shards primários ou novos shards de réplica.

O shard de réplica torna-se um shard principal

Uma parte da réplica torna-se uma parte primária por dois motivos. A parte primária parou ou falhou, ou uma decisão de equilíbrio foi tomada para mover a parte primária anterior para um novo local.

O serviço de catálogo seleciona uma nova parte primária a partir das partes de réplicas síncronas existentes. A nova parte primária registra todas as réplicas existentes e aceita as transações como a nova parte primária. Se as partes da réplica existentes tiverem o nível correto de dados, os dados atuais serão preservados conforme as partes de réplicas são registradas com a nova parte primária. Se uma parte da réplica assíncrona estiver atrasada, ela receberá uma cópia atual dos dados e registro.

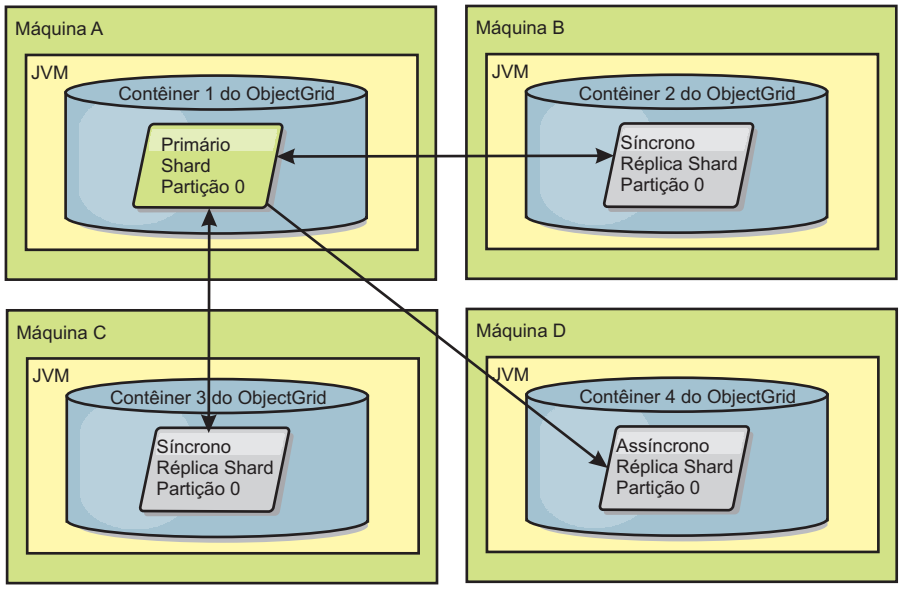


Figura 35. Posicionamento de exemplo de um conjunto de mapas do ObjectGrid para a partição partition0. A política de implementação tem um valor de minSyncReplicas de 1, um valor de maxSyncReplicas de 2, e um valor de maxAsyncReplicas de 1.

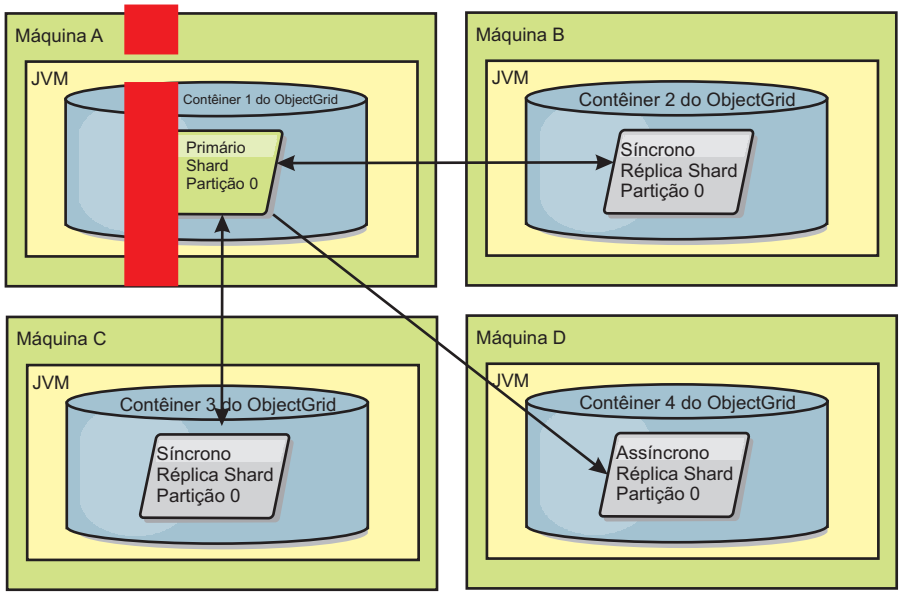


Figura 36. O contêiner para o shard primário falha

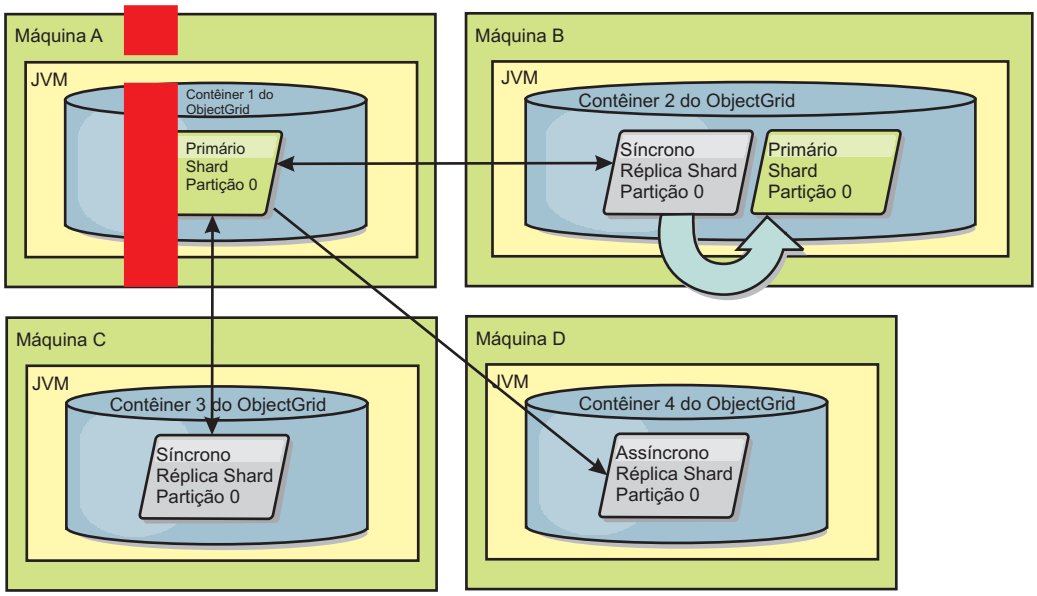


Figura 37. O Shard de Réplica Síncrona no Contêiner 2 do ObjectGrid se Torna o Shard Primário

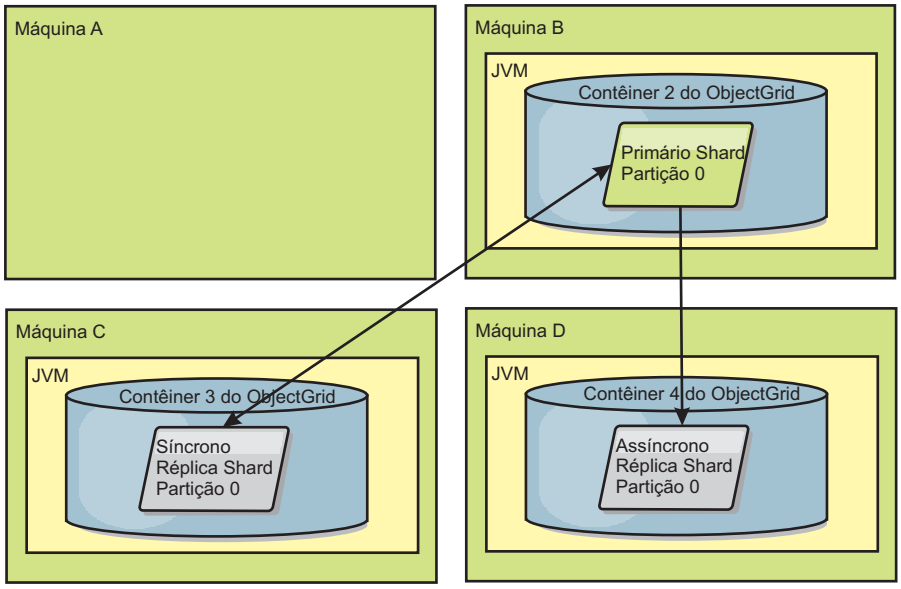


Figura 38. A máquina B contém o shard primário. Dependendo de como o modo de reparo automático é configurado e da disponibilidade dos contêineres, um novo shard de réplica síncrona pode ou não ser posicionado em uma máquina.

Recuperação de shard de réplica

Uma parte da réplica é controlada pela parte primária. No entanto, se uma réplica detectar um problema, ela poderá acionar um evento novamente registrado para corrigir o estado dos dados. A réplica remove os dados atuais e obtém uma cópia atual da primária.

Quando uma parte da réplica inicia um evento novamente registrado, a réplica copia uma mensagem de log.

```
CWOBJ1524I: Listener de réplica
objectGridName:mapSetName:partition deve registrar novamente com o shard primário.
Razão: Exceção listada
```

Se uma transação provocar um erro em uma parte da réplica durante o processamento, então a parte da réplica estará em um estado desconhecido. A transação foi processada com êxito na parte primária, mas aconteceu algo errado na réplica. Para corrigir essa situação, a réplica inicia um evento novamente registrado. Com uma nova cópia de dados da primária, a parte da réplica pode continuar. Se ocorrer novamente o mesmo problema, a parte da réplica não será novamente registrada de forma contínua. Consulte “Eventos de Falha” para obter detalhes adicionais.

Eventos de Falha

Uma réplica pode parar a replicação de dados se ela encontrar situações de erro nas quais ela não pode ser recuperada.

Muitas tentativas de registro

Se uma réplica acionar várias vezes um novo registro sem confirmar dados com êxito, ela irá parar. A parada evita que uma réplica entre em um loop de novo registro sem fim. Por padrão, uma parte da réplica tenta registrar novamente três vezes seguidas antes de parar.

Se uma parte da réplica registrar novamente muitas vezes, ela copiará a mensagem a seguir no log.

```
CWOBJ1537E: objectGridName:mapSetName:partition excedeu o número máximo
de vezes para registrar novamente (timesAllowed) sem transações com êxito.
```

Se a réplica não conseguir ser recuperada pelo novo registro, poderá existir um problema predominante com as transações relativas à parte da réplica. Um possível problema poderia ser recursos ausentes no caminho de classe, se ocorrer um erro ao aumentar as chaves ou valores da transação.

Falha ao entrar no modo peer

Se uma réplica tentar digitar o modo de mesmo nível e tiver um erro no processamento de dados existentes em massa a partir do primário (os dados do ponto de verificação), a réplica será encerrada. O encerramento evita que uma réplica seja iniciada com dados iniciais incorretos. Como ela recebe os mesmos dados do shard primário, caso seja novamente registrada, a réplica não tentará novamente.

Se uma parte da réplica falhar ao entrar no modo de mesmo nível, ela copiará a mensagem a seguir no log:

```
CWOBJ1527W A réplica objectGridName:mapSetName:partition:mapName ao entrar no modo de mesmo nível depois de numSeconds segundos.
```

Uma mensagem adicional é exibida no log que explica o motivo pelo qual a réplica falhou ao entrar no modo de mesmo nível.

Recuperação após falha ao registrar novamente ou do modo de mesmo nível

Se uma réplica falhar ao registrar novamente ou ao entrar no modo de mesmo nível, a réplica estará em um estado inativo até um novo evento de posicionamento ocorrer. Um novo evento de posicionamento pode ser um novo servidor que está iniciando ou parando. Também é possível iniciar um evento de

posicionamento usando o método triggerPlacement no Mbean PlacementServiceMBean.

Lendo a partir de Réplicas

É possível configurar conjuntos de mapas a partir dos quais um cliente pode ler a partir de uma réplica em vez de limitar-se a apenas aos shards primários.

Isso normalmente pode ser vantajoso para permitir que as réplicas atendam mais que apenas shards primários potenciais no caso de falhas. Por exemplo, os conjuntos de mapas podem ser configurados para permitir que as operações de leitura sejam roteadas para réplicas ao configurar a opção replicaReadEnabled no MapSet para true. A configuração padrão é false.

Para obter mais informações sobre o elemento MapSet, consulte o tópico no arquivo XML descritor de política de implementação no *Guia de Administração*.

Ativar a leitura de réplicas pode melhorar o desempenho ao propagar os pedidos de leitura para mais Java™ virtual machines. Se a opção não estiver ativada, todos os pedidos de leitura, tais como os métodos ObjectMap.get ou Query.getResultIterator são roteados para o primário.Quando replicaReadEnabled for configurado para true, alguns pedidos poderão retornar dados obsoletos, então, um aplicativo que usa essa opção deve poder tolerar essa possibilidade. Entretanto, não ocorrerá uma perda de cache. Se os dados não estiverem na réplica, o pedido get será redirecionado para o shard primário e tentado novamente.

A opção replicaReadEnabled pode ser usada com ambas as replicações síncrona e assíncrona.

Utilizando Zonas para Disposição de Réplicas

O suportem a zonas permite configurações sofisticadas para disposição de shards nos centros de dados. Com esse recurso, as grades de milhares de partições podem ser facilmente gerenciadas utilizando algumas regras de localização opcionais. Um datacenter pode estar em diferentes andares de um prédio, diferentes prédios ou até mesmo cidades diferentes ou outras distinções, conforme configurado com as regras de zonas.

Flexibilidade de Zonas

É possível colocar shards em zonas. Esta função permite que você tenha mais controle sobre como o eXtreme Scale coloca shards em uma grade. A Java Virtual Machines que hospeda um servidor eXtreme Scale pode ser identificada com um identificador de zona. O arquivo de implementação agora pode incluir uma ou mais regras de zonas e estas regras de zonas estão associadas com um tipo shard. A melhor maneira de explicar isto é com alguns exemplos acompanhados por mais detalhes.

Controle de zonas de disposição de como o eXtreme Scale designa primários e réplicas para configurar topologias avançadas.

Uma Java Virtual Machine pode ter vários servidores eXtreme Scale ativos. Um contêiner pode hospedar vários shards a partir de um único eXtreme Scale.

Este recurso é útil para certificar-se de que as réplicas e os primários sejam colocados em diferentes locais ou zonas para obter uma melhor alta disponibilidade. Normalmente, o eXtreme Scale não coloca um shard primário e de

réplica na Java Virtual Machines com o mesmo endereço IP. Esta regra simples normalmente evita que dois servidores eXtreme Scale sejam colocados no mesmo computador físico. Entretanto, pode ser necessário um mecanismo mais flexível. Por exemplo, você pode estar utilizando dois chassis blade e desejar que os shards primários sejam *divididos* em ambos os chassis e que a réplica para cada shard primário seja colocada no outro chassi a partir do primário.

Dividido significa que os shards primários são colocados em cada zona e a réplica para cada primário está localizada na zona oposta. O primário 0, por exemplo, poderia estar na zoneA, a a réplica síncrona 0 poderia estar na zoneB. O primário 1 poderia estar na zoneB, e a réplica síncrona 1 poderia estar na zoneA.

O nome do chassi poderia ser o nome da zona neste caso. Alternativamente, você pode chamar as zonas de acordo com os andares em um prédio e utilizar as zonas para ter certeza de que os primários e as réplicas para os mesmos dados estejam em andares diferentes. Prédios e datacenter também são possíveis. Foram feitos testes em datacenters utilizando zonas como um mecanismo para garantir que os dados sejam adequadamente replicados entre os datacenters. Se você estiver utilizando o HTTP Session Manager para eXtreme Scale, também é possível utilizar zonas. Com este recurso, é possível implementar um aplicativo da Web único em três datacenters e garantir que as sessões HTTP para usuários sejam replicas nos datacenters para que as sessões possam ser recuperadas, mesmo se um datacenters inteiro falhar.

O WebSphere eXtreme Scale está ciente da necessidade de gerenciar uma grade grande sobre vários datacenters. Isso pode assegurar que os primários e os backups da mesma partição estejam localizados em datacenters diferentes, se isto for necessário. Ele pode colocar todos os primários no datacenter 1 e todas as réplicas no datacenter 2, ou pode fazer um round robin com os primários e as réplica em ambos os datacenters. As regras são flexíveis de modo que muitos cenários são possíveis. O eXtreme Scale também pode gerenciar milhares de servidores, que juntos com a disposição completamente automática e o reconhecimento do datacenter tornam tais grades grandes financeiramente suportáveis de um ponto de vista administrativo. Os administradores podem especificar o que desejam fazer de maneira simples e eficiente.

Como um administrador, utilize zonas de disposição para controlar onde os shards primários e de réplica são colocados, o que permite a configuração de topologias avançadas de alto desempenho e altamente disponíveis. É possível definir uma zona para qualquer agrupamento lógico de processos do eXtreme Scale, conforme observado acima: Estas zonas podem corresponder a locais de estação de trabalho físicas, tais como um datacenter, um andar de um datacenter ou um chassi blade. É possível dividir dados em zonas, o que fornece aumento de disponibilidade ou você dividir os primários e as réplicas em zonas separadas quando um hot standby é necessário.

Associando um Servidor eXtreme Scale com um Zona que não está utilizando o WebSphere Extended Deployment

Se o eXtreme Scale for usado com o Java Standard Edition ou um servidor de aplicativos que não seja baseado no WebSphere Extended Deployment Versão 6.1, então um JVM que seja um contêiner shard pode ser associado a uma zona se forem usadas as técnicas a seguir.

Aplicativos utilizando o script startOgServer

O script startOgServer é utilizado para iniciar um aplicativo eXtreme Scale quando ele não está sendo integrado em um servidor existente. O parâmetro **-zone** é utilizado para especificar a zona a utilizar para todos os contêineres no servidores.

Especificando a zona ao iniciar um contêiner utilizando APIs

Associando Nós do WebSphere Extended Deployment com Zonas

Se você estiver utilizando o eXtreme Scale com aplicativos WebSphere Extended Deployment Java Platform, Enterprise Edition, pode utilizar o recurso de grupo de nós do WebSphere Extended Deployment para automaticamente colocar o membro de cluster nas zonas específicas das Java Virtual Machines. Uma JVM pode ser um membro de uma única zona. Os servidores em execução em um nó que é parte de um grupo de nós são incluídos na zona especificada pelo nome do grupo de nós. É necessário garantir que estes membros de cluster não sejam membros dos dois ou de mais de grupos de nós como estes. Uma JVM membro de cluster verifica a associação da zona apenas na inicialização. Incluir um novo grupo de nós ou alterar a associação tem impacto apenas em Java Virtual Machines recentemente iniciadas ou reiniciadas.

Regras de Zonas

Uma partição do eXtreme Scale possui um shard primários e zero ou mais shards de réplica. Para este exemplo, considere a seguinte convenção de nomenclatura para estes shards. `P` é o shard primário, `S` é uma réplica assíncrona e `A` é uma réplica assíncrona. Uma regra de zona possui três componentes:

- Um nome de regra
- Uma lista de zonas
- Um sinalizador inclusivo ou exclusivo

Uma regra de zona especifica o possíveis conjuntos de zonas nos quais um shard pode ser colocado. O sinalizador inclusivo indica que após um shard ser colocado na lista, então, todos os outros shards também são colocados em tal zona. Uma configuração exclusiva indica que cada shard para uma partição é colocada em uma zona diferente na lista de zonas. Por exemplo, utilizar uma configuração exclusiva significa que se houver três shards (um primário e duas réplicas síncronas), então, a lista de zonas deve ter três zonas.

Cada shard pode ser associado com uma regra de zona. Uma regra de zona pode ser compartilhada entre dois shards. Quando uma regra é compartilhada, então o sinalizador inclusivo ou exclusivo é estendido pelos shards de todos os tipos compartilhando uma regra única.

Exemplos

A seguir, está um conjunto de exemplos mostrando diversos cenários e a configuração de implementação para implementar os cenários.

Dividindo primários e réplicas entre zonas

É possível ter três chassis blade e desejar primários distribuídos para os três, com uma réplica síncrona única colocada em um chassi diferente do primário. Defina cada chassis como uma zona com nomes de chassi `ALPHA`, `BETA` e `GAMMA`. A seguir, está um exemplo de XML de implementação:

```
<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
 xsi:schemaLocation=
 "http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
    xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
  <objectgridDeployment objectgridName="library">
   <mapSet name="ms1" numberOfPartitions="37" minSyncReplicas="1"
    maxSyncReplicas="1" maxAsyncReplicas="0">
   <map ref="book" />
   <zoneMetadata>
    <shardMapping shard="P" zoneRuleRef="stripeZone"/>
    <shardMapping shard="S" zoneRuleRef="stripeZone"/>
    <zoneRule name ="stripeZone" exclusivePlacement="true" >
     <zone name="ALPHA" />
     <zone name="BETA" />
     <zone name="GAMMA" />
    </zoneRule>
   </zoneMetadata>
  </mapSet>
 </objectgridDeployment>
</deploymentPolicy>
```

Este XML de implementação contém uma grade chamada library com um Mapa único denominado book. Ele utiliza quatro partições com uma única réplica síncrona. A cláusula metadata da zona mostra a definição de uma única regra de zona e a associação das regras de zona com os shards. Os shards primários e síncronos estão ambos associados com a regra de zona ″stripeZone″. A regra de zona possui as três zonas e utiliza a disposição exclusiva. Esta regra significa que se o primário para a partição 0 for colocado no ALPHA, então, a réplica para a partição 0 será colocada no BETA ou GAMMA. De maneira semelhante, os primários para outras partições são colocados em outras zonas e as réplicas serão colocadas.

Réplica assíncrona em uma zona diferente da primária e réplica síncrona

Neste exemplo, existem dois prédios com uma alta conexão de latência entre eles. Você não deseja alta disponibilidade de perda de dados para todos os cenários. Entretanto, o impacto de desempenho da replicação síncrona entre os prédios o leva a um dilema. Você deseja um primário com réplica síncrona em um prédio e uma réplica assíncrona em outro prédio. Normalmente, as falhas são paralisações da JVM ou falhas de computador ao invés de problemas de larga escala. Com esta topologia, é possível sobrevier a falhas normais sem nenhuma perda de dados. A perda de um prédio é rara o suficiente que alguma perda de dados é aceitável neste caso. É possível criar duas zonas, uma para cada prédio. A seguir, está o arquivo XML de implementação:

```
<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
  xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">

 <objectgridDeployment objectgridName="library">
  <mapSet name="ms1" numberOfPartitions="13" minSyncReplicas="1"
   maxSyncReplicas="1" maxAsyncReplicas="1">
   <map ref="book" />
   <zoneMetadata>
    <shardMapping shard="P" zoneRuleRef="primarySync"/>
    <shardMapping shard="S" zoneRuleRef="primarySync"/>
    <shardMapping shard="A" zoneRuleRef="aysnc"/>
    <zoneRule name ="primarySync" exclusivePlacement="false" >
      <zone name="BldA" />
      <zone name="BldB" />
    </zoneRule>
    <zoneRule name="aysnc" exclusivePlacement="true">
      <zone name="BldA" />
      <zone name="BldB" />
    </zoneRule>
   </zoneMetadata>
  </mapSet>
 </objectgridDeployment>
</deploymentPolicy>
```

O primário e a réplica síncrona compartilham uma regra de zona primaySync com uma configuração de sinalizador exclusivo de false. Portanto, após o primário ou síncrono ser colocado em uma zona, então o outro também é colocado na mesma

zona. A réplica assíncrona utiliza uma segunda regra de zona com as mesmas zonas que a regra de zona primarySync, mas ela utiliza o atributo **exclusivePlacement** configurado como true. Este atributo indica que um shard não pode ser colocado em uma zona com outro shard a partir da mesma partição. Como resultado, a réplica assíncrona não é colocada na mesma zona que o primário e as réplicas síncronas.

Colocar todos os primários em uma zona e todas réplicas em outra zona

Aqui, todos os primários estão em uma zona específica e todas as réplicas em uma zona diferente. Teremos um primário e uma réplica assíncrona única. Todas as réplicas estarão na zona A e os primários na B.

```
<?xml version="1.0" encoding="UTF-8"?>

<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation=
  "http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
 xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">

 <objectgridDeployment objectgridName="library">
  <mapSet name="ms1" numberOfPartitions="13" minSyncReplicas="0"
   maxSyncReplicas="0" maxAsyncReplicas="1">
   <map ref="book" />
   <zoneMetadata>
    <shardMapping shard="P" zoneRuleRef="primaryRule"/>
    <shardMapping shard="A" zoneRuleRef="replicaRule"/>
    <zoneRule name ="primaryRule">
     <zone name="A" />
    </zoneRule>
    <zoneRule name="replicaRule">
     <zone name="B" />
      </zoneRule>
     </zoneMetadata>
    </mapSet>
  </objectgridDeployment>
</deploymentPolicy>
```

Aqui, você pode ver duas regras, uma para os primários (P) e outra para a réplica (A).

Zonas sobre Wide Area Networks (WAN)

Você pode desejar implementar um eXtreme Scale único em vários prédios ou datacenters com interconexões de rede mais lentas. Conexões de rede mais lentas levam a uma largura da banda menor e mais conexões de latência. A possibilidade de partições de rede também aumenta neste modo devido ao congestionamento de rede e outros fatores. O eXtreme Scale aborda este ambiente severo das seguintes maneiras.

Pulsação limitada entre zonas

As Java Virtual Machines agrupadas em grupos principais realizam pulsação uma com a outra. Quando o serviço de catálogo organiza as Java Virtual Machines nos grupos, tais grupos não se estendem sobre zonas. Um líder neste grupo executa o push das informações de associação no serviço de catálogo. O serviço de catálogo verifica quaisquer falhas relatadas antes de tomar a ação. Ele faz isto ao tentar conectar-se às Java Virtual Machines suspeitas. Se o serviço de catálogo visualiza uma falsa detecção de falha, então, ele não executa nenhuma ação já que a partição do grupo principal irá resolver o problema em um curto período de tempo.

O serviço de catálogo também executar pulsação nos líderes do grupo principal periodicamente em uma taxa mas baixa para tratar o caso de isolamento do grupo principal.

Serviço de catálogo como o desempatador de grade

O serviço de catálogo é o desempatador para uma grade do eXtreme Scale. É essencial que ele atue como uma única voz para que a grade execute. O serviço de catálogo é executado em um conjunto fixo de Java Virtual Machines replicando dados a partir de um primário eleito para todas as outras Java Virtual Machines em tal conjunto. O serviço de catálogo deve ser distribuído entre as zonas físicas ou datacenters para diminuir a probabilidade de ficar isolado da grade e poder sobreviver a cenários de falha antecipados.

O serviço de catálogo comunica-se com a Java Virtual Machines do contêiner na grade utilizando operações idempotentes ou recuperáveis. Ele faz isto comunicando-se utilizando o IIOP. Todas as alterações de estado no serviço de catálogo são sincronamente replicadas entre os membros atuais hospedando o serviço de catálogo. Esta replicação é bem-sucedida apenas se a maioria das Java Virtual Machines aceitar a alteração. Isto significa que se o serviço de catálogo é particionado, então, apenas a a partição de maioria pode executar commit de alterações. O primário do serviço enviará comandos apenas para as Java Virtual Machines do contêiner ocorrer o commit da alteração de estado que cria tal comando. Isto significa que uma partição de minoria avança seu estado ou emite comandos para os contêineres.

Um serviço de catálogo particionado não interrompe o funcionamento da grade. A grade ainda aceitará pedidos do cliente e executará operações. Se não houver uma partição de serviço de catálogo de maioria, então, as falhas na grade não serão recuperadas até o serviço de catálogo recupere a maioria. Se a recuperação for atrasada, então, ao longo do tempo, enquanto ocorrerem falhas, determinadas partições ficarão off-line até que o serviço de catálogo retome a maioria.

As Java Virtual Machines do líder do grupo principal relata alterações de associação no serviço de catálogo. Se o serviço de catálogo for particionado, então o serviço executará um push back de uma tabela de rotas atualizada para o serviço de catálogo. Esta tabela de rotas de uma partição de minoria não incluirá o local para um primário. O líder precisará iterar em todas as Java Virtual Machines de serviço de catálogo possíveis para localizar a partição primária. Ele precisará fazer isto periodicamente enquanto está aguardando que a partição seja resolvida. Depois, ele recebe uma tabela de rotas com um primário, em seguida, ações de recuperação pendentes a serem direcionadas pelo primário do serviço de catálogo de maioria.

Se o grupo principal não puder conectar-se a um primário do serviço de catálogo por um período de tempo, então, ele está fisicamente desconectado do resto da grade (possivelmente com uma partição de serviço de catálogo de minoria) ou o serviço de catálogo está preso nas partições de minoria. É impossível notar a diferença. Se houver uma partição de serviço de catálogo de maioria, então, ela pode estar se recuperando da aparente perda do grupo principal desconectado. Isto pode levar a dois primários na mesma partição, o primário existente antigo e o novo primário no resto da rede. A partição de serviço de catálogo de maneira não tem como executar o 'kill' dos primários antigos já que está em um estado de rede desconectado com os primários antigos. Quando o serviço de catálogo se recupera e o grupo principal desconectado descobre os novos primários, então, o serviço de catálogo não notará que há dois primários. Ele instruirá os grupos principais anteriormente desconectados para excluir todos os shards e, então, ocorrerá um equilíbrio.

Se o serviço de catálogo for particionado em duas partições de minoria ou uma única partição de minoria sobrevivente, então, o cliente precisará envolver-se para ajudar com a recuperação. Um comando JMX (Java Management Extensions) é

necessário para especificar que uma única partição de menoridade pode tomar uma ação. É necessário garantir que as outras partições de minoria sejam interrompidas.

Roteamento para Zonas Preferenciais

Com o roteamento para zonas preferenciais, o eXtreme Scale pode direcionar transações para zonas com base em especificações preferidas que você faz.

O WebSphere eXtreme Scale permite exercitar uma quantidade significativa de controle sobre onde os shards de um ObjectGrid são colocados. Consulte "Utilizando Zonas para Disposição de Réplicas" na página 105 para obter informações sobre alguns cenários básicos e como configurar corretamente sua política de implementação.

O roteamento de zona preferencial permite que os clientes do eXtreme Scale especifiquem uma propensão para uma zona específica ou conjunto de zonas. Assim o eXtreme Scale tentará rotear as transações de cliente para zonas preferidas antes de tentar rotear para qualquer outra zona.

Requisitos para Roteamento de Zona Preferencial

Há diversos fatores a serem considerados antes de tentar o roteamento de zona preferencial. Certifique-se de que o aplicativo possa satisfazer os requisitos de seu cenário.

O posicionamento de partição por contêiner é necessário para aproveitar o roteamento de zona preferencial. Esta estratégia de posicionamento é uma boa opção para aplicativos que são dados de sessão de armazenamento no ObjectGrid. A estratégia de posicionamento de partição padrão do WebSphere eXtreme Scale é de partição fixa. As chaves são colocadas em hash no momento de consolidação de transação para determinar qual partição hospeda o par chave-valor do mapa ao usar o posicionamento de partição fixa.

O posicionamento por contêiner designa seus dados a uma partição aleatória no momento da consolidação da transação através do SessionHandle. Você deve poder reconstruir o SessionHandle para recuperar seus dados a partir do ObjectGrid.

Como você pode usar zonas para ter mais controle através de onde os primários e suas réplicas residirão em seu domínio, uma implementação de múltiplas zonas é vantajosa quando seus dados residem em múltiplas localizações físicas. A separação geográfica de primários e réplicas é uma forma de garantir que a perda catastrófica de 1 datacenter não causará impacto na disponibilidade dos dados.

Quando os dados são espalhados por uma topologia em múltiplas zonas, é mais provável que os clientes também sejam espalhados pela topologia. O roteamento de clientes para sua zona local ou datacenter possui o benefício óbvio de desempenho reduzido de latência de rede. Tire vantagem deste cenário quando possível.

Configuração de sua Topologia para Roteamento de Zona Preferencial

Considere o seguinte cenário. Você possui 2 datacenters: Chicago e Londres. Para minimizar o tempo de resposta de clientes, você deseja que os clientes leiam e escrevam dados em seus datacenter locais.

Os shards primários do ObjectGrid devem ser colocados em cada datacenter para que as transações possam ser escritas localmente a partir de cada local. Além disso, os clientes precisarão estar cientes de zonas para rotear para a zona local.

O posicionamento por contêiner localiza os novos shards primários em cada contêiner iniciado. As réplicas são posicionadas de acordo com a zona e as regras de posicionamento especificado pela política de implementação. Pelo padrão, uma réplica é colocada em uma zona diferente da zona que seu primário. Considere a política de implementação a seguir para este cenário.

```
<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
 xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
 <objectgridDeployment objectgridName="universe">
  <mapSet name="mapSet1" placementStrategy="PER_CONTAINER"
   numberOfPartitions="3" maxAsyncReplicas="1">
   <map ref="planet" />
  </mapSet>
 </objectgridDeployment>
</deploymentPolicy>
```

Cada contêiner que inicia com a política de implementação receberá 3 novos primários. Cada primário terá 1 réplica assíncrona. Inicie cada contêiner com o nome de zona adequado. Use o parâmetro -zone se você estiver ativando seus contêineres com o script startOgServer.

