WebSphere® eXtreme Scale Versão 7.1
Visão Geral do Produto

Visão Geral do Produto WebSphere eXtreme Scale

IBM

Esta edição aplica-se à versão 7, release 1, 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. Com WebSphere eXtreme Scale, você também pode obter qualidades de serviço como integridade transacional, alta disponibilidade e tempos de respostas previsíveis.

A escalabilidade elástica é possível por meio do uso do armazenamento em cache de objeto distribuído. Com escalabilidade elástica, a grade de dados monitora e gerencia a si mesma. Ela pode executar o ajuste da escala da topologia incluindo e removendo servidores, o que aumenta ou diminui a memória, o rendimento da rede e a capacidade de processamento conforme necessário. Quando um processo de scale-out é iniciado, a capacidade é incluída na grade de dados enquanto ela está em execução, sem exigir um reinício. De modo inverso, um processo de scale-in imediatamente remove a capacidade. A grade de dados também é auto-recuperável recuperando-se automaticamente de falhas.

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. É possível 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, que são equivalentes aos esquemas de árvore restritos. Os esquemas em árvore restritos descrevem o relacionamento entre entidades. Quando você está usando partições, os relacionamentos da entidade devem modelar uma estrutura de dados em árvore, na qual a cabeça da árvore é a entidade raiz e é a única entidade que é particionada. Todos os outros filhos da entidade raiz são armazenados na mesma partição que a entidade raiz. Cada partição existe como uma cópia primária ou shard. Uma partição também contém shards de réplica para fazer backup de 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 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 são de 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.

Com o recurso de cache write-behind, o WebSphere eXtreme Scale pode servir de cache front-end para um banco de dados. Com o uso desse cache front-end, o rendimento aumenta enquanto reduz a contenção e o carregamento do banco de dados. WebSphere eXtreme Scale fornece ampliação e adição previsíveis a um custo de processamento previsível.

A imagem a seguir mostra que, em um ambiente em cache distribuído e coerente, os clientes do eXtreme Scale enviam e recebem dados da grade de dados, que pode ser automaticamente sincronizada com um armazenamento de dados de 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 contém mais dados à medida em que mais servidores são incluídos na grade de dados e escala linearmente conforme a grade de dados cresce em tamanho. Os dados também podem ser opcionalmente replicados para se obter tolerância a falhas opcional.

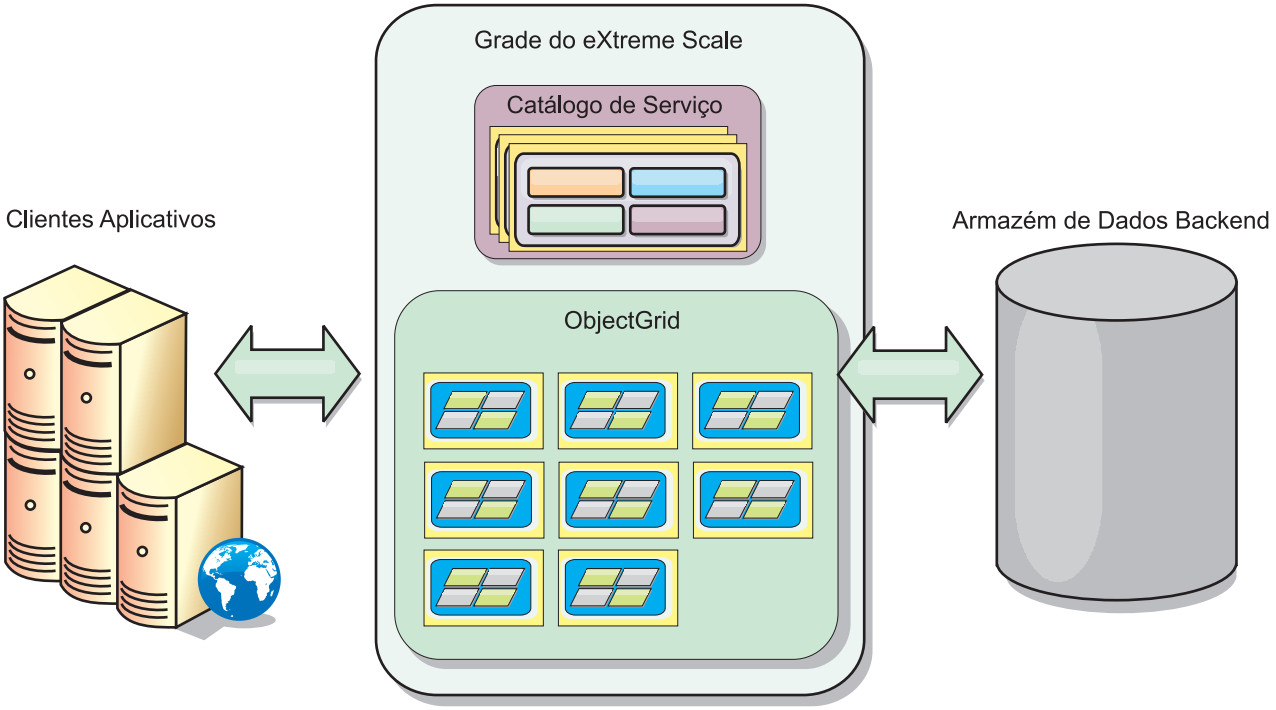


Figura 1. Topologia de Alto Nível

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 Machine 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. A grade de dados não é limitada pela memória ou pelo espaço de endereço do aplicativo ou do servidor de aplicativos e também não tem um impacto sobre eles. A memória pode ser a soma da memória de centenas, ou milhares, de Java Virtual Machines, em execução em diferentes máquinas.

Assim como no espaço de processamento do banco de dados de memória, o WebSphere eXtreme Scale ainda pode ser suportado por 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 de dados é um par de chave-valor, no qual a grade de dados armazena valores (objetos Java), com uma chave associada (outro objeto Java). A chave é utilizada posteriormente para recuperar o valor. No eXtreme Scale, um mapa consiste em entradas desses pares de chave-valor.

O WebSphere eXtreme Scale oferece várias configurações da grade de dados, de um cache local único e simples até um cache grande distribuído, usando várias Java virtual machines ou servidores.

Além de armazenar objetos Java simples, é possível armazenar objetos com relacionamentos. É possível utilizar uma linguagem de consulta como SQL, com instruções `SELECT ... FROM ... WHERE` para recuperar esses objetos. Por exemplo, um objeto de ordem pode ter um objeto de cliente e vários objetos de item associados a ele. WebSphere eXtreme Scale suporta relacionamentos um-para-um, um-para-muitos, muitos-para-um e muitos-para-muitos.

WebSphere eXtreme Scale também suporta uma interface de programação EntityManager para armazenar entidades no cache. Essa interface de programação é semelhante às entidades 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 pela chave primária usando o método find na interface EntityManager. As entidades podem ser persistidas para a grade de dados ou removidas dela dentro de um limite de transação.

O WebSphere eXtreme Scale fornece recursos Extreme Transaction Processing (XTP) que garantem uma infraestrutura de aplicativos mais inteligente para suportar os aplicativos críticos de negócios com maior demanda. É possível 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 suporte para WebSphere Real Time, a oferta Java em tempo real líder de mercado, WebSphere eXtreme Scale permite que aplicativos XTP tenham tempos de resposta mais consistentes e previsíveis. Consulte as informações sobre Suporte em Tempo Real na *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 (ou grade de dados ou ObjectGrid no vocabulário do eXtreme Scale), você configura o número de servidores eXtreme Scale que conterão os dados da grade

de dados e o eXtreme Scale distribui os dados em shards sobre essas 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 contendo um shard de réplica falhar, o eXtreme Scale alocará 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 eXtreme Scale mais simples é ObjectMap, que é uma interface de mapa simples: um método map.put(key,value) para colocar um valor no cache, e um método map.get(key) para recuperar subsequentemente o valor.

Para uma discussão das boas práticas que é possível usar ao projetar seus aplicativos WebSphere eXtreme Scale, leia o seguinte artigo em developerWorks: Princípios e boas práticas para construir aplicativos WebSphere eXtreme Scale de alta execução e alta resiliência.

Recursos Novos e Reprovados neste Release

WebSphere eXtreme Scale inclui vários novos recursos na Versão 7.1, incluindo integração com cache dinâmico, mapas da matriz de byte e mais.

O que Há de Novo no WebSphere eXtreme Scale Versão 7.1

Tabela 1. Novos Recursos no WebSphere eXtreme Scale Versão 7.1

Recurso	Descrição
Integração de informações do cliente do DB2	Integre plug-ins do Carregador JPA do eXtreme Scale com o DB2, para que quando o DB2 for usado como o banco de dados backend, as informações do WebSphere eXtreme Scale (nome de usuário, nome da estação de trabalho, nome do aplicativo e informações da conta) possam ser disponibilizadas no Monitor de Desempenho do DB2. Esse recurso permite ativar e desativar a configuração de informações do cliente para o DB2. Essa função fica desativada por padrão. Para obter mais informações, consulte "Utilitários de Carga" na página 40.
Configuração de domínio do serviço de catálogo	Os domínios do serviço de catálogo podem ser configurados usando o console administrativo do WebSphere Application Server ou usando tarefas administrativas. Os domínios do serviço de catálogo definem um grupo. Para obter mais informações, consulte as informações sobre como criar domínios do serviço de catálogo no *Guia de Administração*.
Replicação multimestre	Vários datacenters podem ser vinculados assincronamente, permitindo o acesso local do datacenter a dados e mantendo uma alta disponibilidade. Consulte "Topologias de Replicação de Grade Multimestre (AP)" na página 21 para obter mais informações.
Horário da última atualização do evictor TTL	O evictor TTL foi atualizado para rastrear o horário no qual uma entrada foi atualizada, expandindo no evictor time-to-live. Para obter mais informações, consulte as informações sobre o evictor TimeToLive (TTL) no *Guia de Programação*
Estatísticas usedBytes para grades na memória	A quantia de memória usada por entradas de cache em um BackingMap pode ser rastreada usando todos os provedores de estatísticas. Para obter mais informações, consulte as informações sobre o dimensionamento do consumo do cache de memória no *Guia de Administração*.
Estatísticas dinâmicas	As estatísticas podem ser ativadas e desativadas on demand. Para obter mais informações, consulte as informações sobre como monitorar com MBeans no *Guia de Administração*.
Console de monitoramento	O console de monitoramento gráfico fornece visualizações atuais e históricas nas estatísticas do servidor WebSphere eXtreme Scale. Para obter mais informações, consulte as informações sobre o console da Web no *Guia de Administração*.
Gerenciador de Sessões HTTP Melhorado	A configuração do gerenciador de sessões HTTP foi simplificada. É possível configurar o gerenciador de sessão HTTP no console administrativo do WebSphere Application Server. Para obter mais informações, consulte as informações sobre como configurar o gerenciador de sessões HTTP no *Guia de Administração*.
Suporte a clientes multihomed	Os clientes podem ser configurados para usar um adaptador de rede específico. Para obter mais informações, consulte as informações sobre o arquivo de propriedades do cliente no *Guia de Administração*.
ISA Lite	O IBM® Support Assistant Lite for WebSphere eXtreme Scale fornece uma coleta de dados automática e suporte à análise de sintomas para cenários de determinação de problemas. Para obter mais informações, consulte as informações sobre o IBM Support Assistant for WebSphere eXtreme Scale no *Guia de Administração*.

Tabela 1. Novos Recursos no WebSphere eXtreme Scale Versão 7.1 (continuação)

Recurso	Descrição
REST	O serviço de dados REST fornece aos clientes não Java acesso a dados do eXtreme Scale, dando suporte ao Open Data Protocol (OData), fornecendo compatibilidade total com o Microsoft® WCF Data Services. Para obter mais informações, consulte Capítulo 8, "Visão Geral do Serviço de Dados REST", na página 175.
Instalação apenas do cliente	Os clientes do WebSphere eXtreme Scale podem ser instalados de forma independente, diminuindo a área de cobertua de instalação para aplicativos WebSphere eXtreme Scale. Para obter mais informações, consulte as informações sobre instalação e implementação do WebSphere eXtreme Scale no *Guia de Administração*.

Recursos Reprovados

Tabela 2. Recursos Reprovados

Reprovação	Ação de migração recomendada
	Crie um domínio do serviço de catálogo no console administrativo do WebSphere Application Server, que cria a mesma configuração que usando a propriedade customizada. Consulte as informações sobre como criar domínios do serviço de catálogo no *Guia de Administração* para obter mais informações.
	Use CatalogServiceManagementMBean no lugar.
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.

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 na 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, é possível 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.

Visão Geral da Implementação do Aplicativo

Antes de utilizar o WebSphere eXtreme Scale em um ambiente de produção, considere os seguintes problemas para otimizar sua implementação.

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?

Integração com Outros Produtos WebSphere Application Server

É possível 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 Java 2 Enterprise Edition (J2EE) 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.

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.

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.

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.

Capítulo 2. Visão Geral do 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 usado como um cache secundário, o backend é usado junto com a grade de dados. Esta seção descreve vários conceitos e cenários de cache e discute as topologias disponíveis para implementar uma grade de dados.

Arquitetura de Armazenamento em Cache: Mapas, Contêineres, Clientes e Catálogos

Com WebSphere eXtreme Scale, sua arquitetura pode utilizar armazenamento em cache de dados em memória local ou armazenamento em cache de dados de cliente/servidor distribuídos.

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.

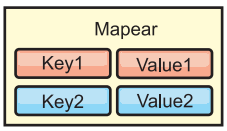


Figura 2. Mapa

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.

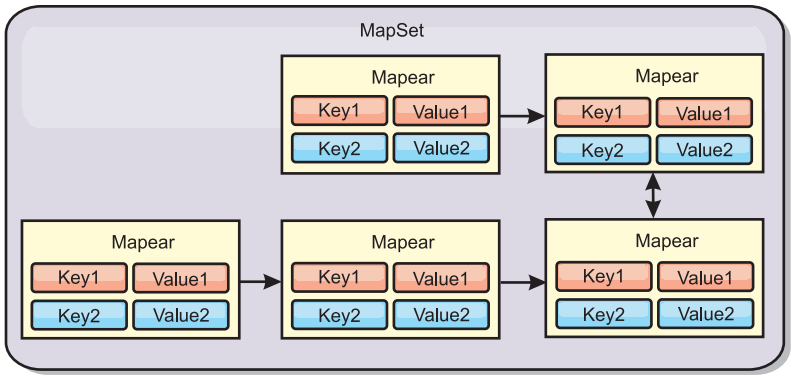


Figura 3. Conjuntos de Mapas

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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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.

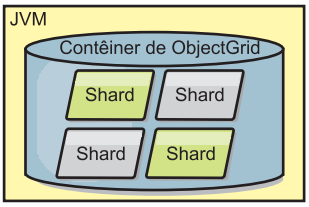


Figura 4. Contêiner

Partições hospedam um subconjunto de dados na grade. O WebSphere 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 utilizado para localizar partições na rede e o eXtreme Scale não pode re-hash o conjunto de dados inteiro após ele ser implementado. Como uma regra geral, é possível superestimar o número de partições

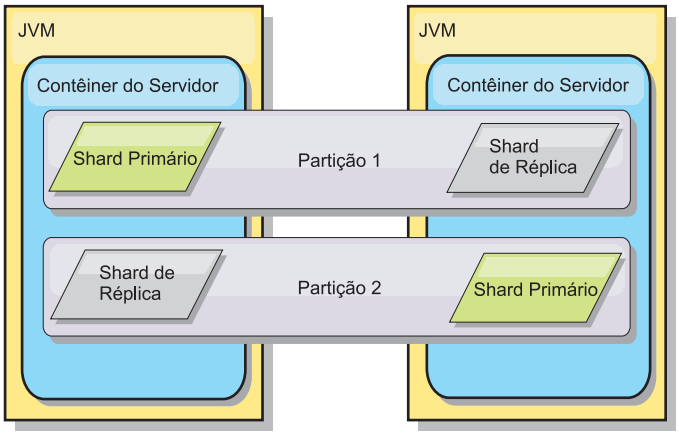


Figura 5. Partição

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.

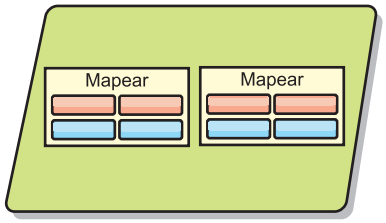


Figura 6. Shard

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.

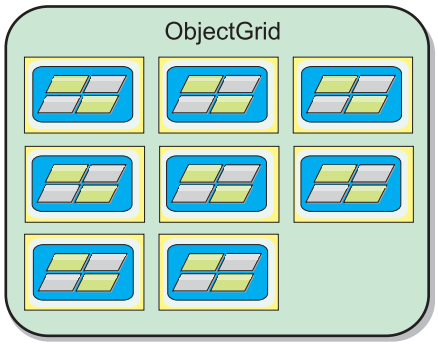


Figura 7. ObjectGrid

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 foram incluídos ou porque servidores existentes falharam, o serviço de catálogo dinâmico roteia o cliente 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 vários clientes ou servidores 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.

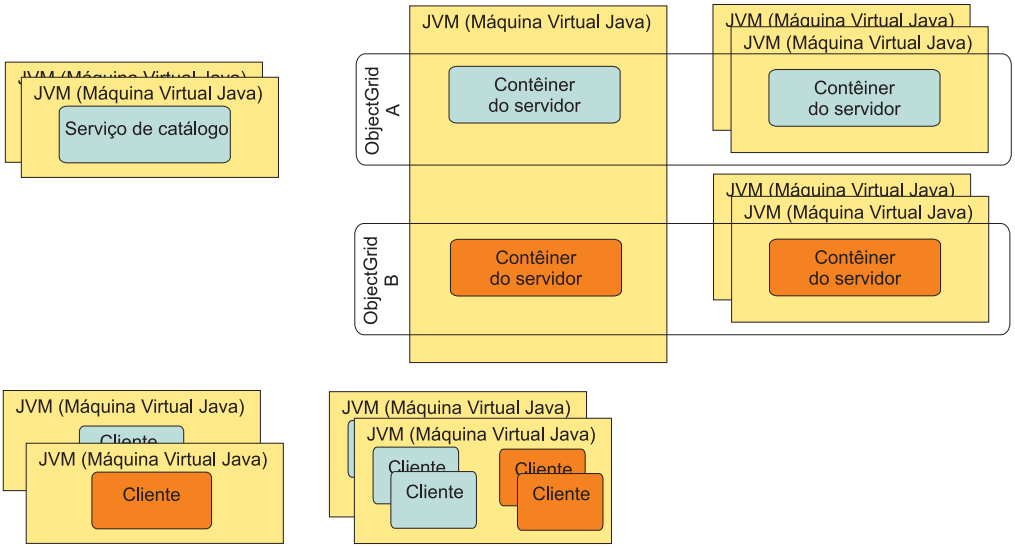


Figura 8. Possíveis Topologias

Serviços de Catálogos (Servidores de Catálogos)

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.

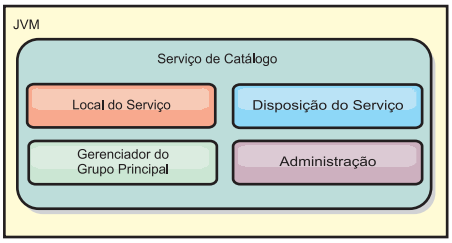


Figura 9. Serviço de Catálogo

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 posicionamento é executado como Um de N serviços eleitos no cluster. Como a política Um de N é usada, há sempre exatamente uma instância do serviço de posicionamento 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 por meio 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 um domínio do serviço de catálogo. Um domínio do serviço de catálogo consiste em várias Java virtual machines, incluindo uma JVM principal e várias Java virtual machines de backup.

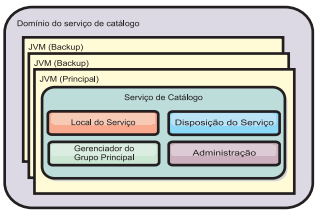


Figura 10. Domínio do Serviço de Catálogo

Topologia de Armazenamento em Cache: Armazenamento em Cache em Memória e Distribuído

Com WebSphere eXtreme Scale, sua arquitetura pode utilizar armazenamento em cache de dados em memória local ou armazenamento em cache de dados de cliente/servidor distribuídos.

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.

Cache de Memória Local

Em um caso mais simples, o eXtreme Scale pode ser utilizado como um cache de grade de dados em memória local (não distribuído). O caso local pode beneficiar especialmente aplicativos de alta simultaneidade nos quais vários encadeamentos precisam acessar e modificar 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.

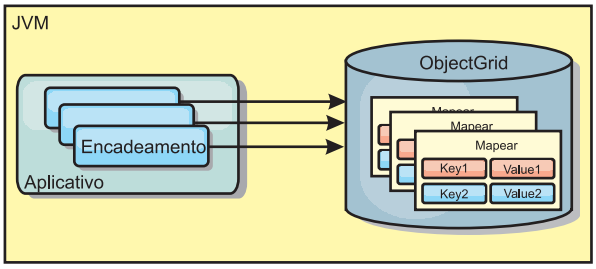


Figura 11. Cenário de Cache em Memória Local

Vantagens

- Configuração simples: Um ObjectGrid pode ser criado programaticamente ou declarativamente com o arquivo XML do 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

Para um cache do WebSphere eXtreme Scale local, você deve garantir que o cache seja sincronizado se houver vários processos com instâncias de cache independentes. Para isso, ative um cache replicado por peer com JMS.

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

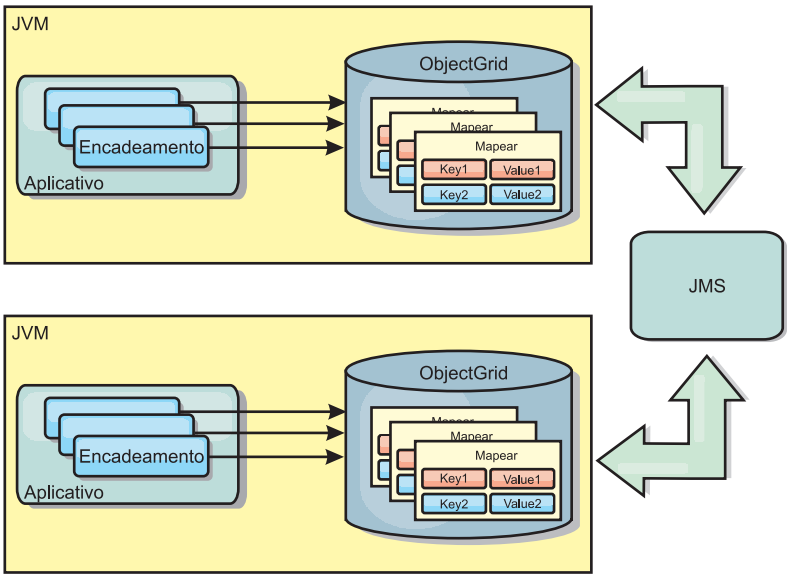


Figura 12. Cache Replicado pelo Peer com Alterações que são Propagadas com JMS

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.