Para um servidor de contêineres Chicago:

- UNIX Linux

```
startOgServer.sh s1 -objectGridFile ../xml/universeGrid.xml
-deploymentPolicyFile ../xml/universeDp.xml
-catalogServiceEndpoints MyServer1.company.com:2809
-zone Chicago
```

- Windows

```
startOgServer.bat s1 -objectGridFile ../xml/universeGrid.xml
-deploymentPolicyFile ../xml/universeDp.xml
-catalogServiceEndpoints MyServer1.company.com:2809
-zone Chicago
```

Se seus contêineres estiverem em execução no WebSphere Application Server, será necessário criar um grupo de nós e nomeá-lo com o prefixo “ReplicationZone”. Os servidores em execução nos nós nestes grupos de nós serão colocados na zona adequada. Por exemplo, os servidores em execução em um nó Chicago podem estar em um grupo de nós nomeado como “ReplicationZoneChicago”. Consulte “Associando os nós do WebSphere Extended Deployment com zonas” em “Utilizando Zonas para Disposição de Réplicas” na página 105.

Os primários para a zona Chicago terão réplicas na zona Londres. Os primários para a zona Londres terão réplicas na zona Chicago.

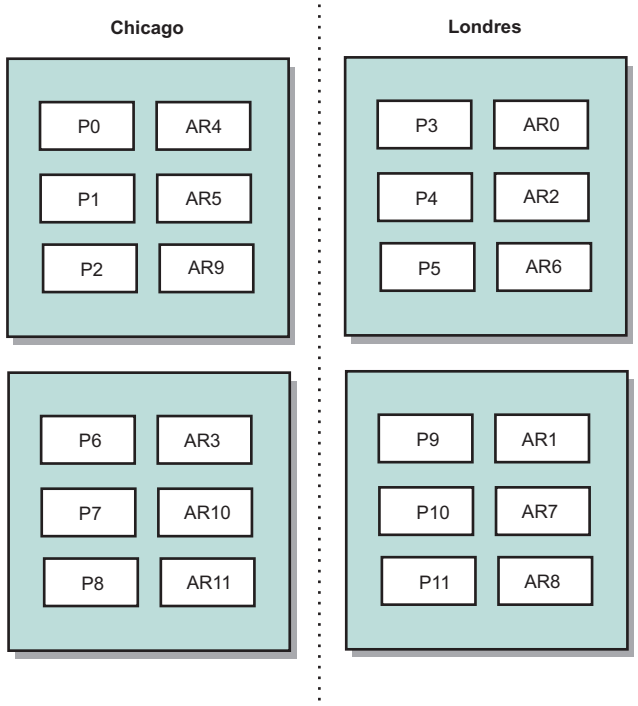


Figura 39. Primários e Réplicas em Zonas

Configure as zonas preferenciais para os clientes. Estas informações podem ser fornecidas em uma das seguintes formas diferentes. A forma mais direta é fornecer um arquivo de propriedades do cliente na JVM do cliente. Crie um arquivo nomeado como objectGridClient.properties e certifique-se de que ele esteja em seu caminho da classe. Para obter informações adicionais, consulte o tópico no arquivo de propriedades do cliente no *Guia de Administração.*

Inclua a propriedade preferZones no arquivo. Consulte o valor da propriedade para a zona adequada. Os clientes em Chicago devem ter o seguinte no arquivo objectGridClient.properties.

```
preferZones=Chicago
```

O arquivo de propriedades para clientes Londres devem conter

```
preferZones=Londres
```

Esta propriedade instrui cada cliente para rotear transações à sua zona local se possível. Dados que são inseridos em um primário na zona local serão assincronamente replicados em uma zona estrangeira.

Uso do SessionHandle para Rotear para a Zona Local

A estratégia de posicionamento por contêiner não usa um algoritmo baseado em hash para determinar o local de seus pares chave-valor no ObjectGrid. Seu ObjectGrid deve aproveitar SessionHandles para garantir que as transações sejam roteadas para o local correto ao usar esta estratégia de posicionamento. Quando uma transação é consolidada, um SessionHandle é limitado à Sessão se uma ainda não tiver sido configurada. Alternativamente, o SessionHandle pode ser limitado à

Sessão através da chamada de Session.getSessionHandle antes da consolidação da transação. O trecho de código a seguir mostra um SessionHandle sendo limitado antes da consolidação da transação.

```
Session ogSession = objectGrid.getSession();

// binding the SessionHandle
SessionHandle sessionHandle = ogSession.getSessionHandle();

ogSession.begin();
ObjectMap map = ogSession.getMap("planet");
map.insert("planet1", "mercury");

// tran is routed to partition specified by SessionHandle
ogSession.commit();
```

Assuma que o código acima estava executando em um cliente em seu datacenter Chicago. Como atributo preferZones foi configurado como Chicago para este cliente, esta transação seria roteada para uma das partições primárias na zona Chicago, partição 0, 1, 2, 6, 7 ou 8.

Este SessionHandle é seu caminho de volta para a partição que armazena estes dados consolidados. O SessionHandle deve ser reutilizado ou reconstruído e configurado na Sessão para voltar para a partição contendo os dados consolidados.

```
ogSession.setSessionHandle(sessionHandle);
ogSession.begin();

// value returned will be "mercury "
String value = map.get("planet1");
ogSession.commit();
```

Como está transação reutiliza o SessionHandle que foi criado durante a transação insert, a transação get será roteada para a partição que hospeda os dados inseridos. Esta transação não poderá recuperar os dados inseridos anteriormente se o SessionHandle não tiver sido configurado.

Como as Falhas de Contêiner e Zonas Afetam o Roteamento Baseado em Zona

Um cliente com o conjunto de propriedades preferZones terá todas as suas transações roteadas para a zona (ou zonas) especificada(s) sob circunstâncias normais. Porém, a perda de um contêiner resultará em uma réplica sendo promovida para primário em uma zona estrangeira. Um cliente que estava anteriormente roteando para partições na zona local pode ser forçado para a zona estrangeira para recuperar dados inseridos anteriormente.

Considere o seguinte cenário. Um contêiner na zona Chicago foi perdido. Ele anteriormente continha primários para partições 0, 1 e 2. Os novos primários para estas partições estarão na zona Londres pois esta zona estava hospedando as réplicas para estas partições.

Qualquer cliente Chicago que estiver usando um SessionHandle apontando para uma das partições falhas agora serão roteados para Londres. Os clientes Chicago usando os novos SessionHandles serão roteados para os primários baseados em Chicago.

Da mesma forma, se toda a zona Chicago for perdida, todas as réplicas na zona Londres se tornarão primárias. Neste caso, todos os clientes Chicago terão suas transações roteadas para Londres.

Tipos de Detecção de Failover

O WebSphere eXtreme Scale usa dois métodos para detecção de failover.

Pulsação

1. Os soquetes são mantidos abertos entre o Java Virtual Machines e se um soquete for fechado inesperadamente, esse fechamento inesperado será detectado como uma falha do peer Java Virtual Machine. Essa detecção captura casos de falha, como o Java Virtual Machine saindo muito rapidamente,e também permite que esses tipos de falhas sejam recuperados normalmente em menos de um segundo.
2. Outros tipos de falhas incluem: um pânico no sistema operacional, falha física do servidor ou falha de rede. Estas falhas são tratadas utilizando pulsação.

Pulsações são enviadas periodicamente entre pares de processos: Quando um número fixo de pulsações está ausente, então é assumida uma falha. Essa abordagem detecta falhas em N*M segundos, em que N é o número de pulsações ausentes e M é o intervalo no qual as pulsações devem ser configuradas. A especificação direta de M e N não é suportada, e ao invés disso, um mecanismo de régua de controle é utilizado para permitir um intervalo de combinações de M e N testadas a ser utilizado.

Falhas

Um processo pode falhar de várias maneiras. o processo poderia falhar porque algum limite de recurso foi atingido, tal como o tamanho máximo do heap ou alguma lógica de controle de processo terminou um processo. O sistema operacional poderia falhar, fazendo com que todos os processos em execução no mesmo sistema fossem perdidos. O hardware poderia falhar, embora com menos freqüência, como a NIC (placa da interface de rede), fazendo com que o sistema operacional fosse desconectado da rede. Muitos outros pontos de falha podem ocorrer, fazendo com que o processo se torne indisponível. Neste contexto, todas estas falhas podem ser categorizadas em um dos dois tipos: falha de processo e perda de conectividade.

Falha de Processo

O WebSphere eXtreme Scale reage para processar falhas muito rapidamente. Quando um processo falha, o sistema operacional é responsável pela limpeza de qualquer recurso pendente que o processo estava utilizando. Essa limpeza inclui a alocação e conectividade da porta. Quando um processo falha, um sinal é enviado para as conexões que estavam sendo utilizadas por esse processo para fechar cada conexão. Com estes sinais, uma falha de processo pode ser detectada instantaneamente por qualquer outro processo que está conectado ao processo falho.

Perda de Conectividade

A perda de conectividade ocorre quando o sistema operacional se desconecta. Como resultado, o sistema operacional não pode enviar sinais a outros processos. Há diversos motivos pelos quais pode ocorrer uma perda de conectividade, mas eles podem ser divididos em duas categorias: falha de host e ilhamento.

Falha de Host

Se a máquina for desconectada da tomada de energia, ela desligará instantaneamente.

Insulamento

Este cenário apresenta a condição de falha mais complicada para o software tratar corretamente porque o processo é presumido como indisponível, embora não esteja.

Falha de Contêiner

Falhas contêiner geralmente são descobertas por contêineres peer através do mecanismo de grupo principal. Quando um contêiner ou conjunto de contêineres falha, o serviço de catálogo migra os fragmentos que foram hospedados nesse contêiner ou contêineres. O serviço de catálogo procura uma réplica síncrona primeiro, antes de migrar para uma réplica assíncrona. Depois da migração dos fragmentos primários para os novos contêineres de host, o serviço de catálogo procura novos contêineres de host para as réplicas que agora estão faltando.

Nota: Ilhamento de contêiner - O serviço de catálogo migra shards dos contêineres quando descobre-se que o contêiner está indisponível. Se esses contêineres se tornarem disponíveis, o serviço de catálogo considerará os contêineres elegíveis para disposição como no fluxo de inicialização normal.

Latência de detecção de failover de contêiner

As falhas podem ser categorizadas em falhas brandas e falhas graves. Falhas brandas normalmente são causadas quando um processo falha. Tais falhas são detectadas pelo sistema operacional, que pode recuperar recursos utilizados, tais como soquetes de rede, muito rapidamente. A detecção de falha típica para falhas brandas é de menos de um segundo. A falhas graves podem levar até 200 segundos até detectar utilizando o ajuste de pulsação padrão. Tais falhas incluem: falhas de máquina física, desconexões de cabo de rede ou falhas de sistema operacional. Deste modo, o eXtreme Scale deve contar com a pulsação para detectar falhas graves que podem ser configuradas. Consulte “Tipos de Detecção de Failover” na página 115 para obter detalhes sobre a diminuição do tempo necessário para detectar uma falha grave.

Várias Falhas de Contêiner

Uma réplica nunca é colocada no mesmo processo que seu primário porque, se o processo for perdido, isso resultará na perda do primário e da réplica. A política de implantação define um atributo que o serviço de catálogo utiliza para determinar se uma réplica pode ser colocada na mesma máquina que o primário. Em um ambiente de implantação em uma única máquina, talvez você queira ter dois contêineres e replicar entre eles. Entretanto, em produção, utilizar uma máquina única é suficiente porque a perda de tal host resulta na perda de ambos os contêineres. Para alterar entre o modo de desenvolvimento em uma máquina única e um modo de produção com várias máquinas, desative o modo de desenvolvimento no arquivo de configuração da política de implementação.

Falha de Serviço de Catálogo

Como a grade de serviço do catálogo é uma grade do eXtreme Scale, ela também usa o mecanismo de agrupamento principal da mesma forma que o processo de falha de contêiner. A diferença primária é que a grade do serviço de catálogo usa

um processo de eleição peer para definir o shard primário em vez do algoritmo do serviço de catálogo que é usado para os contêineres.

Note que o serviço de disposição e o serviço de agrupamento principal são serviços um-entre-N, mas o serviço e a administração de local são executados em qualquer lugar. O serviço de disposição e o serviço de agrupamento principal são singletons porque são responsáveis pela configuração do sistema. O serviço de local e a administração são serviços somente leitura e existe em qualquer lugar para fornecer escalabilidade.

O serviço de catálogo usa a replicação para tornar-se tolerante a falhas. Se um processo de serviço de catálogo falhar, o serviço deverá ser reiniciado para restaurar o sistema para o nível de disponibilidade desejado. Se todos os processos que estão hospedando o serviço de catálogo falharem, o eXtreme Scale possui uma perda de dados críticos. Essa falha resulta em um reinício necessário de todos os contêineres. Como o serviço de catálogo pode ser executado em muitos processos, essa falha é um evento improvável. Entretanto, se você estiver executando todos os processos em uma única caixa, dentro de um único chassi de blade, ou de um único comutador de rede, será mais provável que uma falha ocorra. Tente remover os modos de falha comuns das caixas que estão hospedando o serviço de catálogo para reduzir a possibilidade de falha.

Serviço de Catálogo de Alta Disponibilidade

Um serviço de catálogo é a grade de servidores de catálogo que você está usando, que retém as informações de topologia para todos os contêineres no seu ambiente do eXtreme Scale. O serviço de catálogo controla o equilíbrio e o roteamento de todos os clientes. Para implementar o eXtreme Scale como um espaço de processamento de banco de dados de memória, poderá armazenar em cluster o serviço de catálogo em uma grade para alta disponibilidade.

Componentes do Serviço de Catálogo

Quando múltiplos servidores de catálogo iniciam, um dos servidores é eleito como o servidor de catálogo principal que aceita pulsações do Internet Inter-ORB Protocol (IIOP) e manipula as alterações dos dados do sistema em resposta a qualquer serviço de catálogo ou as alterações de contêiner.

Quando os clientes entram em contato com qualquer um dos servidores de catálogo, a tabela de roteamento para a grade do servidor de catálogos é propagada para os clientes através do contexto de serviço Common Object Request Broker Architecture (CORBA).

Configure pelo menos três servidores de catálogos. Se a sua configuração tem zonas, é possível configurar um servidor de catálogos por zona.

Quando um servidor e contêiner do eXtreme Scale entra em contato com um dos servidores de catálogos, a tabela de roteamento para a grade do servidor de catálogos também é propagada para o servidor e contêiner do eXtreme Scale através do contexto de serviço CORBA. Além disso, se o servidor de catálogos contactado atualmente não for o servidor de catálogos principal, o pedido é automaticamente roteado para o servidor de catálogos principal atual e a tabela de roteamento para o servidor de catálogos é atualizada.

Nota: Uma grade do servidor de catálogos e a grade do servidor de contêineres são muito diferentes. A grade do servidor de catálogos é para a alta

disponibilidade dos seus dados do sistema. A grade do contêiner é para a alta disponibilidade de dados, escalabilidade e gerenciamento de carga de trabalho. Portanto, existem duas diferentes tabelas de roteamento: a tabela de roteamento para a grade do servidor de catálogos e a tabela de roteamento para os shards da grade do servidor.

As responsabilidades do catálogo são divididas em uma série de serviços. O gerenciador do grupo principal executa agrupamento peer para monitoramento de funcionamento, o serviço de posicionamento executa alocação, o serviço de administração fornece acesso à administração e o serviço de local gerencia a localidade.

Implementação da Grade de Catálogo

Gerenciador de grupo principal

O serviço de catálogo usa o gerenciador de alta disponibilidade (gerenciador HA) para agrupar processos para o monitoramento de disponibilidade. Cada agrupamento dos processos é um grupo principal. Com o eXtreme Scale, o gerenciador do grupo principal agrupa dinamicamente os processos. Estes processos são mantidos pequenos para permitir a escalabilidade. Cada grupo principal elege um líder que possui a responsabilidade adicional de enviar o status para o gerenciador do grupo principal quando membros individuais falham. O mesmo mecanismo de status é usado para descobrir quando todos os membros de um grupo falham, o que causa a falha da comunicação com o líder.

O gerenciador do grupo principal é um serviço completamente automático responsável pela organização de contêineres em pequenos grupos de servidores que são então automaticamente associados de maneira livre para criar um ObjectGrid. Quando um contêiner entra em contato com o serviço de catálogo pela primeira vez, ele aguarda para ser designado a um grupo novo ou existente de vários. O eXtreme Scale consiste de diversos destes grupos e este agrupamento é um importante ativador de escalabilidade. Cada grupo é um grupo do Java Virtual Machines que usa a pulsação para monitorar a disponibilidade dos outros grupos. Um destes membros do grupo é eleito o líder e possui uma responsabilidade adicional de retransmitir informações de disponibilidade para o serviço de catálogo para permitir reação através de realocação e encaminhamento de rotas.

Serviço de disposição

O serviço de catálogo gerencia o posicionamento de shards por todo o conjunto de contêineres disponíveis. O serviço de disposição é responsável pela manutenção de um equilíbrio de recursos entre recursos físicos. O serviço de disposição é responsável pela alocação de shards individuais em seu contêiner de host. Ele executa como um serviço eleito ″um-entre-N″ na grade de forma que sempre exista exatamente uma instância do serviço em execução. Se tal instância falhar, outro processo é então eleito e assume. Para redundância, o estado do serviço de catálogo é replicado entre todos os servidores que estão hospedando o serviço de catálogo.

Administração

O serviço de catálogo também é o ponto de entrada lógico para administração do sistema. O serviço de catálogo hospeda um Managed Bean (MBean) e fornece as URLs do Java Management Extensions (JMX) para qualquer um dos servidores que o serviço está gerenciando.

Serviço de local

O serviço de local atua como o ponto de contato para ambos os clientes que estão procurando contêineres que hospedam o aplicativo que procuram, bem como os contêineres que estão registrando aplicativos hospedados com o serviço de disposição. O serviço de local é executado em todos os membros da grade para efetuar o scale out desta função.

O serviço de catálogo hospeda lógica que é tipicamente inativa durante estados estáveis. Como resultado, o serviço de catálogo influencia a escalabilidade minimamente. O serviço é baseado em centenas de serviços de contêineres que se tornam disponíveis simultaneamente. Para disponibilidade, configure o serviço de catálogo numa grade.

Planejamento

Após o início de uma grade de catálogo, os membros da grade se ligam. Cuidadosamente, planeje a topologia da sua grade de catálogo porque você não pode modificar a configuração do catálogo no tempo de execução. Disperse a grade o mais diversamente possível para evitar erros.

Iniciando uma grade do servidor de catálogo

Conectando os contêineres integrados no eXtreme Scale no WebSphere Application Server para uma grade de catálogo independente

É possível configurar os contêineres doeXtreme Scale que estão integrados em um ambiente do WebSphere Application Server para conectar-se a uma grade de catálogo independente. Use a mesma propriedade para conectar o servidor de aplicativos à grade de catálogo. Entretanto, a propriedade não gerencia o ciclo de vida do catálogo com os servidores. Em vez disso, a propriedade permite que os contêineres localizem a grade de catálogo remota. Para configurar a propriedade, consulte.

Nota: Conflito de nome do servidor: Como esta propriedade é usada para iniciar o servidor de catálogos do eXtreme Scale, assim como para se conectar a ele, os servidores de catálogo não devem ter o mesmo nome de nenhum servidor do WebSphere Application Server.

Quorums de Servidores de Catálogo

Quorum é o número mínimo de servidores de catálogo necessário para conduzir as operações de colocação para a grade. O número mínimo é o conjunto completo de servidores de catálogo, a menos que o quorum tenha sido substituído.

Termos Importantes

A seguir há uma lista de termos relacionados às considerações de quorum para o eXtreme Scale.

- Queda de energia: Uma queda de energia é a perda temporária de conectividade entre um ou mais servidores.
- Blecaute: Um blecaute é a perda permanente de conectividade entre um ou mais servidores.
- Datacenter: Um datacenter é um grupo de servidores localizado geograficamente conectado a uma rede local (LAN).

- Zona: Uma zona é uma opção de configuração usada para agrupar servidores que compartilham alguma característica física. Por exemplo, os servidores em um datacenter podem todos serem marcados em uma zona.
- Pulsação: Uma pulsação é o ato de efetuar ping de uma Java virtual machine (JVM) para determinar sua atividade.

Topologia

Esta seção explica como o IBM WebSphere eXtreme Scale opera através de uma rede que inclui componentes não-confiáveis. Os exemplos dessa rede incluem uma rede que se estende por vários datacenters.

Espaço de endereço IP

O WebSphere eXtreme Scale requer uma rede onde qualquer elemento endereçável na rede possa se conectar a qualquer outro elemento endereçável na rede desimpedida. Isso significa que o WebSphere eXtreme Scale requer um espaço de nomenclatura de endereço IP simples e requer que todos os firewalls para todo o tráfego fluem entre os endereços IP e as portas usados pelos elementos de host do Java virtual machines (JVM) do WebSphere eXtreme Scale.

LANs Conectadas

Cada LAN recebe um identificador de zona para os requisitos do WebSphere eXtreme Scale. O WebSphere eXtreme Scale pulsará rigorosamente as JVMs em uma única zona e uma pulsação perdida resultará em um evento de failover enquanto o serviço de catálogo possuir quorum.

Grade de serviço de catálogo e servidores de contêiner

Uma grade é uma coleta de JVMs semelhantes. Um serviço de catálogo é uma grade composta de servidores de catálogo e possui tamanho fixo. Entretanto, o número de servidores de contêiner é dinâmico. Os servidores de contêiner podem ser incluídos e removidos sob demanda. Em uma configuração de três datacenter, o WebSphere eXtreme Scale requer uma JVM de serviço de catálogo por datacenter.

A grade de serviço de catálogo usa um mecanismo de quorum completo. Isso significa que todos os membros da grade devem concordar em alguma ação.

As JVMs do servidor de contêiner são identificadas com um identificador de zona. A grade das JVMs de contêiner é dividida automaticamente em pequenos grupos principais de JVMs. Um grupo principal incluirá apenas as JVMs da mesma zona. JVMs de zonas diferentes nunca devem estar no mesmo grupo principal.

Um grupo principal tentará rigorosamente detectar a falha das JVMs de membro. As JVMs de contêiner de um grupo principal nunca devem se estender por várias LANs conectadas com links, como em uma ampla rede local.Isso significa que um grupo de núcleo não pode ter contêineres na mesma zona em execução em datacenters diferentes.

Ciclo de Vida do Servidor

Inicialização do servidor de catálogo

Os servidores de catálogo são iniciados usando o comando startOgServer. O mecanismo de quorum é desativado por padrão. Para ativar o quorum, transmita o

sinalizador ativado `-quorum` no comando startOgServer ou inclua a propriedade `enableQuorum=true` no arquivo de propriedades. Todos os servidores de catálogo devem receber a mesma configuração de quorum.

```
# bin/startOgServer cat0 –serverProps objectGridServer.properties
```

```
objectGridServer.properties file
catalogClusterEndPoints=cat0:cat0.domain.com:6600:6601,
cat1:cat1.domain.com:6600:6601
catalogServiceEndPoints= cat0.domain.com:2809, cat1.domain.com:2809
enableQuorum=true
```

Inicialização do servidor de contêiner

Os servidores de contêiner são iniciados usando o comando startOgServer. Ao executar uma grade entre os datacenters, os servidores deverão usar a tag de zona para identificar o datacenter no qual eles residem. Configurar a zona nos servidores de grade permite que o WebSphere eXtreme Scale monitore o funcionamento dos servidores com escopo definido no datacenter, minimizando o tráfego entre o datacenter.

```
# bin/startOgServer gridA0 –serverProps objectGridServer.properties –
objectgridfile xml/objectgrid.xml –deploymentpolicyfile xml/
deploymentpolicy.xml
```

```
objectGridServer.properties file
catalogServiceEndPoints= cat0.domain.com:2809, cat1.domain.com:2809
zoneName=ZoneA
```

Encerramento do servidor de grade

Os servidores de grade são interrompidos usando o comando stopOgServer. Ao encerrar um datacenter inteiro para manutenção, transmita a lista de todos os servidores que pertencerem a essa zona. Isso permitirá que a zona mude normalmente do estado teardown para uma ou mais zonas no estado surviving.

```
# bin/startOgServer gridA0,gridA1,gridA2 –catalogServiceEndPoints
cat0.domain.com:2809,cat1.domain.com:2809
```

Detecção de Falha

O WebSphere eXtreme Scale detecta a inatividade do processo através de eventos de encerramento anormal de soquete. O serviço de catálogo será informado imediatamente quando um processo se encerrará. Um blecaute é detectado através de pulsações perdidas. O WebSphere eXtreme Scale protege a si mesmo contra as condições de queda de energia entre os datacenters usando uma implementação de quorum.

Implementação de pulsação

Essa seção descreve como a verificação de atividade é implementada no WebSphere eXtreme Scale.

Pulsação de membro de grupo principal

O serviço de catálogo coloca as JVMs de contêiner em grupos principais de tamanho limitado. Um grupo principal tentará detectar a falha de seus membros usando dois métodos. Se um soquete de JVM for fechado, essa JVM será considerada inativa. Cada membro também efetua pulsação sobre esses soquetes a

uma taxa determinada pela configuração. Se uma JVM não responder a essas pulsações dentro de um período máximo de tempo especificado, a JVM será considerada inativa.

Um único membro de um grupo principal é sempre escolhido para ser o líder. O core group leader (CGL) é responsável por informar periodicamente ao serviço de catálogo que o grupo principal está ativo e relatar quaisquer alterações de associação no serviço de catálogo. Uma alteração de associação pode ser uma JVM com falha ou uma JVM recém incluída que se une ao grupo principal.

Se o core group leader não conseguir contatar nenhum membro da grade de serviço de catálogo, ele continuará tentando novamente.

Pulsação da grade de serviço de catálogo

O serviço de catálogo é semelhante a um grupo principal privado com associação estática e um mecanismo de quorum. Ele detecta falhas da mesma forma que um grupo principal normal. Porém, o comportamento é modificado para incluir a lógica de quorum. O serviço de catálogo também usa uma configuração de pulsação menos rigorosa.

Pulsação grupo principal

O serviço de catálogo precisa saber quando os servidores de contêineres falharão. Cada grupo principal é responsável por determinar a falha da JVM de contêiner e relatar isso para o serviço de catálogo através do core group leader. Também é possível uma falha completa de todos os membros de um grupo principal. Se o grupo principal inteiro falhar, o serviço de catálogo é responsável por detectar essa perda.

Se o serviço de catálogo marcar uma JVM de contêiner como falha e o contêiner for posteriormente relatado como ativo, a JVM de contêiner será instruída a encerrar os servidores de contêiner do WebSphere eXtreme Scale. Uma JVM nesse estado não estará visível nas consultas xsadmin. Os logs da JVM do contêiner exibirão mensagens indicando o fato ocorrido. Essas JVMs precisam ser reiniciadas manualmente.

Se ocorrer um evento de perda de quorum, a pulsação será suspensa até o quorum ser reestabelecido.

Comportamento do Quorum de Serviço de Catálogo

Normalmente, os membros do serviço de catálogo possuem conectividade completa. A grade de serviço de catálogo é um conjunto estático de JVMs. O WebSphere eXtreme Scale espera que todos os membros do serviço de catálogo sempre estejam on-line. O serviço de catálogo responderá apenas aos eventos de contêiner enquanto o serviço de catálogo possuir quorum.

Se o serviço de catálogo perder o quorum, ele aguardará até o quorum ser restabelecido. Enquanto o serviço de catálogo não possuir quorum, ele ignorará os eventos dos servidores de contêiner. Os servidores de contêiner tentarão novamente quaisquer pedidos rejeitados pelo servidor de catálogos durante esse tempo em que o WebSphere eXtreme Scale espera que o quorum seja reestabelecido.

A seguinte mensagem indica que o quorum foi perdido. Procure por essa mensagem nos logs de serviço de catálogo.

```
CWOBJ1254W: O serviço de catálogo está aguardando pelo quorum.
```

O WebSphere eXtreme Scale espera perder o quorum pelos seguintes motivos:

- Falha do membro da JVM de serviço de catálogo
- Queda de energia da rede
- perda do datacenter

Parar uma instância do servidor de catálogos usando`stopOgServer` não causa perda de quorum porque o sistema sabe que a instância do servidor parou, o que é diferente de uma falha de JVM ou queda de energia.

Perda de quorum a partir de uma falha de JVM

Uma falha do servidor de catálogos causará perda de quorum. Nesse caso, o quorum deverá ser substituído o mais rápido possível. O serviço de catálogo com falha não pode unir novamente a grade até o quorum ser substituído.

Perda de quorum a partir de uma queda de energia de rede

O WebSphere eXtreme Scale foi designado para esperar a possibilidade de quedas de energia. Uma queda de energia é quando ocorre uma perda temporária de conectividade entre os datacenters. Geralmente essa perda é temporária por natureza e a conectividade logo é restabelecida dentro de alguns segundos ou minutos. Enquanto o WebSphere eXtreme Scale tenta manter a operação normal durante uma queda de energia, uma queda de energia é considerada uma falha de evento única. Espera-se que essa falha seja corrigida e que a operação seja normalizada sem nenhuma ação necessária do WebSphere eXtreme Scale.

Uma queda de energia de longa duração pode ser classificada como um blacaute apenas sob intervenção do usuário. Substituir o quorum em uma queda de energia é necessário para que o evento seja classificado como um blecaute.

Ciclo JVM de serviço de catálogo

Se um servidor de catálogos for parado usando o comando stopOgServer, o quorum diminuirá um servidor a menos. Isso significa que os servidores restantes ainda possuem quorum. Reiniciar o servidor de catálogos retornará o quorum imediatamente para o número anterior.

Consequências de perda de quorum

Se uma JVM de contêiner tiver que falhar durante a perda de um quorum, a recuperação não ocorrerá até voltar a energia ou, no caso de um blecaute, o cliente executa um comando de substituição de quorum. O WebSphere eXtreme Scale considera um evento de perda de quorum e uma falha de contêiner como uma falha dupla, o que é raro de acontecer. Isso significa que os aplicativos podem perder o acesso de gravação dos dados que foram armazenados na JVM com falha até o quorum ser restaurado no momento em que a recuperação normal ocorrer.

Da mesma forma, se você tentar iniciar um contêiner durante um evento de perda de quorum, o contêiner não iniciará.

A conectividade completa do cliente é permitida durante a perda de quorum. Se não ocorrer nenhuma falha de contêiner ou problemas de conectividade durante o evento de perda de quorum, os clientes ainda poderão interagir completamente com os servidores de contêineres.

Se ocorrer uma queda de energia, alguns clientes poderão não ter acesso às cópias primárias ou réplicas dos dados até voltar a energia.

Novos clientes poderão ser iniciados, já que deve haver uma JVM de serviço de catálogo em cada datacenter, portanto, pelo menos uma JVM de serviço de catálogo pode ser obtida pelo cliente mesmo durante uma queda de energia.

Recuperação de quorum

Se o quorum for perdido por algum motivo, quando o quorum for reestabelecido, um protocolo de recuperação será executado. Quando ocorrer um evento de perda de quorum, toda a verificação de atividade dos grupos principais será suspensa e os relatórios de falha também serão ignorados. Quando o quorum retornar, o serviço de catálogo executará uma verificação de atividade de todos os grupos principais para determinar imediatamente a associação. Quaisquer shards anteriormente hospedados nas JVMs de contêiner relatados como falha serão recuperados nesse ponto. Se os shards primários forem perdidos, as réplicas sobreviventes serão promovidas para primárias. Se os shards de réplicas foram perdidos, réplicas adicionais serão criadas nos que sobreviveram.

Substituindo Quorum

Isso deve ser usado apenas quando ocorrer uma falha de datacenter. A perda de quorum é causada por uma falha de JVM de serviço de catálogo ou por uma queda de energia de rede, o que deverá ser recuperado automaticamente quando a JVM de serviço de catálogo for reiniciada ou a energia de rede for restabelecida.

Os administradores são os únicos que sabem quando ocorre uma falha do datacenter. O WebSphere eXtreme Scale trata uma queda de energia e um blecaute da mesma forma. É necessário informar ao ambiente do eXtreme Scale dessas falhas usando o comando xsadmin para substituir o quorum. Isso informará ao serviço de catálogo a assumir que o quorum foi obtido com a associação atual e que uma recuperação completa ocorrerá. Ao emitir um comando de substituição de quorum, você estará garantindo que as JVMs no datacenter com falha realmente falharam e não serão recuperadas.

A lista a seguir considera alguns cenários para substituição de quorum. Suponha que você possua três servidores de catálogo: A, B e C.

- Queda de energia: Suponha que ocorra uma queda de energia e o servidor C fique isolado temporariamente. O serviço de catálogo perde quorum e aguarda o término da queda de energia e, então, o servidor C une novamente a grade de serviço de catálogo e o quorum é reestabelecido. Seu aplicativo não terá problemas durante esse tempo.
- Falha temporária: Aqui, o servidor C falha e o serviço de catálogo perde o quorum, portanto, o quorum deve ser substituído.Quando o quorum for reestabelecido, o servidor C poderá ser reiniciado. O servidor C unirá novamente a grade de serviço de catálogo quando for reiniciado. O aplicativo não terá problemas durante esse tempo.
- Falha do datacenter: Verifique se o datacenter realmente falhou e se ele foi isolado na rede. Em seguida, emita o comando xsadmin de substituição de

quorum. Os dois datacenters sobreviventes executam recuperação completa ao substituir os shards que estavam hospedados no datacenters com falha. O serviço de catálogo agora está em execução com um servidor de quorum completo A e B. Pode ocorrer atrasos ou exceções no aplicativo durante o intervalo entre o início do blecaute e a substituição do quorum. Quando o quorum for substituído, a grade será recuperada e a operação normalizada.

- Recuperação do datacenter: Os datacenters sobreviventes já estão em execução com o quorum substituído. Quando o datacenter que contém o servidor C for reiniciado, todas as JVMs no datacenter deverão ser reiniciadas. Em seguida, o servidor C unirá novamente a grade de serviço de catálogo existente e o quorum voltará para o estado normal sem nenhuma intervenção do usuário.
- Falha e queda de energia do datacenter: O datacenter que contém o servidor C falha. O quorum é substituído e recuperado nos datacenters restantes. Se ocorrer uma queda de energia entre os servidores A e B, as regras de recuperação normal da queda de energia serão aplicadas. Quando voltar a energia, o quorum será restabelecido e a recuperação necessária da perda do quorum ocorrerá.

Comportamento do contêiner

Essa seção descreve como as JVMs do servidor de contêiner se comportam enquanto o quorum for perdido e recuperado.

Os contêineres hospedam um ou mais shards. Os shards são primários ou réplicas de uma partição específica. O serviço de catálogo designa shards para um contêiner e o contêiner respeitará essa designação até novas instruções chegarem a partir do serviço de catálogo. Isso significa que, se um shard primário em um contêiner não puder se comunicar com um shard réplica devido a uma queda de energia, ele continuará tentando novamente até receber novas instruções do serviço de catálogo.

Se cair a energia da rede e um shard primário perder comunicação com a réplica, ele tentará novamente a conexão até o serviço de catálogo fornecer novas instruções.

Comportamento de réplica síncrona

Enquanto a conexão estiver dividida, o shard primário pode aceitar novas transações enquanto houver, pelo menos, quantas réplicas on-line que a propriedade minsync possuir para o conjunto de mapa. Se alguma das novas transações for processada no shard primário enquanto o link para a réplica síncrona estiver quebrado, todos os dados da réplica serão excluídos e a réplica será ressincronizada com o estado atual do shard primário quando o link for reestabelecido.

A replicação síncrona é altamente não recomendada entre os datacenter ou através de um link de estilo WAN.

Comportamento de réplica assíncrona

Enquanto a conexão estiver interrompida, o shard primário pode aceitar novas transações. O shard primário armazenará em buffer as alterações até um limite. Se a conexão com a réplica for reestabelecida antes de atingir esse limite, a réplica será atualizada com as alterações em buffer. Se esse limite for atingido, o shard primário destruirá a lista armazenada em buffer e, quando a réplica for reconectada, ela será limpa e ressincronizada.

Comportamento do Cliente

Os clientes sempre podem se conectar com o servidor de catálogos para autoinicialização com a grade, independente se a grade de serviço de catálogo possuir quorum ou não. O cliente tentará se conectar com qualquer instância do servidor de catálogos para obter uma tabela de rota e, em seguida, interagir com a grade. A conectividade de rede pode impedir que o cliente interaja com algumas partições devido à configuração da rede. O cliente pode se conectar com réplicas locais para dados remotos se ele for configurado para isso.Os clientes não poderão atualizar os dados se a partição primária para esses dados não estiver disponível.

SSL desativado

É altamente recomendado especificar um tempo limite de transação no arquivo XML do descritor ObjectGrid. O cliente tentará novamente uma transação única até esse tempo limite exceder. Se um tempo limite não for especificado, o cliente tentará novamente infinitas vezes, o que não é recomendado, já que o encadeamento do cliente será bloqueado por tempo indeterminado. O tempo limite da transação deve ser configurado para um múltiplo do tempo máximo de transação esperado.

Também é altamente recomendado configurar o tempo limite do pedido ORB no arquivo orb.properties. O cliente será bloqueado em um soquete que caiu a anergia durante esse tempo máximo. Se o tempo decorrido de quando o pedido foi emitido ainda for menor que o tempo limite da transação, o WebSphere eXtreme Scale tentará o pedido novamente. O cliente tentará novamente até o tempo total decorrido exceder o tempo limite da transação. Assim, o tempo de bloqueio máximo do cliente é o tempo limite da transação mais o tempo limite do pedido do ORB.

O ORB será bloqueado até a pilha TCP fechar o soquete que está sem energia, se o tempo limite de pedido do ORB não for especificado. Esse tempo depende do ajuste da pilha TCP e pode durar horas, dependendo do TCP FIN_WAIT e de outros parâmetros relacionados.

Comandos do Quorum com xsadmin

Essa seção descreve os comandos xsadmin úteis para situações de quorum.

Consultando status de quorum

O status de quorum de uma instância do servidor de catálogos pode ser examinado usando o comando xsadmin.

```
xsadmin –ch cathost –p 1099 –quorumstatus
```

Há cinco resultados possíveis.

- Quorum está desativado: Os servidores de catálogos estão em execução em um modo de quorum desativado. Esse é um desenvolvimento ou apenas um modo de datacenter único. Ele não é recomendado para configurações de vários datacenters.
- O Quorum está ativado e o servidor de catálogos possui quorum: O quorum é ativado e o sistema trabalha normalmente.
- O Quorum está ativado mas o servidor de catálogos está aguardando quorum: O quorum está ativado mas o quorum foi perdido.

- O Quorum está ativado mas o quorum foi substituído: O quorum está ativado mas o quorum foi substituído.
- O status do quorum é declarado ilegal: Em uma queda de energia, o servidor de catálogos é dividido em duas partições, A e B. O servidor de catálogos A possui quorum substituído. A partição de rede é resolvida e o servidor na partição B é declarado ilegal, requerendo um reinício da JVM. Isso também ocorre se a JVM de catálogo na partição B for reiniciada durante a queda de energia e, em seguida, a energia voltar.

Substituindo Quorum

O comando xsadmin pode ser utilizado para substituir o quorum. Qualquer instância do servidor de catálogos sobrevivente pode ser usada. Todas as instâncias sobreviventes são notificadas quando uma for instruída para substituir o quorum. A sintaxe para isso é a seguinte.

```
xsadmin –ch cathost –p 1099 –overridequorum
```

Comandos de diagnóstico

- Status de quorum: Conforme descrito na seção anterior.
- Lista de grupo principal: Exibe uma lista de todos os grupos principais. Os membros e líderes dos grupos principais são exibidos.

  ```
  xsadmin –ch cathost –p 1099 –coregroups
  ```
- Servidores em estado teardown: Esse comando remove um servidor manualmente da grade. Normalmente isso não é necessário já que os servidores são automaticamente removidos quando forem detectados como falhas, mas o comando é fornecido para uso na ajuda de suporte IBM.

  ```
  xsadmin –ch cathost –p 1099 –g Grid –teardown server1,server2,server3
  ```
- Exibir tabela de rota: Esse comando mostra a tabela de rota atual ao simular uma nova conexão do cliente com a grade. Ele também valida a tabela de rota ao confirmar que todos os servidores de contêineres reconhecem sua função na tabela de rota, como o tipo de shard para uma determinada partição.

  ```
  xsadmin –ch cathost –p 1099 –g myGrid -routetable
  ```
- Exibir shards não-designados: Se algum shard não puder ser colocado na grade, isso poderá ser usado para listá-los. Isso acontece apenas quando o serviço de colocação possuir uma restrição que impede a colocação. Por exemplo, se você iniciar as JVMs em uma única caixa física enquanto estiver no modo de produção, apenas os shards primários poderão ser colocados. As réplicas não serão designadas até o início das JVMs em uma segunda caixa. O serviço de colocação coloca apenas as réplicas nas JVMs com endereços IP diferentes das JVMs que hospedam os shards primários. Não ter nenhuma JVM em uma zona também fazem com que os shards não sejam designados.

  ```
  xsadmin –ch cathost –p 1099 –g myGrid –unassigned
  ```
- Definir configurações de rastreio: Esse comando define as configurações de rastreio para todas as JVMs que correspondem ao filtro especificado para o comando xsadmin. Essa definição altera apenas as configurações de rastreio até outro comando ser usado ou as JVMs modificadas falharem ou pararem.

  ```
  xsadmin –ch cathost –p 1099 –g myGrid –fh host1 –settracespec
  ObjectGrid*=event=enabled
  ```

  Isso ativa o rastreio para todas as JVMs na caixa com o nome do host especificado que, neste caso, é host1.

- Verificando tamanhos de mapa: O comando de tamanhos de mapa é útil para verificar se a distribuição de chaves é uniforme entre os shards na chave. Se algum contêiner contiver significativamente mais chaves do que outros, provavelmente a função hash nos objetos de chave está distribuindo incorretamente.

  ```
  xsadmin -ch cathost -p 1099 -g myGrid -m myMapSet -mapsizes myMap
  ```

Utilizando JMS para Distribuir Alterações de Transações

Use o Java Message Service (JMS) para alterações distribuídas entre diferentes camadas ou em ambientes em plataformas mistas.

O JMS é um protocolo ideal para alterações distribuídas entre diferentes camadas ou em ambientes em plataformas mistas. Por exemplo, alguns aplicativos que usam o eXtreme Scale podem ser implementados noIBM WebSphere Application Server Community Edition, Apache Geronimo ou Apache Tomcat, considerando que outros aplicativos podem executar no WebSphere Application Server Versão 6.x. O JMS é ideal para alterações distribuídas entre peers do eXtreme Scale nesses diferentes ambientes. O transporte de mensagens do gerenciador de alta disponibilidade é muito rápido, mas pode apenas distribuir alterações para as Java Virtual Machines que estão em um grupo principal único. O JMS é mais lento, mas permite que conjuntos maiores e mais diversos de aplicativos clientes compartilhem um ObjectGrid. O JMS é ideal no compartilhamento de dados em um ObjectGrid entre um cliente Swing rápido e um aplicativo implementado no WebSphere Extended Deployment.

O Mecanismo de Invalidação do Cliente e a Replicação Ponto a Ponto incorporados são exemplos da distribuição de alterações transacionais com base no JMS. Consulte as informações sobre a configuração da replicação ponto a ponto com o JMS no *Guia de Administração* para obter mais informações.

Implementando o JMS

O JMS é implementado para distribuir alterações de transação usando um objeto Java que se comporta como um ObjectGridEventListener. Este objeto pode propagar o estado nas quatro maneiras a seguir:

1. Invalidar: Qualquer entrada que é despejada, atualizada ou excluída é removida em todas as Java Virtual Machines peer quando elas recebem a mensagem.
2. Invalidar condicional: A entrada é despejada somente se a versão local for a mesma ou mais antiga que a versão no publicador.
3. Push: Qualquer entrada que foi despejada, atualizada, excluída ou inserida é incluída ou sobrescrita em todas as Java Virtual Machines peer quando elas recebem a mensagem JMS.
4. Push condicional: A entrada é atualizada ou incluída no lado de recebimento apenas se a entrada local for menos recente que a versão que está sendo publicada.

Atender Alterações de Publicação

O plug-in implementa a interface ObjectGridEventListener para interceptar o evento transactionEnd. Quando o eXtreme Scale chama este método, o plug-in tenta converter a lista LogSequence list para cada mapa que é acessado pela transação em uma mensagem JMS e, então, a publica. O plug-in pode ser configurado para publicar alterações para todos os mapas ou um subconjunto de

mapas. Os objetos LogSequence são processados para os mapas com a publicação ativada. A classe LogSequenceTransformer do ObjectGrid serializa um LogSequence filtrado para cada mapa em um fluxo. Após todas as LogSequences serem serializadas para o fluxo, então, um ObjectMessage JMS é criado e publicado em um tópico bem conhecido.

Atender Mensagens JMS e Aplicá-las ao ObjectGrid Local

Guia de Administração

O mesmo plug-in também inicia um encadeamento que gira em um loop, recebendo todas as mensagens publicadas no tópico bem conhecido. Quando chega uma mensagem, ele transmite o conteúdo da mensagem para a classe LogSequenceTransformer para convertê-lo em um conjunto de objetos LogSequence. Em seguida, uma transação não-write-through é iniciada. Cada objeto LogSequence é fornecido ao método Session.processLogSequence, que atualiza os Mapas locais com as alterações. O método processLogSequence entende o modo de distribuição. A transação é confirmada e o cache local agora reflete as alterações. Para obter informações adicionais sobre a utilização do JMS para distribuir alterações de transações, consulte as informações sobre a distribuição das alterações entre as Java Virtual Machines do peer no *Guia de Administração*.

Capítulo 6. Segurança

O WebSphere eXtreme Scale é um sistema de armazenamento em cache distribuído. Se você deseja proteger o acesso para proteger seus dados, é possível ativar a segurança e integrar-se com provedores de segurança externos.

Nota: Em um armazenamento de dados fora do cache existente, como um banco de dados, provavelmente é necessário ter recursos de segurança integrados que podem não ser necessários para configuração ou ativação de modo ativo. Entretanto, após ter armazenado seus dados em cache com o eXtreme Scale, você deve considerar a importante situação resultante de que seus recursos de segurança de backend não estão mais em vigor. É possível configurar a segurança do eXtreme Scale no níveis necessários para que a nova arquitetura armazenada em cache para os seus dados também fique segura.
A seguir é apresentado um breve resumo sobre os recursos de segurança do eXtreme Scale. Para obter mais informações detalhadas sobre como configurar a segurança, consulte o *Guia de Administração* e o *Guia de Programação*.

Fundamentos sobre Segurança Distribuída

A segurança distribuída do eXtreme Scale é baseada em três conceitos fundamentais:

Autenticação confiável
: A habilidade de determinar a identidade do solicitante. O WebSphere eXtreme Scale suporta autenticação cliente-para-servidor e servidor-para-servidor.

Autorização
: A habilidade de dar permissões para conceder direitos de acesso ao solicitante. O WebSphere eXtreme Scale suporta diferentes autorizações para várias operações.

Transporte Seguro
: A transmissão segura dos dados sobre uma rede. O WebSphere eXtreme Scale suporta os protocolos TLS/SSL (Transport Layer Security/Secure Sockets Layer).

Autenticação

O WebSphere eXtreme Scale suporta uma estrutura de cliente e servidor distribuída. Uma infraestrutura de segurança de cliente e servidor está estabelecida para proteger o acesso aos servidores eXtreme Scale. Por exemplo, quando a autenticação é necessária pelo servidor do eXtreme Scale, um cliente do eXtreme Scale deve fornecer credenciais para se autenticar no servidor. Essas credenciais podem ser um par de nome de usuário e senha, um certificado cliente, um ticket Kerberos ou dados que são apresentados em um formato acordado entre o cliente e o servidor.

Autorização

As autorizações do WebSphere eXtreme Scale são baseadas em assuntos e permissões. É possível utilizar o Java Authentication and Authorization Services (JAAS) para autorizar o acesso ou é possível conectar uma abordagem

customizada, tal como Tivoli Access Manager (TAM), para tratar as autorizações. As seguintes autorizações podem ser fornecidas a um cliente ou grupo:

Autorização de mapa
Execute operações insert, read, update, evict ou delete nos Mapas.

Autorização do ObjectGrid
Execute consultas em objetos ou entidades e consultas em fluxos nos objetos do ObjectGrid.

Autorização do agente do DataGrid
Permita que os agentes do DataGrid sejam implementados em um ObjectGrid.

Autorização do mapa do lado do servidor
Replique um mapa de servidor para o lado do cliente ou crie um índice dinâmico para o mapa do servidor.

Autorização de administração
Execute tarefas de administração.

Segurança do Transporte

Para garantir a segurança da comunicação entre cliente e o servidor, o WebSphere eXtreme Scale suporta TLS/SSL. Estes protocolos fornecem segurança da camada de transporte com autenticidade, integridade e confidencialidade para uma conexão segura entre um cliente e um servidor do eXtreme Scale.

Segurança da Grade

Em um ambiente seguro, um servidor deve poder verificar a autenticidade de outro servidor. O WebSphere eXtreme Scale utiliza um mecanismo de cadeia de chave secreta compartilhado para este propósito. Este mecanismo de chave secreta é semelhante a uma senha compartilhada. Todos os servidores eXtreme Scale aceitam uma cadeia secreta compartilhada. Quando um servidor se junta à grade, ele é desafiado a apresentar a cadeia secreta. Se a cadeia secreta do servidor que está se juntando corresponder a uma cadeia no servidor principal, então o servidor que está se juntando pode ser unir á grade. Caso contrário, o pedido de junção será rejeitado.

Não é seguro enviar um segredo em texto não-criptografado. A infraestrutura de segurança do eXtreme Scale fornece um plug-in SecureTokenManager para possibilitar que o servidor faça a segurança deste segredo antes de enviá-lo. É possível escolher como implementar a operação segura. O WebSphere eXtreme Scale fornece uma implementação, na qual a operação segura é implementada para criptografar e assinar o segredo.

Segurança Java Management Extensions (JMX) em uma Topologia de Implementação Dinâmica

A segurança JMX MBean é suportada em todas as versões do eXtreme Scale. Clientes dos MBeans do servidor de catálogos e MBeans do servidor de contêineres podem ser autenticados e o acesso às operações do MBean podem ser impostos.

Segurança Local do eXtreme Scale

A segurança local do eXtreme Scale é diferente do modelo distribuído do eXtreme Scale porque o aplicativo instancia diretamente e utiliza uma instância do

ObjectGrid. Seu aplicativo e as instâncias do eXtreme Scale estão na mesma Java virtual machine (JVM). Como não há nenhum conceito de cliente/servidor neste modelo, a autenticação não é suportada. Seu aplicativo deve gerenciar sua própria autenticação e, então, passar o objeto Subject autenticado para o eXtreme Scale. Porém, o mecanismo de autorização usado para o modelo de programação do eXtreme Scale local é o mesmo que o usado para o modelo cliente/servidor.

Configuração e Programação

Para obter informações adicionais sobre a configuração e programação para segurança, consulte o *Guia de Administração* e *Guia de Programação*.

Capítulo 7. Processamento de Transações

O WebSphere eXtreme Scale usa transações de acordo com seu mecanismo de interação com os dados.

Sessões e Transações

Para interagir com os dados, o encadeamento em seu aplicativo precisa de sua própria Sessão. Quando o aplicativo desejar usar o ObjectGrid em um encadeamento, chame um dos métodos ObjectGrid.getSession para obter um encadeamento. Com a sessão, o aplicativo pode trabalhar com dados que são armazenados nos mapas ObjectGrid.

Quando um aplicativo usa um objeto de Sessão, a sessão deve estar no contexto de uma transação. Uma transação inicia e é consolidada ou inicia e é recuperada usando os métodos begin, commit e rollback no objeto de Sessão. Os aplicativos também podem trabalhar em modo de auto-consolidação, no qual a Sessão inicia e consolida automaticamente uma transação sempre que uma operação é executada no mapa. O modo de auto-confirmação não pode agrupar várias operações em uma única transação, assim, ele é a opção mais lenta se você estiver criando um lote de várias operações em uma única transação. Porém, para transações que contêm uma operação, a auto-consolidação é a opção mais rápida.

Vantagens das Transações

Usando as transações, você pode:

- Recuperar alterações se ocorrer uma exceção ou a lógica de negócios precisar desfazer mudanças de estado.
- Mantém e libera bloqueios em dados para aplicar múltiplas alterações como uma unidade atômica no momento da consolidação.
- Protege um encadeamento de alterações concorrentes.
- Implementa um ciclo de vida para bloqueios nas alterações.
- Produz uma unidade atômica de replicação.

Transações

As transações possuem muitas vantagens para o armazenamento e a manipulação de dados. É possível usar transações para se proteger a grade contra alterações simultâneas, aplicar várias alterações como uma unidade simultânea, replicar dados e para implementar um ciclo de vida para bloqueios sobre alterações.

Visão Geral da Transação

Utilize as transações pelos seguintes motivos:

- Para proteger um encadeamento de alterações simultâneas.
- Para aplicar várias alterações como uma unidade atômica no momento commit.
- Para implementar um ciclo de vida para bloqueios em alterações.
- Para atuar como a unidade de replicação.

Quando uma transação inicia, o WebSphere eXtreme Scale aloca um mapa de diferença especial para conter as alterações atuais ou cópias dos pares chave e valor que a transação utiliza. Normalmente, quando um par de chave e valor é acessado, o valor é copiado antes de o aplicativo receber o valor. O mapa de diferenças controla todas as alterações de operações, como inserir, atualizar, obter, remover e assim por diante. As chaves não são copiadas porque elas são assumidas como imutáveis. Se um objeto ObjectTransformer for especificado, então, ele será utilizado para copiar o valor. Se a transação estiver utilizando o bloqueio optimistic, as imagens anteriores dos valores também serão rastreadas para comparação quando a transação for confirmada.

Se uma transação for recuperada, as informações do mapa de diferenças serão descartadas e os bloqueios nas entradas serão liberados. Quando uma transação é consolidada, as alterações são aplicadas nos mapas e os bloqueios são liberados. Se o bloqueio otimista estiver sendo utilizado, o eXtreme Scale compara as versões de imagens anteriores dos valores com os valores que estão no mapa. Esses valores devem corresponder para que a transação seja confirmada. Essa comparação permite um esquema de bloqueio de várias versões, mas a um custo de duas cópias sendo feitas quando a transação acessa a entrada. Todos os valores são copiados novamente e a nova cópia é armazenada no mapa. O WebSphere eXtreme Scale executa esta cópia para se proteger do aplicativo alterando a referência do aplicativo para o valor após um commit.

É possível evitar o uso de diversas cópias das informações. O aplicativo pode salvar uma cópia, utilizando o bloqueio pessimistic em vez do bloqueio optimistic como o custo da limitação da simultaneidade. A cópia do valor no momento da confirmação também pode ser evitada se o aplicativo concordar em não alterar um valor após uma confirmação.

Tamanho da Transação

Transações maiores são mais eficientes, especificamente para replicação. No entanto, as transações maiores podem causar impacto adverso na simultaneidade porque os bloqueios nas entradas são retidos por um período maior de tempo. Se você usar transações maiores, é possível aumentar o desempenho de replicação. Este aumento de desempenho é importante quando você estiver pré-carregando um Mapa. Experimente diferentes tipos de batch para determinar qual funciona melhor para o seu cenário.

Transações maiores também ajudam com os utilitários de carga. Se estiver sendo usado um utilitário de carga que possa executar SQL em lote, então ganhos consideráveis no desempenho são possíveis dependendo da transação e de reduções significativas de carga no lado do banco de dados. Esse ganho no desempenho depende da implementação do Carregador.

Atributo CopyMode

É possível ajustar o número de cópias definindo o atributo CopyMode do BackingMap ou objetos ObjectMap. O modo de cópia possui os seguintes valores:

- COPY_ON_READ_AND_COMMIT
- COPY_ON_READ
- NO_COPY
- COPY_ON_WRITE
- COPY_TO_BYTES

O valor COPY_ON_READ_AND_COMMIT é o padrão. O valor COPY_ON_READ copia os dados iniciais recuperados, mas não copia no momento da consolidação. Este modo é seguro se o aplicativo não modificar um valor depois de consolidar uma transação. O valor NO_COPY não copia os dados, que são seguros apenas para dados de leitura. Se ele nunca for alterado, não será necessário copiá-lo por motivos de isolamento.

Seja cauteloso ao usar o valor do atributo NO_COPY com mapas que possam ser atualizados. O WebSphere eXtreme Scale utiliza a cópia no primeiro acesso para permitir o retrocesso da transação. O aplicativo alterou apenas a cópia e, como resultado, o eXtreme Scale descarta a cópia. Se o valor de atributo NO_COPY for utilizado e o aplicativo modificar o valor confirmado, não será possível concluir a recuperação. Modificar o valor confirmado conduz a problemas nos índices, replicação e assim por diante porque os índices e as réplicas são atualizadas quando a transação é confirmada. Se você modificar os dados confirmados e, em seguida, recuperar a transação, o que na realidade não é recuperada, os índices não serão atualizados e a replicação não ocorrerá. Os outros encadeamentos podem ver as alterações não confirmadas imediatamente, mesmo se tiverem bloqueios. Utilize o valor de atributo NO_COPY apenas para mapas somente leitura ou para aplicativos que concluem a cópia apropriada antes de modificar o valor. Se você utilizar o valor de atributo NO_COPY e chamar o suporte IBM com um problema de integridade de dados, será necessário reproduzir o problema com o modo de cópia definido como COPY_ON_READ_AND_COMMIT.

O valor COPY_TO_BYTES armazena os valores no mapa de maneira serializada. No tempo de leitura, o eXtreme Scale aumenta o valor de um formato serializado e, no tempo de consolidação, ele armazena o valor em um formato serializado. Com esse método, uma cópia é feita no tempo de leitura e de consolidação.

O modo de cópia padrão para um mapa pode ser configurado no objeto BackingMap. Também é possível alterar o modo de cópia antes de iniciar uma transação usando o método ObjectMap.setCopyMode.

A seguir há um exemplo de um fragmento de mapa de apoio de um arquivo `objectgrid.xml` que mostra como configurar o modo de cópia para um determinado mapa de apoio. Este exemplo assume que você esteja utilizando `cc` como o espaço de nomes `objectgrid/config`.

```
<cc:backingMap name="RuntimeLifespan" copyMode="NO_COPY"/>
```

Consulte as informações sobre as boas práticas do método copyMode no *Guia de Programação* para obter mais informações.

Modo de Commit Automático

Se nenhuma transação for ativamente iniciada, então quando um aplicativo interage com um objeto ObjectMap, uma operação automática é iniciada e uma consolidação é executada em nome do aplicativo. Esta operação automática de início e consolidação funciona, mas evita que a recuperação e o bloqueio funcionem efetivamente. A velocidade de replicação síncrona sofre um impacto devido ao tamanho de transação muito reduzido. Se estiver usando um aplicativo gerenciador de entidades, então não use o modo de consolidação automática pois os objetos que estiverem bloqueados com o método EntityManager.find se tornarão imediatamente não gerenciados no retorno do método e inutilizáveis.

Modo de Bloqueio e Transações

Os bloqueios são limitados pelas transações. É possível especificar as seguintes configurações de bloqueio:

- **Ausência de bloqueio**: Executar sem a configuração de bloqueio é a opção mais rápida. Se estiver utilizando dados de leitura, você talvez não precise do bloqueio.
- **Bloqueio pessimistic**: Adquire bloqueios em entrada e, em seguida, contém o bloqueio até o momento do commit. Essa estratégia de bloqueio fornece boa consistência à custa do rendimento.
- **Bloqueio optimistic**: Obtém uma imagem anterior de cada registro que a transação acessa e compara a imagem com os valores de entrada atuais quando ocorre o commit da transação. Se os valores de entrada forem alterados, a transação será recuperada. Nenhum bloqueio será retido até o momento da confirmação. Esta estratégia de bloqueio fornece melhor simultaneidade do que a estratégia pessimista, no risco da transação sendo recuperada e no custo da memória de criar a cópia extra da entrada.

Configure a estratégia de bloqueio no BackingMap. Não é possível alterar a estratégia de bloqueio para cada transação. A seguir há um exemplo de fragmento XML que mostra como configurar o modo de bloqueio em um mapa utilizando o arquivo XML, assumindo que `cc` é o espaço de nomes para o espaço de nomes `objectgrid/config`.

```
<cc:backingMap name="RuntimeLifespan" lockStrategy="PESSIMISTIC" />
```

Coordenadores de Transação Externos

Normalmente, as transações iniciam com o método session.begin e terminam com o método session.commit. Porém, quando o eXtreme Scale está incorporado, as transações podem ser iniciadas e encerradas por um coordenador externo de transações. Se você estiver usando um coordenador de transação externo, não é necessário chamar o método session.begin e terminar com o método session.commit. Consulte o *Guia de Programação* para obter informações adicionais sobre o eXtreme Scale e a interação com transações externas. Se você estiver usando o WebSphere Application Server, é possível usar o plug-in WebSphereTranscationCallback. Consulte o *Guia de Programação* para obter informações adicionais sobre os plug-ins que estão disponíveis com o WebSphere eXtreme Scale.

Estratégias de Bloqueio

As estratégias de bloqueio incluem pessimistic, optimistic e ausência de bloqueio. Para escolher uma estratégia de bloqueio, é necessário saber a porcentagem de cada tipo de operações que possui, se você utiliza um utilitário de carga e assim por diante.

Bloqueio Pessimista

Utilize a estratégia de bloqueio pessimistic para mapas de leitura e gravação quando outras estratégias de bloqueio não são possíveis. Quando um mapa do ObjectGrid map é configurado para utilizar a estratégia de bloqueio pessimista, um bloqueio de transação pessimista para uma entrada de mapa é obtido quando uma transação primeiro obtém a entrada do BackingMap. O bloqueio pessimistic fica retido até que o aplicativo conclua a transação. Geralmente, a estratégia de bloqueio pessimistic é utilizada nas seguintes situações:

- Quando o BackingMap é configurado com ou sem um utilitário de carga e as informações de controle de versões não estão disponíveis.
- Quando o BackingMap é utilizado diretamente por um aplicativo que precisa de ajuda do eXtreme Scale para controle de simultaneidade.
- Quando as informações de controle de versões estão disponíveis, mas as transações de atualização colidem freqüentemente nas entradas de suporte, resultando em falhas de atualização otimistas.

Como a estratégia de bloqueio pessimista tem o maior impacto no desempenho e escalabilidade, esta estratégia deve ser utilizada apenas para ler e gravar mapas quando outras estratégias de bloqueio não são viáveis. Por exemplo, essas situações incluem quando ocorrem falhas de atualização otimista com frequência ou quando a recuperação da falha otimista é difícil para um aplicativo manipular.

Bloqueio Otimista

A estratégia de bloqueio optimistic assume que duas transações podem tentar atualizar a mesma entrada de mapa ao executar simultaneamente. Devido a esta convicção, o modo de bloqueio não precisa ser retido pelo ciclo de vida da transação porque é improvável que mais de uma transação possa atualizar a entrada de mapa simultaneamente. A estratégia de bloqueio optimistic geralmente é utilizada nas seguintes situações:

- Quando um BackingMap é configurado com ou sem um utilitário de carga e as informações de controle de versões estão disponíveis.
- Quando um BackingMap possui em sua maior parte, transações que executam operações de leitura. As operações insert, update ou remove nas entradas de mapa não ocorrem com frequência no BackingMap.
- Quando um BackingMap é inserido, atualizado ou removido mais freqüentemente do que é lido, mas as transações raramente colidem na mesma entrada do mapa.

Como a estratégia de bloqueio pessimistic, o métodos na interface ObjectMap determinam como o eXtreme Scale automaticamente tenta adquirir um modo de bloqueio para a entrada de mapa que está sendo acessada. Entretanto, existem as seguintes diferenças entre as estratégias pessimistic e optimistic:

- Como a estratégia de bloqueio pessimistic, um modo de bloqueio S é adquirido pelos métodos get e getAll quando o método é chamado. No entanto, com o bloqueio optimistic, o modo de bloqueio S não fica retido até que a transação seja concluída. Em vez disso, o modo de bloqueio S é liberado antes de o método retornar ao aplicativo. O propósito de adquirir o modo de bloqueio é para que o eXtreme Scale possa garantir que apenas dados com commit de outras transações fiquem visíveis para a transação atual. Após o eXtreme Scale ter verificado que ocorreu commit nos dados, o modo de bloqueio S é liberado. No momento do commit, uma verificação de versão optimistic é executada para garantir que nenhuma outra transação tenha alterado a entrada do mapa após a transação atual ter liberado seu modo de bloqueio S. Se uma entrada não for procurada a partir do mapa antes de ser atualizada, invalidada ou excluída, o tempo de execução do eXtreme Scale implicitamente procura a entrada a partir do mapa. Esta operação get implícita é desempenhada para obter o valor atual no momento em que foi solicitada a modificação da entrada.
- Diferente da estratégia de bloqueio pessimista, os métodos getForUpdate e getAllForUpdate são tratados exatamente como os métodos get e getAll quando a estratégia de bloqueio otimista é utilizada. Ou seja, um modo de bloqueio S é adquirido no início do método e o modo de bloqueio S é liberado antes de retornar para o aplicativo.

Todos os outros métodos ObjectMap são tratados exatamente como são tratados para a estratégia de bloqueio pessimistic. Ou seja, quando o método commit é chamado, um modo de bloqueio X é obtido para qualquer entrada do mapa que tenha sido inserida, atualizada, removida, tocada ou invalidada e o modo de bloqueio X é retido até que a transação tenha concluído o processamento de consolidação.