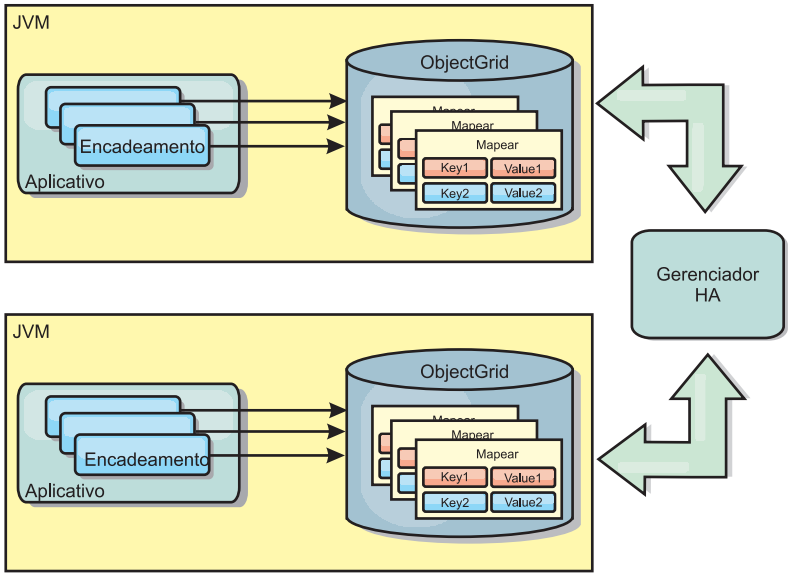


Figura 13. Cache Replicado pelo Peer com Alterações que são Propagadas com o Gerenciador de Alta Disponibilidade

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. O cache compartilhado elimina 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). Pelo 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.

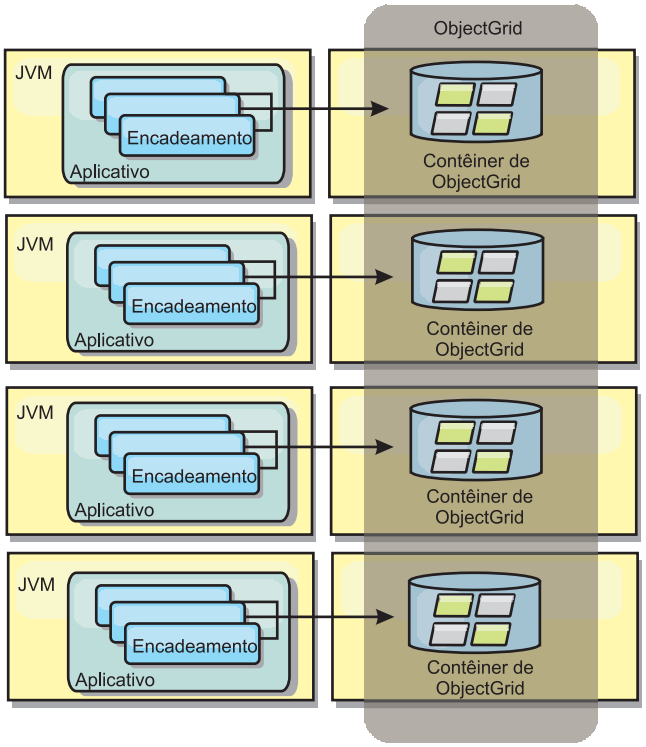


Figura 14. Cache Distribuído

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.

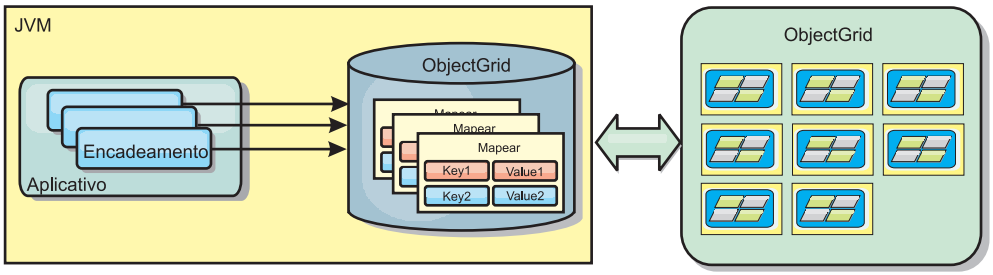


Figura 15. Cache Local

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 o tópico sobre bloqueio de entrada de mapa para obter detalhes sobre 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.

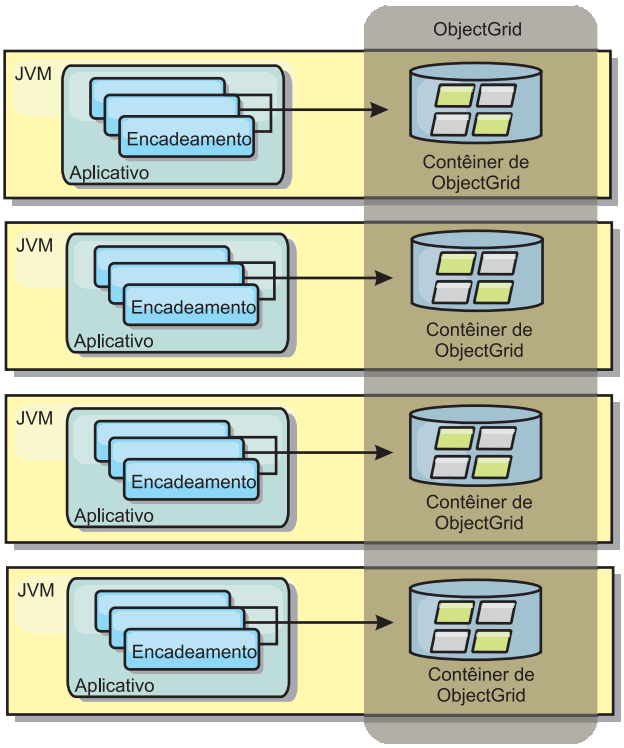


Figura 16. Cache Integrado

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*.

Topologias de Replicação de Grade Multimestre (AP)

Usando o recurso de replicação assíncrona multimestre, duas ou mais grades podem se tornar espelhos exatos umas das outras. Este espelhamento é executado

usando a replicação assíncrona entre links conectando as grades. Cada grade é hospedada dentro de um "domínio" completamente independente, que possui seu próprio serviço de catálogo, servidores de contêiner e um nome de domínio exclusivo. Com o recurso de replicação assíncrona multimestre, É possível usar links para interconectar uma coleta desses domínios e, em seguida, sincronizar os domínios, usando a replicação sobre os links. O eXtreme Scale permite construir quase qualquer topologia, pois a definição dos links entre domínios é deixada para você.

Domínios: Grades com Características Específicas

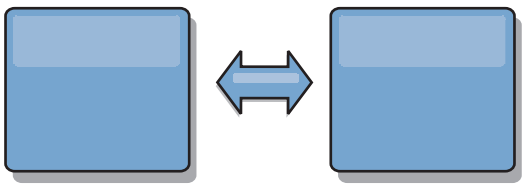
Grades usadas em topologias de replicação multimestre são conhecidas como *domínios*. Cada domínio deve ter as seguintes características:

- Ter um serviço de catálogo privado com um nome de domínio exclusivo
- Ter o mesmo nome de grade que outras grades no domínio
- Ter o mesmo número de partições que outras grades no domínio
- Ser uma grade FIXED_PARTITION (grades PER_CONTAINER não podem ser replicadas)
- Ter o mesmo número de partições (pode o não ter o mesmo número e tipos de réplicas)
- Ter os mesmos tipos de dados sendo replicados como outras grades no domínio
- Ter o mesmo nome do conjunto de mapas, nomes de mapas e modelos de mapas dinâmicos que outras grades no domínio

Depois que os domínios na topologia tiverem sido iniciados, quaisquer grades com as características precedentes serão replicadas. Observe que sua política de replicação será ignorada.

Links Conectando Domínios

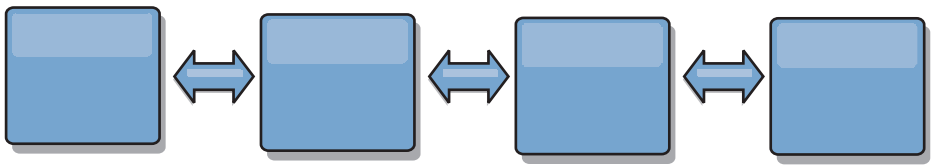
Uma infraestrutura de grade de replicação é um gráfico conectado de domínios com links bidirecionais entre eles. Um link permite que dois domínios troquem mudanças de dados. Por exemplo, a topologia mais simples é um par de domínios com um único link entre eles. Os domínios são denominados começando com “A” e, em seguida, “B” e assim por diante a partir da esquerda. O link pode cruzar uma WAN (Wide Area Network), ampliando grandes distâncias. Se o link for interrompido, as mudanças ainda podem ser feitas nos dados em qualquer um dos domínios. As mudanças são reconciliadas mais tarde, quando o link reconecta os domínios. Os links tentarão automaticamente reconectar se a conexão com a rede for interrompida.



Depois que você configurar os links, o eXtreme Scale primeiro tentará tornar cada domínio idêntico e, em seguida, tentará manter as condições idênticas conforme as mudanças ocorrerem em qualquer domínio. O objetivo do eXtreme Scale é que cada domínio seja um espelho exato de cada outro domínio conectado pelos links. Os links de replicação entre os domínios ajudam a garantir que as mudanças feitas em um domínio sejam copiadas para os outros domínios.

Topologias em Linha

Embora esteja entre as topologias mais simples, uma topologia em linha demonstra algumas qualidades dos links. Primeiro, não é necessário que um domínio esteja conectado diretamente a outro domínio para receber as mudanças. O domínio B puxará as mudanças do Domínio A. O domínio C recebe mudanças do Domínio A por meio do Domínio B, que conecta os Domínios A e C. De forma semelhante, o Domínio D recebe mudanças dos outros domínios por meio do Domínio C. Esta habilidade envia o carregamento das mudanças de distribuição para longe da origem das mudanças.



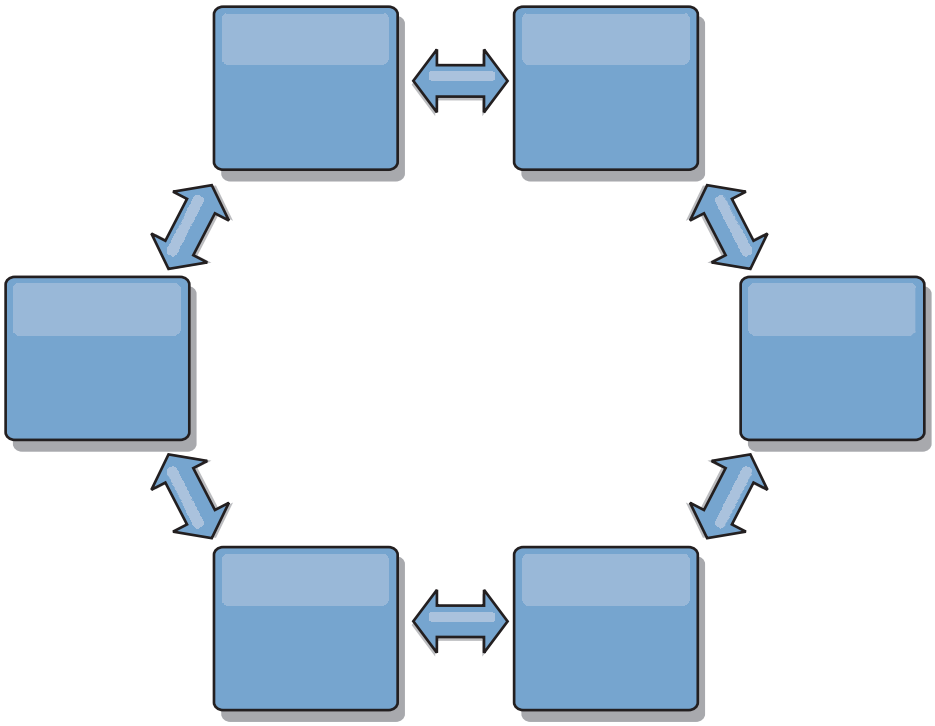
Observe que se o Domínio C falhar, os seguintes eventos poderão ocorrer.

1. O Domínio D pode ficar órfão até o Domínio C ser reiniciado
2. o Domínio C pode se sincronizar com o Domínio B, que é uma cópia do Domínio A
3. O Domínio D pode usar o Domínio C para se sincronizar com as mudanças nos Domínios A e B ocorridas enquanto o Domínio D estava órfão (enquanto o Domínio C estava inativo)

No final, os Domínios A, B, C e D podem se tornar, todos, idênticos uma aos outros novamente.

Topologias em Anel

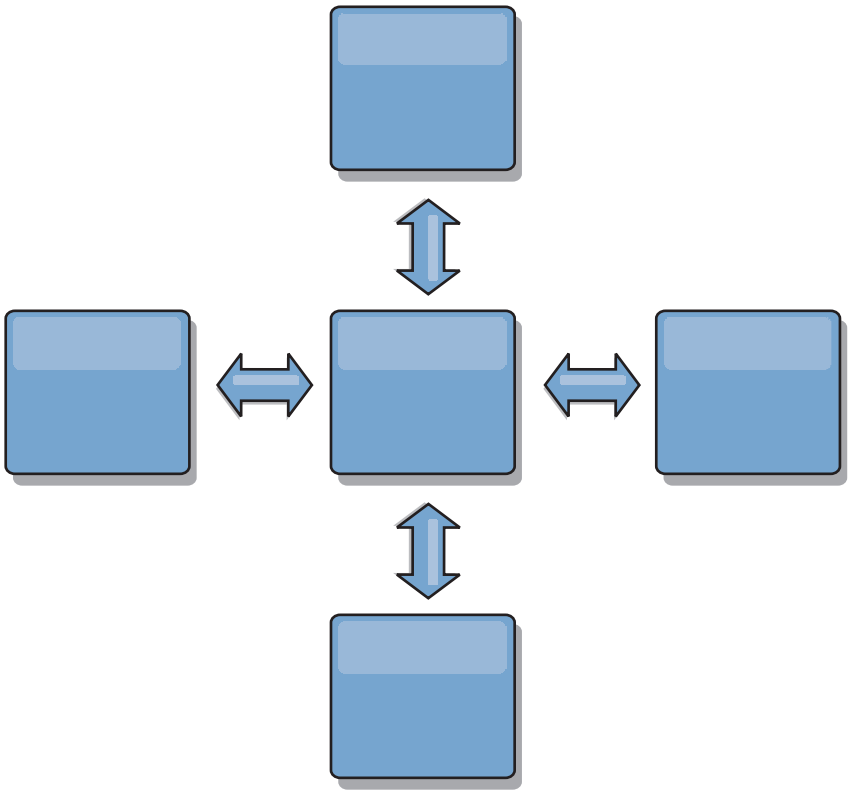
As topologias em anel são um exemplo de uma topologia mais resiliente. Embora um domínio ou um único link possa falhar, os domínios sobreviventes ainda podem obter mudanças viajando ao redor do anel, longe da falha. Cada domínio tem dois links para os outros domínios. Cada domínio tem no máximo dois links, não importa o tamanho de uma topologia em anel. A latência para propagar mudanças pode ser grande, pois as mudanças de um domínio particular podem precisar viajar por meio de vários domínios antes que todos os domínios vejam todas as mudanças. Uma topologia em linha tem o mesmo problema.



Descreva uma topologia em anel mais sofisticada, com um domínio-raiz no centro do anel. O domínio-raiz age como um centro de roteamento central, enquanto os outros domínios agem como centros de roteamento remotos para as mudanças que ocorrem no domínio-raiz. O domínio-raiz pode arbitrar mudanças entre os domínios. Se uma topologia em anel contiver mais de um anel ao redor de um domínio-raiz, o domínio-raiz poderá arbitrar mudanças apenas entre os domínios no anel mais interno. No entanto, os resultados da arbitragem se espalham para os domínios nos outros anéis.

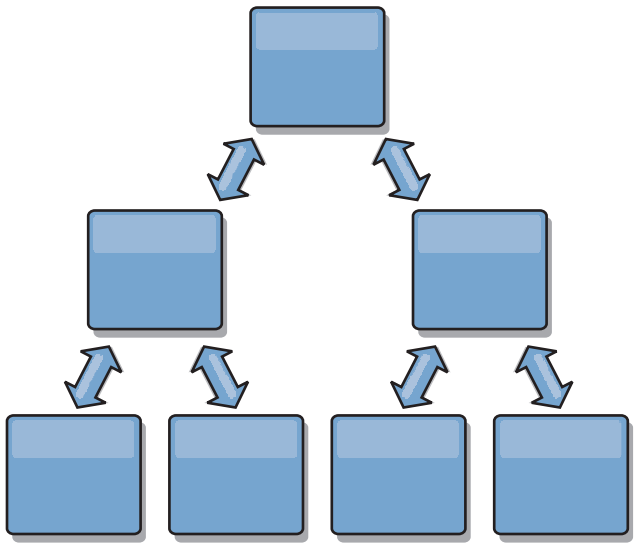
Topologias Hub e Spoke

Uma topologia hub e spoke tem melhor latência, o que significa que as mudanças viajam no máximo por meio de um domínio intermediário (o hub), mas apresenta alguns outros problemas. Ela tem um domínio central que age como um hub. Este "domínio hub" é conectado a cada "domínio spoke" usando um link. Claramente, o encargo de distribuir mudanças entre os domínios fica com o hub. O hub age como um centro de roteamento para colisões, uma configuração que pode ser importante para determinados cenários. Em um ambiente com uma taxa de atualização alta, talvez seja necessário executar o hub em mais hardware que o spoke, para que ele possa continuar ativo. O eXtreme Scale é projetado para escalar linearmente, o que significa que é possível aumentar o hub, conforme necessário, sem dificuldade. No entanto, se o hub falhar, as mudança não serão distribuídas até ele ser reiniciado. As mudanças nos domínios spoke serão distribuídas depois que o hub for reconectado.



Topologias em Árvore

Um último exemplo de topologia é uma árvore direcionada acíclica. Acíclica significa que não há ciclos ou loops. Direcionada significa que existem links apenas entre pais e filhos. Esta configuração pode ser útil para topologias como muitos domínios que não é prático ter um hub central conectado a cada spoke possível ou no qual é necessário ter a habilidade de incluir domínios-filhos sem atualizar o domínio-raiz.



Esta topologia ainda pode ter um centro de roteamento central no domínio-raiz, mas o segundo nível pode agir como centros de roteamento remotos para as mudanças que ocorrem no domínio abaixo deles. O domínio-raiz pode arbitrar as

mudanças entre os domínios no segundo nível apenas. Árvores N-ary também são possíveis. Uma árvore N-ary tem N filhos em cada nível. Cada domínio tem N espalhados.

Considerações sobre Arbitragem no Design de Topologia

Poderão ocorrer colisões de mudanças se os mesmos registros puderem ser alterados simultaneamente em dois locais. Configure cada domínio para ter quase a mesma quantia de CPU, memória, recursos de rede. É necessário observar que os domínios que executam a manipulação da colisão de mudanças (arbitragem) usam mais recursos que outros domínios. As colisões são detectadas automaticamente. Elas são resolvidas usando um entre dois mecanismos:

- **Árbitro de colisão padrão.** O protocolo padrão é usar as mudanças do domínio lexicalmente mais baixo denominado. Por exemplo, se os domínios A e B gerarem um conflito para um registro, a mudança do domínio B será ignorada. O domínio A mantém sua versão e o registro no domínio B é alterado para que corresponda ao registro do domínio A. Isso também se aplica a aplicativos nos quais os usuários ou as sessões são normalmente ligados ou têm afinidade com uma das grades.
- **Árbitro de colisão customizado.** Os aplicativos podem fornecer um árbitro customizado. Quando um domínio detecta uma colisão, ele chama o árbitro. Para obter informações sobre como desenvolver um 'bom' árbitro customizado, consulte Desenvolvendo árbitros customizados para replicação multimestre.

Para topologias nas quais as colisões são possíveis, considere a possibilidade de usar uma topologia hub e spoke ou uma topologia em árvore. Essas duas topologias são úteis para evitar colisões sem fim, que podem acontecer quando:

1. Vários domínios experimentam uma colisão
2. Cada domínio resolve a colisão localmente, produzindo revisões
3. As revisões colidem, resultando em revisões de revisões
4. E assim por diante, uma vez que as revisões são propagadas entre os vários domínios tentando atingir sincronicidade

Para evitar colisões sem fim, escolha um domínio específico – um *domínio de arbitragem* – como o manipulador de colisão para um subconjunto de domínios. Por exemplo, uma topologia hub e spoke pode usar o hub como o manipulador de colisão. O manipulador de colisão spoke ignora quaisquer colisões detectadas pelos domínios spoke. O domínio hub criará revisões, evitando revisões de colisão fora de controle. O domínio designado para tratar colisões deve ser vinculado a todos os domínios para os quais ele é responsável por resolver colisões. Em uma topologia em árvore, os domínios pais internos resolvem colisões para seus filhos imediatos. Em contraste, se usar uma topologia em anel, não será possível designar um domínio no anel como o resolvedor.

A tabela a seguir resume as abordagens de arbitragem que são mais compatíveis com várias topologias.

Tabela 3. Abordagens de Arbitragem. Esta tabela define se a arbitragem de aplicativo é compatível com várias tecnologias.

Topologia	Arbitragem de aplicativo?	Notas
Uma linha de dois domínios	Sim	Escolha um domínio como o árbitro.

Tabela 3. Abordagens de Arbitragem (continuação). Esta tabela define se a arbitragem de aplicativo é compatível com várias tecnologias.

Topologia	Arbitragem de aplicativo?	Notas
Uma linha de três domínios	Sim	O domínio do meio deve ser o árbitro. Pense no domínio do meio como o hub em uma topologia hub e spoke simples.
Uma linha de mais de três domínios	Não	A arbitragem de aplicativo não é suportada.
Um hub com N spokes	Sim	Hub com links para todos os spokes deve ser o domínio de arbitragem.
Um anel de N domínios	Não	A arbitragem de aplicativo não é suportada.
Uma árvore acíclica, direcionada (árvore N-ary)	Sim	Todos os nós-raiz devem arbitrar seus descendentes diretos apenas.