A estratégia de bloqueio optimistic assume que nenhuma transação em execução simultânea tenta atualizar a mesma entrada de mapa. Devido a esta suposição, o modo de bloqueio não precisa ser mantido pela duração da transação porque é improvável que mais de uma transação possa atualizar a entrada de mapa simultaneamente. Entretanto, como um modo de bloqueio não foi mantido, outra transação simultânea poderia potencialmente atualizar a entrada do mapa após a transação atual ter liberado seu modo de bloqueio S.

Para tratar esta possibilidade, o eXtreme Scale obtém um bloqueio X no momento do commit e executa uma verificação de versão optimistic para verificar se nenhuma outra transação alterou a entrada do mapa após a transação atual ter lido a entrada do mapa a partir do BackingMap. Se outra transação alterar a entrada do mapa, a verificação de versão falhará e ocorrerá uma exceção OptimisticCollisionException. Esta exceção força a transação atual a ser retrocedida e o aplicativo deve tentar novamente a transação inteira. A estratégia de bloqueio optimistic é muito útil quando um mapa é lido em sua maior parte e é improvável que ocorram atualizações na mesma entrada do mapa.

Ausência de Bloqueio

Quando um BackingMap é configurado para não utilizar uma estratégia de bloqueio, nenhum bloqueio de transação para uma entrada de mapa será obtido.

Não utilizar uma estratégia de bloqueio é útil quando um aplicativo é um gerenciador de persistência, como um contêiner Enterprise JavaBeans™ (EJB) ou quando um aplicativo utiliza Hibernate para obter dados persistentes. Neste cenário, o BackingMap é configurado sem um utilitário de carga e o gerenciador de persistência utiliza o BackingMap como um cache de dados. Neste cenário, o gerenciador de persistência fornece controle de simultaneidade entre transações que estão acessando as mesmas entradas de Mapa.

O WebSphere eXtreme Scale não precisa obter nenhum bloqueio de transação para o propósito de controle de simultaneidade. Essa situação presume que o gerenciador de persistência não libera os bloqueios da transação antes de atualizar o mapa ObjectGrid com as alterações confirmadas. Se o gerenciador de persistência libera seus bloqueios, então uma estratégia de bloqueio pessimistic ou optimistic deve ser utilizada. Por exemplo, suponha que o gerenciador de persistência de um contêiner EJB esteja atualizando o mapa do ObjectGrid com dados que foram confirmados na transação gerenciada por contêiner de EJB. Se a atualização do mapa do ObjectGrid ocorrer antes dos bloqueios de transação do gerenciador de persistência serem liberados, então é possível não utilizar nenhuma estratégia de bloqueio. Se o mapa do ObjectGrid ocorrer após os bloqueios de transação do gerenciador de persistência serem liberados, será necessário utilizar a estratégia de bloqueio otimista ou pessimista.

Outro cenário onde a ausência de estratégia de bloqueio pode ser utilizada é quando o aplicativo utiliza um BackingMap diretamente e um Utilitário de Carga é configurado para o mapa. Neste cenário, o utilitário de carga utiliza o suporte de controle de simultaneidade que é fornecido por um Relational Database

Management System (RDBMS) utilizando Java Database Connectivity (JDBC) ou Hibernate para acessar dados em um banco de dados relacional. A implementação do utilitário de carga pode utilizar uma abordagem optimistic ou pessimistic. Um utilitário de carga que utiliza um bloqueio optimistic ou uma abordagem de controle de versões ajuda a obter a maior quantidade de simultaneidade e desempenho. Para obter informações adicionais sobre a implementação de uma abordagem de bloqueio optimistic, consulte a seção OptimisticCallback em as informações sobre as considerações do utilitário de carga no *Guia de Administração*. Se estiver usando um utilitário de carga que usa suporte de bloqueio pessimistic de um backend subjacente, você pode querer usar o parâmetro forUpdate que é transmitido no método get da interface do utilitário de carga. Configure este parâmetro como true se o método getForUpdate da interface ObjectMap foi utilizado pelo aplicativo para obter os dados. O utilitário de carga pode utilizar esse parâmetro para determinar se solicitará um bloqueio atualizável na linha que está sendo lida. Por exemplo, o DB2 obtém um bloqueio atualizável quando uma instrução select SQL contém uma cláusula `FOR UPDATE`. Esta abordagem oferece a mesma prevenção de conflito que está descrita em "Bloqueio Pessimista" na página 138.

Capítulo 8. Tutoriais

Você pode usar tutoriais para iniciar o uso de funções específicas do WebSphere eXtreme Scale.

Tutorial do Entity Manager: Visão Geral

Este tutorial do gerenciador de entidade mostra como utilizar o WebSphere eXtreme Scale para armazenar informações de pedido em um Web site. É possível criar um aplicativo Java Platform, Standard Edition 5 simples que utiliza um eXtreme Scale local e de memória. As entidades utilizam anotações e genéricos do Java SE 5.

Antes de Iniciar

Certifique-se de atender aos seguintes requisitos antes de começar o tutorial:

- É necessário ter o Java SE 5.
- É necessário ter o arquivo `objectgrid.jar` em seu caminho de classe.

Tutorial do Entity Manager: Criando uma Classe de Entidade

A primeira etapa do tutorial do gerenciador de entidade mostra como criar um ObjectGrid local com uma entidade ao criar uma classe Entity, registrar o tipo da entidade com o eXtreme Scale e armazenar uma instância da entidade no cache.

Por Que e Quando Desempenhar Esta Tarefa

1. Crie o objeto Order. Para identificar o objeto como uma entidade ObjectGrid, inclua a anotação @Entity. Ao incluir esta anotação, todos os atributos serializáveis no objeto são automaticamente persistidos no eXtreme Scale, a menos que você utilize anotações nos atributos para substituí-los. O atributo orderNumber é anotado com @Id para indicar que este atributo é a chave primária. A seguir, está um exemplo de um objeto Order:

   Order.java

   ```
   @Entity
   public class Order {
       @Id String orderNumber;
       Date date;
       String customerName;
       String itemName;
       int quantity;
       double price;
   }
   ```

2. Execute o aplicativo eXtreme Scale Hello World para demonstrar as operações entity. O programa de exemplo a seguir pode ser emitido no modo independente para demonstrar as operações entity. Use esse programa em um projeto Eclipse Java que tenha o arquivo `objectgrid.jar` incluído no caminho de classe. A seguir, está um exemplo de um aplicativo Hello world simples que utiliza o eXtreme Scale:

   Application.java

   ```
   package emtutorial.basic.step1;

   import com.ibm.websphere.objectgrid.ObjectGrid;
   import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
   import com.ibm.websphere.objectgrid.Session;
   import com.ibm.websphere.objectgrid.em.EntityManager;
   ```

```
public class Application
{

    static public void main(String [] args)
        throws Exception
    {
        ObjectGrid og =
    ObjectGridManagerFactory.getObjectGridManager().createObjectGrid();
        og.registerEntities(new Class[] {Order.class});

        Session s = og.getSession();
        EntityManager em = s.getEntityManager();

        em.getTransaction().begin();

        Order o = new Order();
        o.customerName = "John Smith";
        o.date = new java.util.Date(System.currentTimeMillis());
        o.itemName = "Widget";
        o.orderNumber = "1";
        o.price = 99.99;
        o.quantity = 1;

        em.persist(o);
        em.getTransaction().commit();

        em.getTransaction().begin();
        o = (Order)em.find(Order.class, "1");
        System.out.println("Found order for customer: " + o.customerName);
        em.getTransaction().commit();
    }
}
```

Este aplicativo de exemplo executa as seguintes operações:

a. Inicializa um eXtreme Scale local com um nome gerado automaticamente.
b. Registra as classes entity com o aplicativo utilizando a API do registerEntities, embora utilizar a API do registerEntities não seja sempre necessário.
c. Recupera um objeto Session e uma referência para o entity manager para Session.
d. Associa cada objeto Session do eXtreme Scale com um EntityManager e EntityTransaction únicos. O EntityManager agora é utilizado.
e. O método registerEntities cria um objeto BackingMap que é chamado Order e associa os metadados para o objeto Order com o objeto BackingMap. Esses metadados incluem os atributos chave e não-chave, juntamente com os tipos e nomes de atributo.
f. Uma transação inicia e cria uma instância Order. A transação é preenchida com alguns valores e é persistida utilizando o método EntityManager.persist, que identifica a entidade como aguardando para ser incluída no Mapa ObjectGrid associado.
g. A transação é, então, confirmada e a entidade é incluída no ObjectMap.
h. Outra transação é feita e o objeto Order é recuperado utilizando a chave 1. O cast de tipo no método EntityManager.find é necessário porque os recursos genéricos do Java SE 5 não são utilizados para garantir que o arquivo `objectgrid.jar` funciona em um Java SE 1.4 e Java Virtual Machine posterior.

Tutorial do Entity Manager: Formando Relacionamentos de Entidades

Crie um relacionamento simples entre entidades criando duas classes de entidades com um relacionamento, registrando as entidades com o ObjectGrid e armazenamento as instâncias da entidade no cache.

1. Crie uma entidade do cliente, que é utilizada para armazenar detalhes do cliente independente do objeto Order. A seguir, está um exemplo da entidade do cliente:

Customer.java

```
@Entity
public class Customer
{
    @Id String id;
    String firstName;
    String surname;
    String address;
    String phoneNumber;
}
```

Esta classe inclui informações sobre o cliente, tais como nome, endereço e número de telefone.

2. Crie o objeto Order, que é semelhante ao objeto Order no tópico "Tutorial do Entity Manager: Criando uma Classe de Entidade" na página 143. A seguir, está um exemplo do objeto order:

   Order.java

   ```
   @Entity
   public class Order {
       @Id String orderNumber;
       Date date;
       @ManyToOne(cascade=CascadeType.PERSIST) Customer customer;
       String itemName;
       int quantity;
       double price;
   }
   ```

   Neste exemplo, uma referência para um objeto Customer substitui o atributo customerName. A referência possui uma anotação que indica uma relação muitos-para-um. Um relacionamento muitos-para-um indica que cada pedido possui um cliente, mas vários pedidos podem fazer referência ao mesmo cliente. O modificador de anotação em cascata indica que, se o EntityManager persistir o objeto Order, também deve persistir o objeto Customer. Se você decidir não definir a opção de persistência em cascata, que é a opção padrão, deve persistir manualmente o objeto Customer com o objeto Order.

3. Utilizando as entidades, defina os mapas para a instância do ObjectGrid. Cada mapa é definido para uma entidade específica e uma entidade é denominada Order e a outra é denominada Customer. O aplicativo de exemplo a seguir ilustra como armazenar e recuperar um pedido do cliente:

Application.java

```
public class Application
{
    static public void main(String [] args)
        throws Exception
    {
        ObjectGrid og =
     ObjectGridManagerFactory.getObjectGridManager().createObjectGrid();
        og.registerEntities(new Class[] {Order.class});

        Session s = og.getSession();
        EntityManager em = s.getEntityManager();

        em.getTransaction().begin();

        Customer cust = new Customer();
        cust.address = "Main Street";
        cust.firstName = "John";
        cust.surname = "Smith";
        cust.id = "C001";
        cust.phoneNumber = "5555551212";

        Order o = new Order();
        o.customer = cust;
        o.date = new java.util.Date();
        o.itemName = "Widget";
        o.orderNumber = "1";
        o.price = 99.99;
```

```
        o.quantity = 1;

        em.persist(o);
        em.getTransaction().commit();

        em.getTransaction().begin();
        o = (Order)em.find(Order.class, "1");
        System.out.println("Found order for customer: "
     + o.customer.firstName + " " + o.customer.surname);
        em.getTransaction().commit();
    }
}
```

Este aplicativo é semelhante ao aplicativo de exemplo que está na etapa anterior. No exemplo anterior, apenas uma única classe Order é registrada. O WebSphere eXtreme Scale detecta e automaticamente inclui a referência na entidade Customer e uma instância Customer para `John Smith` é criada e referenciada a partir do novo objeto Order. Como resultado, o novo cliente é persistido automaticamente, porque o relacionamento entre duas ordens inclui o modificador em cascata, que requer que cada objeto seja persistido. Quando o objeto Order é localizado, o entity manager automaticamente localiza o objeto Customer associado e insere uma referência no objeto.

Tutorial do Entity Manager: Esquema da Entidade Order

Crie quatro classes de entidade utilizando relacionamentos únicos e bidirecionais, listas ordenadas e relacionamentos de chave estrangeira. As APIs do EntityManager são utilizadas para persistir e localizar as entidades. Com base nas entidades Order e Customer que estão nas partes anteriores do tutorial, esta etapa do tutorial inclui mais duas entidades: as entidades Item e OrderLine.

Por Que e Quando Desempenhar Esta Tarefa

Figura 40. Esquema da Entidade Order. Uma entidade Order possui uma referência para um cliente e zero ou mais OrderLines. Cada entidade OrderLine possui uma referência para um único item e inclui a quantidade solicitada.

1. Crie a entidade customer, que é semelhante aos exemplos anteriores.

   Customer.java
   ```
   @Entity
   public class Customer
   {
       @Id String id;
       String firstName;
       String surname;
       String address;
       String phoneNumber;
   }
   ```

2. Crie a entidade Item, que contém informações sobre um produto que está incluído no inventário da loja, como a descrição do produto, a quantidade e o preço.

   Item.java
   ```
   @Entity
   public class Item
   {
       @Id String id;
       String description;
       long quantityOnHand;
       double price;
   }
   ```

3. Crie a entidade OrderLine. Cada Order possui zero ou mais OrderLines, que identificam a quantidade de cada item no pedido. A chave para a OrderLine é uma chave composta que consiste no Order que possui o OrderLine e um

número inteiro que designa um número para a linha do pedido. Inclua o modificador de persistência em cascata em cada relacionamento em suas entidades.

OrderLine.java

```
@Entity
public class OrderLine
{
    @Id @ManyToOne(cascade=CascadeType.PERSIST) Order order;
    @Id int lineNumber;
    @OneToOne(cascade=CascadeType.PERSIST) Item item;
    int quantity;
    double price;
}
```

4. Criar o Order Object final, que possui uma referência ao Customer para a ordem e uma coleta de objetos OrderLine.

   Order.java

   ```
   @Entity
   public class Order {
       @Id String orderNumber;
       java.util.Date date;
       @ManyToOne(cascade=CascadeType.PERSIST) Customer customer;
       @OneToMany(cascade=CascadeType.ALL, mappedBy="order")
      @OrderBy("lineNumber") List<OrderLine> lines;  }
   ```

   ALL em cascata é utilizado como o modificador para as linhas. Esse modificador sinaliza o EntityManager para exibir em cascata a operação PERSIST e a operação REMOVE. Por exemplo, se a entidade Order for persistida ou removida, então, todas as entidades OrderLine também são persistidas ou removidas.

   Se uma entidade OrderLine for removida da lista de linhas no objeto de Pedido, a referência então será quebrada. No entanto, a entidade OrderLine não será removida do cache. Você deve utilizar a API de remoção do EntityManager para remover entidades do cache. A operação REMOVE não é utilizada na entidade do cliente ou na entidade de item de OrderLine. Como resultado, a entidade do cliente permanece mesmo que o item ou o item seja removido quando a OrderLine for removida.

   O modificador mappedBy indica um relacionamento inverso com a entidade de destino. O modificador identifica qual atributo na entidade de destino refere-se à entidade de origem, e o lado pertencente de um relacionamento um para um ou muitos para muitos. Geralmente, é possível omitir o modificador. Entretanto, um erro é exibido para indicar que ele deve ser especificado se WebSphere eXtreme Scale não puder descobrí-lo automaticamente. Uma entidade OrderLine que contém dois tipos de atributos Order em uma relacionamento muitos para um normalmente causa o erro.

   A anotação @OrderBy especifica a ordem na qual cada entidade OrderLine deve estar na lista de linhas. Se a anotação não for especificada, então, as linhas são exibida em uma ordem arbitrária. Embora as linhas sejam incluídas na entidade Order emitindo ArrayList, o que preserva o pedido, o EntityManager não necessariamente reconhecerá a pedido. Quando você emite o método de localização para recuperar o objeto Order do cache, o objeto de lista não é um objeto ArrayList.

5. Crie o aplicativo. O exemplo a seguir ilustra o objeto Order final, que possui uma referência para o Customer para o pedido e uma coleta de objetos OrderLine.
   a. Encontre os Itens a serem ordenados, que podem se tornar entidades Gerenciadas.
   b. Crie a OrderLine e anexe-a a cada Item.

c. Crie o Pedido e associe-o a cada OrderLine e ao cliente.
d. Persista o pedido, que persiste automaticamente cada OrderLine.
e. Confirme a transação, que desconecta cada entidade e sincroniza o estado das entidades com o cache.
f. Imprima as informações do pedido. As entidades OrderLine são armazenadas automaticamente pelo ID da OrderLine.

Application.java

```
static public void main(String [] args)
        throws Exception
    {
        ...

        // Add some items to our inventory.
        em.getTransaction().begin();
        createItems(em);
        em.getTransaction().commit();

        // Create a new customer with the items in his cart.
        em.getTransaction().begin();
        Customer cust = createCustomer();
        em.persist(cust);

        // Create a new order and add an order line for each item.
        // Each line item is automatically persisted since the
    // Cascade=ALL option is set.
        Order order = createOrderFromItems(em, cust, "ORDER_1",
     new String[]{"1", "2"}, new int[]{1,3});
        em.persist(order);
        em.getTransaction().commit();

        // Print the order summary
        em.getTransaction().begin();
        order = (Order)em.find(Order.class, "ORDER_1");
        System.out.println(printOrderSummary(order));
        em.getTransaction().commit();
    }

    public static Customer createCustomer() {
        Customer cust = new Customer();
        cust.address = "Main Street";
        cust.firstName = "John";
        cust.surname = "Smith";
        cust.id = "C001";
        cust.phoneNumber = "5555551212";
        return cust;
    }

    public static void createItems(EntityManager em) {
        Item item1 = new Item();
        item1.id = "1";
        item1.price = 9.99;
        item1.description = "Widget 1";
        item1.quantityOnHand = 4000;
        em.persist(item1);

        Item item2 = new Item();
        item2.id = "2";
        item2.price = 15.99;
        item2.description = "Widget 2";
        item2.quantityOnHand = 225;
        em.persist(item2);

    }
```

```
 public static Order createOrderFromItems(EntityManager em,
Customer cust, String orderId, String[] itemIds, int[] qty) {

    Item[] items = getItems(em, itemIds);

    Order order = new Order();
    order.customer = cust;
    order.date = new java.util.Date();
    order.orderNumber = orderId;
    order.lines = new ArrayList<OrderLine>(items.length);
 for(int i=0;i<items.length;i++){
   OrderLine line = new OrderLine();
         line.lineNumber = i+1;
         line.item = items[i];
         line.price = line.item.price;
         line.quantity = qty[i];
         line.order = order;
         order.lines.add(line);
    }
    return order;
 }

 public static Item[] getItems(EntityManager em, String[] itemIds) {
    Item[] items = new Item[itemIds.length];
    for(int i=0;i<items.length;i++){
 items[i] = (Item) em.find(Item.class, itemIds[i]);
    }
    return items;
 }
```

A próxima etapa é excluir uma entidade. A interface EntityManager possui um método de remoção que marca um objeto como excluído. O aplicativo deve remover a entidade de qualquer coleta de relacionamento antes de chamar o método de remoção. Edite as referências e emita o método de remoção ou em.remove(object), como uma etapa final.

Tutorial do Entity Manager: Atualizando Entradas

Se você desejar alterar uma entidade, é possível localizar a instância, atualizar a instância e quaisquer entidades referenciadas, além de executar o commit da transação.

Entradas de atualização. O exemplo a seguir demonstra como localizar a instância Order, alterá-la e qualquer entidade mencionada, e confirmar a transação.

```
public static void updateCustomerOrder(EntityManager em) {
    em.getTransaction().begin();
    Order order = (Order) em.find(Order.class, "ORDER_1");
    processDiscount(order, 10);
    Customer cust = order.customer;
    cust.phoneNumber = "5075551234";
    em.getTransaction().commit();
}

public static void processDiscount(Order order, double discountPct) {
    for(OrderLine line : order.lines) {
        line.price = line.price * ((100-discountPct)/100);
    }
}
```

Executar o flushing da transação sincroniza todas as entidades gerenciadas com o cache. Quando ocorre o commit de uma transação, automaticamente ocorre um flush. Neste caso, Order se torna uma entidade gerenciada. Quaisquer entidades referenciadas de Order, Customer e OrderLine também se tornam entidades gerenciadas. No flush da transação, cada entidade é verificada para determinar se foi modificada. As que foram modificadas são atualizadas no cache. Após a

conclusão da transação, através de commit ou rollback, as entidades se separam e quaisquer alterações feitas nas entidades não são refletidas no cache.

Tutorial do Entity Manager: Atualizando e Removendo Entradas com um Índice

É possível utilizar um índice para localizar, atualizar e remover entidades.

Atualize e remova entidades utilizando um índice. Utilize um índice para localizar, atualizar e remover entidades. Nos exemplos anteriores, a classe de entidade Order é atualizada para utilizar a anotação @Index. A anotação @Index sinaliza ao WebSphere eXtreme Scale para criar um índice de intervalo para um atributo. O nome do índice é o mesmo nome do atributo e é sempre um tipo de índice MapRangeIndex.

Order.java

```
@Entity
public class Order {
    @Id String orderNumber;
    @Index java.util.Date date;
    @OneToOne(cascade=CascadeType.PERSIST) Customer customer;
    @OneToMany(cascade=CascadeType.ALL, mappedBy="order")
    @OrderBy("lineNumber") List<OrderLine> lines;  }
```

O exemplo a seguir demonstra como cancelar todas os pedidos enviados no último minuto. Encontrar o pedido utilizando um índice, incluir os itens no pedido de volta no inventário e remover o pedido e os itens da linha associados do sistema.

```
public static void cancelOrdersUsingIndex(Session s)
  throws ObjectGridException {
    // Cancel all orders that were submitted 1 minute ago
    java.util.Date cancelTime = new
   java.util.Date(System.currentTimeMillis() - 60000);
    EntityManager em = s.getEntityManager();
    em.getTransaction().begin();
    MapRangeIndex dateIndex = (MapRangeIndex)
   s.getMap("Order").getIndex("date");
    Iterator<Tuple> orderKeys = dateIndex.findGreaterEqual(cancelTime);
  while(orderKeys.hasNext()) {
   Tuple orderKey = orderKeys.next();
   // Localizar o Pedido para que possamos removê-lo.
   Order curOrder = (Order) em.find(Order.class, orderKey);
   // Verificar se o pedido não foi atualizado por outra pessoa.
   if(curOrder != null && curOrder.date.getTime() >= cancelTime.getTime()) {
    for(OrderLine line : curOrder.lines) {
     // Incluir o item novamente no inventário.
     line.item.quantityOnHand += line.quantity;
     line.quantity = 0;
    }
   em.remove(curOrder);
   }
  }
 em.getTransaction().commit();
}
```

Tutorial do Entity Manager: Atualizando e Removendo Entradas Utilizando uma Consulta

É possível atualizar e remover entidades utilizando uma consulta.

Atualize e remova entidades utilizando uma consulta.

Order.java

```
@Entity
public class Order {
    @Id String orderNumber;
    @Index java.util.Date date;
    @OneToOne(cascade=CascadeType.PERSIST) Customer customer;
    @OneToMany(cascade=CascadeType.ALL, mappedBy="order")
   @OrderBy("lineNumber") List<OrderLine> lines;  }
```

A classe de entidade order é a mesma que a do exemplo anterior. A classe ainda fornece a anotação @Index, porque a cadeia de consultas utiliza a data para localizar a entidade. O mecanismo de consulta utiliza índices quando eles podem ser utilizados.

```
public static void cancelOrdersUsingQuery(Session s) {
        // Cancel all orders that were submitted 1 minute ago
        java.util.Date cancelTime =
     new java.util.Date(System.currentTimeMillis() - 60000);
        EntityManager em = s.getEntityManager();
        em.getTransaction().begin();

        // Create a query that will find the order based on date.  Since
        // we have an index defined on the order date, the query
    // will automatically use it.
        Query query = em.createQuery("SELECT order FROM Order order
     WHERE order.date >= ?1");
        query.setParameter(1, cancelTime);
        Iterator<Order> orderIterator = query.getResultIterator();
    while(orderIterator.hasNext()) {
     Order order = orderIterator.next();
     // Verificar se o pedido não foi atualizado por outra pessoa.
     // Since the query used an index, there was no lock on the row.
     if(order != null && order.date.getTime() >= cancelTime.getTime()) {
       for(OrderLine line : order.lines) {
        // Incluir o item novamente no inventário.
        line.item.quantityOnHand += line.quantity;
        line.quantity = 0;
       }
       em.remove(order);
     }
   }
  em.getTransaction().commit();
}
```

Como o exemplo anterior, o método cancelOrdersUsingQuery é destinado a cancelar todos os pedidos que foram enviados no último minuto. Para cancelar o pedido, você o localiza utilizando uma consulta, inclui os itens no pedido de volta no inventário e remove o pedido e os itens de linha associados do sistema.

Tutorial do ObjectQuery

Com as seguintes etapas, é possível desenvolver um ObjectGrid de memória local, que pode armazenar informações de pedido para um Web site e demonstrar como utilizar o ObjectQuery para consultar os dados na grade.

Antes de Iniciar

Certifique-se de ter o arquivo `objectgrid.jar` em seu caminho de classe.

Por Que e Quando Desempenhar Esta Tarefa

Cada etapa no tutorial é construída na etapa anterior. Siga cada uma das etapas para criar um aplicativo Java Platform, Standard Edition Versão 1.4 (ou superior) simples que usa um ObjectGrid local na memória.

1. "Tutorial do ObjectQuery - Etapa 1" na página 152
   - Como criar um ObjectGrid local
   - Como definir um esquema para um único objeto utilizando o acesso ao campo
   - Como armazenar o objeto
   - Como consultar o objeto com ObjectQuery
2. "Tutorial do ObjectQuery - Etapa 2" na página 153
   - Como criar um índice que a consulta pode utilizar
3. "Tutorial do ObjectQuery - Etapa 3" na página 154
   - Como criar um esquema com duas entidades relacionadas

- Como armazenar objetos com uma referência de chave estrangeira entre elas
- Como consultar os objetos utilizando uma única consulta com um JOIN

4. "Tutorial do ObjectQuery - Etapa 4" na página 156
   - Como criar um esquema com múltiplas entidades relacionadas
   - Como usar o acesso de método ou de propriedade em vez do acesso de campo

Tutorial do ObjectQuery - Etapa 1

Com as seguintes etapas, você poderá continuar desenvolvendo um ObjectGrid local de memória que armazena informações de pedidos para uma loja varejista on-line usando as APIs do ObjectMap. Defina um esquema para o mapa e execute uma consulta em relação ao mapa.

1. Crie um ObjectGrid com um esquema de mapa.

   Crie um ObjectGrid com um esquema de mapa para o mapa; em seguida, insira um objeto no cache e recupere-o posteriormente, utilizando uma consulta simples.

   OrderBean.java

```
public class OrderBean implements Serializable {
    String orderNumber;
    java.util.Date date;
    String customerName;
    String itemName;
    int quantity;
    double price;
}
```

2. Defina a chave principal.

   O código anterior mostra um objeto OrderBean. Este objeto implementa a interface java.io.Serializable porque todos os objetos no cache devem (por padrão) ser Serializáveis.

   O atributo orderNumber é a chave principal do objeto. O programa de exemplo a seguir pode ser executado no modo independente. É necessário seguir esse tutorial em um projeto Eclipse Java que tenha o arquivo `objectgrid.jar` incluído no caminho de classe.

   Application.java

```
package querytutorial.basic.step1;

import java.util.Iterator;

import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectMap;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.config.QueryConfig;
import com.ibm.websphere.objectgrid.config.QueryMapping;
import com.ibm.websphere.objectgrid.query.ObjectQuery;

public class Application
{

    static public void main(String [] args) throws Exception
    {
        ObjectGrid og = ObjectGridManagerFactory.getObjectGridManager().createObjectGrid();
        og.defineMap("Order");

        // Definir o esquema
        QueryConfig queryCfg = new QueryConfig();
        queryCfg.addQueryMapping(new QueryMapping("Order",
        OrderBean.class.getName(), "orderNumber", QueryMapping.FIELD_ACCESS));
        og.setQueryConfig(queryCfg);

        Session s = og.getSession();
        ObjectMap orderMap = s.getMap("Order");

        s.begin();
```

```
            OrderBean o = new OrderBean();
            o.customerName = "John Smith";
            o.date = new java.util.Date(System.currentTimeMillis());
            o.itemName = "Widget";
            o.orderNumber = "1";
            o.price = 99.99;
            o.quantity = 1;
            orderMap.put(o.orderNumber, o);
            s.commit();

            s.begin();
            ObjectQuery query = s.createObjectQuery("SELECT o FROM Order o WHERE o.itemName='Widget'");
            Iterator result = query.getResultIterator();
            o = (OrderBean) result.next();
            System.out.println("Found order for customer: " + o.customerName);
            s.commit();
        }
    }
```

Esse aplicativo eXtreme Scale primeiro inicializa um ObjectGrid local com um nome gerado automaticamente. Em seguida, o aplicativo cria um BackingMap e um QueryConfig que definem qual tipo Java está associado ao mapa, o nome do campo que é a chave principal para o mapa e como acessar os dados no objeto. A seguir, você obtém uma Sessão para adquirir a instância do ObjectMap e inserir um objeto OrderBean no mapa em uma transação.

Depois que os dados forem confirmados no cache, será possível usar o ObjectQuery para localizar o OrderBean usando qualquer um dos campos persistentes na classe. Campos persistentes são aqueles que não possuem o modificador temporário. Como nenhum índice foi definido no BackingMap, o ObjectQuery deverá varrer cada objeto no mapa usando o reflexo Java.

O que Fazer Depois

O "Tutorial do ObjectQuery - Etapa 2" demonstra como um índice pode ser usado para otimizar a consulta.

Tutorial do ObjectQuery - Etapa 2

Nas seguintes etapas, você continuará criando um ObjectGrid com um mapa e um índice, junto com um esquema para o mapa. Em seguida, você poderá inserir um objeto no cache e, mais tarde, recuperá-lo utilizando uma consulta simples.

Antes de Iniciar

Certifique-se de ter concluído o "Tutorial do ObjectQuery - Etapa 1" na página 152 antes de continuar com esta etapa do tutorial.

Esquema e índice

Application.java

```
// Create an index
    HashIndex idx= new HashIndex();
    idx.setName("theItemName");
    idx.setAttributeName("itemName");
    idx.setRangeIndex(true);
    idx.setFieldAccessAttribute(true);
    orderBMap.addMapIndexPlugin(idx);
}
```

O índice deve ser uma instância com.ibm.websphere.objectgrid.plugins.index.HashIndex com as seguintes configurações:

- O Nome é arbitrário, mas deve ser exclusivo para um BackingMap fornecido.

- O AttributeName é o nome do campo ou propriedade do bean que o mecanismo de indexação utiliza para examinar a classe. Neste caso, este é o nome do campo para o qual você criará um índice.
- RangeIndex deve ser sempre verdadeiro.
- FieldAccessAttribute deve corresponder ao conjunto de valores no objeto QueryMapping quando o esquema de consulta foi criado. Nesse caso, o objeto Java é acessado usando os campos diretamente.

Quando é executada uma consulta que filtra o campo itemName, o mecanismo de consulta utiliza automaticamente esse índice. Isso permite que a consulta seja executada muito mais rápido, e uma varredura de mapa não é necessária. A próxima etapa demonstra como um índice pode ser utilizado para otimizar a consulta.
Próxima etapa

Tutorial do ObjectQuery - Etapa 3

Na etapa a seguir, você criará um ObjectGrid com dois mapas e um esquema para os mapas com um relacionamento e, em seguida, inserirá os objetos no cache e posteriormente irá recuperá-los utilizando uma consulta simples.

Antes de Iniciar

Certifique-se de ter concluído o "Tutorial do ObjectQuery - Etapa 2" na página 153 antes de continuar com esta etapa.

Por Que e Quando Desempenhar Esta Tarefa

Neste exemplo, há dois mapas, cada um com um tipo único Java mapeado para ele. O mapa Order possui objetos OrderBean e o mapa Customer contém objetos CustomerBean.

Defina mapas com um relacionamento.

OrderBean.java

```
public class OrderBean implements Serializable {
    String orderNumber;
    java.util.Date date;
    String customerId;
    String itemName;
    int quantity;
    double price;
}
```

O OrderBean não contém mais o customerName. Ao invés disso, ele contém o customerId, que é a chave principal para o objeto CustomerBean e o mapa Customer.

CustomerBean.java

```
public class CustomerBean implements Serializable{
    private static final long serialVersionUID = 1L;
    String id;
    String firstName;
    String surname;
    String address;
    String phoneNumber;
}
```

O relacionamento entre os dois tipos ou Mapas é:

Application.java

```
public class Application
{

    static public void main(String [] args)
        throws Exception
    {
        ObjectGrid og = ObjectGridManagerFactory.getObjectGridManager().createObjectGrid();
        og.defineMap("Order");
        og.defineMap("Customer");

        // Definir o esquema
        QueryConfig queryCfg = new QueryConfig();
        queryCfg.addQueryMapping(new QueryMapping(
            "Order", OrderBean.class.getName(), "orderNumber", QueryMapping.FIELD_ACCESS));
        queryCfg.addQueryMapping(new QueryMapping(
            "Customer", CustomerBean.class.getName(), "id", QueryMapping.FIELD_ACCESS));
        queryCfg.addQueryRelationship(new QueryRelationship(
             OrderBean.class.getName(), CustomerBean.class.getName(), "customerId", null));
        og.setQueryConfig(queryCfg);

        Session s = og.getSession();
        ObjectMap orderMap = s.getMap("Order");
        ObjectMap custMap = s.getMap("Customer");

        s.begin();
        CustomerBean cust = new CustomerBean();
        cust.address = "Main Street";
        cust.firstName = "John";
        cust.surname = "Smith";
        cust.id = "C001";
        cust.phoneNumber = "5555551212";
        custMap.insert(cust.id, cust);

        OrderBean o = new OrderBean();
        o.customerId = cust.id;
        o.date = new java.util.Date();
        o.itemName = "Widget";
        o.orderNumber = "1";
        o.price = 99.99;
        o.quantity = 1;
        orderMap.insert(o.orderNumber, o);
        s.commit();

        s.begin();
        ObjectQuery query = s.createObjectQuery(
            "SELECT c FROM Order o JOIN o.customerId as c WHERE o.itemName='Widget'");
        Iterator result = query.getResultIterator();
        cust = (CustomerBean) result.next();
        System.out.println("Found order for customer: " + cust.firstName + " " + cust.surname);
        s.commit();
    }
}
```

O XML equivalente no descritor de implementação do ObjectGrid é:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
    <objectGrid name="CompanyGrid">
      <backingMap name="Order"/>
      <backingMap name="Customer"/>

      <querySchema>
         <mapSchemas>
           <mapSchema
             mapName="Order"
             valueClass="com.mycompany.OrderBean"
             primaryKeyField="orderNumber"
             accessType="FIELD"/>
           <mapSchema
             mapName="Customer"
             valueClass="com.mycompany.CustomerBean"
             primaryKeyField="id"
             accessType="FIELD"/>
         </mapSchemas>
         <relationships>
```

```
          <relationship
            source="com.mycompany.OrderBean"
            target="com.mycompany.CustomerBean"
            relationField="customerId"/>
        </relationships>
      </querySchema>
    </objectGrid>
  </objectGrids>
</objectGridConfig>
```

O que Fazer Depois

O "Tutorial do ObjectQuery - Etapa 4" expande a etapa atual ao incluir um campo, objetos de acesso da propriedade e relacionamentos adicionais.

Tutorial do ObjectQuery - Etapa 4

A etapa a seguir mostra como criar um ObjectGrid com quatro mapas e um esquema para os mapas com vários relacionamentos unidirecionais e bidirecionais. Em seguida, você poderá inserir objetos no cache e, mais tarde, recuperá-los utilizando várias consultas.

Antes de Iniciar

Certifique-se de ter concluído o "Tutorial do ObjectQuery - Etapa 3" na página 154 antes de continuar com a etapa atual.

Relacionamentos de mapas múltiplos

OrderBean.java

```
public class OrderBean implements Serializable {
    String orderNumber;
    java.util.Date date;
    String customerId;
    String itemName;
    int quantity;
    double price;
}
```

Na etapa anterior, o OrderBean não possuía mais o customerName nele. Ao invés disso, ele contém o customerId, que é a chave principal para o objeto CustomerBean e o mapa Customer.

CustomerBean.java

```
public class CustomerBean implements Serializable{
    private static final long serialVersionUID = 1L;
    String id;
    String firstName;
    String surname;
    String address;
    String phoneNumber;
}
```

Ao ter criado as classes especificadas acima, é possível executar o aplicativo abaixo.

Application.java

```
public class Application
{

    static public void main(String [] args)
        throws Exception
    {
```

```
        ObjectGrid og = ObjectGridManagerFactory.getObjectGridManager().createObjectGrid();
        og.defineMap("Order");
        og.defineMap("Customer");

        // Definir o esquema
        QueryConfig queryCfg = new QueryConfig();
        queryCfg.addQueryMapping(new QueryMapping(
            "Order", OrderBean.class.getName(), "orderNumber", QueryMapping.FIELD_ACCESS));
        queryCfg.addQueryMapping(new QueryMapping(
            "Customer", CustomerBean.class.getName(), "id", QueryMapping.FIELD_ACCESS));
        queryCfg.addQueryRelationship(new QueryRelationship(
             OrderBean.class.getName(), CustomerBean.class.getName(), "customerId", null));
        og.setQueryConfig(queryCfg);

        Session s = og.getSession();
        ObjectMap orderMap = s.getMap("Order");
        ObjectMap custMap = s.getMap("Customer");

        s.begin();
        CustomerBean cust = new CustomerBean();
        cust.address = "Main Street";
        cust.firstName = "John";
        cust.surname = "Smith";
        cust.id = "C001";
        cust.phoneNumber = "5555551212";
        custMap.insert(cust.id, cust);

        OrderBean o = new OrderBean();
        o.customerId = cust.id;
        o.date = new java.util.Date();
        o.itemName = "Widget";
        o.orderNumber = "1";
        o.price = 99.99;
        o.quantity = 1;
        orderMap.insert(o.orderNumber, o);
        s.commit();

        s.begin();
        ObjectQuery query = s.createObjectQuery(
            "SELECT c FROM Order o JOIN o.customerId as c WHERE o.itemName='Widget'");
        Iterator result = query.getResultIterator();
        cust = (CustomerBean) result.next();
        System.out.println("Found order for customer: " + cust.firstName + " " + cust.surname);
        s.commit();
    }
}
```

Utilizar a configuração XML abaixo (no descritor de implementação ObjectGrid) é equivalente à abordagem programática acima.

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
    <objectGrid name="og1">
      <backingMap name="Order"/>
      <backingMap name="Customer"/>

      <querySchema>
         <mapSchemas>
           <mapSchema
             mapName="Order"
             valueClass="com.mycompany.OrderBean"
             primaryKeyField="orderNumber"
             accessType="FIELD"/>
           <mapSchema
             mapName="Customer"
             valueClass="com.mycompany.CustomerBean"
             primaryKeyField="id"
             accessType="FIELD"/>
         </mapSchemas>
         <relationships>
           <relationship
             source="com.mycompany.OrderBean"
             target="com.mycompany.CustomerBean"
             relationField="customerId"/>
         </relationships>
```

```
      </querySchema>
    </objectGrid>
  </objectGrids>
</objectGridConfig>
```

Tutorial de Segurança do Java SE - Página Principal

Com o seguinte tutorial, é possível criar um ambiente eXtreme Scale distribuído em um ambiente Java Platform, Standard Edition.

Antes de Iniciar

Certifique-se de estar familiarizado com os conceitos básicos de uma configuração distribuída do eXtreme Scale.

Por Que e Quando Desempenhar Esta Tarefa

Neste tutorial, o servidor de catálogo, servidor de contêiner e o cliente são todos executados em um ambiente do Java SE. Cada etapa no tutorial é baseada na anterior. Siga cada etapa para proteger um eXtreme Scale distribuído e a desenvolver um aplicativo simples Java SE para acessar o eXtreme Scale protegido.

Iniciar o tutorial

1. "Tutorial de Segurança do Java SE - Etapa 1"
   - Inicie um servidor de catálogos não-seguro
   - Inicie um servidor de contêineres não-seguro
   - Inicie um cliente para acessar os dados
   - Use o comando xsadmin para mostrar o tamanho do mapa
   - Pare o servidor
2. "Tutorial de Segurança do Java SE - Etapa 2" na página 162
   - Uso do CredentialGenerator
   - Uso do Autenticador
   - Inicie um servidor de catálogos seguro
   - Inicie um servidor de contêineres seguro
   - Inicie o cliente para acessar o ObjectGrid seguro
   - Use o comando xsadmin para mostrar o tamanho do mapa
   - Pare o servidor protegido
3. "Tutorial de Segurança do Java SE - Etapa 3" na página 168
   - Uso da política de autorização do JAAS
4. "Tutorial de Segurança do Java SE - Etapa 4" na página 172
   - Crie um key store e trust store
   - Configure propriedades SSL para o servidor
   - Configure propriedades SSL para o cliente
   - Use o comando xsadmin para mostrar o tamanho do mapa
   - Pare o servidor protegido

Tutorial de Segurança do Java SE - Etapa 1

Este tópico descreve uma *amostra não-segura simples*. Recursos de segurança adicionais são incluídos de maneira incremental nas etapas do tutorial para aumentar a quantidade de segurança integrada que está disponível.

Antes de Iniciar

Nota: Todos os arquivos necessários para essa etapa do tutorial são fornecidos na seguinte seção.

Executando a amostra
Inicie o serviço de catálogo usando os seguintes scripts. Para obter mais informações sobre o início do serviço de catálogo, consulte as informações sobre o início do serviço de catálogo no *Guia de Administração*.

1. Navegue até o diretório bin: `cd objectgridRoot/bin`
2. Inicie um servidor de catálogos denominado catalogServer:
   - UNIX Linux `startOgServer.sh catalogServer`
   - Windows `startOgServer.bat catalogServer`
3. Navegue até o diretório bin `cd objectgridRoot/bin`
4. Em seguida, ative um servidor de contêiner denominado c0 com o script a seguir:
   - UNIX Linux `startOgServer.sh c0 -objectGridFile ../xml/SimpleApp.xml -deploymentPolicyFile ../xml/SimpleDP.xml -catalogServiceEndpoints localhost:2809`
   - Windows `startOgServer.bat c0 -objectGridFile ../xml/SimpleApp.xml -deploymentPolicyFile ../xml/SimpleDP.xml -catalogServiceEndpoints localhost:2809`

Exemplo

Para obter mais informações sobre o início dos servidores de contêiner, consulte as informações sobre o início dos processos do contêiner no *Guia de Administração*.

Após o servidor de catálogos e o servidor de contêineres terem sido iniciados, ative o cliente conforme a seguir.

1. Navegue até o diretório bin mais uma vez.
2. `java -classpath ../lib/objectgrid.jar;../applib/secsample.jar com.ibm.websphere.objectgrid.security.sample.guide.SimpleApp`

O arquivo `secsample.jar` contém a classe SimpleApp.

A saída deste programa é:

```
The customer name for ID 0001 is fName lName
```

Também é possível usar o xsadmin para mostrar os tamanhos de mapa da grade "accounting".

- Navegue até o diretório `objectgridRoot/bin`
- Use o comando `xsadmin` com a opção -mapSizes da seguinte forma.
  - UNIX Linux `xsadmin.sh -g accounting -m mapSet1 -mapSizes`
  - Windows `xsadmin.bat -g accounting -m mapSet1 -mapSizes`

  Você verá a seguinte saída.

  ```
  This administrative utility is provided as a sample only and is not to be considered a fully supported component of the WebSphere eXtreme Scale product.
  Connecting to Catalog service at localhost:1099
  ```

```
********** Displaying Results for Grid - accounting, MapSet - mapSet1
**********
*** Listing Maps for c0 ***
Map Name: customer Partition #: 0 Map Size: 1 Shard Type: Primary
Server Total: 1
Total Domain Count: 1
```

Parando servidores

Servidor de Contêiner

Utilize o seguinte comando para parar o servidor de contêiner c0.

UNIX Linux
```
stopOgServer.sh c0 -catalogServiceEndPoints localhost:2809
```

Windows
```
stopOgServer.bat c0 -catalogServiceEndPoints localhost:2809
```

A seguinte mensagem será exibida.

```
CWOBJ2512I: ObjectGrid server c0 stopped.
```

Servidor de catálogos

É possível parar um servidor de catálogos usando o seguinte comando.

UNIX Linux
```
stopOgServer.sh catalogServer -catalogServiceEndPoints localhost:2809
```

Windows
```
stopOgServer.bat catalogServer -catalogServiceEndPoints localhost:2809
```

Se você encerrar o servidor de catálogos, a seguinte mensagem será exibida.

```
CWOBJ2512I: ObjectGrid server catalogServer stopped.
```

Arquivos necessários

O arquivo a seguir é a classe Java para SimpleApp.

SimpleApp.java

```
// This sample program is provided AS IS and may be used, executed, copied and modified
// without royalty payment by customer
// (a) for its own instruction and study,
// (b) in order to develop applications designed to run with an IBM WebSphere product,
// either for customer's own internal use or for redistribution by customer, as part of such an
// application, in customer's own products.
// Licensed Materials - Property of IBM
// 5724-J34 (C) COPYRIGHT International Business Machines Corp. 2007-2009
package com.ibm.websphere.objectgrid.security.sample.guide;

import com.ibm.websphere.objectgrid.ClientClusterContext;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectMap;
import com.ibm.websphere.objectgrid.Session;

public class SimpleApp {

    public static void main(String[] args) throws Exception  {

        SimpleApp app = new SimpleApp();
        app.run(args);
```

```
    }

    /**
     * read and write the map
     * @throws Exception
     */
    protected void run(String[] args) throws Exception {
        ObjectGrid og = getObjectGrid(args);

        Session session = og.getSession();

        ObjectMap customerMap = session.getMap("customer");

        String customer = (String) customerMap.get("0001");

        if (customer == null) {
            customerMap.insert("0001", "fName lName");
        } else {
            customerMap.update("0001", "fName lName");
        }
        customer = (String) customerMap.get("0001");

        System.out.println("The customer name for ID 0001 is " + customer);
    }

    /**
     * Get the ObjectGrid
     * @return an ObjectGrid instance
     * @throws Exception
     */
    protected ObjectGrid getObjectGrid(String[] args) throws Exception {
        ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();

        // Create an ObjectGrid
        ClientClusterContext ccContext = ogManager.connect("localhost:2809", null, null);
        ObjectGrid og = ogManager.getObjectGrid(ccContext, "accounting");

        return og;

    }

}
```

O método getObjectGrid nesta classe obtém o um ObjectGrid e o método run lê um registro a partir do mapa do cliente e atualiza o valor.

Para executar essa amostra em um ambiente distribuído, um arquivo descritor XML do ObjectGrid `SimpleApp.xml` e um arquivo de implementação XML `SimpleDP.xml` são criados. Os arquivos são apresentados no exemplo a seguir:

SimpleApp.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">
    <objectGrids>
        <objectGrid name="accounting">
            <backingMap name="customer" readOnly="false" copyKey="true"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

O arquivo XML a seguir configura o ambiente de implementação.

SimpleDP.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
 xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">

 <objectgridDeployment objectgridName="accounting">
  <mapSet name="mapSet1" numberOfPartitions="1" minSyncReplicas="0" maxSyncReplicas="2" maxAsyncReplicas="1">
   <map ref="customer"/>
  </mapSet>
 </objectgridDeployment>
</deploymentPolicy>
```

Essa é uma configuração simples do ObjectGrid com uma instância do ObjectGrid chamada ″accounting″ e um mapa chamado ″customer″ (dentro do mapSet ″mapSet1″). O arquivo `SimpleDP.xml` apresenta um conjunto de mapa que é configurado com 1 partição e com o mínimo de 0 réplicas necessárias.

Próxima etapa do tutorial

Tutorial de Segurança do Java SE - Etapa 2

Baseado na etapa anterior, o seguinte tópico mostra como implementar a autenticação do cliente em um ambiente do eXtreme Scale distribuído.

Antes de Iniciar

Certifique-se de ter concluído o “Tutorial de Segurança do Java SE - Etapa 1” na página 158.

Por Que e Quando Desempenhar Esta Tarefa

Com a autenticação de cliente ativada, um cliente é autenticado antes de conectar-se ao servidor eXtreme Scale. Esta seção demonstra como a autenticação de cliente pode ser feita em um ambiente de servidor eXtreme Scale, incluindo código de amostra e scripts para demonstrar.

Como qualquer outro mecanismo de autenticação, a autenticação mínima consiste nas seguintes etapas:

1. O administrador altera as configurações feitas para tornar a autenticação um requisito.
2. O cliente oferece uma credencial para o servidor.
3. O servidor autentica a credencial para o registro.

1. **Credencial de cliente**

   Uma credencial de cliente é representada por uma interface com.ibm.websphere.objectgrid.security.plugins.Credential. Uma credencial de cliente pode ser um par de nome de usuário e senha, um registro do Kerberos, um certificado cliente ou dados em qualquer formato concordado entre o cliente e o servidor. Consulte a a documentação da API de Credencial para obter mais detalhes.

   Esta interface define explicitamente os métodos equals(Object) e hashCode(). Estes métodos são importantes porque os objetos Subject autenticados são armazenados em cache utilizando o objeto Credential como a chave no lado do servidor.

   O eXtreme Scale também fornece um plug-in para gerar uma credencial. Este plug-in é representado pela interface com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator e é utilizado para gerar uma credencial de cliente. Isso é útil quando a credencial pode expirar. Neste caso, o método getCredential() é chamado para renovar uma credencial. Consulte a Documentação da API do CredentialGenerator para obter mais detalhes.

   É possível implementar estas duas interfaces para o tempo de execução do cliente do eXtreme Scale para obter credenciais de cliente.

   Essa amostra usa as seguintes duas implementações de plug-in de amostra fornecidas pelo eXtreme Scale.

   ```
   com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential
   com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator
   ```

Para obter mais informações sobre esses plug-ins, consulte o tópico sobre a programação de autenticação do cliente no *Guia de Programação*.

2. **Autenticação do servidor** Após o cliente do eXtreme Scale recuperar o objeto Credential utilizando o objeto CredentialGenerator, este objeto Credential do cliente é enviado junto o pedido do cliente para o servidor eXtreme Scale. O servidor eXtreme Scale autentica o objeto Credential antes de processar o pedido. Se o objeto Credential for autenticado com êxito, um objeto Subject será retornado para representar este cliente.

   Este objeto Subject é então armazenado em cache e expira após seu tempo de vida alcançar o valor de tempo limite da sessão. O valor de tempo limite da sessão de login pode ser configurado utilizando a propriedade loginSessionExpirationTime no arquivo XML do cluster. Por exemplo, configurar `loginSessionExpirationTime="300"` fará com que o objeto Subject expire em 300 segundos.Esse objeto Subject é, então, utilizado para autorizar o pedido, que é mostrado posteriormente.

   Um servidor eXtreme Scale utiliza o plug-in do Autenticador para autenticar o objeto Credential. Consulte a Documentação da API do Autenticador para obter mais detalhes.

   Este exemplo utiliza uma implementação integrada do eXtreme Scale: KeyStoreLoginAuthenticator, que é propósitos de teste e amostra (um armazenamento de chaves é um registro do usuário simples e não deve ser utilizado para produção). programação de autenticação do cliente no *Guia de Programação*.

   Este KeyStoreLoginAuthenticator utiliza um KeyStoreLoginModule para autenticar o usuário com o armazenamento de chaves utilizando o módulo de login do JAAS ″KeyStoreLogin″. O armazenamento de chaves é configurado como uma opção para a classe KeyStoreLoginModule. O exemplo a seguir ilustra o alias keyStoreLogin definido no arquivo de configuração do JAAS og_jaas.config:

   ```
   KeyStoreLogin{
   com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginModule required
        keyStoreFile="../security/sampleKS.jks" debug = true;
   };
   ```

   Os seguintes comandos criam um armazenamento de chaves sampleKS.jks no diretório %OBJECTGRID_HOME%/security com a senha como sampleKS1. Além disso, três certificados de usuário, representando o usuário administrador, o usuário gerente e o usuário caixa são criados com suas próprias senhas.

   a. Navegue para o diretório-raiz eXtreme Scale.

      ```
      cd objectgridRoot
      ```

   b. Crie um diretório chamado ″security″.

      ```
      mkdir security
      ```

   c. Navegue até o diretório de segurança recém criado.

      ```
      cd security
      ```

   d. Use o keytool (no diretório `javaHOME/bin`) para criar um usuário ″administator″ com a senha ″administrator1″ no armazenamento de chaves sampleKS.jks.

      ```
      keytool -genkey -v -keystore ./sampleKS.jks -storepass sampleKS1
      -alias administrator -keypass administrator1
      -dname CN=administrator,O=acme,OU=OGSample -validity 10000
      ```

   e. Use o keytool (no diretório `javaHOME/bin`) para criar um usuário ″manager″ com a senha ″manager1″ no armazenamento de chaves sampleKS.jks.

      ```
      keytool -genkey -v -keystore ./sampleKS.jks -storepass sampleKS1
      -alias manager -keypass manager1
      -dname CN=manager,O=acme,OU=OGSample -validity 10000
      ```

f. Use o keytool (no diretório javaHOME/bin) para criar um usuário ″cashier″ com a senha ″cashier1″ no armazenamento de chaves sampleKS.jks.

```
keytool -genkey -v -keystore ./sampleKS.jks -storepass sampleKS1
-alias cashier -keypass cashier1 -dname CN=cashier,O=acme,OU=OGSample
-validity 10000
```

A configuração de segurança do cliente é definida no arquivo de propriedades do cliente. Utilize o seguinte comando para criar uma cópia no diretório %OBJECTGRID_HOME%/security:

a. Mude para o diretório security.

```
cd objectgridRoot/security
```

b. Copie o arquivo sampleClient.properties para o arquivo client.properties.

```
cp ../properties/sampleClient.properties client.properties
```

As seguintes propriedades são realçadas no arquivo client.properties no diretório de segurança.

a. **securityEnabled:** Configurar securityEnabled como true (valor padrão) ativa a segurança do cliente, que inclui autenticação.
b. **credentialAuthentication:** Configure credentialAuthentication autenticação de credencial.
c. **transportType:** Configure transportType como TCP/IP, o que significa que nenhum SSL será utilizado.
d. **singleSignOnEnabled:** Configure-o como false (valor padrão). A conexão única não está disponível.

3. **Configuração de segurança do servidor**

A configuração de segurança do servidor é especificada no arquivo XML descritor de segurança e o arquivo de propriedades de segurança do servidor.O arquivo XML descritor de segurança descreve as propriedades de segurança comuns para todos os servidores (incluindo servidores de catálogo e servidores de contêiner). Um exemplo de propriedade é a configuração do autenticador que representa o registro do usuário e o mecanismo de autenticação.

Aqui está o arquivo security.xml a ser utilizado nesta amostra:

```
<?xml version="1.0" encoding="UTF-8"?>
<securityConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ibm.com/ws/objectgrid/config/security ../objectGridSecurity.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config/security">

 <security securityEnabled="true" loginSessionExpirationTime="300" >

        <authenticator className ="com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginAuthenticator">
        </authenticator>
    </security>

</securityConfig>
```

a. **securityEnabled:** Configure para true para ativar a segurança do servidor, incluindo autenticação.
b. **loginSessionExpirationTime:** Configure o valor como 300 (valor-padrão).
c. **authenticator:** Inclua a classe do autenticador KeyStoreLoginAuthenticator no arquivo XML do cluster, conforme a seguir:

```
<authenticator className ="com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginAuthenticator">
        </authenticator>
```

d. **credentialAuthentication:** Configure o atributo credentialAuthentication para Necessário para que o servidor exija autenticação.

Para obter uma explicação mais detalhada sobre o arquivo security.xml, consulte as informações sobre o arquivo descritor XML de segurança no *Guia de Administração*.

Copie o arquivo de propriedades do servidor no diretório de segurança. Neste momento, não é necessário modificar nada neste arquivo.

a. Navegue até o diretório security.

```
cd objectgridRoot/security
```

b. Copie o arquivo `sampleServer.properties` de amostra do objectGrid do diretório de propriedades para o novo arquivo `server.properties`.

```
cp ../properties/containerServer.properties server.properties
```

Faça as seguintes alterações no arquivo `server.properties`:

a. **securityEnabled:** Configure o atributo **securityEnabled** como true.

b. **transportType:** Configure o atributo **transportType** como TCP/IP, o que significa que nenhum SSL será utilizado.

c. **secureTokenManagerType:** Configure o atributo **secureTokenManagerType** como none para não configurar o gerenciador de tokens seguros.

4. **Cliente seguro** Conecte o aplicativo cliente ao servidor de maneira segura conforme demonstrado no exemplo a seguir:

```
// This sample program is provided AS IS and may be used, executed, copied and modified
// without royalty payment by customer
// (a) for its own instruction and study,
// (b) in order to develop applications designed to run with an IBM WebSphere product,
// either for customer's own internal use or for redistribution by customer, as part of such an
// application, in customer's own products.
// Licensed Materials - Property of IBM
// 5724-J34 (C) COPYRIGHT International Business Machines Corp. 2007-2009
package com.ibm.websphere.objectgrid.security.sample.guide;

import com.ibm.websphere.objectgrid.ClientClusterContext;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.security.config.ClientSecurityConfiguration;
import com.ibm.websphere.objectgrid.security.config.ClientSecurityConfigurationFactory;
import com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator;
import com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator;

public class SecureSimpleApp extends SimpleApp {

    public static void main(String[] args) throws Exception  {

        SecureSimpleApp app = new SecureSimpleApp();
        app.run(args);
    }

    /**
     * Get the ObjectGrid
     * @return an ObjectGrid instance
     * @throws Exception
     */
    protected ObjectGrid getObjectGrid(String[] args) throws Exception {
        ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
        ogManager.setTraceFileName("logs/client.log");
        ogManager.setTraceSpecification("ObjectGrid*=all=enabled:ORBRas=all=enabled");

        // Creates a ClientSecurityConfiguration object using the specified file
        ClientSecurityConfiguration clientSC = ClientSecurityConfigurationFactory
                .getClientSecurityConfiguration(args[0]);

        // Creates a CredentialGenerator using the passed-in user and password.
        CredentialGenerator credGen = new UserPasswordCredentialGenerator(args[1], args[2]);
        clientSC.setCredentialGenerator(credGen);

        // Create an ObjectGrid by connecting to the catalog server
        ClientClusterContext ccContext = ogManager.connect("localhost:2809", clientSC, null);
        ObjectGrid og = ogManager.getObjectGrid(ccContext, "accounting");

        return og;

    }

}
```

Existem três coisas diferentes do aplicativo não-seguro:

a. Objeto ClientSecurityConfiguration criado ao transmitir o arquivo `client.properties` configurado.

b. Criou um UserPasswordCredentialGenerator utilizando o ID do usuário e a senha passados.

c. Conectado com o servidor de catálogo para obter um ObjectGrid a partir ClientClusterContext ao transmitir um objeto ClientSecurityConfiguration.

5. **Execute o aplicativo**

Para iniciar o aplicativo, inicie o servidor de catálogos. Emita as opções da linha de comandos -clusterFile e -serverProps para transmitir nas propriedades de segurança:

a. Navegue até o diretório bin:

```
cd objectgridRoot/bin
```

b. Ative o servidor de catálogos:

- UNIX Linux

```
startOgServer.sh catalogServer -clusterSecurityFile ../security/security.xml
-serverProps ../security/server.properties -jvmArgs
-Djava.security.auth.login.config="../security/og_jaas.config"
```

- Windows

```
startOgServer.bat catalogServer -clusterSecurityFile ../security/security.xml
-serverProps ../security/server.properties -jvmArgs
-Djava.security.auth.login.config="../security/og_jaas.config"
```

Em seguida, ative um servidor de contêiner seguro, utilizando o seguinte script:

a. Navegue até o diretório bin novamente:

```
cd objectgridRoot/bin
```

b. Ative o servidor de contêineres seguro:

- Linux UNIX

```
startOgServer.sh c0 -objectgridFile ../xml/SimpleApp.xml
-deploymentPolicyFile ../xml/SimpleDP.xml
-catalogServiceEndpoints localhost:2809
-serverProps ../security/server.properties
-jvmArgs -Djava.security.auth.login.config="../security/og_jaas.config"
```

- Windows

```
startOgServer.bat c0 -objectgridFile ../xml/SimpleApp.xml
-deploymentPolicyFile ../xml/SimpleDP.xml
-catalogServiceEndpoints localhost:2809
-serverProps ../security/server.properties
-jvmArgs -Djava.security.auth.login.config="../security/og_jaas.config"
```

O arquivo de propriedades do servidor é passado executando -serverProps.

Quando o servidor for iniciado, ative o cliente, utilizando o seguinte comando:

a. `cd objectgridRoot/bin`

b.

```
java -classpath ../lib/objectgrid.jar;../applib/secsample.jar
  com.ibm.websphere.objectgrid.security.sample.guide.SecureSimpleApp
  ../security/client.properties manager manager1
```

O arquivo `secsample.jar` contém a classe SimpleApp.

O SecureSimpleApp utiliza três parâmetros que são fornecidos na lista a seguir:

a. O arquivo `../security/client.properties` está no arquivo de propriedades de segurança do cliente.

b. `manager` é o ID do usuário.

c. `manager1` é a senha.

Após executar a classe, o resultado é a seguinte saída:

O nome do cliente para o ID 0001 é fName lName.

Também é possível usar o xsadmin para mostrar os tamanhos de mapa da grade "accounting".

- Navegue até o diretório objectgridRoot/bin
- Use o comando xsadmin com a opção -mapSizes da seguinte forma.
  - UNIX Linux `xsadmin.sh -g accounting -m mapSet1 -username manager -password manager1 -mapSizes`
  - Windows `xsadmin.bat -g accounting -m mapSet1 -username manager -password manager1 -mapSizes`

  A seguinte saída será exibida.

  ```
  This administrative utility is provided as a sample only and is not to
  be considered a fully supported component of the WebSphere eXtreme
  Scale product.
  Connecting to Catalog service at localhost:1099
  ********** Displaying Results for Grid - accounting, MapSet - mapSet1
  **********
  *** Listing Maps for c0 ***
  Map Name: customer Partition #: 0 Map Size: 1 Shard Type: Primary
  Server Total: 1
  Total Domain Count: 1
  ```

Agora é possível usar o comando stopOgServer para parar o processo do servidor de contêiner ou do serviço de catálogo. Porém, é necessário fornecer um arquivo de configuração de segurança. O arquivo de propriedades do cliente de amostra define as seguintes duas propriedades para gerar uma credencial userID/password (manager/manager1).

```
credentialGeneratorClass=com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator
credentialGeneratorProps=manager manager1
```

Pare o contêiner c0 com o seguinte comando.

- UNIX Linux `stopOgServer.sh c0 -catalogServiceEndPoints localhost:2809 -clientSecurityFile ..\security\client.properties`
- Windows `stopOgServer.bat c0 -catalogServiceEndPoints localhost:2809 -clientSecurityFile ..\security\client.properties`

Se você não fornecer a opção -clientSecurityFile, ocorrerá uma exceção com a seguinte mensagem.

```
>> SERVER (id=39132c79, host=9.10.86.47) TRACE START:
>> org.omg.CORBA.NO_PERMISSION: O servidor requer uma autenticação de
credencial mas não há nenhum contexto de segurança a partir do cliente.
Isso geralmente acontece quando o cliente não transmite uma credencial
para o servidor.
vmcid: 0x0
código secundário: 0
completed: No
```

Também é possível encerrar o servidor de catálogos usando o seguinte comando. Porém, se você desejar continuar tentando a próxima etapa do tutorial, poderá deixar que o servidor de catálogo permaneça em execução.

- UNIX Linux `stopOgServer.sh catalogServer -catalogServiceEndPoints localhost:2809 -clientSecurityFile ..\security\client.properties`
- Windows `stopOgServer.bat catalogServer -catalogServiceEndPoints localhost:2809 -clientSecurityFile ..\security\client.properties`

Se você encerrar o servidor de catálogos, a seguinte saída será exibida.