Considerações sobre Links no Design de Topologia

De forma ideal, uma topologia inclui um número mínimo de links enquanto otimiza trade-offs entre latência de mudança, tolerância a falhas e características de desempenho.

- **Latência de mudança**

  A latência de mudança é determinada pelo número de domínios intermediários pelos quais uma mudança deve passar antes de chegar a um domínio específico.

  Uma topologia tem a melhor latência de mudança quando elimina domínios intermediários vinculando cada domínio a outro domínio. No entanto, um domínio deve executar o trabalho de replicação em proporção ao seu número de links. Para topologias grandes, o número completo de links a ser definido pode representar uma carga administrativa.

  A velocidade com que uma mudança é copiada para outros domínios depende de fatores adicionais, como:

  – CPU e largura de banda da rede no domínio de origem
  – O número de domínios e links intermediários entre o domínio de origem e o de destino
  – Os recursos de CPU e de rede disponíveis para os domínios de origem, destino e intermediários

- **Tolerância a Falhas**

  A tolerância a falhas é determinada por quantos caminhos existem entre dois domínios para a replicação de mudança.

  Se existir apenas um único link entre os domínios, se o link falhar, as mudanças não serão propagadas. Se o único link de um domínio para outro domínio passar por domínios intermediários, as mudanças não serão propagadas se qualquer um dos domínios intermediários estiver inativo.

  Considere a topologia em linha com três domínios A, B e C:

  ```
  A <-> B <-> C
  ```

  Se qualquer uma dessas condições for mantida, o Domínio C não verá nenhuma mudança de A:

  – O Domínio A está ativo e o domínio B está inativo
  – O link entre A e B está inativo

– O link entre B e C está inativo

Em contraste, uma topologia em anel permite que cada domínio pegue mudanças de qualquer direção.

```
A <-> B <-> C <-> de volta para A
```

Por exemplo, se o domínio B estiver inativo, o domínio C ainda pode pegar mudanças diretamente do domínio A.

Um design hub e spoke é suscetível a que o hub fique inativo, uma vez que todas as mudanças são pegas por meio do hub. No entanto, vale lembrar que um único domínio ainda é uma grade totalmente tolerante a falhas que pode ter falhas grosseiras como problemas na WAN ou no datacenter físico.

- **Desempenho**

  O número de links definido em um domínio afeta o desempenho. Mais links usam mais recursos e o desempenho de replicação pode diminuir como resultado. A habilidade de pegar mudanças para um domínio A por meio de outros domínios efetivamente libera o domínio A de replicar suas transações em todo lugar. *O carregamento de distribuição de mudança em um domínio é limitado ao número de links que ele usa. Não tem nenhuma ligação com quantos domínios estão na topologia.* Esta propriedade fornece escalabilidade e permite que a carga de distribuição de mudança seja compartilhada pelos domínios na topologia, em vez de colocar a carga em um único domínio.

  Um domínio pode pegar mudanças indiretamente por meio de outros domínios. Considere uma topologia em linha com cinco domínios.

  ```
  A <=> B <=> C <=> D <=> E
  ```

  – A pega mudanças de B, C, D e E por meio de B
  – B pega mudanças de A e C diretamente e mudanças de D e E por meio de C
  – C pega mudanças de B e D diretamente e mudanças de A por meio de B e E por meio de D
  – D pega mudanças de C e E diretamente e mudanças de A e B por meio de C
  – E pega mudanças de D diretamente e mudanças de A, B e C por meio de D

  O carregamento de distribuição nos domínios A e E é o mais baixo, porque eles têm, cada um, um link apenas para um único domínio. O carregamento de distribuição nos domínios B, C e D é o dobro do carregamento nos domínios A e E porque os domínios B, C e D têm, cada um, um link para dois domínios. Essa distribuição de carregamentos poderia permanecer constante, mesmo que a linha contivesse 1000 domínios, pois o carregamento depende do número de links de cada domínio, não do número geral de domínios na topologia.

Considerações sobre Desempenho

Leve em consideração as seguintes limitações quando usar as topologias de replicação multimestre.

- **Ajuste de distribuição de mudança** (discutido anteriormente)
- **Latência de replicação** (discutido anteriormente)
- **Desempenho do link de replicação** O eXtreme Scale cria um único soquete TCP/IP entre qualquer par de JVMs. Todo tráfego entre essas JVMs ocorre por meio do soquete, incluindo a replicação multimestre. Como os domínios são hospedados em pelo menos N JVMs de contêiner, fornecendo pelo menos N links TCP para os domínios do mesmo nível, os domínios com números maiores de contêineres terão níveis mais altos de desempenho de replicação. Mais contêineres significa mais recursos de CPU e de rede.

- **Ajuste da janela de deslizamento TCP e RFC 1323** A ativação do suporte RFC 1323 em ambos os terminais de um link permite mais dados para um roundtrip, resultando em um maior rendimento. A técnica expande a capacidade da janela em um fator de aproximadamente 16.000.

  A rechamada desses soquetes TCP usa um mecanismo de janela de deslizamento para controlar o fluxo de dados em massa, o que geralmente limita o soquete a 64 KB para um intervalo de roundtrip. Se o intervalo de roundtrip for de 100 ms, a largura de banda será limitada a 640 KB/segundo sem ajuste adicional. Usar totalmente a largura de banda disponível em um link pode exigir um ajuste específico para um sistema operacional. A maioria dos sistemas operacionais inclui parâmetros de ajuste, incluindo opções RFC 1323, para aprimorar o rendimento por meio dos links de alta latência.

  Vários fatores podem afetar o desempenho de replicação:

  – A velocidade com que o eXtreme Scale pode pegar mudanças.
  – A velocidade com que o eXtreme Scale pode atender pedidos de replicação de tomada.
  – A capacidade da janela de deslizamento.
  – Ajuste do buffer de rede em ambos os lados de um link para permitir que o eXtreme Scale pegue mudanças por meio do soquete o mais rápido possível.
- **Serialização de Objeto** Todos os dados devem ser serializáveis. Se um domínio não estiver usando COPY_TO_BYTES, o domínio deverá usar a serialização Java ou ObjectTransformers para otimizar o desempenho da serialização.
- **Compactação** O eXtreme Scale compacta todos os dados enviados entre os domínios por padrão. Não há nenhuma opção para desativar a compactação no release atual.
- **Ajuste de memória** *O uso de memória para uma topologia de replicação multimestre é independente do número de domínios na topologia.*

  A ativação da replicação multimestre inclui uma sobrecarga fixa por Entrada de mapa para tratar a versão. Cada contêiner também rastreia uma quantia fixa de dados para cada domínio na topologia. Uma topologia com dois domínios usa aproximadamente a mesma memória que uma topologia com 50 domínios. O eXtreme Scale não usa logs de reprodução ou filas semelhantes em sua implementação, o que significa que se um link de replicação não estiver disponível por um período substancial de tempo, não haverá nenhuma estrutura de dados aumentando de tamanho, aguardando para retomar a replicação quando o link for reiniciado.

Vários Datacenters com FIXED_PARTITION

Agora, é possível usar uma grade FIXED_PARTITION entre dois ou mais datacenters. Cada datacenter precisa de seu próprio domínio, em termos de replicação multimestre. Cada datacenter pode ler e gravar dados em relação ao domínio local. Essas mudanças serão propagadas para os outros datacenters usando os links que você definiu.

Clientes Totalmente Replicados

Essa variação de topologia envolve um par de servidores do eXtreme Scale sendo executados como um hub. Cada cliente cria uma grade autocontida de contêiner único com um catálogo na JVM do cliente. Um cliente usa sua grade para conectar ao catálogo do hub, fazendo com que o cliente sincronize com o hub assim que obtiver uma conexão com o hub.

Todas as mudanças feitas pelo cliente são locais para o cliente e são replicadas de forma assíncrona para o hub. O hub age como um domínio de arbitragem, distribuindo mudanças para todos os clientes conectados. A topologia de clientes totalmente replicada fornece um bom cache L2 para um mapeador relacional de objeto, como OpenJPA. As mudanças serão distribuídas rapidamente entre JVMs de cliente por meio do hub. Contanto que o tamanho do cache possa estar contido no espaço de heap disponível dos clientes, esta topologia será uma boa arquitetura para este estilo de L2.

Use várias partições para escalar o domínio do hub em várias JVMs, se necessário. Como todos os dados ainda devem se ajustar em uma única JVM de cliente, usar várias partições aumenta a capacidade do hub para distribuir e arbitrar mudanças, mas não altera a capacidade de um único domínio.

Limitações

Leve em consideração as seguintes limitações ao decidir se e como usar as topologias de replicação multimestre.

- **Tenha cuidado com a configuração de carregadores de classe com vários domínios**

  Os domínios devem ter acesso a todas as classes que são usadas como chaves e valores. Todas as dependências devem ser refletidas em todos os caminhos da classe para as JVMs do contêiner da grade para todos os domínios. Se um plug-in CollisionArbiter recuperar o valor para uma entrada de cache, as classes para os valores deverão estar presentes para o domínio que está chamando o árbitro.

- **O uso de carregadores não é recomendado**

  Os carregadores podem ser usados para mudanças de interface entre uma grade e um banco de dados. É improvável que todas as grades (domínios) em uma topologia sejam colocadas geograficamente com o mesmo banco de dados. A latência de WAN e outros fatores podem renderizar este caso de uso não desejável.

  O pré-carregamento da grade é outro problema que requer um design cuidadoso. Geralmente, quando uma grade é reiniciada, ela é pré-carregada novamente. O pré-carregamento não é necessário ou mesmo desejável quando estiver usando a replicação multimestre. Assim que um domínio estiver on-line, ele se recarregará automaticamente com o conteúdo dos domínios aos quais está vinculado. Como resultado, não há necessidade de iniciar um pré-carregamento manual para uma grade que é um domínio em uma topologia de replicação multimestre.

  Os carregadores, geralmente, obedecem as regras de inserção e atualização. Com a replicação multimestre, as inserções devem ser tratadas como mesclagens. Quando os dados estiverem sendo pegos remotamente após o reinício de um domínio, os dados existentes serão ‘inseridos' no domínio local. Como esses dados podem já ter estado no banco de dados local, uma inserção típica falhará com uma exceção de chave duplicada no banco de dados. Em vez disso, a semântica de mesclagem deve ser usada.

  O eXtreme Scale pode ser configurado para fazer um pré-carregamento baseado em shard, usando os métodos de pré-carregamento nos plug-ins do Carregador. Não use esta técnica em uma topologia de replicação multimestre. Em vez disso, use um pré-carregamento baseado no cliente quando a topologia for iniciada (inicialmente). Permita que a topologia multimestre atualize os domínios reiniciados com uma cópia atual do que está armazenado em outros domínios

na topologia. Depois que os domínios tiverem sido iniciados, a topologia multimestre será responsável por mantê-los em sincronia.

- **EntityManager não é suportado**

  Um conjunto de mapas contendo um mapa de entidade não é replicado por meio dos domínios.
- **Mapas de matriz de byte não são suportados**

  Um conjunto de mapas contendo um mapa configurado com COPY_TO_BYTES não é replicado por meio dos domínios.
- **Write-behind não é suportado**

  Um conjunto de mapas contendo um mapa configurado com o suporte write-behind não é replicado por meio dos domínios.

Integração com o Banco de Dados: Armazenamento em Cache Write-behind, Sequencial e Lateral

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.

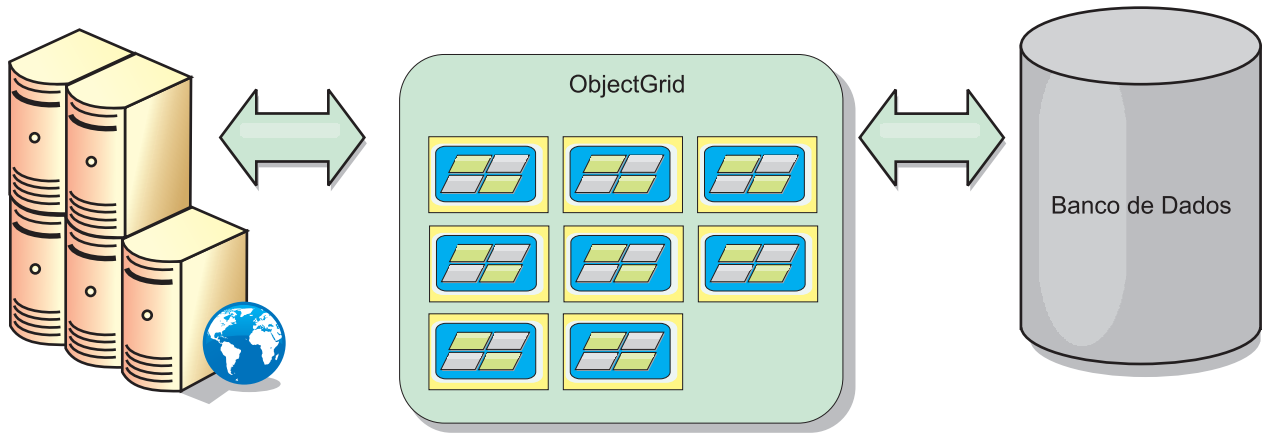


Figura 17. ObjectGrid como um Buffer de Banco de Dados

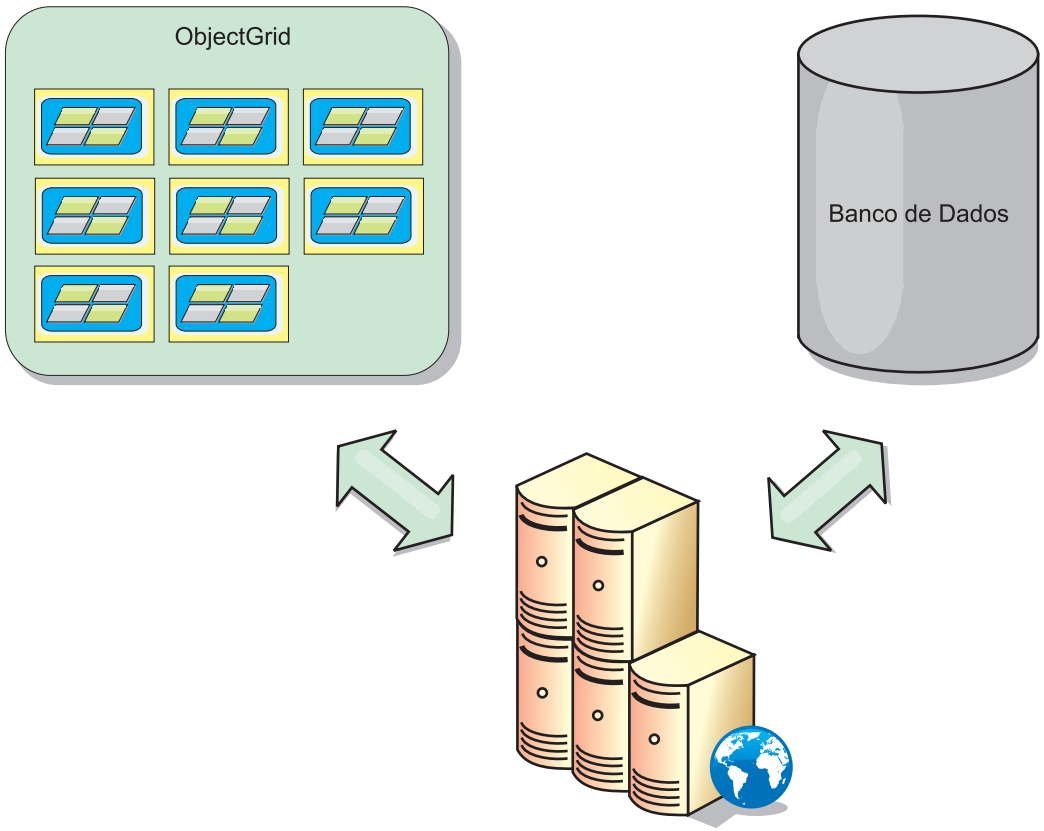


Figura 18. ObjectGrid como um Cache Secundário

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 funcionar efetivamente como uma substituição de 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

WebSphere eXtreme Scale é utilizado para fornecer armazenamento em cache sequencial para um banco de dados backend 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 junto 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

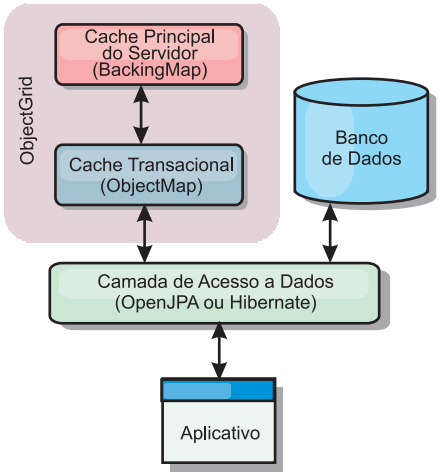


Figura 19. Cache Secundário

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.

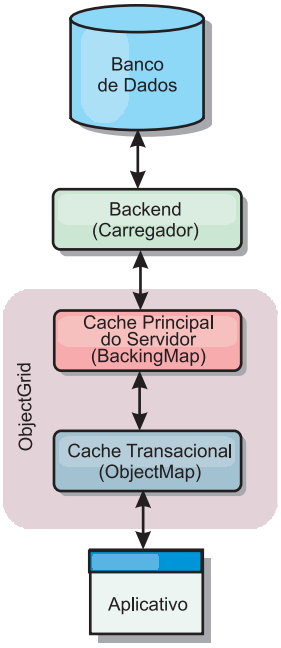


Figura 20. Cache Sequencial

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.

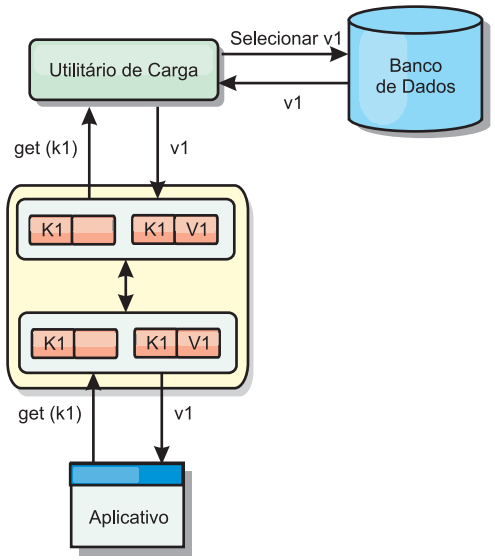


Figura 21. Armazenamento em Cache Read-through

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.

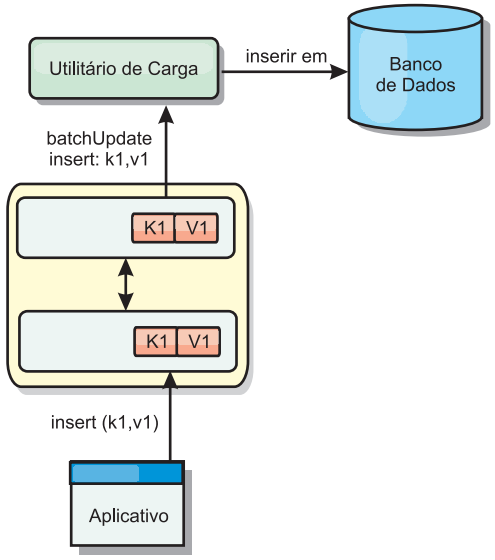


Figura 22. Armazenamento em Cache Write-through

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” para obter mais informações.

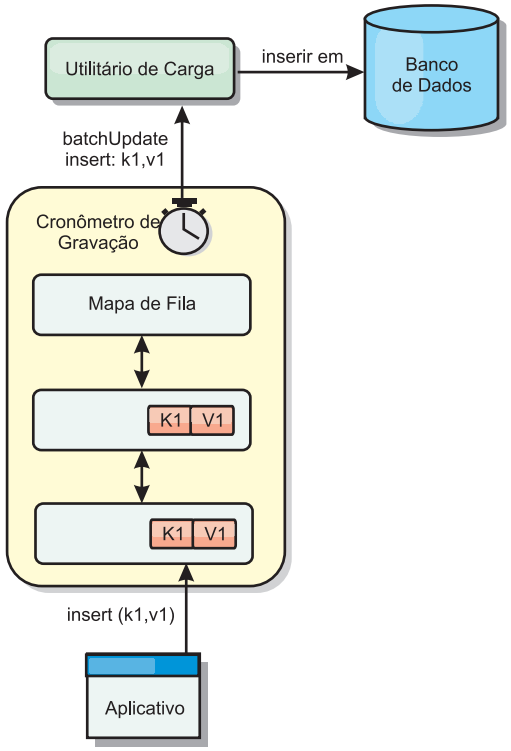


Figura 23. Armazenamento em Cache Write-behind

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

Armazenamento em Cache Write-behind

É possível utilizar armazenamento em cache write-behind para reduzir o gasto adicional que ocorre durante a atualização de um banco de dados que você está utilizando como back end.

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).

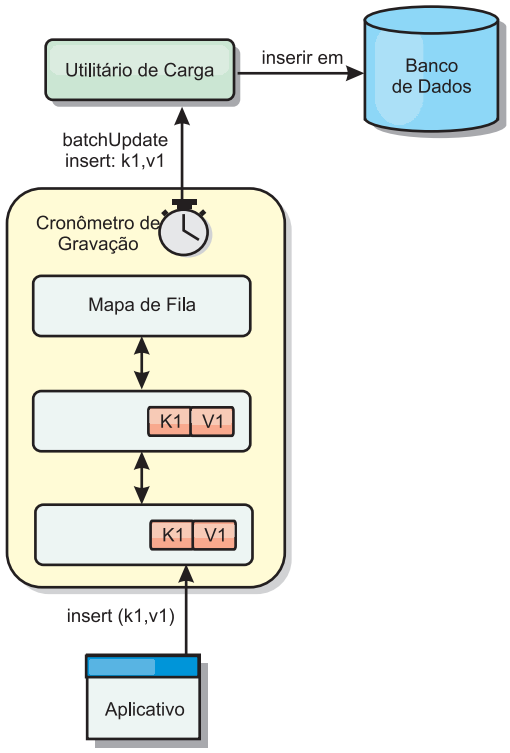


Figura 24. Armazenamento em Cache Write-behind

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 por meio 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.