```
CWOBJ2512I: ObjectGrid server catalogServer stopped
```

Agora, você tornou seu sistema parcialmente seguro com sucesso, ativando a autenticação. Você configurou o servidor para conexão no registro do usuário, configurou o cliente para fornecer credenciais do cliente e alterou o arquivo de propriedades do cliente e o arquivo XML do cluster para ativar autenticação.

Se você fornecer uma senha inválida, verá uma exceção que informa que o nome de usuário ou a senha não está correto.

Para obter mais detalhes sobre a autenticação do cliente, consulte as informações sobre a autenticação do aplicativo cliente no *Guia de Administração*.

Próxima etapa do tutorial

Tutorial de Segurança do Java SE - Etapa 3

Depois de autenticar um cliente, como na etapa anterior, é possível fornecer privilégios de segurança através dos mecanismos de autorização do eXtreme Scale.

Antes de Iniciar

Certifique-se de ter concluído o "Tutorial de Segurança do Java SE - Etapa 2" na página 162 antes de continuar com esta tarefa.

Por Que e Quando Desempenhar Esta Tarefa

A etapa anterior deste tutorial demonstrou como ativar a autenticação em um cluster do eXtreme Scale. Como resultado, nenhum cliente não autenticado pode se conectar a seu servidor e submeter pedidos para seu sistema. Entretanto, todo cliente autenticado tem a mesma permissão ou privilégios para o servidor, como de leitura, gravação ou exclusão de dados armazenados nos mapas do ObjectGrid. Os clientes também podem emitir qualquer tipo de consulta. Esta seção demonstra como utilizar a autorização do eXtreme Scale para conceder vários privilégios de usuários autenticados.

Semelhante a vários outros sistemas, o eXtreme Scale adota um mecanismo de autorização baseado em permissão. O WebSphere eXtreme Scale possui diferentes categorias de permissão representadas por diferentes classes de permissão. Este tópico descreve o MapPermission. Para obter a categoria completa de permissões, consulte a Referência de Autorização do Cliente.

No eXtreme Scale, a classe com.ibm.websphere.objectgrid.security.MapPermission representa permissões para os recursos do eXtreme Scale, especialmente os métodos das interfaces ObjectMap ou JavaMap. O WebSphere eXtreme Scale define as seguintes cadeias de permissão para acessar os métodos de ObjectMap e de JavaMap:

- read: Concede permissão para ler os dados do mapa.
- write: Concede permissão para atualizar os dados no mapa.
- insert: Concede permissão para inserir os dados no mapa.
- remove: Concede permissão para remover os dados do mapa.

- invalidate: Concede permissão para invalidar os dados do mapa.
- all: Concede todas as permissões para ler, gravar, inserir, remover e invalidar.

A autorização ocorre quando um cliente chama um método de ObjectMap ou JavaMap. O tempo de execução do eXtreme Scale verifica diferentes permissões do mapa para métodos diferentes. Se as permissões requeridas não forem concedidas ao cliente, isso resultará em um AccessControlException.

Este tutorial demonstra como utilizar a autorização JAAS para conceder acessos do mapa de autorização para diferentes usuários.

1. **Ativar a autorização do eXtreme Scale** Para ativar a autorização no ObjectGrid. É necessário configurar securityEnabled como true para tal ObjectGrid particular no arquivo XML. A segurança no ObjectGrid significa autorização. Utilize os seguintes comandos para criar um novo XML do ObjectGrid com a segurança ativada.
   a. cd objectgridRoot/bin
   b. cp SimpleApp.xml SecureSimpleApp.xml

   Em seguida, inclua securityEnabled=″true″ no nível do ObjectGrid como mostra o seguinte XML:

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
      xmlns="http://ibm.com/ws/objectgrid/config">
       <objectGrids>
           <objectGrid name="accounting" securityEnabled="true">
               <backingMap name="customer" readOnly="false" copyKey="true"/>
           </objectGrid>
       </objectGrids>
   </objectGridConfig>
   ```

2. **Definir política de autorização** Lembre-se da seção de autenticação pré-cliente da etapa anterior. Criamos três usuários no key store: cashier, manager e administrator. Neste exemplo, mostraremos que o usuário ″cashier″ possui apenas permissões de leitura para todos os mapas, e o usuário ″manager″ possui todas as permissões. A autorização JAAS é usada neste exemplo. A autorização JAAS utiliza o arquivo de políticas de autorização para conceder permissões aos principals. O seguinte arquivo og_auth.policy é definido no diretório desegurança:

   ```
   grant codebase "http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction"
       principal javax.security.auth.x500.X500Principal "CN=cashier,O=acme,OU=OGSample" {
       permission com.ibm.websphere.objectgrid.security.MapPermission "accounting.*", "read ";
   };

   grant codebase "http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction"
       principal javax.security.auth.x500.X500Principal "CN=manager,O=acme,OU=OGSample" {
       permission com.ibm.websphere.objectgrid.security.MapPermission "accounting.*", "all";
   };
   ```

   Nota:

   - O código base ″http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction″ é uma URL reservada especialmente para o ObjectGrid. Todas as permissões do ObjectGrid concedidas a principals devem utilizar esse código base especial.
   - A primeira instrução concede permissão de mapa ″read″ ao principal ″CN=cashier,O=acme,OU=OGSample″, assim o cashier terá permissão, mas apenas de leitura do mapa para todos os mapas na contabilidade do ObjectGrid.

- A segunda instrução concede permissão de mapa ″all″ para o principal ″CN=manager,O=acme,OU=OGSample″, assim o cashier terá todas as permissões aos mapas na contabilidade do ObjectGrid.

Agora você pode ativar um servidor com uma política de autorização. O arquivo da política de autorização JAAS pode ser configurando utilizando a propriedade -D padrão: -Djava.security.auth.policy=../security/ogAuth.policy

3. **Execute o aplicativo**

   Depois de criar os arquivos acima, será possível executar o aplicativo.

   Utilize os seguintes comandos para iniciar o servidor de catálogos. Para obter mais informações sobre o início do serviço de catálogo, consulte as informações sobre o início de um serviço de catálogo no *Guia de Administração*.

   a. Navegue até o diretório bin: `cd objectgridRoot/bin`

   b. Inicie o servidor de catálogos.

      - UNIX Linux
      ```
      startOgServer.sh catalogServer
      -clusterSecurityFile ../security/security.xml -serverProps
      ../security/server.properties -jvmArgs
      -Djava.security.auth.login.config="../security/og_jaas.config"
      ```
      - Windows
      ```
      startOgServer.bat catalogServer -clusterSecurityFile
      ../security/security.xml -serverProps ../security/server.properties
      -jvmArgs -Djava.security.auth.login.config="../security/
      og_jaas.config"
      ```

      Os arquivos `security.xml` e `server.properties` foram criados na etapa anterior deste tutorial.

      Em seguida, ative um servidor de contêiner seguro utilizando o seguinte script:

   c. Navegue até o diretório bin novamente.

   d.

      - UNIX Linux
      ```
      # startOgServer.sh c0 -objectGridFile
      ../xml/SecureSimpleApp.xml -deploymentPolicyFile
      ../xml/SimpleDP.xml -catalogServiceEndpoints localhost:2809
      -serverProps ../security/server.properties -jvmArgs
      -Djava.security.auth.login.config="../security/og_jaas.config"
      -Djava.security.auth.policy="../security/og_auth.policy"
      ```
      - Windows
      ```
      startOgServer.bat c0 -objectGridFile ../xml/
      SecureSimpleApp.xml -deploymentPolicyFile ../xml/SimpleDP.xml
      -catalogServiceEndpoints localhost:2809 -serverProps
      ../security/server.properties -jvmArgs
      -Djava.security.auth.login.config="../security/og_jaas.config"
      -Djava.security.auth.policy="../security/og_auth.policy"
      ```

   Observe as seguintes diferenças do comando para iniciar o servidor de contêiner anterior:

   - Use `SecureSimpleApp.xml` em vez de `SimpleApp.xml`
   - Incluir outro -Djava.security.auth.policy para configurar o arquivo de políticas de autorização ao processo do servidor de contêiner.

   Agora, utilizamos o mesmo comando como na etapa anterior do tutorial:

   a. Navegue até o diretório bin conforme acima.

   b.
   ```
   java -classpath ../lib/objectgrid.jar;../applib/secsample.jar
   com.ibm.websphere.objectgrid.security.sample.guide.SecureSimpleApp
   ../security/client.properties manager manager1
   ```

Como o usuário "manager" possui todas as permissões para mapas no ObjectGrid de contabilidade, o aplicativo é executado apropriadamente.

Agora, no lugar de utilizar o usuário "manager", utilizamos o usuário "cashier" para ativar o aplicativo cliente.

c. Navegue até o diretório bin uma vez mais.

d. `java -classpath ../lib/objectgrid.jar;../applib/secsample.jar com.ibm.ws.objectgrid.security.sample.guide.SecureSimpleApp ../security/client.properties cashier cashier1`

Resulta na seguinte exceção:

```
Exception in thread "P=387313:O=0:CT" com.ibm.websphere.objectgrid.TransactionException:
rolling back transaction, see caused by exception
 at com.ibm.ws.objectgrid.SessionImpl.rollbackPMapChanges(SessionImpl.java:1422)
  at com.ibm.ws.objectgrid.SessionImpl.commit(SessionImpl.java:1149)
  at com.ibm.ws.objectgrid.SessionImpl.mapPostInvoke(SessionImpl.java:2260)
  at com.ibm.ws.objectgrid.ObjectMapImpl.update(ObjectMapImpl.java:1062)
  at com.ibm.ws.objectgrid.security.sample.guide.SimpleApp.run(SimpleApp.java:42)
 at com.ibm.ws.objectgrid.security.sample.guide.SecureSimpleApp.main(SecureSimpleApp.java:27)
Caused by: com.ibm.websphere.objectgrid.ClientServerTransactionCallbackException:
   Client Services - received exception from remote server:
     com.ibm.websphere.objectgrid.TransactionException: transaction rolled back, see caused by Throwable
        at com.ibm.ws.objectgrid.client.RemoteTransactionCallbackImpl.processReadWriteResponse(
            RemoteTransactionCallbackImpl.java:1399)
        at com.ibm.ws.objectgrid.client.RemoteTransactionCallbackImpl.processReadWriteRequestAndResponse(
            RemoteTransactionCallbackImpl.java:2333)
        at com.ibm.ws.objectgrid.client.RemoteTransactionCallbackImpl.commit(RemoteTransactionCallbackImpl.java:557)
        at com.ibm.ws.objectgrid.SessionImpl.commit(SessionImpl.java:1079)
        ... 4 more
Caused by: com.ibm.websphere.objectgrid.TransactionException: transaction rolled back, see caused by Throwable
        at com.ibm.ws.objectgrid.ServerCoreEventProcessor.processLogSequence(ServerCoreEventProcessor.java:1133)
        at com.ibm.ws.objectgrid.ServerCoreEventProcessor.processReadWriteTransactionRequest
        (ServerCoreEventProcessor.java:910)
        at com.ibm.ws.objectgrid.ServerCoreEventProcessor.processClientServerRequest(ServerCoreEventProcessor.java:1285)

        at com.ibm.ws.objectgrid.ShardImpl.processMessage(ShardImpl.java:515)
        at com.ibm.ws.objectgrid.partition.IDLShardPOA._invoke(IDLShardPOA.java:154)
        at com.ibm.CORBA.poa.POAServerDelegate.dispatchToServant(POAServerDelegate.java:396)
        at com.ibm.CORBA.poa.POAServerDelegate.internalDispatch(POAServerDelegate.java:331)
        at com.ibm.CORBA.poa.POAServerDelegate.dispatch(POAServerDelegate.java:253)
        at com.ibm.rmi.iiop.ORB.process(ORB.java:503)
        at com.ibm.CORBA.iiop.ORB.process(ORB.java:1553)
        at com.ibm.rmi.iiop.Connection.respondTo(Connection.java:2680)
        at com.ibm.rmi.iiop.Connection.doWork(Connection.java:2554)
        at com.ibm.rmi.iiop.WorkUnitImpl.doWork(WorkUnitImpl.java:62)
        at com.ibm.rmi.iiop.WorkerThread.run(ThreadPoolImpl.java:202)
        at java.lang.Thread.run(Thread.java:803)
Caused by: java.security.AccessControlException: Access denied (
   com.ibm.websphere.objectgrid.security.MapPermission accounting.customer write)
        at java.security.AccessControlContext.checkPermission(AccessControlContext.java:155)
        at com.ibm.ws.objectgrid.security.MapPermissionCheckAction.run(MapPermissionCheckAction.java:141)
        at java.security.AccessController.doPrivileged(AccessController.java:275)
        at javax.security.auth.Subject.doAsPrivileged(Subject.java:727)
        at com.ibm.ws.objectgrid.security.MapAuthorizer$1.run(MapAuthorizer.java:76)
        em java.security.AccessController.doPrivileged(AccessController.java:242)
        at com.ibm.ws.objectgrid.security.MapAuthorizer.check(MapAuthorizer.java:66)
        at com.ibm.ws.objectgrid.security.SecuredObjectMapImpl.checkMapAuthorization(SecuredObjectMapImpl.java:429)
        at com.ibm.ws.objectgrid.security.SecuredObjectMapImpl.update(SecuredObjectMapImpl.java:490)
        at com.ibm.ws.objectgrid.SessionImpl.processLogSequence(SessionImpl.java:1913)
        at com.ibm.ws.objectgrid.SessionImpl.processLogSequence(SessionImpl.java:1805)
        at com.ibm.ws.objectgrid.ServerCoreEventProcessor.processLogSequence(ServerCoreEventProcessor.java:1011)
        ... 14 more
```

Isso porque o usuário "cashier" não possui permissão de gravação, então ele não pode atualizar o cliente do mapa.

Agora, o seu sistema suporta autorização. É possível definir políticas de autorização para conceder diferentes permissões a diferentes usuários. Para obter mais informações sobre a autorização, consulte as informações sobre a autorização do aplicativo cliente no *Guia de Programação*.

Próxima etapa

Tutorial de Segurança do Java SE - Etapa 4

A seguinte etapa explica como uma camada de segurança pode ser ativada para comunicação entre os terminais do ambiente.

Antes de Iniciar

Certifique-se de ter concluído do "Tutorial de Segurança do Java SE - Etapa 3" na página 168 antes de continuar com esta tarefa.

Por Que e Quando Desempenhar Esta Tarefa

A topologia do eXtreme Scale suporta Transport Layer Security/Secure Sockets Layer (TLS/SSL) para comunicação segura entre terminais do ObjectGrid (cliente, servidores de contêineres e servidores de catálogos). Esta etapa do tutorial é baseada nas etapas anteriores para ativar a segurança do transporte.

1. **Criar chaves TLS/SSL e key stores**

   Para ativar a segurança do transporte, é necessário criar um key store e um trust store. Este exercício cria apenas um par de chave e trust-store. Estes armazéns são utilizados para clientes do ObjectGrid, servidores de contêineres e servidores de catálogos, e são criados com o JDK keytool.

   - *Criar uma chave privada no key store*

     ```
     keytool -genkey -alias ogsample -keystore key.jks -storetype JKS
     -keyalg rsa -dname "CN=ogsample, OU=Your Organizational Unit, O=Your
     Organization, L=Your City, S=Your State, C=Your Country" -storepass
     ogpass -keypass ogpass -validity 3650
     ```

     Utilizando este comando, um key store key.jks é criado com uma chave "ogsample" armazenada nele. Esta key store key.jks será utilizada como o key store SSL.

   - *Exportar o certificado público*

     ```
     keytool -export -alias ogsample -keystore key.jks -file temp.key
     -storepass ogpass
     ```

     Utilizando este comando, o certificado público da chave "ogsample" é extraído e armazenado no arquivo temp.key.

   - *Importar o certificado público do cliente para o trust store*

     ```
     keytool -import -noprompt -alias ogsamplepublic -keystore trust.jks
     -file temp.key -storepass ogpass
     ```

     Utilizando este comando, o certificado público foi incluído no key store trust.jks. Este trust.jks é utilizado como o trust store SSL.

2. **Configurando arquivos de propriedades do ObjectGrid**

   Neste etapa, é necessário configurar os arquivos de propriedades do ObjectGrid para ativar a segurança do transporte.

   Primeiro, copie os arquivos key.jks e trust.jks no diretório objectgridRoot/security.

   As seguintes propriedades foram configuradas no arquivo client.properties e server.properties.

   ```
   transportType=SSL-Required

   alias=ogsample
   contextProvider=IBMJSSE2
   protocol=SSL
   keyStoreType=JKS
   keyStore=../security/key.jks
   ```

```
keyStorePassword=ogpass
trustStoreType=JKS
trustStore=../security/trust.jks
trustStorePassword=ogpass
```

transportType: O valor de transportType é configurado como ″SSL-Required″, o que significa que o transporte requer SSL. Assim, todos os terminais do ObjectGrid (clientes, servidores de catálogos e servidores de contêineres) devem ter a configuração SSL definida e toda a comunicação de transporte será criptografada.

As outras propriedades são utilizadas para definir as configurações SSL. Consulte as informações sobre a segurança de camada de transporte e sobre a camada de soquetes seguros no *Guia de Administração* para obter uma explicação detalhada. Certifique-se de seguir as seguintes instruções neste tópico para atualizar o arquivo orb.properties.

Certifique-se de seguir essa página para atualizar o arquivo `orb.properties`.

No arquivo `server.properties`, é necessário incluir uma propriedade adicional clientAuthentication e configurá-la para false. No lado do servidor, não é necessário confiar o cliente.

```
clientAuthentication=false
```

3. **Execute o aplicativo**

   Os comandos são os mesmos que os comandos no tópico “Tutorial de Segurança do Java SE - Etapa 3” na página 168.

   Utilize os seguintes comandos para iniciar um servidor de catálogos.

   a. Navegue até o diretório bin: `cd objectgridRoot/bin`

   b. Inicie o servidor de catálogos:

   - Linux UNIX

     ```
     startOgServer.sh catalogServer -clusterSecurityFile ../security/security.xml
     -serverProps ../security/server.properties -JMXServicePort 11001
     -jvmArgs -Djava.security.auth.login.config="../security/og_jaas.config"
     ```

   - Windows

     ```
     startOgServer.bat catalogServer -clusterSecurityFile ../security/security.xml
     -serverProps ../security/server.properties -JMXServicePort 11001 -jvmArgs
     -Djava.security.auth.login.config="../security/og_jaas.config"
     ```

     Os arquivos `security.xml` e `server.properties` foram criados na página “Tutorial de Segurança do Java SE - Etapa 2” na página 162.

     Use a opção -JMXServicePort para especificar explicitamente a porta JMX para o servidor. Essa opção é necessária para usar o comando xsadmin.

     Execute um servidor de contêiner ObjectGrid:

   c. Navegue até o diretório bin novamente: `cd objectgridRoot/bin`

   d.

   - Linux UNIX

     ```
     startOgServer.sh c0 -objectGridFile ../xml/SecureSimpleApp.xml
     -deploymentPolicyFile ../xml/SimpleDP.xml -catalogServiceEndpoints
     localhost:2809 -serverProps ../security/server.properties
     -JMXServicePort 11002 -jvmArgs
     -Djava.security.auth.login.config="../security/og_jaas.config"
     -Djava.security.auth.policy="../security/og_auth.policy"
     ```

   - Windows

     ```
     startOgServer.bat c0 -objectGridFile ../xml/SecureSimpleApp.xml
     -deploymentPolicyFile ../xml/SimpleDP.xml
     -catalogServiceEndpoints localhost:2809
     -serverProps ../security/server.properties -JMXServicePort 11002
     -jvmArgs -Djava.security.auth.login.config="../security/og_jaas.config"
     -Djava.security.auth.policy="../security/og_auth.policy"
     ```

   Observe as seguintes diferenças do comando para iniciar o servidor de contêiner anterior:

- Utilizar SecureSimpleApp.xml em vez de SimpleApp.xml
- Incluir outro -Djava.security.auth.policy para configurar o arquivo de políticas de autorização ao processo do servidor de contêiner.

Execute o seguinte comando para autenticação de cliente:

a. `cd objectgridRoot/bin`

b.
```
javaHome/java -classpath ../lib/objectgrid.jar;../applib/secsample.jar
com.ibm.websphere.objectgrid.security.sample.guide.SecureSimpleApp
../security/client.properties manager manager1
```

Como o usuário "manager" tem permissão para todos os mapas no ObjectGrid de contabilidade, o aplicativo é executado com êxito.

Também é possível usar o xsadmin para mostrar os tamanhos de mapa da grade "accounting".

- Navegue até o diretório `objectgridRoot/bin`
- Use o comando `xsadmin` com a opção -mapSizes da seguinte forma.
  - UNIX Linux
    ```
    xsadmin.sh -g accounting -m mapSet1 -mapsizes -p 11001 -ssl
    -trustpath ..\security\trust.jks -trustpass ogpass -trusttype jks
    -username manager -password manager1
    ```
  - Windows
    ```
    xsadmin.bat -g accounting -m mapSet1 -mapsizes -p 11001 -ssl
    -trustpath ..\security\trust.jks -trustpass ogpass -trusttype jks
    -username manager -password manager1
    ```

  Observe que especificamos a porta JMX do serviço de catálogo usando -p 11001 aqui.

  A seguinte saída será exibida.

```
This administrative utility is provided as a sample only and is not to
be considered a fully supported component of the WebSphere eXtreme Scale product.
Connecting to Catalog service at localhost:1099
*********** Displaying Results for Grid - accounting, MapSet - mapSet1 ***********
*** Listing Maps for c0 ***
Map Name: customer Partition #: 0 Map Size: 1 Shard Type: Primary
Server Total: 1
Total Domain Count: 1
```

Executando o aplicativo com um key store incorreto

Se o seu trust store não contiver o certificado público da chave privada no key store, será obtida uma exceção reclamando que a chave pode não ser confiável.

Para mostrar isso, crie outro key store, key2.jks.

```
keytool -genkey -alias ogsample -keystore key2.jks -storetype JKS
-keyalg rsa -dname "CN=ogsample, OU=Your Organizational Unit, O=Your
Organization, L=Your City, S=Your State, C=Your Country" -storepass
ogpass -keypass ogpass -validity 3650
```

Em seguida, modifique o arquivo server.properties para que o keyStore aponte para esse novo armazenamento de chaves key2.jks:

```
keyStore=../security/key2.jks
```

Execute o seguintes comando para iniciar o servidor de catálogos:

a. Navegue até o bin: `cd objectgridRoot/bin`

b. Inicie o servidor de catálogos:

Linux UNIX

```
startOgServer.sh c0 -objectGridFile ../xml/SecureSimpleApp.xml
-deploymentPolicyFile ../xml/SimpleDP.xml
-catalogServiceEndpoints localhost:2809
-serverProps ../security/server.properties -jvmArgs
-Djava.security.auth.login.config="../security/og_jaas.config"
-Djava.security.auth.policy="../security/og_auth.policy"
```

Windows

```
startOgServer.bat c0 -objectGridFile ../xml/SecureSimpleApp.xml
-deploymentPolicyFile ../xml/SimpleDP.xml
-catalogServiceEndpoints localhost:2809
 -serverProps ../security/server.properties -jvmArgs
-Djava.security.auth.login.config="../security/og_jaas.config"
-Djava.security.auth.policy="../security/og_auth.policy"
```

A seguinte exceção será exibida:

```
Caused by: com.ibm.websphere.objectgrid.ObjectGridRPCException:
    com.ibm.websphere.objectgrid.ObjectGridRuntimeException:
        SSL connection fails and plain socket cannot be used.
```

Por fim, altere o arquivo `server.properties` de volta para usar o arquivo `key.jks`.

Capítulo 9. Glossário

Este glossário inclui termos e definições para o WebSphere eXtreme Scale.

As referências cruzadas a seguir são utilizadas nesse glossário:

1. Consulte encaminha o leitor para um termo sinônimo preferencial, ou um acrônimo ou uma abreviação para a forma completa definida.
2. Consulte também encaminha o leitor a um termo relacionado ou constrastante.

Para visualizar glossários de outros produtos IBM, vá para www.ibm.com/software/globalization/terminology.

acordo de nível de serviço (SLA). Um contrato entre um cliente e um provedor de serviços que especifica as expectativas para o nível de serviço em relação à disponibilidade, ao desempenho e a outros objetivos mensuráveis.

administrador. Uma pessoa responsável pelas tarefas administrativas como autorização de acesso e gerenciamento de conteúdo. Os administradores também podem alterar os níveis de concessão de autoridade aos usuários.

administrador de segurança. A pessoa que controla o acesso a dados comerciais e funções de programas.

Afinidade de sessão. Um método para configurar aplicativos em que um cliente sempre está conectado ao mesmo servidor. Essas configurações desativam o gerenciamento da carga de trabalho após uma conexão inicial, forçando um pedido do cliente a sempre ir para o mesmo servidor.

Agente. Um programa que executa uma ação em nome de um usuário ou outro programa sem intervenção do usuário ou em um planejamento regular e relata os resultados para o usuário ou programa.

Agente do nó. Um agente administrativo que gerencia todos os servidores de aplicativos em um nó e representa o nó na célula de gerenciamento.

agente iterativo. Uma classe ou construção que é usada para navegar através de uma coleção de objetos, um por vez.

alias de autenticação. Um alias que autoriza acesso aos adaptadores de recursos e origens de dados. Um alias de autenticação contém dados de autenticação, incluindo um ID do usuário e senha.

alta disponibilidade (HA). Pertencente a um sistema em cluster que é reconfigurado quando ocorrem falhas de nó ou de daemon, para que as cargas de trabalho possam ser redistribuídas aos nós restantes no cluster.

ambiente. Uma coleta de recursos lógicos e físicos nomeada usada para suportar o desempenho de uma função.

ambiente de implementação. Um conjunto de clusters, servidores e middleware configurados que colaboram com o fornecimento de um ambiente para hospedar os módulos de software. Por exemplo, um ambiente de implementação pode incluir um host para destinos de mensagem, um processador ou separador de eventos de negócio e programas administrativos.

analista de sistemas. Um especialista responsável pela conversão de requisitos de negócios em definições e soluções do sistema.

APAR. Consulte relatório de análise de programa autorizado.

API. Consulte interface de programação de aplicativos.

API (Application Programming Interface). Uma interface que permite que um programa de aplicativo gravado em uma linguagem de alto nível utilize dados ou funções específicos do sistema operacional ou de um outro programa.

API JavaMail. Uma plataforma e quadro independente do protocolo para construir aplicativos de clientes de correio com base em Java.

aplicativo. Um ou mais programas de computador ou componentes de software que fornecem uma função em suporte direto de um processo de negócios específico ou processos.

aplicativo cliente. Um aplicativo, em execução em uma estação de trabalho e vinculado a um cliente, que fornece ao aplicativo acesso aos serviços de fila em um servidor.

Aplicativo cliente thin. Um tempo de execução de aplicativo Java leve e que pode ser obtido por download capaz de interagir com beans corporativos.

Aplicativo Java EE. Qualquer unidade implementável de funcionalidade Java EE. Esta unidade pode ser um único módulo ou um grupo de módulos empacotados em um arquivo EAR (Enterprise Archive) com um descritor de implementação de aplicativos Java EE. (Sun)

área do editor. Em produtos Eclipse e baseados no Eclipse, a área na janela do ambiente de trabalho em que os arquivos são abertos para edição.

Armazenamento de certificados de coleta. Uma coleta de certificados intermediários ou CRLs (Certificate Revocation Lists) que são utilizadas por um caminho de certificado para construir uma cadeia de certificados para validação.

armazenamento de dados persistente. Um armazenamento não-volátil para dados do evento, como um sistema de banco de dados, que é mantido entre limites de sessão e que continua a existir após a execução do programa ou processo que o criou.

arquivo de armazenamento de dados. Um arquivo de banco de dados de chave que contém as chaves públicas para uma entidade confiável.

arquivo de classe. Um arquivo de origem Java compilado.

arquivo de definição de construção. Um arquivo XML que identifica componentes e características de um pacote de instalação customizada (CIP).

arquivo de exportação.

1. Um arquivo criado durante o processo de desenvolvimento para operações de entrada que contém as definições de configuração para o processamento de entrada.

2. Os dados contendo arquivos que foram exportados.

arquivo de Java. Um formato de arquivo compactado para armazenar todos os recursos necessários para instalar e executar um programa Java em um único arquivo. Consulte também arquivamento Web, archive corporativo.

arquivo delimitado por vírgula. Um arquivo cujos registros contêm campos que são separados por uma vírgula.

Arquivo JAR. Um arquivo archive Java. Consulte também arquivamento Web, archive corporativo.

Arquivo JAR EJB. Um Java Archive que contém um módulo EJB. (Sun)

Arquivo Java. Um arquivo fonte editável (com extensão .java) que pode ser compilado em bytecode (um arquivo .class).

Arquivo JSP. Um arquivo HTML de script que possui uma extensão .jsp e que permite a inclusão de conteúdo dinâmico em páginas da Web. Um arquivo JSP pode ser diretamente solicitado como um URL, chamado por um servlet, ou chamado a partir de uma página HTML.

assíncrono. Relativo a eventos que não estão sincronizados no tempo ou não ocorrem em intervalos de tempo regulares ou previsíveis.

atributo global. No XML, um atributo declarado como um filho do elemento esquema, em vez de parte de uma definição de tipo complexo. Atributos globais podem ser referidos em um ou mais modelos de conteúdo utilizando o atributo ref.

Autenticação. Um serviço de segurança que prova que o usuário de um sistema de computador é de fato quem ele afirma ser. Os mecanismos comuns para implementação deste serviço são as senhas e as assinaturas digitais. A autenticação é diferente da autorização; a autenticação não está relacionada a conceder ou negar acesso aos recursos do sistema.

authorized program analysis report (APAR). Um pedido de correção de um defeito em um release suportado de um programa fornecido pela IBM.

auto-inicialização. Um pequeno programa que carrega programas maiores durante a inicialização do sistema.

autônomos. Independente de qualquer outro dispositivo, programa ou sistema. Em uma rede ou ambiente, uma máquina independente acessa todos os recursos necessários localmente.

Autorização. O processo de concessão a um usuário, sistema ou processo de acesso completo ou restrito a um objeto, recurso ou função.

Balanceamento de carga. A monitoração de servidores de aplicativos e o gerenciamento da carga de trabalho nos servidores. Se um servidor exceder sua carga de trabalho, os pedidos serão encaminhados a outro servidor com mais capacidade.

banco de dados local. Um banco de dados localizado na estação de trabalho em uso.

Bandeja. Um encadeamento que aguarda conexão.

Bean. Uma definição ou instância de um componente JavaBeans. Consulte também JavaBeans, enterprise bean.

Bean corporativo. Um componente que implementa uma tarefa de negócios ou uma entidade de negócios e reside em um contêiner EJB. Beans de entidade, beans de sessão e beans orientados a mensagens são todos beans corporativos. (Sun) Consulte também bean.

bean de comando. Um proxy que pode chamar uma operação única utilizando um método execute().

Bean de entidade. Na programação EJB, um enterprise bean que representa dados persistentes mantidos em um banco de dados. Cada bean de entidade transporta sua própria identidade. (Sun)

Bean Scripting Framework. Uma arquitetura para incorporar funções da linguagem de script a aplicativos Java.

biblioteca.

1. Uma coleta de elementos de modelo, incluindo itens de negócios, processos, tarefas, recursos e organizações.

2. Um projeto que é utilizado para desenvolvimento, gerenciamento de versão, e organização de recursos compartilhados. Apenas um subconjunto de tipos de artefatos pode ser criado e armazenado em uma biblioteca, como objetos de negócios e interfaces.

bifurcação. Um elemento de processo que faz cópias de sua entrada e as redireciona por vários caminhos de processamento paralelamente.

bloqueio. Um meio de evitar que mudanças não confirmadas feitas por um processo aplicativo sejam percebidas por outro processo aplicativo e para evitar que um processo aplicativo atualize dados que estão sendo acessados por outro processo. Um bloqueio garante a integridade dos dados evitando que usuários simultâneos acessem dados inconsistentes.

bloqueio atualizável. Um bloqueio que identifica o intento de atualizar uma entrada armazenada em cache ao usar um bloqueio pessimista.

bloqueio compartilhado. Um bloqueio que limita processos aplicativos que executam simultaneamente às operações somente leitura nos dados do banco de dados.

bloqueio pessimista. Uma estratégia de bloqueio por meio da qual um bloqueio é mantido entre o horário em que uma linha é selecionada e o horário em que uma atualização pesquisada ou operação de exclusão é tentada nesta linha.

bloqueio restrito. Um bloqueio que evita que os processos de aplicativo que executam simultaneamente acessem os dados do banco de dados. Consulte também bloqueio compartilhado.

BMP. Consulte persistência gerenciada por bean.

BMP (Bean-managed Persistence). O mecanismo através do qual a transferência de dados entre as variáveis de um bean de entidade e um gerenciador de recursos é gerenciada pelo bean de entidade. (Sun)

BMT. Consulte transação gerenciada por bean.

BMT (Transação Gerenciada por Bean). O recurso do bean de sessão, servlet ou componente do aplicativo cliente que gerencia suas próprias transações diretamente, em vez de fazê-lo através de um contêiner.

bootstrapping. O processo pelo qual uma referência inicial do serviço de nomenclatura é obtido. A definição de bootstrap e o nome do host formam o contexto inicial das referências de JNDI (Java Naming and Directory Interface).

bytecode. Código independente da máquina gerado pelo compilador Java e executado pelo interpretador Java. (Sun)

cache coerente. O cache que mantém a integridade para que todos os clientes vejam os mesmos dados.

Cache dinâmico. A consolidação de várias atividades de cache, inclusive servlets, serviços da Web e comandos do WebSphere em um serviço no qual essas atividades compartilham parâmetros de configuração e funcionam juntas para melhorar o desempenho.

cache read-through. Um cache escasso que carrega entradas de dados por chave à medida que são solicitadas. Quando os dados não podem ser localizados no cache, os dados ausentes são recuperados com o utilitário de carga, que carrega os dados do repositório de dados de backend e insere os dados no cache.

cache write-behind. Um cache que grava assincronicamente cada operação de gravação para o banco de dados usando um utilitário de carga.

cache write-through. Um cache que grava sincronicamente cada operação de gravação para o banco de dados usando um utilitário de carga.

Caminho de classe. Uma lista de diretórios e arquivos JAR que contém os arquivos de recursos ou classes Java que um programa pode carregar dinamicamente no tempo de execução.

caminho de construção. O caminho utilizado durante a compilação do código fonte Java para localizar classes referenciadas que residem em outros projetos.

Canal de contêiner da Web. Um tipo de canal dentro de uma cadeia de transporte que cria uma ponte na cadeia de transporte entre um canal de entrada HTTP e um servlet ou mecanismo JSP (JavaServer Pages).

Canal SSL. Um tipo de canal dentro de uma cadeia de transporte que associa um repertório de configuração SSL (Secure Sockets Layer) com a cadeia de transporte.

Canal TCP. Um tipo de canal dentro de uma cadeia de transporte que fornece aplicativos clientes com conexões persistentes em uma LAN (Rede Local).

carregador de classes. Parte da JVM (Java Virtual Machine) que é responsável por localizar e carregar arquivos de classe. Um carregador de classes afeta o pacote dos aplicativos e o comportamento do tempo de execução dos aplicativos empacotados implementados em servidores de aplicativos.

catálogo. Um contêiner que, dependendo do tipo de contêiner, suspende processos, dados, recursos, organizações ou relatórios na árvore de projeto.

categoria. Um contêiner usado em um diagrama de estrutura para agrupar elementos do grupo com base em um atributo compartilhado ou qualidade.

cell.

1. Um grupo de processos gerenciados que são associados ao mesmo gerenciador de implementação e podem incluir grupos principais de alta disponibilidade.

2. Um ou mais processos, cada com componentes de tempo de execução de host. Cada uma tem um ou mais grupos principais denominados.

centro de informações. Uma coleta de informações que fornece suporte para os usuários de um ou mais produtos pode ser ativada separadamente do produto e inclui uma lista de tópicos para navegação e um mecanismo de procura.

Certificado de assinante. A entrada do certificado confiável que normalmente está em um arquivo de armazenamento confiável.

Certificado digital. Um documento eletrônico usado para identificar um indivíduo, um sistema, um servidor, uma empresa ou alguma outra entidade e para associar uma chave pública à entidade. Um certificado digital é emitido por uma autoridade de certificação e é assinado digitalmente por essa autoridade.

chamada. A ativação de um programa ou procedimento.

chassi. A estrutura de metal na qual vários componentes eletrônicos são montados.

chave.

1. Um valor matemático criptografado que é utilizado para atribuir digitalmente, verificar, criptografar ou decriptografar uma mensagem.

2. Informações que caracterizam e identificam de forma exclusiva a entidade de mundo real que está sendo rastreada por um contexto de monitoramento.

Chave primária.

1. Um objeto que identifique unicamente um bean de entidade de um tipo específico.

2. Em um banco de dados relacional, uma chave que identifica exclusivamente uma linha de uma tabela de banco de dados.

ciclo de vida. Uma passagem completa pelas quatro fases de desenvolvimento de software: encetamento, elaboração, construção e transição.

CIP. Consulte o pacote de instalação customizada.

classe. Na programação ou no design orientado pelo objeto, um modelo ou modelo que pode ser utilizado para criar objetos com uma definição, propriedades, operações e comportamento comuns. Um objeto é uma instância de uma classe.

classe de bean. Na programação EJB (Enterprise JavaBeans), uma classe Java que implementa uma classe javax.ejb.EntityBean ou classe javax.ejb.SessionBean.

Classe Java. Uma classe que é gravada na linguagem Java.

classificador. Um atributo especializado usado para agrupamento e processo de codificação por cor dos elementos.

cliente. Um programa de software ou computador que solicita serviços de um servidor. Consulte também host.

Cliente fino. Um cliente com pouco ou nenhum software instalado, mas que tem acesso a software gerenciado e fornecido por servidores de rede conectados a ele. Um cliente thin é uma alternativa a um cliente com plena funcionalidade, como um estação de trabalho.

cliente/servidor. Pertencente ao modelo de interação em processamento de dados distribuídos, no qual um programa em um computador envia um pedido para um programa em outro computador e aguarda uma resposta. O programa solicitante é chamado de cliente; o programa de resposta é chamado de servidor.

Cloudscape. Um ORDBMS (Object-Relational Database Management System) incorporável totalmente Java.

cluster. Um grupo de servidores de aplicativos que colabora com o equilíbrio de carga de trabalho e failover.

Cluster de proxy. Um grupo de servidores de proxy que distribui pedidos HTTP através do cluster.

Cluster de servidores. Um grupo de servidores que geralmente estão em máquinas físicas diferentes e que possuem os mesmos aplicativos neles configurados, mas que operam como um único servidor lógico.

cluster dinâmico. Um cluster de servidor que utiliza pesos para equilibrar as cargas de trabalho de seus membros de cluster dinamicamente, com base nas informações de desempenho coletadas dos membros do cluster.

código de implementação. Código adicional que permite que o código de implementação do bean gravado por um desenvolvedor de aplicativos funcione em um determinado ambiente de tempo de execução EJB. O código de implementação pode ser gerado por ferramentas fornecidas pelo fornecedor de servidor de aplicativos.

coleta de lixo. Uma rotina que pesquisa a memória para reclamar espaço de segmentos do programa ou dados inativos.

componente.

1. Um objeto reutilizável ou programa que executa uma função específica e trabalha com outros componentes e aplicativos.

2. No Eclipse, um ou mais plug-ins que funcionam em conjunto para fornecer um conjunto distinto de funções.

Componente da Web. Um servlet, arquivo JSP (JavaServer Pages), ou arquivo HTML (HyperText Markup Language). Um ou mais componentes da Web formam um módulo da Web.

conflito. Uma condição na qual dois encadeamentos de controle independentes estão bloqueados, cada um aguardando pelo outro tomar uma ação. O conflito geralmente surge da inclusão de mecanismos de sincronização para evitar condições de disputa.

Consulta EJB. Na linguagem de consulta EJB, uma cadeia que contém uma cláusula SELECT opcional, que especifica os objetos EJB a serem retornados, uma cláusula FROM, que nomeia as coletas de beans, uma cláusula WHERE opcional, que contém predicados de procura nas coletas e uma cláusula ORDER BY opcional, que especifica a ordem da coleta de resultados e dos parâmetros de entrada que correspondem aos argumentos do método localizador.

consultar.

1. Uma solicitação de informações de um banco de dados baseada em condições específicas: por exemplo, uma solicitação para uma lista de todos os clientes em uma tabela de clientes cujos saldos sejam maiores que USD1000.

2. Um pedido reutilizável de informações sobre um ou mais elementos de modelo

consulta SQL. Um componente de determinadas instruções SQL que especifica uma tabela de resultados.

Contêiner da Web. Um contêiner que implementa o contrato do componente Web da arquitetura Java EE. (Sun)

Contêiner de EJB. Um contêiner que implementa o contrato do componente EJB da arquitetura Java EE. Esse contrato especifica um ambiente de tempo de execução para os beans corporativos que incluem segurança, conformidade, gerenciamento do ciclo de vida, transação, implementação e outros serviços. (Sun)

contenção de encadeamento. Uma condição na qual um encadeamento está aguardando uma trava ou um objeto que um outro encadeamento possui.

Contexto EJB. Nos beans corporativos, um objeto que permite a um enterprise bean chamar serviços fornecidos pelo contêiner e para obter informações sobre o responsável pela chamada de um método chamado pelo cliente. (Sun)

conversor. Na programação EJB (Enterprise JavaBeans), uma classe que converte uma representação do banco de dados em um tipo de objeto e vice-versa.

correção temporária. Uma correção certificada que geralmente está disponível para todos os clientes entre fix packs, pacotes de atualização ou releases regularmente planejados. Consulte também fix pack.

crawler da Web. Um tipo de crawler que explora a Web recuperando um documento da Web e seguindo os links nesse documento.

Credencial. No quadro JAAS (Java Authentication and Authorization Service), uma classe de assunto que possui atributos relacionados à segurança. Esses atributos podem conter informações utilizadas para autenticar o assunto para novos serviços.

cronômetro. Uma tarefa que produz saída em determinados pontos no tempo.

Customized Installation Package (CIP). Uma imagem de instalação customizada que pode incluir um ou mais pacotes de manutenção, um arquivo arquivado de configuração a partir de um perfil de servidor independente, um ou mais arquivos arquivados corporativos, scripts e outros arquivos que ajudam a customizar a instalação resultante.

dados da hora de construção. Objetos que não são utilizados pelo conversor, como padrões EDI, tipos de documentos de dados orientados por registros e mapas.

daemon. Um programa que é executado de modo não assistido para executar funções contínuas ou periódicas, como controle de rede.

DB2. Uma família de programas licenciados da IBM para gerenciamento de banco de dados relacional.

definição de documento DTD. Uma descrição ou um layout de um documento XML baseado em um DTD XML.

demilitarized zone (DMZ). Uma configuração que inclui múltiplos firewalls para incluir camadas de proteção entre uma intranet corporativa e uma rede pública, como a Internet.

Depósito de informações do provedor. Na programação orientada a objetos, uma classe que é utilizada para criar instâncias de uma outra classe. Uma fábrica é utilizada para isolar a criação de objetos de uma determinada classe em um local para que novas funções possam ser fornecidas sem difundir as alterações do código.

derivação. Na programação orientada a objetos, o refinamento ou a extensão de uma classe em outra.

descoberta automática. A descoberta de artefatos de serviço em um sistema de arquivos, registro externo ou outra origem.

Descrição, Descoberta e Integração Universal (UDDI). Um conjunto de especificações baseadas em padrões que permite às companhias e aos aplicativos localizarem e utilizarem os serviços da Web na Internet, de maneira rápida e fácil.

Descritor de implementação. Um arquivo XML que descreve como implementar um módulo ou aplicativo, especificando opções de configuração e de contêiner. Por exemplo, um descritor de implementação EJB transmite informações para um contêiner de EJB sobre como gerenciar e controlar um enterprise bean.

desenvolvimento de baixo para cima. Nos serviços da Web, o processo de desenvolvimento de um serviço com base em um artefato existente, como um Java bean ou enterprise bean, em vez de um arquivo WSDL (Web Services Description Language).

desserialização. Um método para converter uma variável serializada em dados de objetos.

destino. Um ponto de saída usado para fornecer entregar documentos a um sistema backend ou a um parceiro comercial.

destino de instalação. O sistema no qual os pacotes de instalação selecionados são instalados.

diretório de implementação. O diretório em que a configuração de servidor publicada e o aplicativo da Web estão localizados na máquina em que o servidor de aplicativos está instalado.

diretório LDAP. Um tipo de repositório que armazena informações sobre pessoas, organizações e outros recursos e que é acessado utilizando o protocolo LDAP. As entradas do repositório são organizadas em uma estrutura hierárquica e, em alguns casos, a estrutura hierárquica reflete a estrutura ou a geografia de uma organização.

disparar. Na programação orientada por objetos, causar uma transição de estado.

disponibilidade.

1. A condição que permite aos usuários acessar e usar seus aplicativos e dados.

2. Os períodos de tempo durante os quais um recurso está acessível. Por exemplo, um contratado pode ter uma disponibilidade das 9h às 17h, todos os dias da semana e das 9h às 15h aos sábados.

DMZ. Consulte zona desmilitarizada.

DNS. Consulte Domain Name System.

Document Type Definition (DTD). As regras que especificam a estrutura de uma classe específica de documentos SGML ou XML. O DTD define a estrutura com elementos, atributos e notações e estabelece restrições para como cada elemento, atributo e notação pode ser utilizado na classe de documentos específica.

domain. Um objeto, ícone ou contêiner que contém outros objetos que representam os recursos de um domínio. O objeto de domínio pode ser utilizado para gerenciar esses recursos.

Domain Name System (DNS). O sistema de banco de dados distribuído que mapeia nomes de domínio para endereços IP.

Drop-down. Consulte pull-down.

DTD. Consulte definição de tipo de documento.

EAR. Consulte archive corporativo.

EAR (Enterprise Archive). Um tipo especializado de arquivo JAR, definido pelo padrão Java EE, utilizado para implementar aplicativos Java EE em servidores de aplicativos do Java EE. Um arquivo EAR contém componentes do EJB, um descritor de implementação e arquivos WAR (Web Archive) para aplicativos individuais da Web. Consulte também arquivamento Web.

Eclipse. Uma iniciativa de código aberto que fornece a ISVs e a outros desenvolvedores de ferramentas uma plataforma padrão para o desenvolvimento de ferramentas de desenvolvimento de aplicativos compatíveis com conexão.

edição. Uma geração de implementação sucessiva de um conjunto particular de artefatos com versão.

EJB. Consulte JavaBeans Corporativos.

elemento do componente. Uma entidade em um componente, na qual um ponto de interrupção pode ser definido, como uma atividade ou snippet Java em um processo de negócios, ou uma primitiva ou nó de mediação em um fluxo de mediação.

elemento global. No XML, um elemento declarado como um filho do elemento esquema, em vez de parte de uma definição de tipo complexo. Os elementos globais podem ser referidos em um ou mais modelos de conteúdo, utilizando o atributo ref.

encadeamento. Um fluxo de instruções do computador que está no controle de um processo. Em alguns sistemas operacionais, um encadeamento é a menor unidade de operação em um processo. Vários encadeamentos podem ser executados simultaneamente, executando tarefas diferentes.

Enterprise JavaBeans (EJB). Uma arquitetura de componente definida pela Sun Microsystems para o desenvolvimento e a implementação de aplicativos orientados por objetos, distribuídos e de nível corporativo (Java EE).

enterprise service bus (ESB). Uma infra-estrutura de conectividade flexível para integração de aplicativos e serviços; oferece uma abordagem flexível e gerenciável para a implementação de arquitetura orientada a serviços.

Entidade.

1. Uma classe Java simples que representa uma linha em uma tabela de banco de dados ou entrada em um mapa.

2. Em linguagens de marcação como XML, uma coleção de caracteres que podem se referenciados como uma unidade, por exemplo, para incorporar texto frequentemente repetido ou caracteres especiais dentro de um documento.

envio de dados. Relativo à direção do fluxo, que é do início do processo (envio de dados) para o fim do processo (recebimento de dados).

Erro. Uma discrepância entre um valor ou condição calculada, observada ou medida e o valor ou condição verdadeira, especificada ou teoricamente correta.

ESB. Consulte barramento de serviço corporativo.

escalabilidade. A capacidade de um sistema expandir quando recursos, como processadores, memória ou armazenamento, são incluídos.

Escopo.

1. Uma especificação do limite dentro do qual os recursos do sistema podem ser usados.

2. Em serviços da Web, uma propriedade que identifica o tempo de vida do objeto que atende a solicitação de chamada.

espaço de nomes. Um contêiner lógico no qual todos os nomes são exclusivos. O identificador exclusivo para um artefato é composto do espaço de nomes e do nome local do artefato.

espaço de trabalho.

1. Um diretório no disco que contém todos os arquivos de projeto, assim como as informações como preferências.

2. Um repositório temporário de informações de configuração que clientes administrativos utilizam.

. No Eclipse, o conjunto de projetos e de outros recursos que o usuário está desenvolvendo no momento no ambiente de trabalho. Metadados sobre este recursos residem em um diretório no sistema de arquivos; os recursos podem residir no mesmo diretório.

específico. Relativo à visualização de um objeto individual em detalhes.

esqueleto. Estrutura para uma classe de implementação.

Esquema de URL. Um formato que contém outra referência de objeto.

Estático. Uma palavra-chave da linguagem de programação Java utilizada para definir uma variável como variável de classe.

Evento.

1. Uma mudança para um estado, como a conclusão ou falha de uma operação, processo de negócios ou tarefa manual, que pode disparar uma ação subsequente, como persistir dados do evento para um repositório de dados ou chamar outro processo de negócios.

2. Uma mudança nos dados em um enterprise information system (EIS) que é processada pelo adaptador e usada para fornecer objetos de negócios do EIS aos terminais (aplicativos) que precisam ser notificados da mudança.

evictor. Um componente que controla a associação de entradas em cada instância de BackingMap. Caches escassos podem usar evictors para remover dados automaticamente do cache sem afetar o banco de dados.

exceção. Uma condição ou evento não pode ser tratada por um processo normal.

exportação. Uma interface exposta de um módulo SCA (Service Component Architecture) que oferece um serviço de negócios externamente. Uma exportação tem uma ligação que define como o serviço pode ser acessado por solicitantes de serviços, por exemplo, um serviço da Web.

expressão. Um operando SQL ou XQuery ou uma coleta de operadores e operandos SQL ou XQuery que produzem um único valor.

Extensible Markup Language (XML). Uma metalinguagem padrão para definir linguagens de marcações com base em SGML (Standard Generalized Markup Language).

eXtreme Scale distribuído. Um padrão de uso para interagir com o eXtreme Scale quando os servidores e os clientes existem em vários processos.

factory EJB. Um bean de acesso que simplifica a criação ou a descoberta de uma instância de enterprise bean.

failover. Uma operação automática que comuta para um sistema redundante ou de espera no caso de uma interrupção de software, hardware ou rede.

fase de implementação. Consulte também fase de implementação.

fase de implementação. Uma fase que inclui uma combinação de criação de ambiente de hosting para seus aplicativos e a implementação desses aplicativos. Isso inclui resolver as dependências de recurso do aplicativo, condições operacionais, requisitos de capacidade e limitadores de integridade e acesso.

Firewall. Uma configuração de rede, geralmente de hardware e software, que impede que um tráfego não autorizado entre e saia de uma rede segura.

Fixpack. Uma coleta acumulativa de correções que é disponibilizada entre pacotes de atualização, atualizações de fábrica ou releases planejados. Ela é destinada a permitir que os usuários se movam para um nível de manutenção específico. Consulte também correção temporária.

Fluxo do log de erros. Um fluxo contínuo de informações de erro que é transmitido utilizando um formato predefinido.

formato binário. Representação de um valor decimal no qual cada campo deve ter 2 ou 4 bytes de comprimento. O sinal (+ ou -) está no bit mais à esquerda do campo e o valor de número está nos bits restantes do campo. Números positivos têm um 0 no bit de sinal e estão na forma verdadeira. Os números negativos têm um 1 no bit de sinal e estão em duas formas de complemento.

fuga de memória. O efeito de um programa que mantém referências a objetos que não são mais necessários e, portanto, precisam ser recuperados.

funcionamento. A condição ou estado geral do ambiente do banco de dados.

Função.

1. Uma descrição de uma função a ser desempenhada por um indivíduo ou recurso em massa, e as qualificações necessárias para preencher a função. Na simulação e na análise, a função do termo também é usada para se referir aos recursos qualificados.

2. Um cargo que identifica as tarefas que um usuário pode executar e os recursos aos quais um usuário possui acesso. Um usuário pode ter uma ou mais funções atribuídas.

. Um grupo lógico de diretores que fornece um conjunto de permissões. O acesso a operações é controlado pela concessão de acesso a uma função.

4. Em um relacionamento, uma função determina a função e a participação de entidades. As funções capturam os requisitos de estrutura e de restrição nas entidades participantes e sua maneira de participação. Por exemplo, em um relacionamento de emprego, as funções são empregador e funcionário.

gargalo. Um local no sistema no qual a contenção para um recurso está afetando o desempenho.

genérico. Em relação à visualização de um grupo de objetos com base em um resumo ou alto nível.

gerenciador autônomo. Um conjunto de componentes de software ou hardware, configurado por políticas, as quais gerenciam o comportamento de outros componentes de software ou hardware como um humano o faria. Um gerenciador autônomo inclui um loop de controle que consiste em componentes de monitor, análise, plano e execução.

Gerenciador de alta disponibilidade. Um quadro dentro do qual a participação do grupo principal é determinado e o status é comunicado entre os membros do grupo principal.

gerenciador de implementação. Um servidor que gerencia operações para um grupo lógico de células de outros servidores.

gerenciamento de carga de trabalho. A otimização da distribuição de pedidos de trabalho de entrada para os servidores de aplicativos, beans corporativos, servlets e outros objetos que podem efetivamente processar o pedido.

GIOP. Consulte General Inter-ORB Protocol.

GIOP (General Inter-ORB Protocol). Um protocolo que a arquitetura CORBA (Common Object Request Broker Architecture) utiliza para definir o formato de mensagens.

global.

1. Pertinente a um elemento que está disponível a qualquer processo na área de trabalho. Um elemento global aparece na árvore de projeto e pode ser usado em vários processos. Tarefas, processos, repositórios e serviços podem ser globais (com referência de qualquer processo no projeto) ou locais (específicas para um único processo).

2. Pertinente às informações disponíveis a mais de um programa ou subrotina.

grade de dados. Um sistema para acessar terabytes ou petrabytes de dados.

grade de eXtreme Scale. Um padrão que é usado para interagir com o eXtreme Scale quando todos os dados e clientes estão em um processo.

grupo.

1. Uma coleta de usuários que podem compartilhar autoridades de acesso para recursos protegidos.

2. Um conjunto de documentos relacionados dentro de um intercâmbio. Um intercâmbio pode conter de zero a vários grupos.

. Em locais, duas ou mais pessoas que são agrupadas para associação em um local.

grupo HA. Um conjunto de um ou mais membros usados para fornecer alta disponibilidade para um processo.

HA. Consulte alta disponibilidade.

Herança. Uma técnica de programação orientada a objetos na qual as classes existentes são usadas como base para criar outras classes. Por intermédio da herança, os elementos mais específicos incorporam a estrutura e o comportamento de elementos mais gerais.

Herança EJB. Uma forma de herança na qual um enterprise bean herda propriedades, métodos e atributos do descritor de controle do nível do método de outro enterprise bean que reside no mesmo grupo.

Hierarquia de classe. As relações entre as classes que compartilham uma herança única.

Host.

1. Um computador que está conectado a uma rede e que fornece um ponto de acesso para essa rede. O host pode ser um cliente, um servidor ou ambos um cliente e servidor simultaneamente.

2. Na definição do perfil de desempenho, uma máquina que possui processos cujos perfis estão sendo definidos. Consulte também servidor.

Host virtual. Uma configuração que permite que uma única máquina host se pareça com várias máquinas host. Os recursos associados a um host virtual não podem compartilhar dados com os recursos associados a um outro host virtual, mesmo que os hosts virtuais compartilharem da mesma máquina física.

HTTP através de SSL (HTTPS). Um protocolo da Web para transações seguras que criptografa e decriptografa pedidos de páginas do usuário e páginas retornadas pelo servidor da Web.

HTTPS.

1. Consulte HTTP sobre SSL.

2. Consulte Segurança de Protocolo de Transporte de Hipertexto.

Hypertext Transfer Protocol Secure (HTTPS). Um protocolo da Internet que é utilizado por servidores da Web e navegadores da Web para transferir e exibir documentos de hipermídia com segurança através da Internet.

IDE. Consulte ambiente de desenvolvimento integrado.

IDE (Integrated Development Environment). Um conjunto de ferramentas de desenvolvimento de software como editores de origem, compiladores e depuradores, que são acessíveis a partir de uma interface com o usuário única.

identificador de instância global. Um identificador globalmente exclusivo que é gerado pelo aplicativo ou pelo emissor e é utilizado como uma chave primária para identificação de eventos.

IIOP. Consulte Internet Inter-ORB Protocol.

IIOP (Internet Inter-ORB Protocol). Um protocolo utilizado para a comunicação entre intermediários de pedidos de objetos CORBA (Common Object Request Broker Architecture).

implementar. Colocar arquivos ou instalar software em um ambiente operacional. No J2EE (Java 2 Platform, Enterprise Edition), essa ação envolve a criação de um descritor de implementação adequado ao tipo de aplicativo que está sendo implementado.

importação.

1. Um artefato de desenvolvimento que importa um serviço que é externo a um módulo.

2. O ponto no qual um módulo SCA acessa um serviço externo, (um serviço externo ao módulo SCA) como se ele fosse local. Uma importação define as interações entre o módulo SCA e o provedor de serviços. Uma importação possui uma ligação e uma ou mais interfaces.

índice. Um conjunto de ponteiros que são logicamente ordenados pelos valores de uma chave. Os índices fornecem rápido acesso aos dados e podem impingir exclusividade dos valores da chave às linhas na tabela.

instalação silenciosa. Uma instalação que não envia mensagens ao console, mas que armazena mensagens e erros nos arquivos de log. Uma instalação silenciosa pode utilizar arquivos de resposta para a entrada de dados.

instância. Uma ocorrência específica de um objeto que pertence a uma classe.

instância do componente. Um componente em execução, que pode estar em execução paralelamente a outras instâncias do mesmo componente.

instanciar. Representar uma abstração com uma instância concreta.

interface. Um conjunto de operações usadas para especificar um serviço de uma classe ou um comportamento.

IP. Consulte Protocolo da Internet.

IP sprayer. Um dispositivo localizado entre os pedidos de entrada dos usuários e os nós do servidor de aplicativos que roteia novamente os pedidos através dos nós.

iteração. Consulte loop.

JAAS. Consulte Java Authentication and Authorization Service.

JAAS (Java Authentication and Authorization Service). Na tecnologia Java EE, uma API padrão para executar operações baseadas em segurança. Através de JAAS, os serviços podem autenticar e autorizar usuários enquanto ativam os aplicativos para permanecer independentes das tecnologias subjacentes.

JAF. Consulte JavaBeans Activation Framework.

JAF (JavaBeans Activation Framework). Uma extensão padrão para a plataforma Java que determina os tipos de dados arbitrários e operações disponíveis e que pode instanciar um bean para executar serviços pertinentes.

(Java. Uma linguagem de programação orientada a objetos para código interpretativo portátil que suporta a interação entre objetos remotos. Java foi desenvolvido e especificado por Sun Microsystems, Incorporated.

JavaBeans. Conforme definido para Java pela Sun Microsystems, um modelo de componente portátil, reutilizável, independente da plataforma. Consulte também bean.

Java Command Language. Uma linguagem de script para o ambiente Java utilizada para criar conteúdo da Web e controlar aplicativos Java.

Javadoc.

1. Uma ferramenta que analisa as declarações e os comentários da documentação em um conjunto de arquivos de origem e produz um conjunto de páginas HTML que descreve as classes, classes internas, interfaces, construções, os métodos e campos. (Sun)

2. Pertinente à ferramenta que analisa as declarações e os comentários da documentação em um conjunto de arquivos de origem e produz um conjunto de páginas HTML que descreve as classes, classes internas, interfaces, construtores, os métodos e campos.

Java EE. Consulte Plataforma Java, Enterprise Edition.

Java EE Connector Architecture (JCA). Uma arquitetura padrão para conectar a plataforma Java EE ao enterprise information systems (EIS) heterogêneo.

Java Message Service (JMS). Uma interface de programação de aplicativo que fornece funções de linguagem Java para manipulação de mensagens.

Java Platform, Enterprise Edition (Java EE). Um ambiente para desenvolvimento e implementação de aplicativos corporativos, definido pela Sun Microsystems Inc. A plataforma Java EE consiste em um conjunto de serviços, APIs (interface de programação de aplicativos) e protocolos que fornecem a funcionalidade para desenvolver aplicativos com multicamadas, baseados na Web. (Sun)

Java runtime environment. Um subconjunto do Java developer kit que contém os programas principais executáveis e os arquivos que constituem a plataforma Java padrão. O JRE inclui a JVM (Java Virtual Machine), classes de núcleo e arquivos de suporte.

JavaScript. Uma linguagem de script da Web utilizada em navegadores e servidores da Web. (Sun)

JavaScript Object Notation. Formato de troca de dados simples baseado na notação de objeto literal do JavaScript. JSON é uma linguagem de programação neutra, porém utilizando convenções de linguagens que incluem C, C++, C#, Java, JavaScript, Perl, Python.

Java SE. Consulte Java Platform, Standard Edition.

Java SE Development Kit (JDK). O nome do kit de desenvolvimento de software fornecido pela Sun Microsystems para a plataforma Java.

Java Specification Request (JSR). Uma especificação proposta formalmente para a plataforma Java.

Java virtual machine Profiler Interface (JVMPI). Uma ferramenta de perfil com suporte à coleta de informações, como dados sobre coleta de lixo e a API da Java virtual machine (JVM) que executa o servidor de aplicativos.

JAX. Consulte Java API for XML.

JAX (Java API para XML). Um conjunto de APIs baseadas em Java para manipular várias operações envolvendo dados definidos através de XML (Extensible Markup Language).

JCA. Consulte Java EE Connector Architecture.

JDBC. Consulte Java Database Connectivity.

JDBC (Java Database Connectivity). Um padrão de mercado para conectividade independente de banco de dados entre a plataforma e um amplo intervalo de bancos de dados. A interface JDBC fornece uma interface de nível de chamada para acesso ao banco de dados baseado em SQL e baseado em XQuery.

JDK. Consulte Java SE Development Kit.

JMS. Consulte Java Message Service.

JMX. Consulte Java Management Extensions.

JMX (Java Management Extensions). Um meio de executar gerenciamento e através de tecnologia Java. JMX é uma extensão universal aberta da linguagem de programação Java para gerenciamento que pode ser implementada entre todas as indústrias, onde o gerenciamento for necessário.

JNDI. Consulte Java Naming and Directory Interface.

JNDI (Java Naming and Directory Interface). Uma extensão para a plataforma Java que fornece uma interface padrão para serviços de nomenclatura heterogêneas e de diretório.

JSP. Consulte JavaServer Pages.

JSP (JavaServer Pages). Uma tecnologia de script no lado do servidor que permite que código Java seja incorporado dinamicamente dentro de páginas da Web (arquivos HTML) e executado quando a página for apresentada, para retornar conteúdo dinâmico ao cliente.

JSR. Consulte Java Specification Request.

JSSE. Consulte Java Secure Socket Extension.

JSSE (Java Secure Socket Extension). Um pacote Java que permite comunicações seguras na Internet. Ele implementa uma versão Java dos protocolos SSL (Secure Sockets Layer) e TLS (Transport Layer Security) e suporta criptografia de dados, autenticação de servidor, integridade de mensagens e autenticação de cliente opcional.

junção.

1. Um elemento de processo que recombina e sincroniza os caminhos de processamento paralelo depois de uma decisão ou bifurcação. Uma junção aguarda a chegada de uma entrada em cada uma de suas ramificações antes de permitir que o processo continue.

2. Uma operação relacional SQL na qual os dados podem ser recuperados de duas tabelas, tipicamente com base em uma condição de junção especificando as colunas de junção.

3. A configuração em um link de recebimento que determina o comportamento do link.

JVM. Consulte Java virtual machine.

JVM (Java Virtual Machine). Uma implementação de software de um processador que executa código Java compilado (applets e aplicativos).

JVMPI. Consulte Java virtual machine Profiler Interface.

Jython. Uma implementação da linguagem de programação Python integrada com a plataforma Java.

kit de desenvolvimento de software (SDK). Um conjunto de ferramentas, APIs e documentação para assistir no desenvolvimento do software em uma linguagem de computador específica ou para um ambiente operacional particular.

LDAP. Consulte Lightweight Directory Access Protocol.

LDAP (Lightweight Directory Access Protocol). Um protocolo baseado em padrão aberto que utiliza o TCP/IP para fornecer acesso a diretórios que suportam um modelo X.500 e que não está sujeito aos requisitos de recursos do DAP (Directory Access Protocol) X.500 mais complexo. Por exemplo, o LDAP pode ser utilizado para localizar pessoas, organizações e outros recursos em um diretório da Internet ou da intranet.

LED com login. Conforme definido em um documento WSDL (Web Services Description Language), um único nó de extremidade que é definido como uma combinação de uma ligação e um endereço de rede.

leitura suja. Um pedido de leitura que não envolve nenhum mecanismo de bloqueio. Isso significa que os dados que podem ser lidos podem ser recuperados posteriormente, resultando em uma inconsistência entre o que foi lido e o que está no banco de dados.

Ligação com escopo em célula. Um escopo de ligação no qual a ligação não é específica e não está associada a nenhum nó ou servidor. Esse tipo de ligação de nomes é criado no contexto de raiz persistente de uma célula.

ligação de dados JMS. Uma ligação de dados que fornece um mapeamento entre o formato utilizado por uma mensagem JMS externa e a representação SDO (Service Data Object) utilizada por um módulo SCA (Service Component Architecture).

ligação de protocolo. Uma ligação que permite que o barramento de serviço corporativo processe mensagens, independentemente do protocolo de comunicação.

limite. Uma configuração que se aplica a uma interrupção em uma simulação que define quando uma simulação de processo deve ser parada com base em uma condição existente para uma proporção especificada de ocorrências de algum evento.

linha de comandos. A linha em branco em uma exibição em que os comandos, os números de opção ou as seleções podem ser digitadas.

Listener. Um programa que detecta pedidos de entrada e inicia o canal associado.

listener do terminal. O ponto ou endereço no qual as mensagens que chegam para um serviço da Web são recebidas por um barramento de integração de serviço.

Local.

1. Relativo a um dispositivo, arquivo ou sistema que é acessado diretamente de um sistema do usuário, sem o uso de uma linha de comunicação.