Utilitários de Carga

Com um plug-in de utilitário de carga do eXtreme Scale, um mapa do eXtreme Scale pode se comportar como um cache de memória para dados que normalmente são mantidos em um armazenamento persistente no mesmo sistema ou em outro. Geralmente, um banco de dados ou sistema de arquivos é utilizado como o armazenamento persistente. Uma JVM (Java Virtual Machine) também pode ser usada como a origem de dados, permitindo que caches baseados em hub seja construído usando o eXtreme Scale. Um utilitário de carga possui a lógica para leitura e escrita de dados no armazenamento persistente.

Visão Geral

Os utilitários de carga são plug-ins de mapa de apoio que são chamados quando são feitas alterações no mapa de apoio ou quando o mapa de apoio não pode atender a um pedido de dados (um erro de cache). 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.

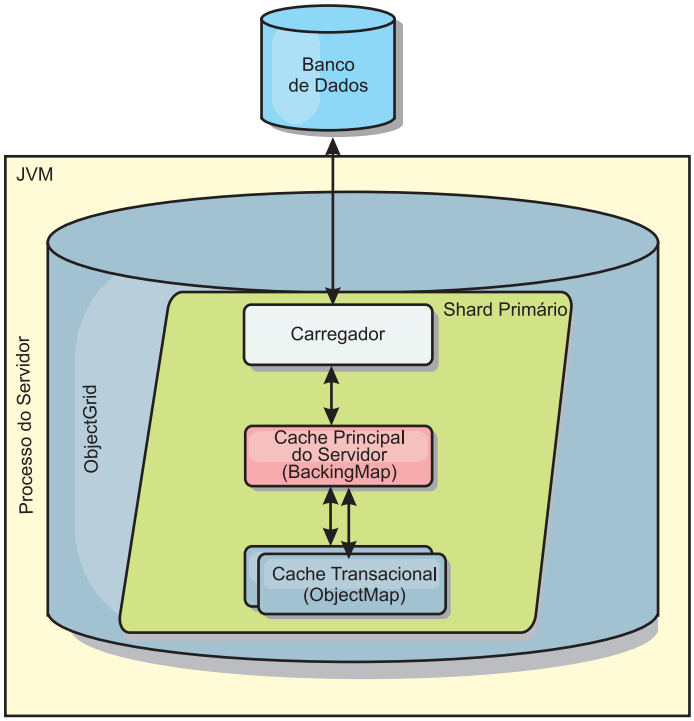


Figura 25. Utilitário de Carga

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.

Dois utilitários de carga integrados podem simplificar muito a integração com back ends de banco de dados relacional. Os utilitários de carga JPA utilizam os recursos ORM (Object-Relational Mapping) de ambas as implementações OpenJPA e Hibernate da especificação JPA (Java Persistence API). Consulte as informações sobre utilitários de carga JPA no *Visão Geral do Produto* para obter mais informações.

Utilizando um Utilitário de Carga

Para incluir um Utilitário de Carga na configuração do BackingMap, é possível utilizar a configuração programática ou a configuração do XML. Um utilitário de carga possui o seguinte relacionamento com um mapa de apoio.

- Um mapa de apoio pode ter apenas um utilitário de carga.
- Um mapa de apoio de cliente (cache local) não pode ter um utilitário de carga.

- Um definição do utilitário de carga pode ser aplicada a múltiplos mapas de apoios, mas cada mapa de apoio possui sua própria instância do utilitário de carga.

Para obter mais informações, consulte o tópico sobre gravação de um utilitário de carga no *Visão Geral do Produto*.

Abordagem da Configuração XML para Conectar um Utilitário de Carga

Um utilitário de carga fornecido pelo aplicativo pode ser plugado usando um arquivo XML. O exemplo a seguir demonstra como conectar o utilitário de carga "MyLoader" ao mapa de apoio "map1": É necessário especificar o className para seu utilitário de carga, os detalhes de nome e conexão do banco de dados e as propriedades do nível de isolamento. É possível usar a mesma estrutura XML se você estiver usando apenas um pré-utilitário de carga ao especificar o nome da classe de pré-utilitário de carga em vez de um nome de classe de utilitário de carga completo.

configuração do utilitário de carga com 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="grid">
        <backingMap name="map1" pluginCollectionRef="map1" lockStrategy="OPTIMISTIC" />
    </objectGrid>
</objectGrids>
<backingMapPluginCollections>
    <backingMapPluginCollection id="map1">
        <bean id="Loader" className="com.myapplication.MyLoader">
            <property name="dataBaseName"
                      type="java.lang.String"
                      value="testdb"
                      description="database name" />
            <property name="isolationLevel"
                      type="java.lang.String"
                      value="read committed"
                      description="iso level" />
        </bean>
    </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

Conectando um Utilitário de Carga Programaticamente

É possível usar somente uma configuração programática com grades locais na memória. O trecho de código a seguir demonstra como conectar o Utilitário de Carga fornecido pelo aplicativo ao mapa de apoio para map1, por meio da API do ObjectGrid:

configuração programática de um utilitário de carga

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid("grid");
BackingMap bm = og.defineMap( "map1" );
MyLoader loader = new MyLoader();
loader.setDataBaseName("testdb");
loader.setIsolationLevel("read committed");
bm.setLoader( loader );
```

Este fragmento assume que a classe MyLoader é a classe fornecida pelo aplicativo que implementa a interface com.ibm.websphere.objectgrid.plugins.Loader. Como a associação de um Loader com um mapa de suporte não pode ser alterada após a

inicialização de um ObjectGrid, o código deve ser executado antes de chamar o método initialize da interface ObjectGrid que está sendo chamada. Uma exceção IllegalStateException ocorre em uma chamada de método setLoader se ela for chamada depois de a inicialização ocorrer.

O Utilitário de Carga fornecido pelo aplicativo pode ter propriedades configuradas. No exemplo, o utilitário de carga MyLoader é utilizado para ler e gravar dados de uma tabela em um banco de dados relacional. O utilitário de carga deve especificar o nome do banco de dados e o nível de isolamento do SQL. O loader MyLoader possui os métodos setDataBaseName e setIsolationLevel que permitem que o aplicativo configure estas duas propriedades do Loader.

- O usuário 'bob' é autenticado como um usuário do eXtreme Scale. O aplicativo está acessando uma grade chamada 'mygrid' utilizando o nome da unidade de persistência de 'DB2Hibernate'. O servidor de contêiner é 'XS_Server1'. As informações resultantes seriam as seguintes:
  - **Usuário**=bob
  - **Nome da Estação de Trabalho**=XS_Server1,192.168.1.101
  - **Nome do Aplicativo**=mygrid,DB2Hibernate
  - **Dados da Conta**=1, DEFAULT,FE7954BD-0126-4000-E000-2298094151DB,com.ibm.db2.jcc.t4.b@71787178
- O usuário 'bob' é autenticado como um token do WAS. O aplicativo está acessando uma grade chamada 'mygrid" utilizando o nome da unidade de persistência de 'DB2OpenJPA'. O servidor de contêiner é 'XS_Server2'. As informações resultantes seriam as seguintes:
  - **Usuário**

    =acme.principal.UserPrincipal[Bob],acme.principal.GroupPrincipal[admin]
  - **Nome da Estação de Trabalho**=XS_Server2,192.168.1.102
  - **Nome do Aplicativo**=mygrid,DB2OpenJPA
  - **Dados da Conta**=188,DEFAULT,FE72BC63-0126-4000-E000-851C092A4E33,com.ibm.ws.rsadapter.jdbc.WSJccSQLJConnection@2b432b43

Leia sobre DB2 Performance Expert para saber como monitorar o acesso ao banco de dados.

Pré-carregamento de Dados e Aquecimento

Em vários cenários que incorporam o uso de um utilitário de carga, é possível preparar sua grade fazendo seu pré-carregamento 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.

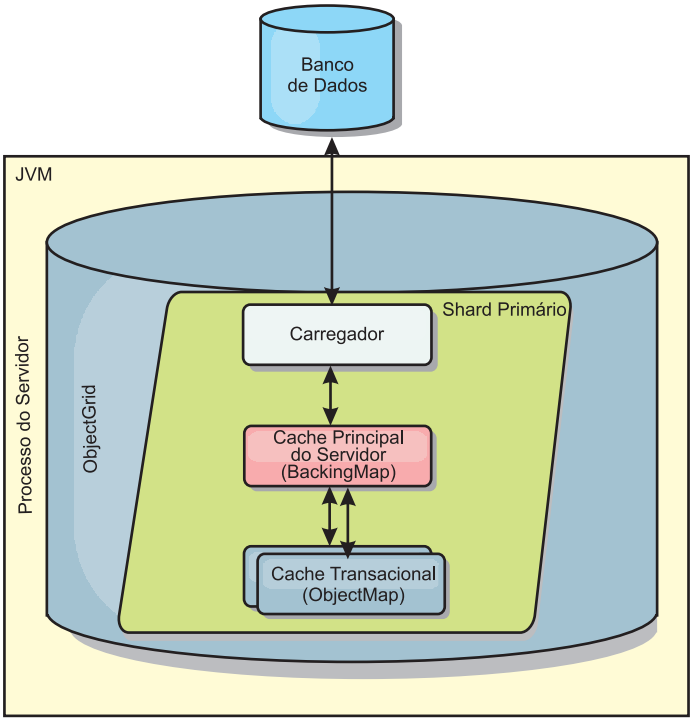


Figura 26. Plug-in do Utilitário de Carga

Consulte os detalhes sobre a utilização de um utilitário de carga no *Guia de Programação* para obter informações adicionais.

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. É possível 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 é possível usar para carregar automaticamente a grade com os provedores JPA OpenJPA ou Hibernate.

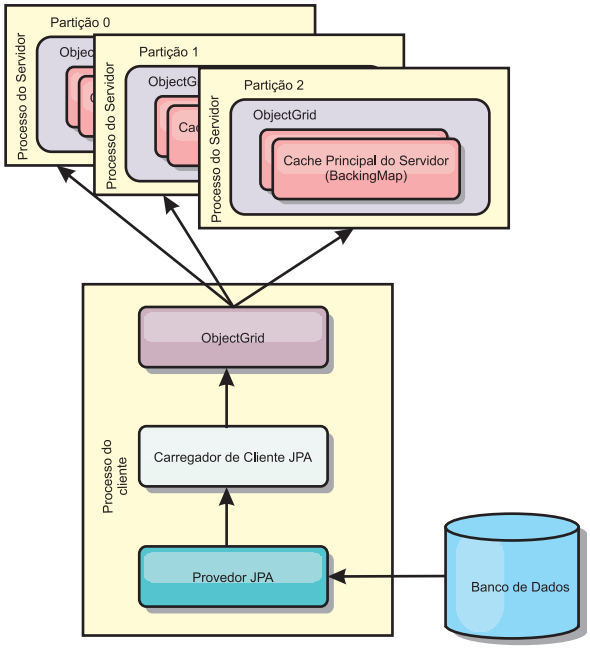


Figura 27. Utilitário de Carga do Cliente

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.

Consulte os detalhes sobre como escrever um utilitário de carga com um controlador de pré-carga de réplica no *Guia de Programação* para obter um exemplo sobre como implementar um utilitário de carga para um eXtreme Scale replicado.

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, após a transação incluir blocos de mais de 100 entradas, o benefício do desempenho diminui 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 4. 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 5. 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 6. 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 mudança. 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.

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.

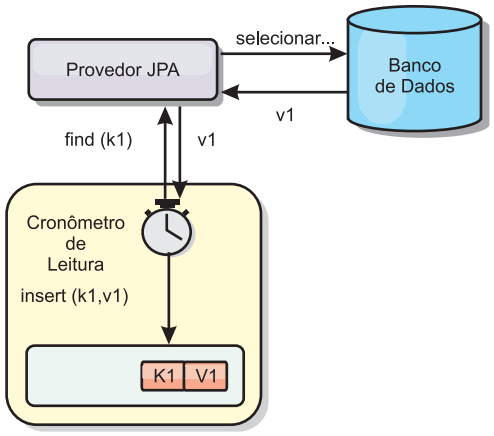


Figura 28. Atualização Periódica

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.

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 mudança. 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.

Invalidando Dados em Cache Antigos

Para reduzir a janela de tempo em que clientes podem ver dados antigos, é possível utilizar um mecanismo de invalidação programático ou baseado em evento.

Invalidação Baseada em Evento

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. É fornecido um plug-in JMS ObjectGridEventListener no eXtreme Scale que pode notificar os clientes quando o cache do servidor mudar. Esse tipo de notificação diminui a quantidade de tempo em que o cliente pode ver dados antigos.

A invalidação baseada em evento normalmente consiste nos três componentes a seguir.

- **Fila de eventos:** Uma fila de eventos armazena os eventos de mudança de dados. Ela pode ser uma fila JMS, um banco de dados, uma fila FIFO em memória ou qualquer tipo de manifesto, contanto que possa gerenciar os eventos de mudança de dados.
- **Publicador de evento:** Um publicador de evento publica os eventos de mudança de dados na fila de eventos. Um publicador de evento geralmente é um aplicativo que você cria ou uma implementação de plug-in do eXtreme Scale. O publicador de evento sabe quando os dados são alterados ou quando ele mesmo altera os dados. Quando uma transação é confirmada, eventos são gerados para os dados alterados e o publicador de eventos publica esses eventos na fila de eventos.
- **Consumidor de evento:** Um consumidor de evento consome eventos de mudança de dados. O consumidor de evento geralmente é um aplicativo para garantir que os dados da grade de destino sejam atualizados com a mudança mais recente das outras grades. Esse consumidor de evento interage com a fila de eventos para obter a mudança de dados mais recente, além de aplicar as mudanças de dados na grade de destino. Os consumidores de evento podem utilizar APIs do eXtreme Scale para invalidar dados antigos ou atualizar a grade com os dados mais recentes.

Por exemplo, JMSObjectGridEventListener tem uma opção para um modelo de cliente/servidor, no qual a fila de eventos é um destino JMS designado. Todos os processos do servidor são publicadores de eventos. Quando uma transação é confirmada, o servidor obtém as mudanças de dados e as publica no destino JMS designado. Todos os processos do cliente são consumidores de evento. Eles recebem as mudanças de dados do destino JMS designado e aplicam as mudanças no cache perto do cliente.

Consulte o tópico sobre a ativação do mecanismo de invalidação de cliente no *Guia de Administração* para obter mais informações.

Invalidação Programática

As APIs do WebSphere 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.

É possível utilizar invalidação programática com outras técnicas para determinar quando invalidar os dados. Por exemplo, esse método de invalidação utiliza mecanismos de invalidação baseados em evento para receber eventos de mudança de dados e depois utiliza as APIs para invalidar os dados antigos.

Indexação

Utilize MapIndexPlugin para construir um índice ou vários índices em um BackingMap para suportar acesso a dados sem chave.

Tipos e Configuração de Índice

O recurso de indexação é representado por MapIndexPlugin ou Index for short. O Índice é um plug-in BackingMap. Um BackingMap pode ter múltiplos plug-ins de Índice configurados, enquanto cada um seguir as regras de configuração de Índice.

É possível usar o recurso de indexação para construir um índice ou diversos índices em um BackingMap. Um índice é construído a partir de um atributo ou uma lista de atributos de um objeto no BackingMap. Este recurso fornece uma maneira para os aplicativos localizarem determinados objetos mais rapidamente. Com o recurso de indexação, os aplicativos podem localizar objetos com um valor específico ou em um intervalo com os valores de atributos indexados.

Dois tipos de indexação são possíveis: estática e dinâmica. Com a indexação estática, é necessário configurar o plug-in de índice no BackingMap antes de inicializar a instância do ObjectGrid. É possível fazer esta configuração com a configuração XML ou programática do BackingMap. A indexação estática inicia a construção de um índice durante a inicialização do ObjectGrid. O índice é sempre sincronizado com o BackingMap e está pronto para utilização. Depois de o processo de indexação estático iniciar, a manutenção do índice é parte do processo de gerenciamento de transação do eXtreme Scale. Quando as consolidações de transações mudam, estas alterações também atualizam o índice estático, e as alterações de índice são recuperadas se a transação for recuperada.

Com a indexação dinâmica, é possível criar um índice num BackingMap antes ou depois da inicialização da instância do ObjectGrid que o contém. Os aplicativos possuem controle de ciclo de vida sobre o processo de indexação dinâmica para que você possa remover um índice dinâmico quando ele não for mais necessário. Quando um aplicativo cria um índice dinâmico, o índice pode não estar pronto para utilização imediata devido ao tempo gasto na conclusão do processo de construção do índice. Como a quantidade de vezes depende dos dados indexados, a interface DynamicIndexCallback será fornecida para os aplicativos que desejem receber notificações na ocorrência de determinados eventos de indexação. Estes eventos incluem ready, error e destroy. Os aplicativos podem implementar esta interface de retorno de chamada e registrar-se no processo de indexação dinâmica.

Se um BackingMap tiver um plug-in de índice configurado, é possível obter o objeto de proxy do índice do aplicativo a partir do ObjectMap correspondente. A

chamada do método getIndex no ObjectMap e a transmissão do nome do plug-in de índice retornam o objeto de proxy do índice. Você deve efetuar o cast do objeto de proxy do índice para uma interface de índice adequada do aplicativo, como MapIndex, MapRangeIndex ou uma interface de índice customizada. Após obter o objeto de proxy do índice, é possível utilizar métodos definidos na interface do índice do aplicativo para localizar objetos armazenados em cache.

As etapas para utilizar a indexação estão resumidas na lista a seguir:

- Incluir plug-ins de indexação, estáticos ou dinâmicos, no BackingMap;
- Obter um objeto de proxy de indexação do aplicativo, emitindo o método getIndex do ObjectMap.
- Direcione o objeto de proxy de índice a uma interface de índice de aplicativo apropriado, como MapIndex, MapRangeIndex, ou uma interface de índice customizada.
- Utilizar os métodos definidos na interface Index do aplicativo para localizar objetos no cache.

A classe HashIndex é a implementação de plug-in de índice integrada que pode suportar ambas as interfaces integradas de índice do aplicativo: MapIndex e MapRangeIndex. Também é possível criar seus próprios índices. É possível incluir o HashIndex como um índice estático ou dinâmico no BackingMap, obter o objeto proxy do índice MapIndex ou MapRangeIndex e usar o objeto proxy do índice para localizar objetos em cache.

Para obter informações sobre a configuração do HashIndex, consulte Configurando o HashIndex.

Para obter mais informações sobre como escrever seu próprio plug-in de índice, consulte as informações sobre como escrever um plug-in de índice no *Guia de Programação*..

Para obter informações sobre como usar a indexação, consulte as informações sobre como usar a indexação para acesso a dados não chave no *Guia de Programação* e HashIndex Composto.

Consideração sobre a Qualidade dos Dados

Os resultados dos métodos de consulta de índice somente representar uma captura instantânea de dados em um determinado ponto no tempo. Nenhum bloqueio contra as entradas de dados é obtido depois do retorno dos resultados para o aplicativo. O aplicativo deve estar ciente de que podem ocorrer atualizações de dados em um conjunto de dados retornado. Por exemplo, o aplicativo obtém a chave de um objeto armazenado em cache executando o método findAll de MapIndex. Este objeto de chave retornado está associado a uma entrada de dados no cache. O aplicativo deve poder executar o método get no ObjectMap para localizar um objeto fornecendo o objeto chave. Se outra transação remover o objeto de dados do cache imediatamente antes de o método get ser chamado, o resultado retornado será nulo.

Considerações sobre Desempenho de Indexação

Um dos principais objetivos do recurso de indexação é melhorar o desempenho geral do BackingMap. Se a indexação não for utilizada corretamente, o desempenho do aplicativo poderá ficar comprometido. Considere os seguintes fatores antes de usar este retorno.

- **A quantidade de transações de gravação concorrentes:** O processamento de índices pode ocorrer todas as vezes que uma transação gravar dados em um BackingMap. O desempenho será afetado se muitas transações gravarem dados no mapa simultaneamente quando um aplicativo tentar operações de consulta ao índice.
- **O tamanho do conjunto de resultados que é retornado por uma operação de consulta:** Como o tamanho do conjunto de resultados aumenta, o desempenho da consulta cai. O desempenho tente a degradas quando o tamanho do conjunto de resultados é de 15% (ou mais) do BackingMap.
- **A quantidade de índices construídos sobre o mesmo BackingMap:** Cada índice consome os recursos do sistema. Conforme a quantidade dos índices construídos sobre o BackingMap aumenta, o desempenho diminui.

A função de indexação pode aumentar significativamente o desempenho do BackingMap. Os casos ideais ocorrem quando o BackingMap possui a maioria de operações de leitura, o conjunto de resultados da consulta é de uma porcentagem pequena das entradas do BackingMap, e somente poucos índices são construídos sobre o BackingMap.

Conceitos de Armazenamento em Cache de Objetos Java

O WebSphere eXtreme Scale é primariamente usado como uma grade de dados e cache para objetos Java. É possível utilizar várias APIs 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 de Armazenamento em Cache: Mapas, Contêineres, Clientes e Catálogos" 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, você deve definir classes 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 por meio 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 por meio 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 por meio 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 parte da chave precisar ser alterada, a entrada do cache deverá ser removida e reinserida.
- As chaves devem ser pequenas. Como as chaves são utilizadas em toda operação de acesso a dados, é recomendável manter a chave pequena para que ela possa ser serializada eficientemente e utilize menos memória.

- Implementa um bom hash e algoritmo de igualdade. Os métodos hashCode e equals(Object o) devem ser sempre substituídos para cada objeto-chave.
- Armazene 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

WebSphere eXtreme Scale utiliza vários processos Java para conter dados. Esses processos serializam os dados: ou seja, convertem os dados (que estão no formato de instâncias de objeto Java) em bytes e volta em objetos novamente, conforme necessário, para mover os dados entre processos do cliente e do servidor. Delegar os dados é a operação mais dispendiosa e deve ser endereçada pelo desenvolvedor de aplicativos ao projetar o esquema, configurar a grade e interagir 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 a 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.

Para obter mais informações, consulte Plug-ins para Serializar e Copiar Objetos em Cache e Boas Práticas da Interface de ObjectTransformer.

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 por meio 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.

Consulte as melhores práticas do método CopyMode no *Guia de Programação* para obter mais informações.

Inserindo Dados para Fusos Horários Diferentes

Ao inserir dados com os atributos calendar, java.util.Date e timestamp em um ObjectGrid, você deve garantir que esses atributos de data e hora sejam criados com base no mesmo fuso horário, principalmente quando implementados em vários servidores em vários fusos horários. O uso dos mesmos objetos de data e hora baseados em fuso horário pode garantir que o aplicativo esteja protegido por fuso horário e que os dados possam ser consultados pelos predicados calendar, java.util.Date e timestamp.

Sem especificar explicitamente um fuso horário ao criar objetos de data e hora, o Java utilizará o fuso horário local e poderá causar valores de data e hora inconsistentes nos clientes e servidores.

Considere um exemplo em uma implementação distribuída na qual o client1 está no fuso horário [GMT-0] e o client2 está no [GMT-6], e ambos querem criar um objeto java.util.Date com o valor '1999-12-31 06:00:00'. Então, o client1 criará o

objeto java.util.Date com o valor '1999-12-31 06:00:00 [GMT-0]' e o client2 criará o objeto java.util.Date com o valor '1999-12-31 06:00:00 [GMT-6]'. Os objetos java.util.Date não são iguais porque o fuso horário é diferente. Um problema semelhante ocorre quando você pré-carrega os dados nas partições que residem em servidores em fusos horários diferentes se o fuso horário local for utilizado para criar objetos de data e hora.

Para evitar o problema descrito, o aplicativo pode escolher um fuso horário como [GMT-0] como fuso horário base para criar objetos calendar, java.util.Date e timestamp.

Para obter mais informações, consulte o tópico sobre como consultar dados em vários fusos horários no *Guia de Programação.*

Capítulo 3. Visão Geral da Integração de Cache: JPA, Sessões e Armazenamento em Cache Dinâmico

O elemento crucial que dá ao WebSphere eXtreme Scale a capacidade para executar com tal versatilidade e confiabilidade é seu aplicativo de conceitos de armazenamento em cache para otimizar a persistência e a nova coleta de dados em virtualmente qualquer ambiente de implementação.

Carregadores 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. Inúmeras implementações comerciais e de software livre estão disponíveis.

É possível utilizar uma implementação de plug-in de utilitário de carga do Java Persistence API (JPA) com eXtreme Scale para interagir com qualquer banco de dados suportado por seu utilitário de carga escolhido. 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.

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).

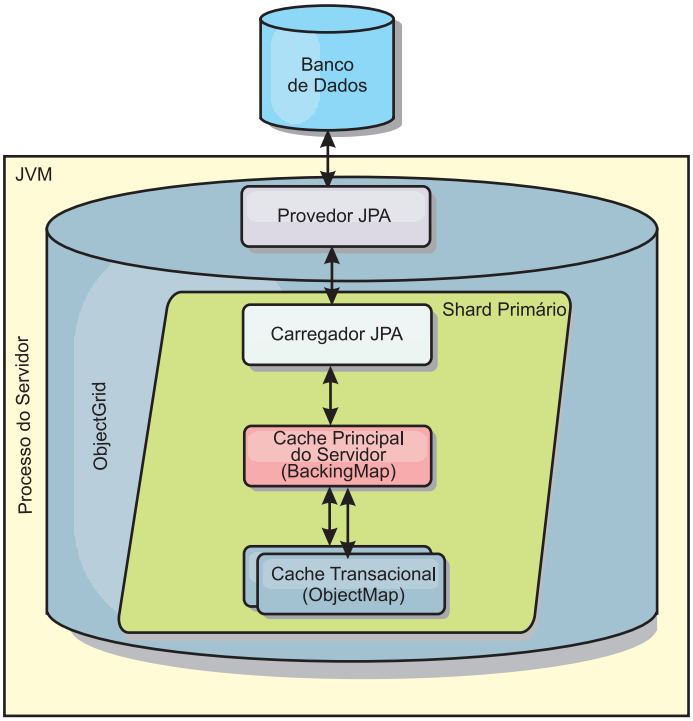


Figura 29. Arquitetura do Utilitário de Carga do JPA

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 detalhes, consulte as informações sobre o plug-in JPAEntityLoader no *Guia de Programação*.

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.

Configuração e Programação

Para obter mais informações sobre como configurar os utilitários de carga do JPA, consulte emas informações sobre os utilitários de carga do JPA no*Guia de Administração*. Para obter mais informações sobre utilitário de carga do JPA de programação, consulte o *Guia de Programação*.

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.

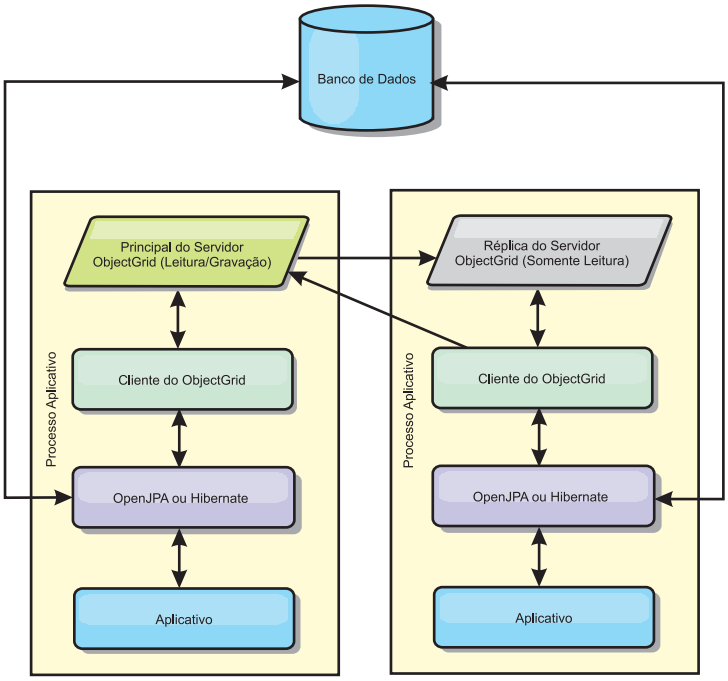


Figura 30. Topologia Integrado do JPA

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.

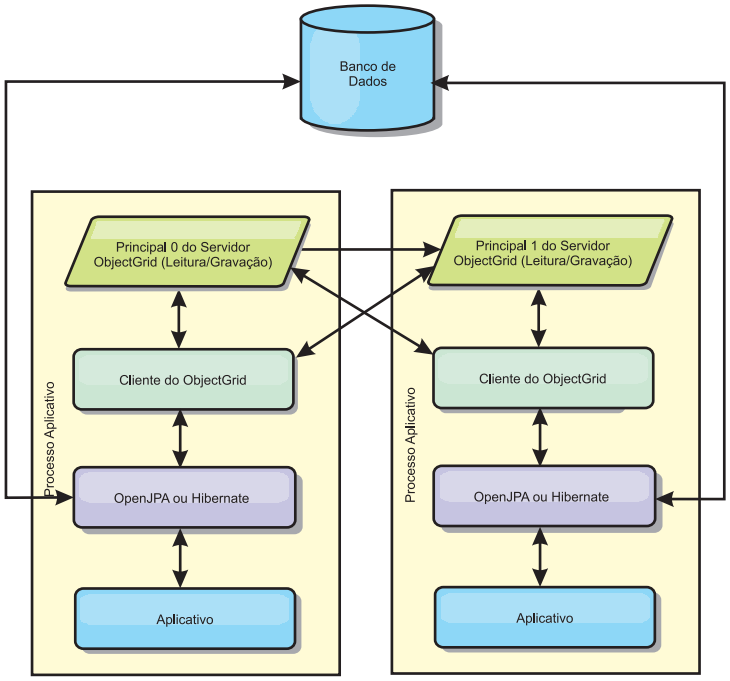


Figura 31. Topologia Particionada Integrada do JPA

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 replicação, será necessário aumentar o número de Java Virtual Machines no sistema. Caso contrário, cada JVM também armazenará uma partição de réplica, que consome tanta memória quanto uma partição primária.

Leia sobre o dimensionamento de memória e o cálculo da contagem de partições no *Guia de Administração* para maximizar o desempenho da sua configuração escolhida.

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 aproveitar as vantagens da distribuição de dados em processos separados implementando uma grade do eXtreme Scale replicada particionada. Ao contrário das configurações integradas e particionadas integradas descritas nas seções anteriores, se quiser gerenciar a grade remota, você deverá fazer isso independentemente do aplicativo e provedor JPA. Leia sobre o monitoramento do seu ambiente de implementação para obter mais informações sobre o gerenciamento de uma implementação em grade do eXtreme Scale.

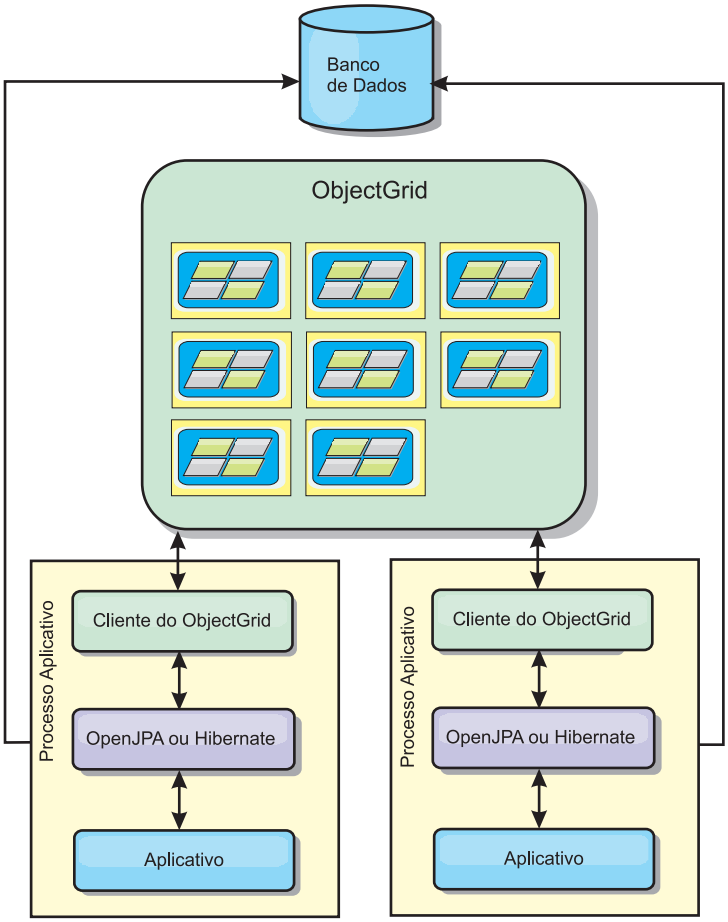


Figura 32. Topologia Remota do JPA

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.

Configuração

Para obter mais informações sobre como configurar os plug-ins de cache do JPA, consulte na seção de plug-ins no*Guia de Programação*.

Gerenciando de Sessões HTTP

O gerenciador de replicação de sessão enviado com o WebSphere eXtreme Scale pode trabalhar com o gerenciador de sessões padrão no servidor de aplicativos para replicar dados da sessão de um processo para outro para dar suporte à alta disponibilidade dos dados da sessão do usuário.

Características

O gerenciador de sessões foi projetado de forma que possa ser executado em qualquer contêiner do Java Platform, Enterprise Edition Versão 1.4. Como o gerenciador de sessões não tem nenhuma dependência nas APIs do WebSphere, ele pode suportar várias versões do WebSphere Application Server e também ambientes do servidor de aplicativos do fornecedor.

O gerenciador de sessões HTTP fornece recursos de replicação de sessão para um aplicativo associado. O gerenciador de replicação de sessão trabalha com o gerenciador de sessões do contêiner de Web para criar sessões HTTP e gerenciar os ciclos de vida de sessões HTTP que estão associadas ao 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 é hospedado 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 esse 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.
- Em datacenters múltiplos ou zonas múltiplas.

Como o Gerenciador de Sessões Funciona

O gerenciador de replicação de sessão utiliza um listener de sessão padrão para atender às mudanças de dados da sessão, além de fazer a persistência dos dados da sessão em uma instância do ObjectGrid local ou remotamente. Os dados da sessão são recarregados no caminho do pedido por meio do servlet padrão a partir da instância de ObjectGrid local ou remotamente. É possível incluir o listener de sessão e o filtro de servlet em cada módulo da Web em seu aplicativo com o conjunto de ferramentas que é fornecido com o WebSphere eXtreme Scale. Você também pode incluir manualmente esses listeners e filtros no descritor de implementação da Web de seu aplicativo.

Este gerenciador de replicação de sessão trabalha com o gerenciador de sessões do contêiner de Web de cada fornecedor para replicar dados da sessão por meio de Java virtual machines. Quando o servidor original para de funcionar, os usuários podem recuperar dados da sessão de outros servidores.

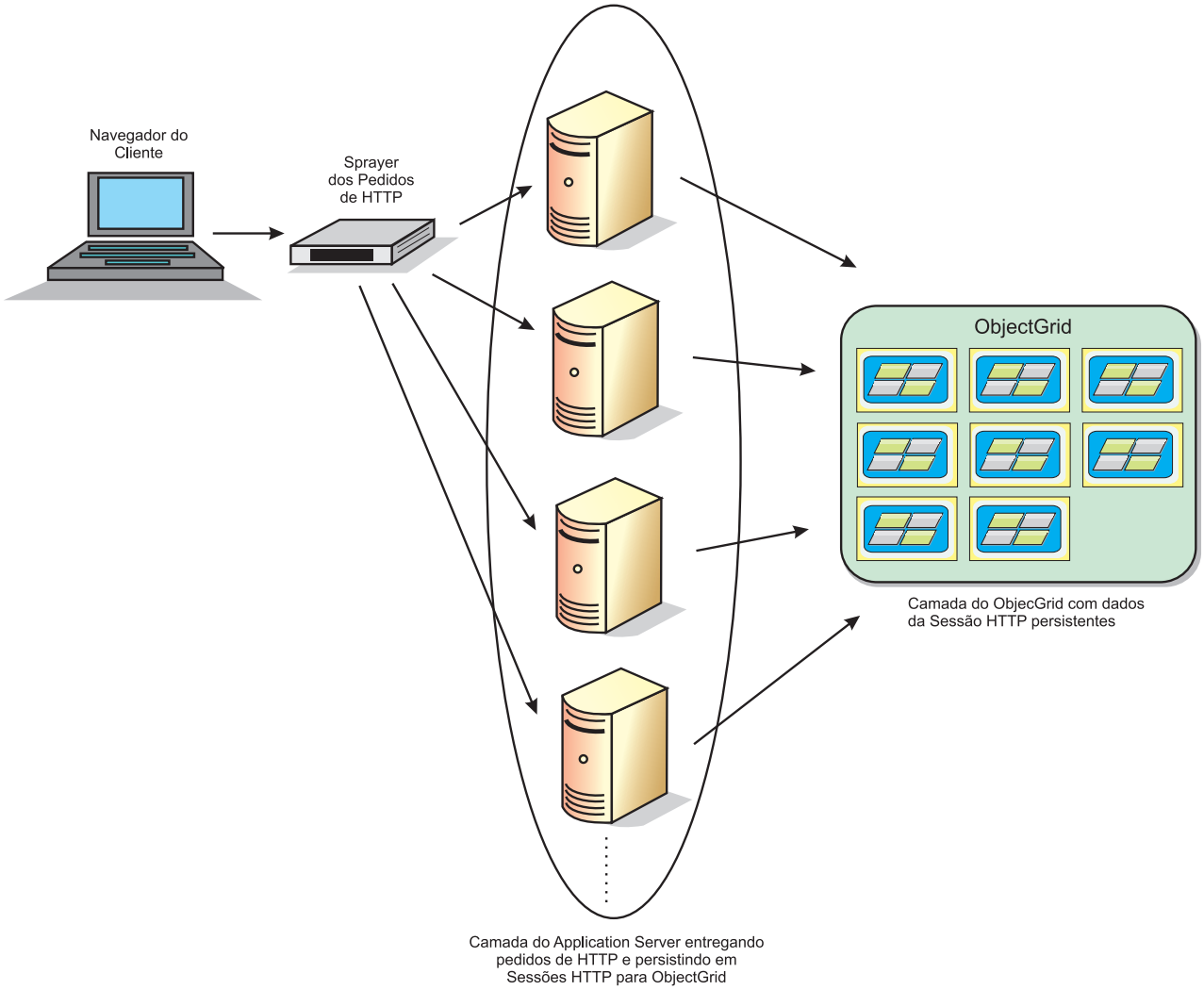


Figura 33. Topologia do Gerenciamento de Sessões HTTP com uma Configuração de Contêiner Remoto

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. Este cenário é preferível ao executar com afinidade e o desempenho for crítico
- **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 se comunica com uma grade do servidor eXtreme Scale remoto. Este cenário é preferível quando a camada do contêiner de Web não tem a memória para armazenar os dados da sessão. Os dados da sessão serão transferidos para uma camada separada, o que resultará em um uso menor de memória na camada do contêiner de Web, mas em maior latência devido à localização remota dos dados.

Inicialização do Contêiner Integrado Genérico

O eXtreme Scale inicia automaticamente um contêiner ObjectGrid integrado dentro de qualquer processo de aplicativo-servidor quando o contêiner de Web inicializa o listener de sessão ou o filtro de servlet, se a propriedade objectGridType for configurada para EMBEDDED. Consulte Parâmetros de inicialização do contexto de servlet para obter detalhes.

Não é obrigatório que você compacte um arquivo ObjectGrid.xml e um arquivo objectGridDeployment.xml no arquivo WAR ou EAR do seu aplicativo da Web com o eXtreme Scale Versão 7.1. Os arquivos ObjectGrid.xml e objectGridDeployment.xml padrão são compactados no JAR do produto. Mapas dinâmicos são criados para vários contextos de aplicativos da Web por padrão. Mapas estáticos do eXtreme Scale continuam a ser suportados.

Esta abordagem para iniciar contêineres do ObjectGrid integrados é aplicada a qualquer tipo de servidor de aplicativos. As abordagens envolvendo um componente do WebSphere Application Server ou GBean do WebSphere Application Server Community Edition são reprovadas.

Gerenciador de Replicação de Sessão Baseada em Listener

O gerenciador de replicação de sessão do eXtreme Scale que é fornecido com o WebSphere eXtreme Scale pode trabalhar com o gerenciador de sessão padrão no servidor de aplicativos para replicar dados da sessão de um processo para outro para dar suporte à alta disponibilidade dos dados da sessão do usuário.

O gerenciador de sessões foi projetado para poder ser executado em qualquer contêiner Java™ Platform, Enterprise Edition Versão 1.4. O gerenciador de sessões não depende de nenhuma API do WebSphere, portanto, ele é capaz de suportar várias versões do WebSphere Application Server, bem como ambientes do servidor de aplicativos do fornecedor.

O gerenciador de sessões HTTP fornece recursos de replicação de sessão para um aplicativo associado. O gerenciador de replicação de sessão trabalha com o gerenciador de sessão de contêiner de Web para criar sessões HTTP e gerenciar os ciclos de vida de sessões HTTP que estão associadas com o aplicativo. Essas atividades de gerenciamento de ciclo de vida incluem: a invalidação de sessões baseada em um tempo limite ou um servlet explícito ou chamada de 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 diferentes versões do WebSphere Application Server, como em um cenário de migração clássico.

- 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 é hospedado 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 qualidade de serviço (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.
- Em datacenters múltiplos ou zonas múltiplas.

Detalhes do Gerenciador de Sessão

O gerenciador de replicação de sessão utiliza um listener de sessão padrão para atender às mudanças de dados da sessão, além de fazer a persistência dos dados da sessão em uma instância do ObjectGrid local ou remotamente. Os dados da sessão são recarregados no caminho do pedido por meio do servlet padrão a partir da instância de ObjectGrid local ou remotamente. É possível incluir um listener de sessão e um filtro de servlet em cada módulo da Web em seu aplicativo com o conjunto de ferramentas que é fornecido com o WebSphere eXtreme Scale. Você também pode incluir manualmente esses listeners e filtros no descritor de implementação da Web do seu aplicativo.

O gerenciador de replicação de sessão trabalha com o gerenciador de sessão de base de cada fornecedor para replicar dados da sessão de aplicativo. Observe as seguintes considerações.

- Escolha contêineres ObjectGrid integrados ou contêineres ObjectGrid remotos, dependendo dos requisitos de desempenho e do tamanho dos dados. O cenário integrado fornece a configuração mais fácil e um melhor desempenho.
- Escolha o número de contêineres ObjectGrid remotos por contêiner de Web de acordo com os tamanhos dos dados dos usuários.
- Escolha armazenar todos os dados da sessão juntos ou armazenar cada atributo separadamente, dependendo do número e do tamanho dos dados do usuário dos atributos e das frequências de mudança.
- Escolha o intervalo de replicação. Um intervalo de replicação menos agressivo proporciona um melhor desempenho.
- Escolha o tamanho da tabela da sessão para equilibrar o tamanho e o desempenho da memória local. O produto transfere dados do usuário da sessão quando o tamanho máximo do cache da sessão local é atingido.
- O produto suporta sessões HTTP dentro do contexto de aplicativo, de acordo com a especificação do Servlet, para evitar problemas de conflitos de nomenclatura de atributos de segurança e de uso. É possível compartilhar sessões por meio do contexto de aplicativo pela classe singleton.

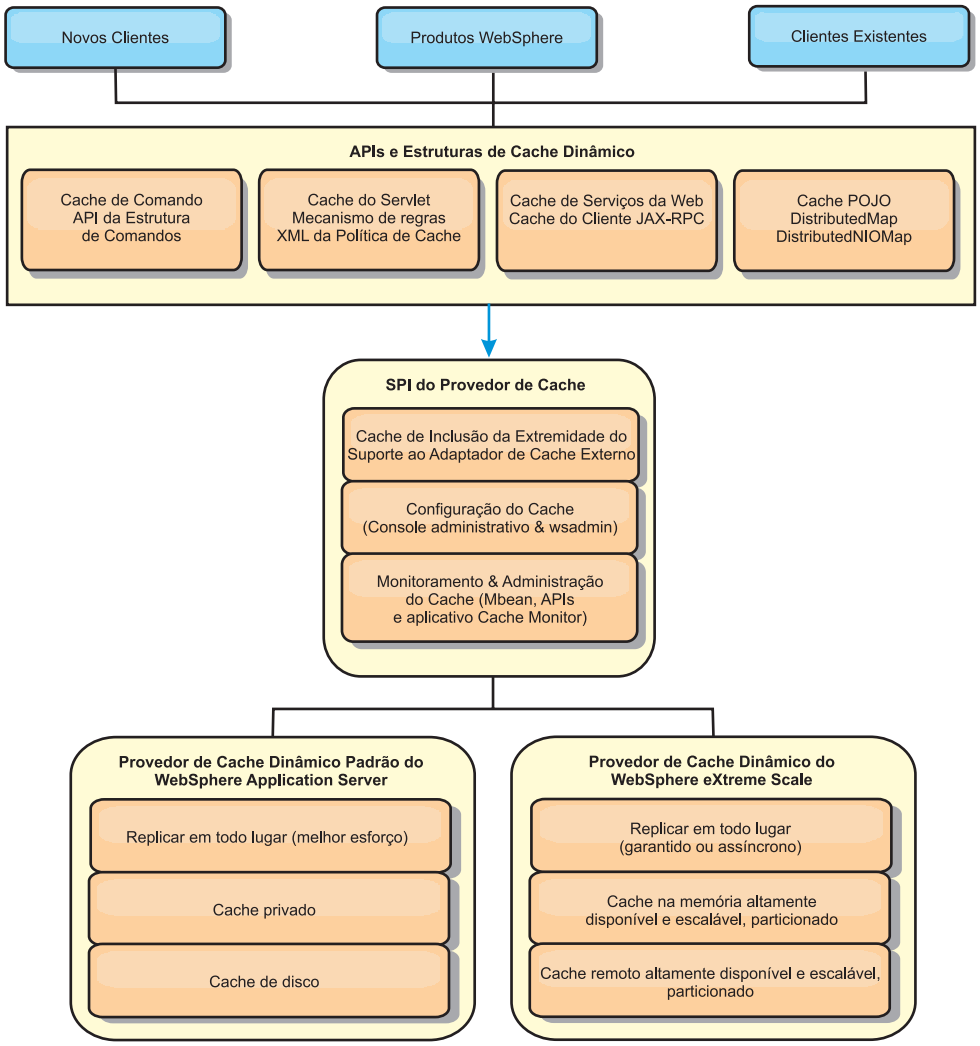
- O gerenciador de replicação de sessão replica dados da sessão para alta disponibilidade, atendendo as mudanças de dados da sessão e recarregando os dados da sessão armazenados on demand. Isso é feito reutilizando o gerenciador de sessões de base do contêiner de Web de cada fornecedor. A sessão é vinculada por meio de vários IDs de sessão nativos. A falha de dados da sessão ocorre de um contêiner de Web para outro recarregando dados da sessão a partir da instância ObjectGrid. O ID de sessão e o tempo de criação podem diferir antes e depois do failover.

Provedor de Cache Dinâmico

A API de Cache Dinâmico está disponível para aplicativos Java EE implementados no WebSphere Application Server. O provedor de cache dinâmico pode ser potencializado para dados de negócios em cache, HTML gerado ou para sincronizar os dados em cache na célula utilizando 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.



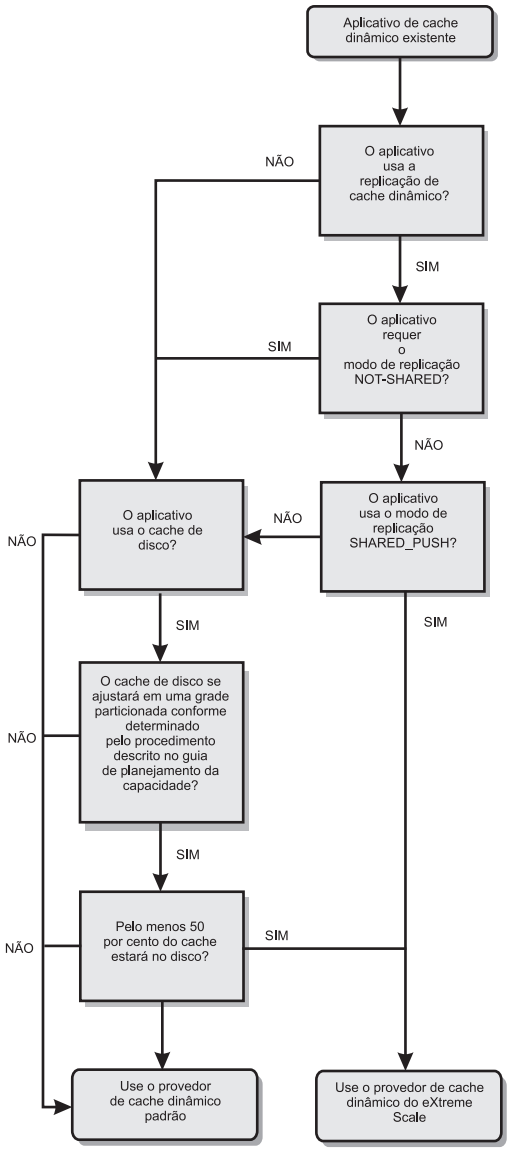
É possível instalar e configurar o provedor de cache dinâmico como descrito em Configurando o Provedor de Cache Dinâmico para o WebSphere eXtreme Scale.

Decidindo como Potencializar o WebSphere eXtreme Scale

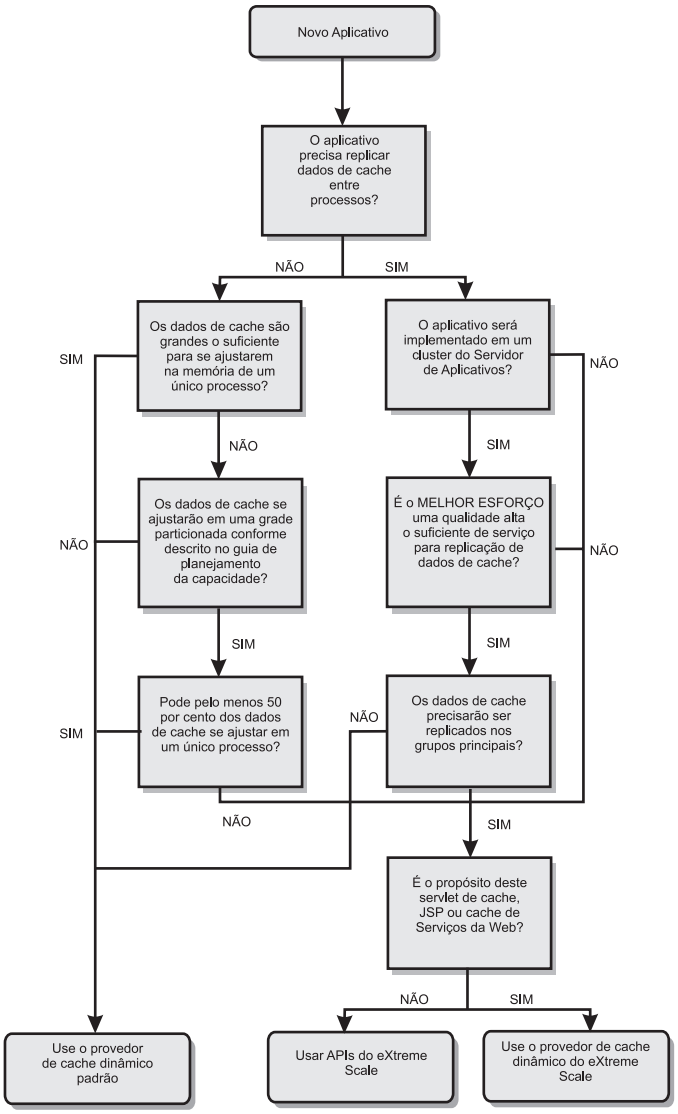
Os recursos disponíveis noWebSphere eXtreme Scale aumentam significantemente os recursos distribuídos da API do cache dinâmico além do que é oferecido pelo mecanismo de cache dinâmico e serviço de replicação de dados. Com o eXtreme Scale, é possível criar caches que são verdadeiramente distribuídos entre 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 um Provedor de Cache para Novos Aplicativos



Comparação de Recursos

Tabela 7. 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 7. 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 8. 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 9. 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 funcionam os cache distribuídos do eXtreme Scale, consulte as informações de configuração de implementação no *Guia de Administração*.

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.

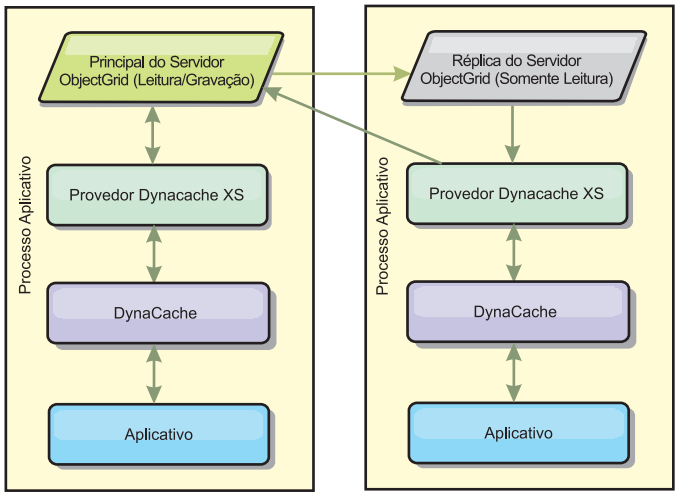
Tipos de topologia

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

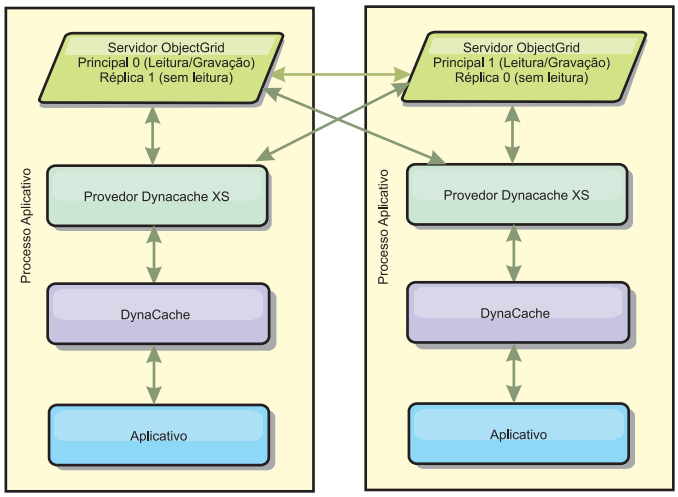
A topologia integrada é similar ao cache dinâmico padrão e ao provedor DRS. Instâncias de cache distribuído criados com a topologia integrada mantêm uma cópia integral do cache dentro de cada processo doeXtreme Scale que acessa o serviço de cache dinâmico, permitindo que todas as operações de leitura 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 estará visível, mesmo para o servidor que a gerou, até ser propagada por meio dos serviços de replicação assíncrona 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

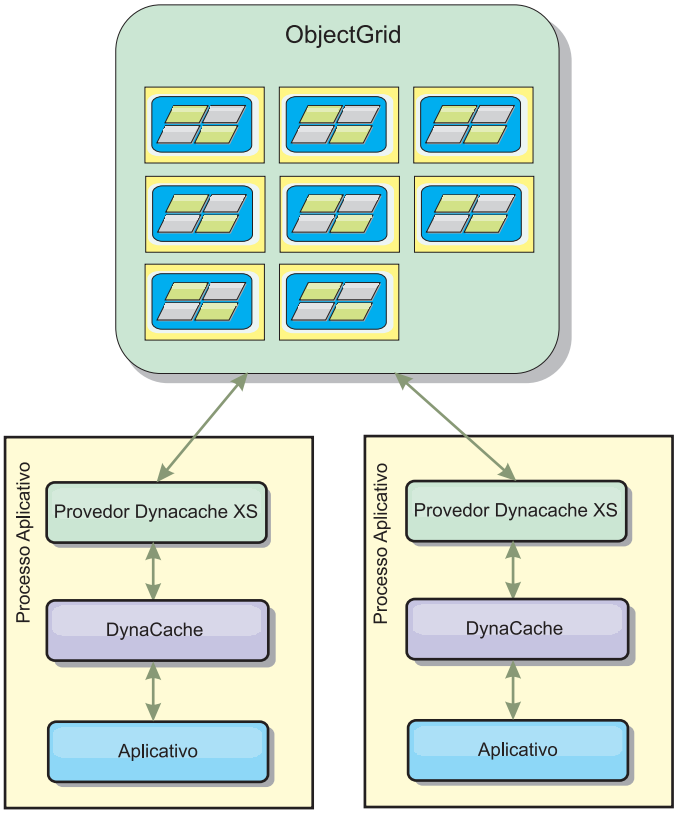
Para cargas de trabalho nas quais as gravações em cache ocorrem tão frequentemente quanto ou mais frequentemente que as leituras, as topologias remotas ou particionadas integradas são recomendadas. A topologia particionada integrada mantém todos os dados do cache dentro dos processos do WebSphere Application 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 de cache dinâmico do WebSphere eXtreme Scale também pode ser usado para retornar as instâncias do cache dinâmico que têm a **replicação desativada**. Como o provedor de cache dinâmico padrão, esses caches podem armazenar dados não serializáveis. Eles também podem oferecer melhor desempenho que o provedor de cache dinâmico padrão em servidores corporativos de multiprocessador grande porque o caminho do código do eXtreme Scale é projetado para maximizar a simultaneidade do cache em memória.

Mecanismo de Cache Dinâmico e Diferenças Funcionais do eXtreme Scale

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

retornados pelo WebSphere eXtreme Scale não suportam a transferência de disco ou estatísticas e operações relacionadas ao tamanho do cache em 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 de cache dinâmico são relatadas por meio do aplicativo CacheMonitor ou do MBean de cache dinâmico. Quando o provedor de cache dinâmico do eXtreme Scale for usado, as estatísticas ainda serão reportadas por meio 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 é suportada apenas por caches que usam as topologias remota, integrada ou integrada_particionada. Ela relata o número de megabytes do espaço de heap Java consumido pelo cache, na grade inteira. Esta estatística relata o uso de heap apenas para as partições primárias; você deve levar as réplicas em conta. Como a configuração padrão para as topologias remota e integrada_particionada é uma réplica assíncrona, dobre este número para obter o consumo real de memória do cache.
- **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. Portanto, se um cache tiver 11 partições e cada servidor em uma grade do WebSphere Application Server Network Deployment de 11 servidores hospedar uma das partições, cada servidor receberá os eventos do cache dinâmico para as entradas de cache que hospedar. 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

É possível 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 WebSphere eXtreme Scalesegurança 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

Planejamento de Capacidade e Alta Disponibilidade (Armazenamento em Cache Dinâmico)

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 é possível 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 o ajuste de desempenho com o provedor de cache dinâmico, consulte Ajustando o Provedor de Cache Dinâmico.

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.

Capítulo 4. Visão Geral de Conceitos de Escalabilidade

A escalabilidade permite que os dados em uma implementação do WebSphere eXtreme Scale sejam distribuídos em um conjunto de servidores (contêineres), com base em sua opção de configuração.

Escalabilidade

O WebSphere eXtreme Scale é escalável por meio do uso de dados particionados, e pode ser escalado para milhares de contêineres se necessário, pois cada contêiner é independente um do outro.

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.

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.

Grades, Partições e Shards

Uma grade distribuída do eXtreme Scale é dividida em partições. Uma partição retém um subconjunto exclusivo de dados. Uma partição é constituída de um ou mais shards: um shard primário e shards réplicas. Não é necessário ter shards réplicas em uma partição, mas os shards réplicas fornecem alta disponibilidade. Independente se sua implementação for um espaço de processamento de grade de dados ou de banco de dados de memória independente, o acesso aos dados no eXtreme Scale depende fortemente dos conceitos dos shards.

Os dados para uma partição são armazenados no tempo de execução em um conjunto de shards. Este conjunto de shards inclui um shard principal e, possivelmente, um ou mais shards de réplica. Um shard é a menor unidade que o eXtreme Scale pode incluir ou remover de uma Java Virtual Machine.

Existem duas estratégias de posicionamento: FIXED_PARTITIONS (padrão) e PER_CONTAINER. A seguinte abordagem foca o uso da estratégia FIXED_PARTITIONS.

Número de Shards

Se o seu ambiente incluía dez partições que continham um milhão de objetos sem réplica, então, existiriam 10 shards com cada um armazenando 100.000 objetos. Se você incluir uma réplica neste cenário, então, existe um shard extra em cada partição. Neste caso, existem 20 shards - dez shard primários e dez shards de réplica. Novamente, cada um destes shards armazenaria 100.000 objetos. Cada partição consiste de um shard primário e uma ou mais (N) shards de réplica. Determinar a contagem de shards ideal é crítica. Se você configurar um pequeno número de shards, os dados não são distribuídos igualmente entre os shards, resultando em erros de falta de memória e problemas de sobrecarga do processador. É necessário ter pelo menos dez shards para cada JVM à medida que escala. Quando você está implementando a grade inicialmente, poderia potencialmente utilizar um grande número de partições.

Número de Shards por JVM

Cenário: pequeno número de shards para cada JVM

Os dados são incluídos e removidos de uma JVM utilizando unidades de shard. Os shards nunca são divididos em partes. Se existirem 10 GB de dados e 20 shards para conterem estes dados, cada shard conterá 500 MB de dados em média. Se nove Java Virtual Machines hospedam a grade, então, em média, cada JVM possui dois shards. Como 20 não é igualmente divisível por 9, algumas Java Virtual Machines possuem três shards, na seguinte distribuição:

- 7 Java Virtual Machines com 2 shards
- 2 Java Virtual Machines com 3 shards

Como cada shard contém 500 MB de dados, a distribuição de dados é desigual. As sete Java Virtual Machines com dois shards contêm cada uma 1 GB de dados. As duas Java Virtual Machines com três shards possuem 50% mais dados ou 1.5 GB, o que é uma carga de memória muito maior. Como estas duas Java Virtual Machines estão hospedando três shards, elas também recebem 50% mais pedidos para seus dados. Como resultado, um pequeno número de shards para cada JVM causa desequilíbrio. Para aumentar o desempenho, aumente o número de shards para cada JVM.

Cenário: número aumentado de shards por JVM

Neste cenário, considere um número muito maior de shards. Neste cenário, há 101 shards com 9 Java Virtual Machines hospedando 10 GB de dados. Neste caso, cada shard contém 99 MB de dados. A Java Virtual Machines possui a seguinte distribuição de shards:

- 7 Java Virtual Machines com 11 shards
- 2 Java Virtual Machines com 12 shards

As duas Java Virtual Machines com 12 shards agora possuem apenas mais 99 MB de dados do que os outros shards, o que é uma diferença de 9%. Este cenário é muito mais igualmente distribuído do que a diferença de 50% no cenários com um pequeno número de shards. A partir de uma perspectiva de uso do processador, existe apenas 9% mais de trabalho para as duas Java Virtual Machines com os 12 shards comparado às sete Java Virtual Machines que possuem 11 shards. Ao aumentar o número de shards em cada JVM, o uso dos dados e do processador é distribuído de uma maneira proporcional e equilibrada.

Quando você está criando seu sistema, utilize 10 shards para cada JVM em seu cenário com dimensionamento máximo ou quando o sistema está executando seu número máximo de Java Virtual Machines em seu horizonte de planejamento.

Fatores Adicionais de Posicionamento

O número de partições, a estratégia de posicionamento e o número e o tipo de réplicas são configurados na política de implementação. O número de shards posicionados dependerá da política de implementação definida. numInitialContainers, minSyncReplicas, developmentMode, maxSyncReplicas e maxAsyncReplicas afetarão onde e quando as partições e as réplicas podem ser posicionadas. Se a inicialização do servidor inicial não permitir que maxSyncReplicas e maxAsyncReplicas sejam posicionados, réplicas adicionais poderão ser posicionadas se você iniciar servidores adicionais posteriormente. Ao planejar o número de shards por JVM, o número máximo de shards, incluindo réplicas, dependerá de ter JVMs suficientes iniciadas para suportar o número máximo de réplicas solicitadas. 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. Quando developmentMode é falso, o principal e as réplicas não serão posicionados na mesma máquina.

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. Com o WebSphere eXtreme Scale, o particionamento aumenta a escalabilidade e a disponibilidade.

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 por meio de grandes conjuntos de dados não expandem e não podem ser particionados efetivamente, assim o eXtreme Scale não suporta atualmente consolidação de duas fase por meio 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.

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 elasticamente 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.

ObjectMap e Particionamento

Com a estratégia de posicionamento FIXED_PARTITION padrão, os mapas são divididos em partições e chaves hash para diferentes partições. O cliente não precisa saber à qual partição as chaves pertencem. Se um mapSet tiver vários mapas, os mapas deverão ser confirmados em transações separadas.

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 por meio 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. O gráfico de objetos define os relacionamentos entre uma ou mais entidades. Cada entidade vinculada deve resolver para a mesma partição. Todas as entidades filha são assumidas como estando na mesma partição que a raiz. As entidades filhas no gráfico de objetos são 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 por meio de acesso direto e local, e não por meio do mecanismo de cliente e servidor.

Colocação e Partições

Você tem duas estratégias de posicionamento disponíveis para WebSphere eXtreme Scale: partição fixa e por contêiner. A escolha da estratégia de posicionamento afeta a forma como sua configuração de implementação posiciona partições na grade remota.

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 reconectar à mesma sessão usada para localizar uma chave mais uma vez, 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.

O WebSphere eXtreme Scale permite posicionamento por contêiner como uma alternativa para aquilo que poderia ser chamado de estratégia de posicionamento "típica", uma abordagem de partição fixa com a chave de um Mapa com hash para uma dessas partições. Em um caso por contêiner (que você configura com PER_CONTAINER), sua implementação coloca as partições no conjunto de servidores de contêiner on-line e efetua seu scale out ou scale in automaticamente, conforme os contêineres são incluídos ou removidos da grade do servidor. Uma grade com a abordagem de partição fixa funciona bem para grades baseadas em chave, na qual o aplicativo utiliza um objeto chave para localizar dados na grade. A seguir está uma discussão sobre a alternativa.

Exemplo de Grade por Contêiner

As grades PER_CONTAINER são diferentes. Você especifica que a grade utiliza PER_CONTAINER por meio do atributo placementPolicy em seu arquivo XML de implementação. Em vez de configurar o total de partições que quer na grade, você especifica quantas partições quer por contêiner que você iniciar.

Por exemplo, se configurar o número de partições por contêiner para 5, quando você iniciar um contêiner, o eXtreme Scale criará 5 novas partições anônimas primárias nesse contêiner e criará todas as réplicas necessárias nos outros contêineres já implementados.

A seguir está uma possível sequência em um ambiente por contêiner conforme a grade aumenta.

1. Iniciar contêiner C0 hospedando 5 primárias (P0 - P4).
   - C0 hospeda: P0, P1, P2, P3, P4.
2. Iniciar contêiner C1 hospedando mais 5 primárias (P5 - P9). As réplicas são balanceadas nos contêineres.
   - C0 hospeda: P0, P1, P2, P3, P4, R5, R6, R7, R8, R9.
   - C1 hospeda: P5, P6, P7, P8, P9, R0, R1, R2, R3, R4.
3. Iniciar contêiner C2 hospedando mais 5 primárias (P10 - P14). As réplicas são mais balanceadas.
   - C0 hospeda: P0, P1, P2, P3, P4, R7, R8, R9, R10, R11, R12.
   - C1 hospeda: P5, P6, P7, P8, P9, R2, R3, R4, R13, R14.
   - C2 hospeda: P10, P11, P12, P13, P14, R5, R6, R0, R1.

O padrão continua conforme mais contêineres são iniciados, criando 5 novas partições primárias toda vez e reequilibrando as réplicas nos contêineres disponíveis na grade.

Nota: WebSphere eXtreme Scale não move primárias quando utiliza a estratégia PER_CONTAINER, apenas réplicas.

Lembre-se de que os números de partições são arbitrários e não têm nada a ver com chaves, portanto, não é possível utilizar roteamento baseado em chave. Se um contêiner parar, os IDs de partição criados para esse contêiner não serão mais utilizados, portanto, haverá um intervalo nos IDs de partição. No exemplo, não haveria mais as partições P5 - P9 se o contêiner C2 falhasse, deixando apenas P0 - P4 e P10 - P14, por isso o hash baseado em chave é impossível.

O uso de números como 5 ou, mais provavelmente, 10 para o número de partições por contêiner funciona melhor se você considerar as consequências de uma falha do contêiner. Para propagar o carregamento de shards de hosting de maneira uniforme na grade, você precisa de mais do que apenas uma partição para cada contêiner. Se você tivesse uma única partição por contêiner, quando um contêiner falhasse, apenas um contêiner (aquele hospedando o shard de réplica correspondente) deveria suportar o carregamento total da primária perdida. Nesse caso, o carregamento é duplicado imediatamente para o contêiner. Entretanto, se você tiver 5 partições por contêiner, então 5 contêineres selecionarão o carregamento do contêiner perdido, diminuindo em 80 por cento o impacto em cada um. O uso de várias partições por contêiner geralmente diminui o possível impacto em cada contêiner substancialmente. Mais diretamente, considere o caso em que um contêiner para inesperadamente - o carregamento da replicação desse contêiner é espalhado sobre os 5 contêineres em vez de apenas um.

Utilizando a Política por Contêiner

Vários cenários tornam a estratégia por contêiner uma configuração ideal, como com a replicação da sessão HTTP ou o estado da sessão de aplicativo. Nesse caso, um roteador HTTP designa uma sessão a um contêiner do servlet. O contêiner do servlet precisa criar uma sessão HTTP e escolher uma das 5 partições locais primárias para a sessão. O "ID" da partição escolhida é armazenado em um cookie. O contêiner do servlet agora tem acesso local ao estado da sessão que significa acesso de latência zero aos dados para esse pedido, contanto que você mantenha a afinidade da sessão. E o eXtreme Scale replica quaisquer mudanças para a partição.

Na prática, lembre-se da repercussão do caso no qual você tem várias partições por contêiner (por exemplo, 5 novamente). Certamente, com cada novo contêiner iniciado, você tem mais 5 partições primárias e mais 5 réplicas. Com o tempo, mais partições devem ser criadas, e elas não devem ser movidas ou destruídas. Mas não é assim que os contêineres se comportam de fato. Quando um contêiner é iniciado, ele hospeda 5 shards primários, que podem ser chamados de primários "iniciais", existentes nos respectivos contêineres que os criaram. Se o contêiner falhar, as réplicas se tornarão primárias e o eXtreme Scale criará mais 5 réplicas para manter a alta disponibilidade (a menos que você desative o reparo automático). Os novos primários estão em um contêiner diferente daquele que os criou, que podem ser chamados de primários "estrangeiros". O aplicativo nunca deve colocar novos estados ou sessões em um primário estrangeiro. Eventualmente, o primário estrangeiro não tem entradas, e o eXtreme Scale o exclui automaticamente, junto com suas réplicas associadas. O propósito dos primários estrangeiros é permitir que sessões existentes continuem disponíveis (mas não as novas sessões).

Um cliente pode interagir com uma grade que não conte com chaves. O cliente apenas inicia uma transação e armazena na grade dados independentes de quaisquer chaves. Ele solicita de Session um objeto SessionHandle, um identificador serializável que permite que o cliente interaja com a mesma partição quando necessário. Para obter mais informações, consulte o tópico sobre o uso de um SessionHandle para roteamento no *Guia de Programação*. O WebSphere eXtreme Scale escolhe uma partição para o cliente da lista de partições primárias iniciais.

Ele não retorna uma partição primária estrangeira. O SessionHandle pode ser serializado em um cookie HTTP, por exemplo, e depois converter o cookie novamente em um SessionHandle. As APIs do WebSphere eXtreme Scale podem obter um Session ligado à mesma partição novamente, utilizando SessionHandle.

Nota: Não é possível utilizar agentes para interagir com uma grade PER_CONTAINER.

Vantagens

A descrição anterior é diferente de uma grade hash ou FIXED_PARTITION normal, pois o cliente por contêiner armazena dados em um local na grade, obtém um identificador para eles e utiliza o identificador para acessá-los novamente. Não existe uma chave fornecida por aplicativo como existe no caso de uma partição fixa.

Sua implementação não cria uma nova partição para cada Session. Portanto, em uma implementação por contêiner, as chaves utilizadas para armazenar dados na partição devem ser exclusivas dentro dessa partição. Por exemplo, é possível fazer o cliente gerar um SessionID exclusivo e depois utilizá-lo como chave para localizar informações em Mapas nessa partição. Várias sessões do cliente interagem com a mesma partição, portanto o aplicativo precisa utilizar chaves exclusivas para armazenar dados da sessão em cada partição fornecida.

Os exemplos anteriores utilizaram 5 partições, mas o parâmetro numberOfPartitions no arquivo XML de objectgrid pode ser utilizado para especificar as partições conforme necessário. Em vez de por grade, a configuração é por contêiner. (O número de réplicas é especificado da mesma maneira que com a política de partição fixa.)

A política por contêiner também pode ser utilizada com várias zonas. Se possível, o eXtreme Scale retorna um SessionHandle para uma partição cuja primária está localizada na mesma zona que esse cliente. O cliente pode especificar a zona como um parâmetro para o contêiner ou utilizando uma API. O ID da zona do cliente pode ser configurado por meio de `serverproperties` ou `clientproperties`.

A estratégia PER_CONTAINER para uma grade adequa aplicativos armazenando o estado do tipo de conversação, e não dados orientados por banco de dados. A chave para acessar os dados seria um ID de conversa e não tem relação com um registro do banco de dados específico. Ela fornece melhor desempenho (porque as partições primárias podem ser colocadas junto com os servlets, por exemplo) e configuração mais fácil (sem precisar calcular partições e contêineres).

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 construir 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 mudança 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.

Ampliação de Unidades ou Pods

Este tópico discute escalabilidade, mas não da maneira comum, como de que forma efetuar adição da grade para um número X de JVMs. Em vez disso, a perspectiva aqui discute escalabilidade em termos de operações, planejamento e gerenciamento de risco. Embora normalmente esses fatores sejam mais importantes que as considerações de escalabilidade tradicionais do produto, infelizmente na maioria das vezes eles são ignorados. A construção de sistemas altamente disponíveis requer tipos de considerações de escalabilidade para implementar o eXtreme Scale em um processo de implementação confiável.

Implementando uma Grade Única Grande

Testes verificaram que o eXtreme Scale pode adicionar para mais de 1000 JVMs. Tais testes encorajam a construção de aplicativos para implementar grades únicas em grandes números de caixas. Embora seja possível, isso não é recomendado por vários motivos mostrados a seguir.

1. **Questões de orçamento:** Seu ambiente não pode testar realisticamente uma grade de 1000 servidores. Entretanto, ele pode testar uma grade muito menor, considerando os motivos de orçamento, portanto você não precisa comprar duas vezes o hardware, principalmente para um número tão grande de servidores.
2. **Versões de aplicativo diferentes:** Exibir um grande número de caixas para cada encadeamento de teste não é considerado prático. O risco é que você não esteja testando os mesmos fatores que estaria em um ambiente de produção.
3. **Perda de dados:** A execução de um banco de dados em um único disco rígido não é confiável. Qualquer problema com o disco rígido faz você perder dados. A execução de um aplicativo em crescimento em uma única grade é semelhante. Você provavelmente terá erros em seu ambiente e em seus aplicativos. Portanto, colocar todos os dados em um único sistema grande muitas vezes levará à perda de grandes quantidades de dados.

Dividindo a Grade

A divisão da grade de aplicativo em pods (unidades) é mais confiável. Um pod é um grupo de servidores executando uma pilha de aplicativos homogêneos. Pods podem ser de qualquer tamanho, mas o ideal é que eles consistam em cerca de 20 caixas. Em vez de ter 500 caixas em uma única grade, é possível ter 25 pods de 20 caixas. Uma única versão de uma pilha de aplicativos deve ser executada em um determinado pod, mas diferentes pods podem ter suas próprias versões de uma pilha de aplicativos.

Geralmente, uma pilha de aplicativos considera os níveis dos seguintes componentes.

- Sistema Operacional
- Hardware
- JVM
- eXtreme Scale versão
- Aplicativo

- Outros componentes necessários

Um pod é uma unidade de implementação de tamanho conveniente para testes. Em vez de ter centenas de servidores para testes, é mais prático ter 20 servidores. Nesse caso, você ainda está testando a mesma configuração que teria em uma produção. A produção utiliza grades com um tamanho máximo de 20 servidores, constituindo um pod. É possível fazer testes de tensão de um único pod e determinar sua capacidade, número de usuários, quantidade de dados e rendimento da transação. Isso facilita o planejamento e segue o padrão de escala previsível a um custo previsível.

Configurando um Ambiente Baseado em Pod

Em casos diferentes, o pod não precisa ter necessariamente 20 servidores. O propósito do tamanho do pod são os testes práticos. Um tamanho de pod deve ser pequeno o bastante para que se encontrar problemas na produção, a fração de transações afetadas seja tolerável.

O ideal é que qualquer erro tenha impacto em apenas um pod. No exemplo anterior, um erro só teria impacto em quatro por cento das transações do aplicativo, e não em 100 por cento. Além disso, os upgrades são mais fáceis porque podem ser lançados por um pod por vez. Isso simplifica o cenário, de modo que se um upgrade para um pod criar problemas, o usuário possa mudar esse pode de volta para o nível anterior. Os upgrades incluem quaisquer mudanças no aplicativo, na pilha de aplicativos ou nas atualizações do sistema. Enquanto possível, os upgrades só devem alterar um único elemento da pilha por vez para tornar o diagnóstico do problema mais preciso.

Para implementar um ambiente com pods, você precisa de uma camada de roteamento acima dos pods compatíveis com versões anteriores e posteriores, caso os pods sofram upgrades de software. Além disso, você deve criar um diretório que inclua informações sobre qual pod contém dados. (É possível utilizar outra grade do eXtreme Scale para isso com um banco de dados por trás, de preferência utilizando o cenário write-behind.) Isso gera uma solução de duas camadas. A camada 1 é o diretório e é utilizado para localizar qual pod trata uma transação específica. A camada 2 é composta pelos próprios pods. Quando a camada 1 identifica um pod, a configuração roteia cada transação para o servidor correto no pod, que geralmente é o servidor contendo a partição para os dados utilizados pela transação. Opcionalmente, você também pode utilizar um cache local na camada 1 para diminuir o impacto associado à consulta do pod correto.

O uso do pods é um pouco mais complexo do que ter uma única grade, mas as melhorias operacionais, de teste e de confiabilidade o tornam parte crucial dos testes de escalabilidade.

Capítulo 5. Visão Geral de Disponibilidade

Alta Disponibilidade

Com alta disponibilidade, o WebSphere eXtreme Scale fornece redundância de dados confiáveis 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 "Arquitetura de Armazenamento em Cache: Mapas, Contêineres, Clientes e Catálogos" na página 9 para obter mais informações.

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 por meio 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 113 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 principal diferença é que o domínio do serviço de catálogo usa um processo de eleição de peer para definir o shard principal em vez do algoritmo do serviço de catálogo usado para os contêineres.

Observe que o serviço de posicionamento e o serviço de agrupamento principal são serviços Um de N. Um serviço Um de N é executado em um membro do grupo de alta disponibilidade. O serviço de local e a administração são executados em todos os membros do grupo de alta disponibilidade. 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 91 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.

Consulte os detalhes sobre como escrever um utilitário de carga com um controlador de pré-carga de réplica no *Guia de Programação* para obter um exemplo sobre como implementar um utilitário de carga para um eXtreme Scale replicado.

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, após a transação incluir blocos de mais de 100 entradas, o benefício do desempenho diminui 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 10. 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 11. 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

Tabela 11. Sequência de Commit no Primário (continuação)

Etapa com o Utilitário de Carga	Etapa sem o Utilitário de Carga
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 12. 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 mudança. 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. É possível 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 por meio 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 solicitar uma cópia local somente leitura de um mapa do lado do servidor usando 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.

Tipos de Detecção de Failover

WebSphere eXtreme Scale pode detectar falhas de forma confiável.

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 a Java Virtual Machine saindo rapidamente. Tal detecção permite a recuperação desses tipos de falhas 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. Essas falhas são descobertas por meio de pulsação.

Pulsações são enviadas periodicamente entre pares de processos: Quando um número fixo de pulsações está ausente, 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 pode falhar, embora com menos

frequência, como a placa da interface de rede (NIC), o que faria o sistema operacional ser desconectado da rede. 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 imediatamente por meio das conexões que estavam sendo utilizadas por esse processo para fechar cada conexão.

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 uma máquina host perder a energia, ela ficará indisponível imediatamente.

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 por meio 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.

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

Observe que o serviço de posicionamento e o serviço de agrupamento principal são serviços Um de N. Um serviço Um de N é executado em um membro do grupo de alta disponibilidade. O serviço de local e a administração são executados em todos os membros do grupo de alta disponibilidade. 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.

Tabela 13. Resumo de Descoberta de Falha e Recuperação

Tipo de perda	**Mecanismo de descoberta (detecção)**	**Método de recuperação**
Perda de processo	E/S	Reiniciar
Perda de servidor	Pulsação	Reiniciar
Interrupção da rede	Pulsação	Reestabelecer rede e conexão
Interrupção do lado do servidor	Pulsação	Parar e reiniciar servidor
Servidor ocupado	Pulsação	Aguardar até servidor estar disponível

Serviço de Catálogo de Alta Disponibilidade

Um domínio do serviço de catálogo é a grade de servidores de catálogos que você está usando, que retém informações de topologia para todos os contêineres no 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 em memória, você deve armazenar em cluster o serviço de catálogo em um domínio do serviço de catálogo para alta disponibilidade.

Componentes do Domínio 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 catálogos, a tabela de roteamento para o domínio do serviço de catálogo é propagada para os clientes por meio do contexto de serviço de 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 entram em contato com um dos servidores de catálogos, a tabela de roteamento para o domínio do serviço de catálogo também é propagada para o servidor e contêiner do eXtreme Scale por meio do contexto de serviço de 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. O domínio do serviço de catálogo é para alta disponibilidade dos dados do sistema. A grade do contêiner é para a alta disponibilidade de dados, escalabilidade e gerenciamento de carga de trabalho. Portanto, existem duas tabelas de roteamento diferentes: a tabela de roteamento para o domínio do serviço de catálogo 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 do Domínio do Serviço 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 o processo. 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 contata pela primeira vez o serviço de catálogo, ele aguarda para ser designado a um grupo novo ou existente. Uma implementação do eXtreme Scale consiste em vários desses grupos, e esse 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 por meio 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. O serviço de posicionamento é executado como um serviço eleito Um de N na grade de dados, portanto, exatamente uma instância do serviço está 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

Depois que um domínio do serviço de catálogo é iniciado, os membros da grade são ligados juntos. Planeje com cuidado sua topologia de domínio do serviço de

catálogo, pois não é possível modificar a configuração do domínio do serviço de catálogo no tempo de execução. Disperse a grade o mais diversamente possível para evitar erros.

Iniciando um domínio do serviço de catálogo

Para obter mais informações sobre como criar um domínio do serviço de catálogo, consulte os detalhes sobre como iniciar um domínio do serviço de catálogo no *Guia de Administração*.

Conectando contêineres do eXtreme Scale integrados no WebSphere Application Server a um domínio do serviço de catálogo independente

É possível configurar contêineres do eXtreme Scale integrados em um ambiente do WebSphere Application Server para conectar a um domínio do serviço de catálogo independente.

-  (reprovado) Em releases anteriores, você conectou os serviços de catálogo a um domínio do serviço de catálogo criando uma propriedade customizada. Esta propriedade ainda pode ser usada, mas é reprovada. Para obter mais informações sobre esta propriedade customizada, consulte as informações sobre como iniciar o processo do serviço de catálogo em um WebSphere Application Server no *Guia de Administração*..

  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.

Consulte o "Quorums de Servidores de Catálogo" para obter mais informações.

Quorums de Servidores de Catálogo

Quorum é o número mínimo de servidores de catálogo que são necessários para conduzir operações de posicionamento para a grade de dados. O número mínimo é o conjunto completo de servidores de catálogo que você substitui o valor do quorum.

Termos Importantes

A seguir, é apresentada uma lista de termos relacionados a considerações do quorum para o WebSphere eXtreme Scale.

- **Brown out**: Um brown out é a perda temporária de conectividade entre um ou mais servidores.
- **Black out**: Um black out é a perda permanente de conectividade entre um ou mais servidores.
- **Datacenter**: Um datacenter é um grupo de servidores localizado geograficamente geralmente conectado a uma rede local (LAN).
- **Zona**: Uma zona é uma opção de configuração usada para agrupar servidores que compartilham alguma característica física. A seguir estão alguns exemplos de zonas para um grupo de servidores: um datacenter, conforme descrito no marcador anterior, uma rede local, um edifício, o andar de um edifício, entre outras.
- **Pulsação**: Pulsação é um mecanismo usado para determinar se uma determinada Java virtual machine (JVM) está ou não em execução.

Topologia

Esta seção explica como o WebSphere eXtreme Scale opera por meio 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 em que qualquer elemento endereçável na rede possa conectar a qualquer outro elemento endereçável na rede não impedida. Isso significa que o WebSphere eXtreme Scale requer um espaço de nomenclatura de endereço IP simples e requer que todos os firewalls permitam que todo o tráfego flua entre os endereços IP e portas usados pelas Java virtual machines (JVM) que hospedam elementos do WebSphere eXtreme Scale.

LANs Conectadas

Cada LAN recebe um identificador de zona para os requisitos do WebSphere eXtreme Scale. O WebSphere eXtreme Scale faz pulsar agressivamente as JVMs em uma única zona. Uma pulsação ausente resultará em um evento de failover se o serviço de catálogo tiver quorum.

Domínio do serviço de catálogo e servidores de contêiner

Um domínio do serviço de catálogo é uma coleta de JVMs semelhantes. Um domínio do serviço de catálogo é uma grade composta de servidores de catálogos e é fixa no tamanho. 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 datacenters, o WebSphere eXtreme Scale requer uma JVM do serviço de catálogo por datacenter.

O domínio do serviço de catálogo usa um mecanismo de quorum completo. Por causa deste mecanismo de quorum completo, todos os membros da grade de dados deve concordar em qualquer 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 inclui apenas JVMs da mesma zona. JVMs de zonas diferentes nunca estão no mesmo grupo principal.

Um grupo principal tenta agressivamente detectar a falha das JVMs de seus membros. 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 catálogos são iniciados usando o comando startOgServer. O mecanismo de quorum é desativado por padrão. Para ativar o quorum, passe o sinalizador ativado `-quorum` no comando startOgServer ou inclua a propriedade `enableQuorum=true` no arquivo de propriedade. 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 de dados por meio de datacenters, os servidores devem usar a marcação de zona para identificar o datacenter no qual residem. Configurar a zona nos servidores da grade permite que o WebSphere eXtreme Scale monitore o funcionamento dos servidores com escopo no datacenter, minimizando o tráfego pelo 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 parados 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/stopOgServer gridA0,gridA1,gridA2 –catalogServiceEndPoints
cat0.domain.com:2809,cat1.domain.com:2809
```