2. Relativo a um elemento que está disponível somente em seu próprio processo.

log. O registro de dados sobre eventos específicos no sistema, como erros.

loop. Uma seqüência de instruções desempenhadas repetidamente.

loop do-while. Um loop que repete a mesma seqüência de atividades enquanto uma condição é atendida. Diferentemente de um loop while, um loop do-while testa sua condição no final do loop. Isso significa que sua seqüência de atividades sempre é executada pelo menos uma vez.

loop for. Um loop que repete a mesma seqüência de atividades por um determinado número de vezes.

loop while. Um loop que repete a mesma seqüência de atividades enquanto uma condição é atendida. O loop while testa sua condição no início de cada loop. Se a condição for false desde o início, a seqüência de atividades contida no loop nunca será executada.

LTPA. Consulte Lightweight Third Party Authentication.

LTPA (Lightweight Third Party Authentication). Um protocolo que utiliza criptografia para suportar a segurança em um ambiente distribuído.

Manipulador de exceção. Um conjunto de rotinas que respondem a uma condição anormal. Um manipulador de exceções é capaz de interromper e retomar a execução normal dos processos.

mapa.

1. Uma estrutura de dados que mapeia chaves em valores.

2. Um arquivo que define a transformação entre origens e destinos.

. No ambiente de desenvolvimento do EJB, a especificação de como os campos persistentes gerenciados por contêiner de um enterprise bean corresponde às colunas em uma tabela de banco de dados relacional ou outro armazenamento persistente.

máquina virtual. Uma especificação abstrata para um dispositivo de computação que pode ser implementada de formas diferentes no software e no hardware.

MBean. Consulte Managed Bean.

MBean (Managed Bean). Na especificação JMX (Java Management Extensions), os objetos Java que implementam os recursos e suas instrumentações.

método. Na programação orientada a objetos, uma operação que um objeto pode desempenhar. Um objeto pode ter muitos métodos.

Método create. Em beans corporativos, um método definido na interface inicial e chamado por um cliente para criar um enterprise bean. (Sun)

Método getter. Um método cujo objetivo é obter o valor de uma instância ou variável de classe. Isso permite que outro objeto descubra o valor de uma de suas variáveis.

Método setter. Um método cuja finalidade é definir o valor de uma instância ou variável de classe. Esse recurso permite que um outro objeto defina o valor de uma de suas variáveis.

metric. Um portador de informações, geralmente uma medida de desempenho de negócios, em um contexto de monitoramento.

modo de manutenção. Um estado de um nó ou servidor que um administrador pode utilizar para diagnosticar, manter ou ajustar o nó ou o servidor sem interromper o tráfego de entrada em um ambiente de produção.

modo silencioso. Um método para instalação ou desinstalação de um componente do produto a partir da linha de comandos sem exibição de GUI. Ao usar o modo silencioso, você especifica os dados necessários pelo programa de instalação ou desinstalação diretamente na linha de comandos ou em um arquivo (chamado um arquivo de opções ou arquivo de resposta).

Módulo EJB. Uma unidade de software que consiste em um ou mais beans corporativos e um descritor de desenvolvimento EJB. (Sun)

Navegador da Web. Um programa do cliente que inicia pedidos para um servidor Web e exibe as informações que o servidor retorna.

node.

1. Um agrupamento lógico de servidores gerenciados.

2. Qualquer item em um controle em árvore, incluindo um único elemento, elemento composto, comando de mapeamento, comentário ou nó de grupo.

3. Em XML, a menor unidade de uma estrutura válida e completa em um documento.

4. As formas fundamentais que compõem um diagrama.

nó-filho. Um nó dentro do escopo de outro nó.

Nome de Recurso Uniforme (URN). Um nome que identifica exclusivamente um serviço da Web para um cliente.

nome do host.

1. Na comunicação de Internet, o nome dado a um computador. O nome do host pode ser um nome completo de domínio como mycomputer.city.company.com, ou pode ser um subnome específico como mycomputer.

2. O nome da rede para um adaptador de rede em uma máquina física na qual o nó está instalado.

nome longo. A propriedade que especifica o nome lógico para o servidor na plataforma z/OS.

número da porta. Nas comunicações da Internet, o identificador de um conector lógico entre uma entidade do aplicativo e o serviço de transporte.

ObjectGrid. Um banco de dados de memória ativado por grade para aplicativos que são gravados em Java. ObjectGrid pode ser utilizado como um banco de dados em memória ou para distribuir dados pela rede.

objeto. Em design ou programação orientada por objeto, uma realização concreta (instância) de uma classe que consiste em dados e operações associadas aos dados. Um objeto contém os dados da ocorrência que são definidos pela classe, mas a classe pertence às operações que são associadas com os dados.

Objeto EJB. Nos beans corporativos, um objeto cuja classe implementa a interface remota do enterprise bean (Sun).

objeto genérico. Um objeto utilizado na API chama as expressões do XPATH para se referir a conceitos, entidades customizadas ou coletas. Por exemplo, a expressão do XPATH /WSRR/GenericObject recuperará todos os conceitos do WebSphere Service Registry and Repository.

objeto inicial de EJB. No EJB (Enteprise JavaBeans), um objeto que fornece as operações de ciclo de vida (criar, remover, localizar) de um enterprise bean. (Sun)

ODBC. Consulte Open Database Connectivity.

Open Database Connectivity (ODBC). Uma API (Interface de Programação de Aplicativo) padrão para acessar dados em sistemas de gerenciamento de bancos de dados relacionais e não-relacionais. Utilizando essa API, os aplicativos de bancos de dados podem acessar os dados armazenados em sistemas de gerenciamento de bancos de dados em vários computadores, mesmo se cada sistema de gerenciamento de bancos de dados utilizar um formato de armazenamento de dados e uma interface de programação diferente.

operação. Uma implementação de funções ou consultas para a qual um objeto pode ser chamado para desempenhar.

ORB. Consulte Object Request Broker.

ORB (Agente de Pedido de Objetos). Na programação orientada por objeto, o software que funciona como um intermediário, permitindo de maneira transparente que os objetos troquem pedidos e respostas.

organização. Uma entidade na qual as pessoas cooperam para realizar objetivos especificados, como uma corporação, uma empresa ou uma fábrica.

Pacote.

1. Em programação Java, um grupo de tipos. Os pacotes são declarados com a palavra-chave pacote. (Sun)

2. O wrapper em torno do conteúdo do documento que define o formato utilizado para transmitir um documento via Internet, por exemplo, RNIF, AS1 e AS2.

3. Para montar componentes em módulos e módulos em aplicativos corporativos.

pacote de instalação. Uma unidade instalável de um produto de software. Os pacotes de produto de software são unidades instaláveis separadamente que podem operar independente de outros pacotes desse produto de software.

Página JSP. Um documento baseado em texto que utiliza dados corrigidos do modelo e elementos JSP que descrevem como processar um pedido para criar uma resposta. (Sun)

painel. Uma página da Web que pode conter um ou mais visualizadores que representam graficamente os dados de negócio.

palavra-chave. Uma das palavras predefinidas de uma linguagem de programação, linguagem artificial, aplicativo ou comando.

pasta. Um contêiner utilizado para organizar objetos.

Perfil. Dados que descrevem as características de um usuário, grupo, recurso, programa, dispositivo ou local remoto.

permissão. Autorização para executar atividades, como ler e gravar arquivos locais, criar conexões de rede e carregar código nativo.

Persistência.

1. Uma característica de dados que é mantida pelos limites de sessões ou de um objeto que continua existindo após a execução do programa ou processo que o criou, geralmente em um armazenamento não-volátil como um sistema de banco de dados.

2. Em Java EE, o protocolo para transferir o estado de um bean de entidade entre suas variáveis de instância e um banco de dados subjacente. (Sun)

Persistir. Ser mantido além dos limites da sessão, geralmente em armazenamento não-volátil, como um sistema de banco de dados ou um diretório.

plano de construção. Um arquivo XML que define o processamento necessário para construir saídas de geração e que especifica a máquina em que o processamento ocorre.

Plataforma Java. Um termo coletivo para a linguagem Java para gravar programas; um conjunto de APIs, bibliotecas de classe e outros programas utilizados em programas de desenvolvimento, compilação e verificação de erros; e um Java virtual machine que carrega e executa arquivos de classe. (Sun)

Plataforma Java, Edição Padrão (Java SE). A plataforma da tecnologia Java principal. (Sun)

Plug-in do. Um módulo de software que pode ser instalado separadamente que inclui função a um programa, aplicativo ou interface existente.

Plug-in do servidor Web. Um módulo de software que suporta o servidor da Web em pedidos de comunicação de conteúdo dinâmico, como servlets, para o servidor de aplicativos.

PMI. Consulte Performance Monitoring Infrastructure.

PMI (Performance Monitoring Infrastructure). Um conjunto de pacotes e bibliotecas atribuídas para reunir, distribuir, processar e exibir dados de desempenho.

política. Um conjunto de considerações que influenciam o comportamento de um recurso gerenciado ou um usuário.

política de autorização. Uma política cujo destino é um serviço de negócios e cujo contrato contém uma ou mais asserções que concedem permissão para executar uma ação de canal.

política de implementação. Uma maneira opcional de configurar um ambiente eXtreme Scale baseado em vários itens, incluindo: número de sistemas, servidores, partições, réplicas (incluindo tipo de réplica), e tamanhos de heap para cada servidor.

política HA. Um conjunto de regras definido para um grupo HA que dita se zero (0) ou mais membros estão ativados. A política está associada a um grupo HA específico, correspondendo os critérios de correspondência de política com o nome do grupo.

Ponto-a-ponto. Um estilo de aplicativo de sistema de mensagens no qual o aplicativo de envio conhece o destino da mensagem.

Ponto de acesso de período do proxy. Um meio de identificar as configurações de comunicação para um ponto de acesso de período que não pode ser acessado diretamente.

ponto de interrupção. Um ponto marcado em um processo ou fluxo programático que faz com que esse fluxo seja pausado quando o ponto é atingido, geralmente para permitir a depuração ou a monitoração.

ponto de interrupção de entrada. Um ponto de interrupção configurado em um elemento do componente, que é selecionado antes do elemento do componente ser chamado.

porta listener. Um objeto que define a associação entre uma fábrica de conexões, um destino e um bean orientado por mensagem implementado. As portas de atendentes simplificam a administração das associações entre esses recursos.

processo.

1. Um procedimento progressivamente contínuo que consiste em uma série de atividades controladas que estão direcionadas sistematicamente para um determinado resultado ou fim.

2. A seqüência de documentos ou mensagens a serem trocadas entre Gerenciadores de Comunidade e participantes para executar uma transação de negócios.

processo síncrono. Um processo que é iniciado chamando uma operação de pedido-resposta. O resultado do processo é retornado pela mesma operação.

programação orientada a objetos. Uma abordagem de programação baseada nos conceitos de abstração e herança de dados. Diferente das técnicas de programação de procedimentos, a programação orientada a objetos concentra-se não em como algo é realizado, mas em quais objetos de dados consistem o problema e como eles são manipulados.

program temporary fix (PTF). Para produtos System i, System p e System z, uma correção testada pela IBM e disponibilizada para todos os clientes. Consulte também fix pack.

projeto do aplicativo corporativo (projeto EAR). Uma estrutura e hierarquia de pastas e arquivos que contém um descritor de desenvolvimento e documento de extensão IBM, além de arquivos que são comuns para todos os módulos Java EE que são definidos no descritor de desenvolvimento.

projeto EAR. Consulte projeto do aplicativo corporativo.

Projeto EJB. Um projeto que contém os recursos necessários para aplicativos EJB, incluindo beans corporativos; interfaces home, local e remota; arquivos JSP; servlets; e descritores de implementação.

Projeto Java. No Eclipse, um projeto que contém código fonte Java compilável e é um contêiner para pastas ou pacotes de origem.

prompt. Um componente de uma ação, que indica que a entrada do usuário é requerida para um campo antes da transição para uma tela de saída.

Propriedade. Uma característica de um objeto que descreve o objeto. Uma propriedade pode ser alterada ou modificada. As propriedades podem descrever um nome, tipo, valor ou comportamento de objeto, entre outras coisas.

Protocolo da Internet (IP). Um protocolo que roteia dados por meio de uma rede ou redes interconectadas. Este protocolo age como um intermediário entre as camadas mais altas do protocolo e a rede física.

Protocolo de Controle de Transmissões/Protocolo da Internet (TCP/IP). Um conjunto de protocolos de comunicação não-proprietário padrão de mercado que fornece conexões de ponta a ponta confiáveis entre aplicativos por redes interconectadas de tipos diferentes.

Provedor MBean. Uma biblioteca que contém uma implementação de um MBean JMX (Java Management Extensions) e seu arquivo descritor MBean XML (Extensible Markup Language).

proxy. Um gateway de aplicativo de uma rede para outra para um aplicativo de rede específico, como Telnet ou FTP por exemplo, em que um servidor proxy Telnet de firewall desempenha autenticação do usuário e, em seguida, permite que o tráfego flua pelo proxy como se não estivesse lá. A função é desempenhada no firewall e não na estação de trabalho do cliente, gerando maior carga no firewall.

PTF. Consulte correção temporária do programa .

public.

1. Na programação orientada a objetos, relativo a um membro de classe que é acessível a todas as classes.

2. Em linguagem de programação Java, relativo a um método ou variável que pode ser acessada por elementos que residem em outras classes. (Sun)

pulsação. Um sinal que uma entidade envia para outra para informar que ainda está ativa.

QoS. Consulte quality of service.

qualidade de serviço (QoS). Um conjunto de características de comunicação que um aplicativo precisa. A QoS (Quality of Service) define uma prioridade de transmissão específica, o nível de confiabilidade da rota e o nível de segurança.

qualificador. Um elemento simples que fornece um significado específico a outro elemento genérico, composto ou simples. Qualificadores são utilizados em ocorrências múltiplas ou única de mapeamento. Um qualificador pode ser utilizado também para denotar o espaço de nomes utilizado para interpretar a segunda parte do nome, geralmente referenciada como o ID.

rastreio de execução. Uma cadeia de eventos, registrados e exibidos em um formato hierárquico na página Eventos do cliente de teste de integração.

recebimento de dados. Relativo à direção do fluxo, que é do primeiro nó no processo (envio de dados) para o último nó no processo (recebimento de dados).

recursivo. Uma técnica de programação na qual um programa ou rotina chama a si mesmo para executar etapas sucessivas em uma operação, em que cada etapa utiliza a saída da etapa anterior.

recurso.

1. Um ativo discreto como, por exemplo, conjuntos de aplicativos, aplicativos, serviços de negócios, interfaces, terminais e eventos de negócio.

2. Um recurso de um sistema de cálculo ou sistema operacional necessário por um trabalho, tarefa ou programa em execução. Os recursos incluem armazenamento principal, dispositivos de entrada/saída, a unidade de processamento, conjuntos de dados, arquivos, bibliotecas, servidores de aplicativos e programas de controle ou processamento.

3. Uma pessoa, parte de equipamento ou material que é usado para execução de uma tarefa ou um projeto. Cada recurso é uma ocorrência ou exemplo em particular de uma definição de recurso.

Recurso da instância de cache. Um local onde qualquer aplicativo J2EE (Java 2 Platform, Enterprise Edition) pode armazenar, distribuir e compartilhar dados.

recurso de particionamento. Uma estrutura de programação e uma infra-estrutura de gerenciamento de sistemas que suporta o conceito de particionamento para enterprise bean, tráfego de HTTP e acesso ao banco de dados.

referência do EJB. Um nome lógico utilizado por um aplicativo para localizar a interface inicial de um enterprise bean no ambiente operacional de destino.

Refresh pack. Uma coleta acumulativa de correções que contêm novas funções. Consulte também fix pack, correção temporária.

Região. Uma área contígua de armazenamento virtual que possui características comuns e que pode ser compartilhada entre os processos.

Região servant. Uma área contígua de armazenamento virtual que é iniciada dinamicamente conforme a carga aumenta e é automaticamente parada conforme a carga se ameniza.

regra if-then. Uma regra na qual a ação (parte then) é executada somente quando a condição (parte if) é verdade.

Rendimento do processamento. A medida da quantidade de trabalho executado por um dispositivo, como um computador ou impressora, por um período e tempo, por exemplo, número de jobs por dia.

réplica. Um servidor que contém uma cópia do diretório ou diretórios de outro servidor. As réplicas fazem backup dos servidores para melhora o desempenho e os tempos de resposta para garantir a integridade dos dados.

réplica assíncrona. Um shard que recebe atualizações após os as confirmações de transação. Este método á mais rápido que uma réplica síncrona, mas tem a possibilidade de perda de dados porque a réplica assíncrona pode estar várias transações atrás do shard primário.

replicação. O processo de manutenção de um conjunto definido de dados em mais de um local. A replicação envolve a cópia de alterações designadas de um local (uma origem) para outro (um destino) e sincronização de dados nos dois locais.

Replicação de cache. O compartilhamento de IDs de cache, entradas de cache e invalidações de cache com outros servidores no mesmo domínio de replicação.

réplica síncrona. Um shard que recebe atualizações como parte da transação no shard primário para garantir a consistência de dados, o que pode aumentar o tempo de resposta comparado com uma réplica assíncrona.

Reprovado. Relativo a uma entidade, como um recurso ou um elemento de programação, que é suportada mas não mais recomendada e que pode tornar-se obsoleta.

restrição de cronometragem. Uma ação de validação especializada utilizada para medir a duração de uma chamada de método ou uma seqüência de chamadas de métodos.

root. O nome do usuário para o usuário do sistema com a autoridade maior.

script. Uma série de comandos, combinados em um arquivo, que executa uma função específica quando o arquivo é executado. Os scripts são interpretados conforme eles são executados.

Scripts. Um estilo de programação que reutiliza componentes existentes como base para criar aplicativos.

script shell. Um programa, ou script, que é interpretado pelo shell de um sistema operacional.

SDK. Consulte kit de desenvolvimento de software.

Segurança do Java Connector. Uma arquitetura projetada para estender o modelo de segurança de ponta a ponta para aplicativos com base em Java EE para incluir EIS (Enterprise Information Systems).

segurança global. Relativa a todos os aplicativos em execução no ambiente e determina se a segurança é utilizada, o tipo de registro utilizado para autenticação e outros valores, muitos dos quais atuam como padrões.

Separação do servidor Web. Uma topologia na qual o servidor Web é fisicamente separado do servidor de aplicativos.

Será impresso exatamente o que está na tela (WYSIWYG). Um recurso de um editor para exibir páginas continuamente exatamente como serão impressas ou apresentadas.

serialização. Em programação orientada a objetos, a gravação de dados de maneira seqüencial em uma mídia de comunicações a partir da memória do programa.

serializador. Um método de conversão de dados de objetos para outro formato, como binário ou XML.

server. Um programa de software ou um computador que fornece serviços a outros programas de software ou outros computadores. Consulte também host.

serviço de catálogo. Um serviço que controla o posicionamento dos shards e descobre e monitora o funcionamento de contêineres.

servidor da Web. Software capaz de atender a pedidos de HTTP (Hypertext Transfer Protocol).

servidor de aplicativos. Um programa do servidor em uma rede distribuída que fornece o ambiente de execução para um programa de aplicativo.

servidor de contêineres. Uma instância do servidor que pode hospedar vários shards. Uma Java virtual machine (JVM) pode hospedar vários servidores de contêineres.

servidor de monitoramento de TCP/IP. Um ambiente de tempo de execução que monitora todos os pedidos e respostas entre um navegador da Web e um servidor de aplicativos, bem como uma atividade de TCP/IP.

Servidor EJB. O software que fornece serviços a um contêiner EJB. Um servidor EJB pode hospedar um ou mais contêineres EJB. (Sun)

servidor independente. Um serviço de catálogo ou servidor de catálogo que é gerenciado a partir do sistema operacional que inicia e finaliza o processo do servidor.

servidor integrado. Um serviço de catálogo ou servidor de contêineres que reside em um processo existente e é iniciado e parado dentro do processo.

Servidor Java EE. Um ambiente de tempo de execução que fornece contêineres EJB ou de Web.

Servidor Proxy.

1. Um servidor que atua como um intermediário para as solicitações HTTP da Web que são hospedadas por um aplicativo ou um servidor da Web. Um servidor proxy atua como um substituto para os servidores de conteúdo na empresa.

2. Um servidor que recebe pedidos destinados a outro servidor e que atua em nome do cliente (como o proxy do cliente) para obter o serviço solicitado. Um servidor de proxy é freqüentemente utilizado quando o cliente e o servidor são incompatíveis para a conexão direta. Por exemplo, o cliente não consegue atender aos requisitos de autenticação de segurança do servidor, mas deve ter permissão para alguns serviços.

servlet. Um programa Java executado em um servidor da Web e que estende as funções do servidor através da geração de conteúdo dinâmico em resposta aos pedidos do Web client. Os servlets são utilizados geralmente para conectar os bancos de dados à Web.

sessão.

1. Uma conexão lógica ou virtual entre duas estações, programas de software ou dispositivos em uma rede que permite que os dois elementos se comuniquem e troquem dados.

2. Uma série de solicitações para um servlet de origem do mesmo usuário no mesmo navegador.

. No Java EE, um objeto usado por um servlet para controlar a interação de um usuário com um aplicativo da Web através de vários pedidos HTTP.

shard. Uma instância de uma partição. Um shard pode ser primário ou réplica.

sincronizar. Incluir, subtrair ou alterar um recurso ou artefato para corresponder ao outro.

Sintaxe. As regras para a construção de um comando ou instrução.

Sistema de mensagens assíncronas. Um método de comunicação entre programas em que um programa coloca uma mensagem em uma fila de mensagens e, em seguida, prossegue com seu próprio processamento sem esperar por uma resposta da sua mensagem.

Sistema de mensagens gerenciado por bean. Uma função de sistema de mensagens assíncronas que fornece a um enterprise bean controle completo sobre a infra-estrutura do sistema de mensagens.

sistema host. Um sistema de computador mainframe corporativo que hospeda aplicativos 3270. Nas ferramentas de desenvolvimento de terminal service 3270, o desenvolvedor usa o gravador do terminal service 3270 para conectar-se ao sistema host.

SLA. Consulte contrato de nível de serviço.

software livre. Software cujo código-fonte está publicamente disponível para uso ou modificação. O software livre geralmente é desenvolvido como uma colaboração pública e disponibilizado livremente, embora seu uso e sua redistribuição possam estar sujeitos a restrições de licenciamento. O Linux é um exemplo bem conhecido de software livre.

SQL. Consulte Linguagem de consulta estruturada.

SQL (Structured Query Language). Uma linguagem padronizada para definir e manipular dados em um banco de dados relacional.

SSL. Consulte Secure Sockets Layer.

SSL (Secure Sockets Layer). Um protocolo de segurança que fornece privacidade de comunicação. O SSL permite que aplicativos de cliente/servidor se comuniquem de forma designada para evitar escuta maliciosa, violação e falsificação de mensagens.

stack. Uma área na memória que geralmente armazena informações como registro temporário, valores de parâmetros e endereços de retorno de sub-rotinas e é baseada no princípio último a entrar, primeiro a sair (LIFO).

string. Em linguagens de programação, o formato dos dados utilizados para armazenar e manipular texto.

subclasse. Em Java, uma classe derivada de uma classe específica, por meio de herança.

subconsulta. Em SQL, uma subseleção utilizada em um predicado, por exemplo, uma instrução de seleção dentro da cláusula WHERE ou HAVING de outra instrução SQL.

suporte baseado em zona. Um função que ativa o posicionamento compartilhado baseado em regras para melhorar a disponibilidade da grade por meio da colocação de shards através de diferentes centros de dados, seja em andares diferentes ou mesmo em prédios ou geografias diferentes.

Tabela de autorizações. Uma tabela que contém a função para informações de mapeamento de usuário ou grupo que identificam o acesso permitido de um cliente a um recurso específico.

TCP. Consulte Protocolo de Controle de Transmissões.

TCP/IP. Consulte Protocolo de Controle de Transmissões/Protocolo da Internet.

TCP (Transmission Control Protocol). Um protocolo de comunicação utilizado na Internet e em qualquer rede que segue os padrões do IETF (Internet Engineering Task Force) para o protocolo de interligação de redes. O TCP oferece um protocolo confiável de host a host em redes de comunicação através da comutação de pacotes e em sistemas interconectados dessas redes.

tempo de ativação. O intervalo de tempo em segundos no qual uma entrada pode existir no cache antes de ser descartada.

tempo de compilação. O período de tempo durante o qual um programa de computador está sendo compilado em um programa executável.

Tempo de execução. O período de tempo durante o qual um programa de computador está em execução.

terminal.

1. Um aplicativo JCA ou outro consumidor cliente de um evento a partir do sistema de informações corporativos.

2. O sistema que é a origem ou o destino de uma sessão.

teste de componente. Um teste automatizado de um ou mais componentes de um aplicativo corporativo, que pode incluir classes Java, beans EJB ou serviços da Web.

timeout. Um intervalo de tempo concedido para que um evento ocorra ou seja concluído, antes da operação ser interrompida.

tipo primitivo. Em Java, uma categoria de tipo de dados que descreve uma variável que contém um valor único do tamanho apropriado e o formato para seu tipo: um número um caractere ou um valor Booleano. Exemplos de tipos primitivos incluem byte, short, int, long, float, double, char, boolean.

Tivoli Performance Viewer. Um cliente Java que recupera os dados PMI (Performance Monitoring Infrastructure) de um servidor de aplicativos e os exibe em diversos formatos.

token.

1. Um marcador usado para controlar o estado atual de uma instância de processo durante uma execução de simulação.

2. Uma mensagem específica ou padrão de bits que significa permissão ou controle temporário para transmitir através de uma rede.

Token de segurança. Uma representação de um conjunto de alegações que são feitas por um cliente que pode incluir um nome, senha, identidade, chave, certificado, grupo, privilégio e assim por diante.

topologia. O mapeamento físico ou lógico o local dos componentes de rede ou nós dentro de uma rede. As topologias de rede incluem barramento, anel, estrela e árvore.

topologia da implementação. A configuração de servidores e clusters em um ambiente de implementação e os relacionamentos físicos e lógicos entre eles.

topologia do tempo de execução. Uma descrição do estado momentâneo do ambiente.

transação. Um processo no qual todas as modificações de dados que foram feitas durante uma transação são confirmadas juntas como uma unidade ou revertidas como uma unidade.

transação global. Uma unidade de trabalho recuperável realizada por um ou mais gerenciadores de recursos em um ambiente de transação distribuído e coordenado por um gerenciador de transações externo.

type.

1. Em programação Java, uma classe ou interface.

2. Em um documento WSDL, um elemento que contém definições de tipo de dados usando algum sistema de tipo (como XSD).

UDDI. Consulte Descrição, Descoberta e Integração Universal.

Unidade de compilação. Uma parte de um programa de computador suficientemente completa para ser compilada corretamente.

Universally Unique Identifier (UUID). O identificador numérico de 128 bits utilizado para assegurar que dois componentes não têm o mesmo identificador.

UNIX System Services. Um elemento de z/OS que cria um ambiente UNIX de acordo com as especificações XPG4 UNIX 1995 e que fornece duas interfaces de sistema aberto no sistema operacional z/OS: uma API (interface de programação de aplicativos) e uma interface shell interativa.

URI. Consulte Identificador Uniforme de Recursos (URI).

URI (Uniform Resource Identifier).

1. Uma cadeia compacta de caracteres para identificação de um recurso abstrato ou físico.

2. Um endereço exclusivo que é usado para identificar o conteúdo na Web, como uma página de texto, um vídeo ou clipe de som, uma imagem parada ou animada, ou um programa. A forma mais comum de URI é o endereço de páginas da Web, que é uma forma particular ou um subconjunto de URIs chamado URL (Uniform Resource Locator). Normalmente, um URI descreve como acessar o recurso, o computador que contém o recurso e o nome do recurso (o nome de um arquivo) no computador.

URL. Consulte Localizador Uniforme de Recursos.

URL (Uniform Resource Locator). O endereço exclusivo de um recurso de informações que pode ser acessado em uma rede como a Internet. A URL inclui o nome abreviado do protocolo utilizado para acessar o recurso de informações e as informações utilizadas pelo protocolo para localizar o recurso de informações.

URN. Consulte Nome Uniforme de Recursos.

usuário autenticado. Um usuário do portal que não efetuou login com uma conta válida (ID do usuário e senha). Os usuários autenticados têm acesso a todos os locais públicos.

utilitário de carga. Um componente que lê dados de um armazenamento persistente e grava dados nele.

UUID. Consulte Identificador Universalmente Exclusivo.

Variável. Uma representação de um valor alterável.

Variável de ambiente. Uma variável que especifica como um sistema operacional ou outro programa é executado ou os dispositivos que o sistema operacional reconhece.

variável global. Uma variável que é utilizada para manter e manipular valores designados a ela durante a conversão e que é compartilhada por meio de mapas e conversões de documentos. Um dos três tipos de variáveis com suporte na linguagem de comandos de mapeamento do Data Interchange Services.

versão. Um programa licenciado separadamente que geralmente possui um novo código ou uma nova função significativa.

virtualização. Uma técnica que encapsula as características de recursos da maneira na qual os outros sistemas interagem com esses recursos.

WAR. Consulte Archive da Web.

WAR (Web Archive). Um formato de arquivo compactado, definido pelo padrão Java EE, para armazenar todos os recursos necessários para instalar e executar um aplicativo da Web em um único arquivo. Consulte também archive corporativo.

WCCM. Consulte WebSphere Common Configuration Model.

Web site. Uma coleta de arquivos relacionados disponível na Web que é gerenciada por uma única entidade (uma organização ou uma pessoa) e contém informações em hipertexto para seus usuários. Um Web site geralmente inclui links de hipertexto para outros Web sites.

WebSphere. Um nome de marca IBM que abrange ferramentas para desenvolvimento de aplicativos de e-business e middleware para execução de aplicativos da Web.

WebSphere Common Configuration Model (WCCM). Um modelo que fornece acesso programático aos dados de configuração.

WLM. Consulte Workload Manager.

Workload Manager (WLM). Um componente do z/OS que fornece a capacidade de executar várias cargas de trabalho ao mesmo tempo dentro de uma imagem ou de várias imagens do z/OS.

WYSIWYG. Consulte Será impresso exatamente o que está na tela .

XA. Uma interface bidirecional entre um ou mais gerenciadores de recursos que fornecem acesso aos recursos compartilhados e um gerenciador de transações que monitora e resolve as transações.

XML. Consulte Linguagem de Marcação Extensível.

X/Open XA. A interface X/Open Distributed Transaction Processing XA. Um padrão proposto para comunicação de transação distribuída. O padrão especifica uma interface bidirecional entre os gerenciadores de recursos que fornecem acesso a recursos compartilhados dentro de transações e entre um serviço de transação que monitora e resolve as transações.

z/OS. Um sistema operacional mainframe IBM que usa armazenamento real de 64 bits.

Avisos

Referências nesta publicação a produtos, programas ou serviços IBM não significam que a IBM pretende torná-los disponíveis em todos os países onde opera. Qualquer referência a um produto, programa ou serviço IBM não significa que apenas um produto, programa ou serviço IBM possa ser utilizado. Qualquer produto, programa ou serviço funcionalmente equivalente, que não infrinja nenhum direito de propriedade intelectual da IBM ou outros direitos legalmente protegidos, poderá ser utilizado em substituição a este produto, programa ou serviço. A avaliação e a verificação da operação em conjunto com outros produtos, exceto aqueles expressamente designados pela IBM, são de inteira responsabilidade do usuário.

A IBM pode ter patentes ou solicitações de patentes pendentes relativas a assuntos tratados nesta publicação. O fornecimento desta publicação não garante ao Cliente nenhum direito sobre tais patentes. Pedidos de licença devem ser enviados, por escrito, para:

Gerência de Relações Comerciais e Industriais da IBM Brasil
IBM Corporation
Rio de Janeiro, RJ
CEP 22290-240

Licenciados deste programa que desejam obter mais informações sobre este assunto com objetivo de permitir: (i) a troca de informações entre programas criados independentemente e outros programas (incluindo este) e (ii) a utilização mútua das informações trocadas, devem entrar em contato com:

IBM Corporation
Av. Pasteur, 138-146, Botafogo
Botafogo
Rio de Janeiro, RJ
CEP 22290-240

Tais informações podem estar disponíveis, sujeitas a termos e condições apropriados, incluindo em alguns casos, o pagamento de uma taxa.

Marcas Registradas

Os termos a seguir são marcas registradas da IBM Corporation nos Estados Unidos e/ou em outros países:

- AIX
- CICS
- Cloudscape
- DB2
- Domino
- IBM
- Lotus
- RACF
- Redbooks
- Tivoli
- WebSphere
- z/OS

Java e todas as marcas registradas baseadas em Java são marcas registradas da Sun Microsystems, Inc. nos Estados Unidos e/ou em outros países.

LINUX é uma marca registrada de Linus Torvalds nos Estados Unidos e/ou em outros países.

Microsoft, Windows, Windows NT e o logotipo Windows são marcas registradas da Microsoft Corporation nos Estados Unidos e/ou em outros países.

UNIX é uma marca registrada do The Open Group nos Estados Unidos e em outros países.

Outros nomes de empresas, produtos e serviços podem ser marcas registradas ou marcas de serviço de terceiros.

Índice Remissivo

T

U

V

W

Z

Impresso em Brazil