Detecção de Falha

O WebSphere eXtreme Scale detecta o fim do processo por meio de eventos anormais de encerramento do soquete. O serviço de catálogo será informado imediatamente quando um processo se encerrará. Um blecaute é detectado por meio de pulsações perdidas. O WebSphere eXtreme Scale se protege contra condições de brown out por meio dos datacenters usando uma implementação de quorum.

Implementação de Pulsação

Esta seção descreve como a verificação de ativação 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 mudança de associação pode ser uma JVM com falha ou uma JVM recém incluída que se une ao grupo principal.

Se o líder do grupo principal não puder entrar em contato com nenhum membro do domínio do serviço de catálogo, ele continuará tentando novamente.

Pulsação do domínio do serviço de catálogo

O domínio do serviço de catálogo é semelhante a um grupo principal privado com uma 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 por meio 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 com falha e o contêiner for posteriormente relatado como ativo, será solicitado que a JVM de contêiner encerre os servidores de contêiner do WebSphere eXtreme Scale. Uma JVM neste estado não é visível nas consultas do comando xsadmin. As mensagens nos logs da JVM de contêiner indicam que a JVM de contêiner falhou. É necessário reiniciar manualmente estas JVMs.

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. O domínio do serviço de catálogo é um conjunto estático de JVMs. O WebSphere eXtreme Scale espera que todos os membros do serviço de catálogo estejam sempre on-line. O serviço de catálogo responde a eventos do contêiner apenas enquanto o serviço de catálogo tem quorum.

Se o serviço de catálogo perde quorum, ele espera até que o quorum seja restabelecido. Durante o período de tempo no qual o serviço de catálogo não tem quorum, ele ignora eventos dos servidores de contêiner. Os servidores de contêiner repetem os pedidos rejeitados pelo servidor de catálogo durante esse tempo uma vez que o WebSphere eXtreme Scale espera o quorum ser restabelecido.

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 é projetado para esperar a possibilidade de brown outs. 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 o período de brown out, um brown out é considerado um evento de falha única. Espera-se que a falha seja corrigida e, em seguida, a operação normal é retomada sem que nenhuma ação do WebSphere eXtreme Scale seja necessária.

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. WebSphere eXtreme Scale considera um evento de perda do quorum e uma falha do contêiner como uma falha dupla, que é um evento raro. 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 devido a uma falha da JVM do serviço de catálogo ou a um brownout da rede deve ser recuperada automaticamente depois que a JVM do serviço de catálogo for reiniciada ou que o brownout da rede for resolvido.

Os administradores são os únicos que sabem quando ocorre uma falha do datacenter. O WebSphere eXtreme Scale trata um brown out e um black out de forma semelhante. 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 perderá quorum e esperará que o brown out seja resolvido e nesse ponto o C une novamente o domínio do serviço de catálogo e o quorum é restabelecido. 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. C unirá novamente o domínio do 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, C unirá novamente o domínio do serviço de catálogo existente e o quorum será revertido para a situação 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 por meio 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 são capazes de conectar ao servidor de catálogo para realizar uma autoinicialização para a grade se o domínio do serviço de catálogo tiver 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.

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 são detectados como com falha, mas o comando é fornecido para uso na ajuda de suporte da 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
  ```

Considerações de Segurança de Transporte

Como normalmente os datacenters são implementados em diferentes localizações geográficas, os usuários podem querer ativar a segurança do transporte entre os datacenters por motivos de segurança.

Leia sobre a segurança da camada de transporte no *Guia de Administração*.

Réplicas e Shards

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. Cada partição tem uma instância que é um shard primário e um número configurável de shards de réplica. 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.

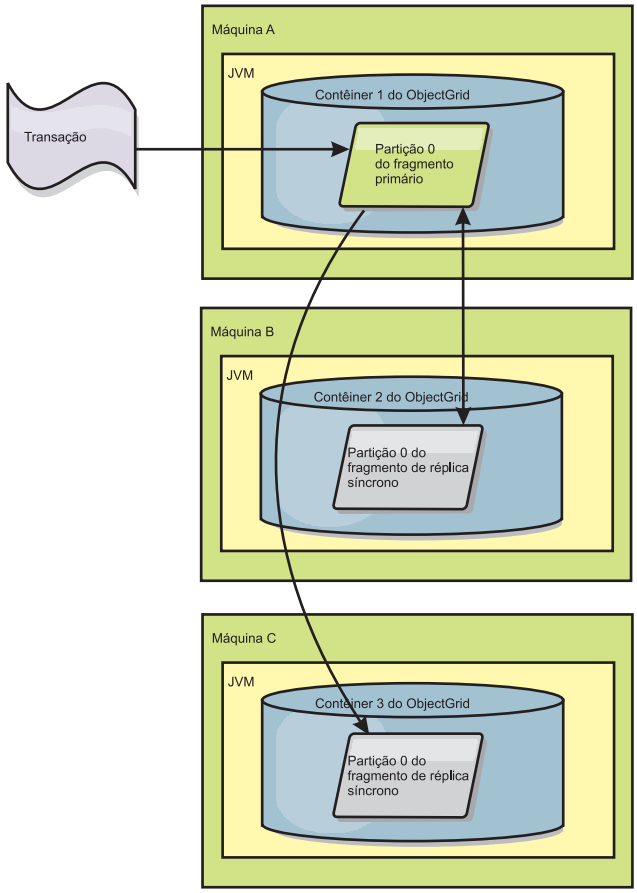


Figura 34. Caminho de Comunicação Entre um Shard Primário e Shards de Réplica

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 mais informações 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 inúmeras partições, sendo que cada uma 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 catálogo aloca os shards equilibrando-os para que eles sejam distribuídos uniformemente nos 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.

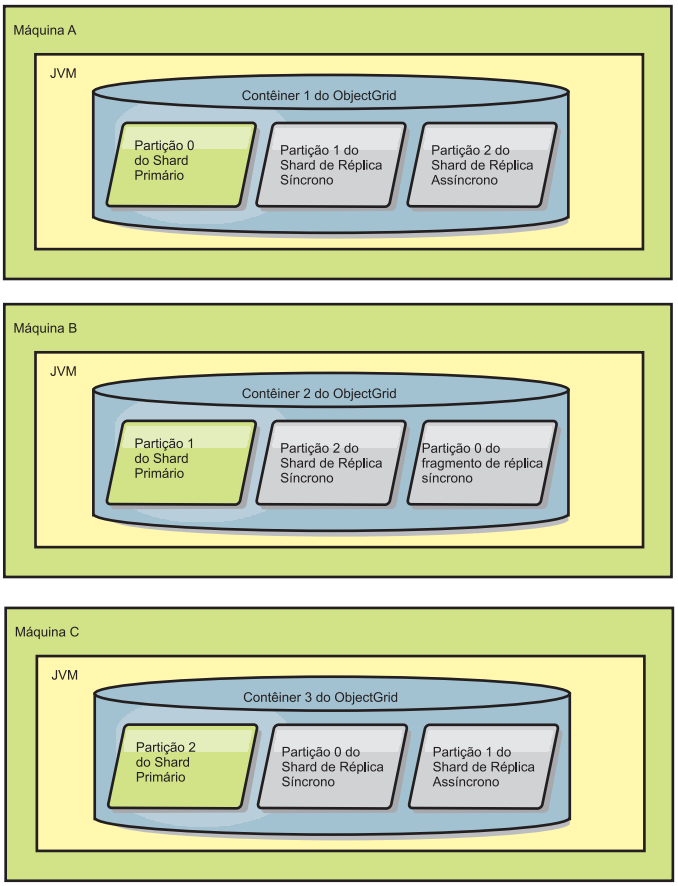


Figura 35. Posicionamento de um Conjunto de Mapas ObjectGrid com um Política de Implementação de 3 Partições com um Valor minSyncReplicas de 1, um Valor maxSyncReplicas de 1, e um Valor maxAsyncReplicas de 1

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.

Equilíbrio de Carga entre Réplicas

Geralmente, o balanceamento de carga por meio 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.

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. É possível 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 por meio 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.

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 primário 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 129 para obter mais informações 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.
```

Shard de réplica assíncrona: Um shard de réplica assíncrona sonda o primário para dados. A réplica automaticamente ajustará a sincronização da sondagem se não receber dados do primário, o que indica que ela é capturada com o primário. Ela também ajustará se receber um erro que possa indicar que o primário falhou ou se houver um problema de rede.

Quando a réplica assíncrona inicia a replicação, ela imprime a seguinte mensagem para o arquivo SystemOut.log para a réplica. Esta mensagem pode ser impressa mais de uma vez por mensagem CWOBJ1511. Ela será impressa novamente se a réplica conectar a um primário diferente ou se os mapas do modelo forem incluídos.

```
CWOBJ1543I: A réplica assíncrona objectGridName:mapsetName:partitionNumber iniciou ou
continuou a replicação a partir do primário. Replicando para mapas: [mapName]
```

Shard de réplica síncrona: Quando o shard de réplica síncrona é iniciado pela primeira vez, ele ainda não está no modo peer. 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.

A réplica síncrona copia dados do shard primário semelhante a uma réplica assíncrona sondando os dados. Quando copia os dados existentes do primário, ela alterna para o modo peer e começa a receber dados como o primário recebe os dados.

Quando um shard de réplica alcança o modo peer, ele imprime uma mensagem no arquivo SystemOut.log para a réplica. O tempo refere-se ao período de tempo que

o shard de réplica leva para obter todos os seus dados iniciais do shard primário. O tempo pode ser exibido como zero ou mínimo, se a parte primária não tiver nenhum dado existente para replicar. Esta mensagem pode ser impressa mais de uma vez por mensagem CWOBJ1511. Ela será impressa novamente se a réplica conectar a um primário diferente ou se os mapas do modelo forem incluídos.

```
CWOBJ1526I: Réplica objectGridName:mapsetName:partitionNumber:mapName entrando no modo peer
após X segundos.
```

Quando o shard de réplica síncrona está no modo peer, o shard primário deve replicar transações para todas as réplicas síncronas do modo peer. Os dados da parte da réplica síncrona permanecem no mesmo nível que os dados da parte primária. Se um número mínimo de réplicas síncronas ou minSync for configurado na política de implementação, esse número de réplicas síncronas deve ser votado para confirmação antes que a transação possa ser confirmada com êxito no primário.

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 primário

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. Se um movimento primário tiver que ocorrer e não houver nenhuma réplica, uma réplica temporária será colocada para concluir a transição. 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. As réplicas assíncronas serão sondadas em relação ao novo primário.

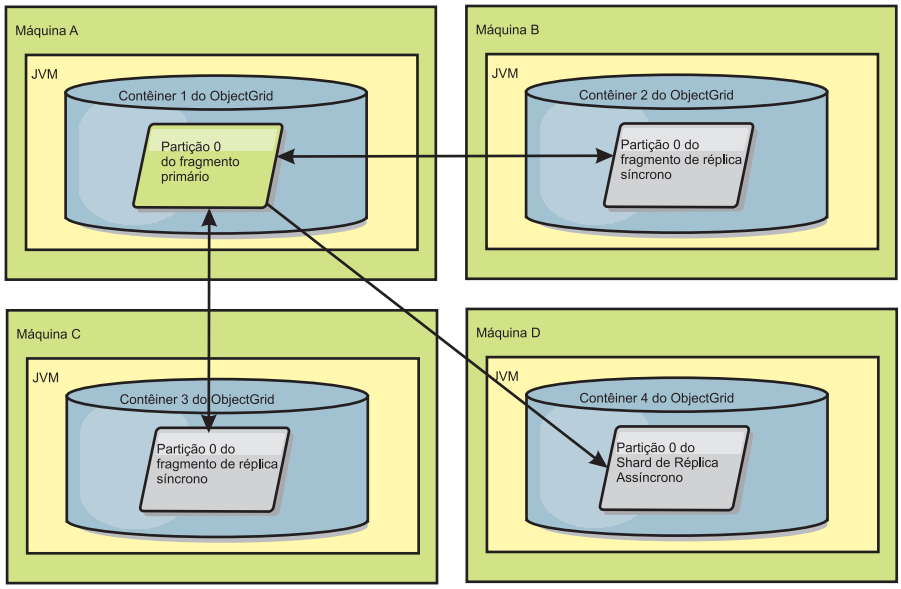


Figura 36. Posicionamento de exemplo de um conjunto de mapas do ObjectGrid para a partição partition0. A política de implementação tem um valor de minSyncReplicas de 1, um valor de maxSyncReplicas de 2, e um valor de maxAsyncReplicas de 1.

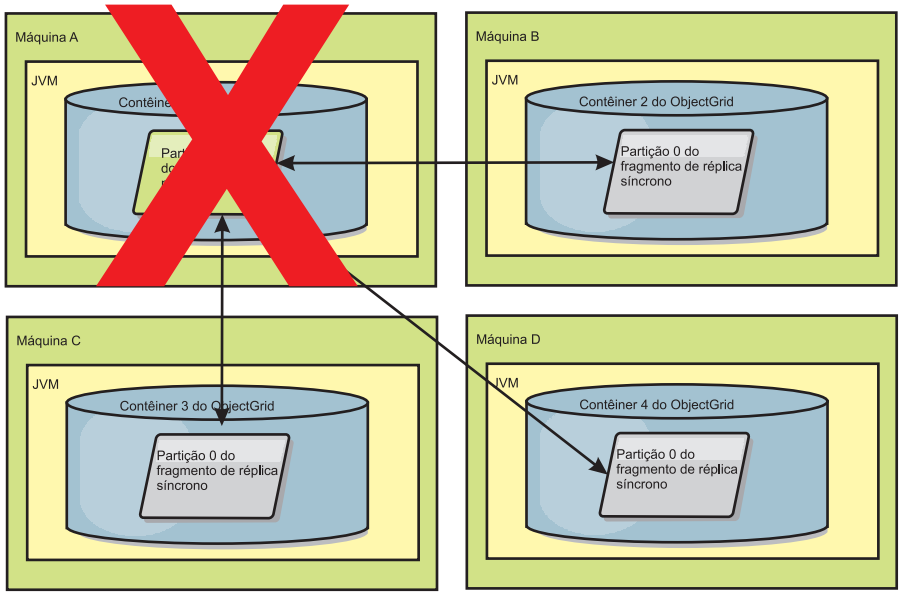


Figura 37. O contêiner para o shard primário falha

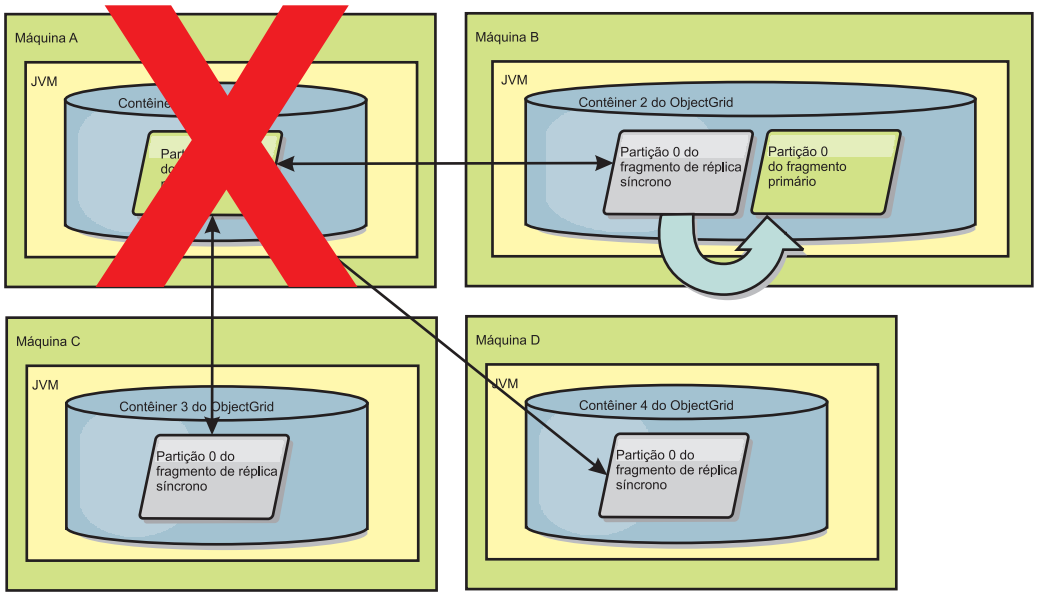


Figura 38. O Shard de Réplica Síncrona no Contêiner 2 do ObjectGrid se Torna o Shard Primário

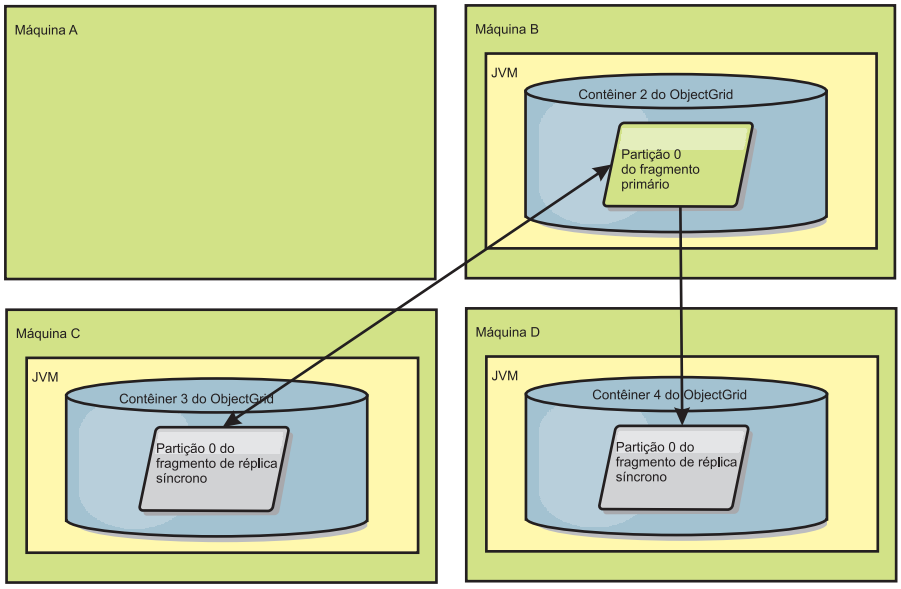


Figura 39. A máquina B contém o shard primário. Dependendo de como o modo de reparo automático é configurado e da disponibilidade dos contêineres, um novo shard de réplica síncrona pode ou não ser posicionado em uma máquina.

Recuperação de shard de réplica

Um shard de réplica síncrona é controlado pelo shard primário. 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.

Roteamento para Zonas Preferenciais

Com o roteamento para zonas preferenciais, o eXtreme Scale pode direcionar transações para zonas com base em suas especificações.

O WebSphere eXtreme Scale permite exercitar uma quantidade significativa de controle sobre onde os shards de um ObjectGrid são colocados.

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 para o WebSphere eXtreme Scale é 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 por meio do SessionHandle. Você deve poder reconstruir o SessionHandle para recuperar seus dados a partir do ObjectGrid.

Como é possível usar zonas para ter mais controle por meio 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”.

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

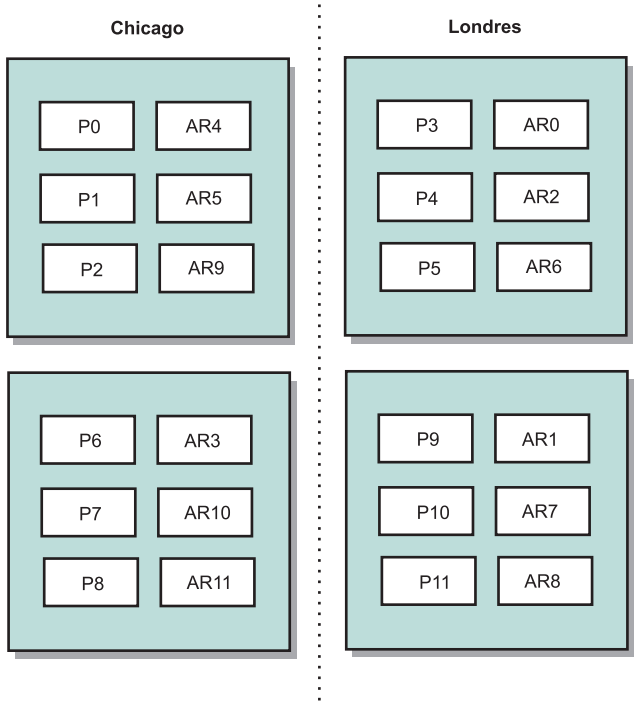


Figura 40. Primários e Réplicas em Zonas

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 mais informações, consulte o tópico sobre o 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 por meio 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.

Topologias de Replicação de Grade Multimestre (AP)

Usando o recurso de replicação assíncrona multimestre, duas ou mais grades podem se tornar espelhos exatos umas das outras. Este espelhamento é executado usando a replicação assíncrona entre links conectando as grades. Cada grade é hospedada dentro de um "domínio" completamente independente, que possui seu próprio serviço de catálogo, servidores de contêiner e um nome de domínio exclusivo. Com o recurso de replicação assíncrona multimestre, É possível usar links para interconectar uma coleta desses domínios e, em seguida, sincronizar os domínios, usando a replicação sobre os links. O eXtreme Scale permite construir quase qualquer topologia, pois a definição dos links entre domínios é deixada para você.

Domínios: Grades com Características Específicas

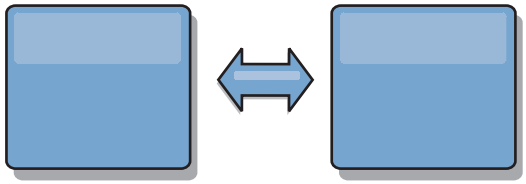
Grades usadas em topologias de replicação multimestre são conhecidas como *domínios*. Cada domínio deve ter as seguintes características:

- Ter um serviço de catálogo privado com um nome de domínio exclusivo
- Ter o mesmo nome de grade que outras grades no domínio
- Ter o mesmo número de partições que outras grades no domínio
- Ser uma grade FIXED_PARTITION (grades PER_CONTAINER não podem ser replicadas)
- Ter o mesmo número de partições (pode o não ter o mesmo número e tipos de réplicas)
- Ter os mesmos tipos de dados sendo replicados como outras grades no domínio
- Ter o mesmo nome do conjunto de mapas, nomes de mapas e modelos de mapas dinâmicos que outras grades no domínio

Depois que os domínios na topologia tiverem sido iniciados, quaisquer grades com as características precedentes serão replicadas. Observe que sua política de replicação será ignorada.

Links Conectando Domínios

Uma infraestrutura de grade de replicação é um gráfico conectado de domínios com links bidirecionais entre eles. Um link permite que dois domínios troquem mudanças de dados. Por exemplo, a topologia mais simples é um par de domínios com um único link entre eles. Os domínios são denominados começando com “A” e, em seguida, “B” e assim por diante a partir da esquerda. O link pode cruzar uma WAN (Wide Area Network), ampliando grandes distâncias. Se o link for interrompido, as mudanças ainda podem ser feitas nos dados em qualquer um dos domínios. As mudanças são reconciliadas mais tarde, quando o link reconecta os domínios. Os links tentarão automaticamente reconectar se a conexão com a rede for interrompida.

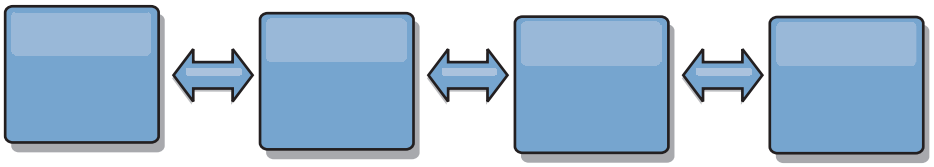


Depois que você configurar os links, o eXtreme Scale primeiro tentará tornar cada domínio idêntico e, em seguida, tentará manter as condições idênticas conforme as mudanças ocorrerem em qualquer domínio. O objetivo do eXtreme Scale é que

cada domínio seja um espelho exato de cada outro domínio conectado pelos links. Os links de replicação entre os domínios ajudam a garantir que as mudanças feitas em um domínio sejam copiadas para os outros domínios.

Topologias em Linha

Embora esteja entre as topologias mais simples, uma topologia em linha demonstra algumas qualidades dos links. Primeiro, não é necessário que um domínio esteja conectado diretamente a outro domínio para receber as mudanças. O domínio B puxará as mudanças do Domínio A. O domínio C recebe mudanças do Domínio A por meio do Domínio B, que conecta os Domínios A e C. De forma semelhante, o Domínio D recebe mudanças dos outros domínios por meio do Domínio C. Esta habilidade envia o carregamento das mudanças de distribuição para longe da origem das mudanças.



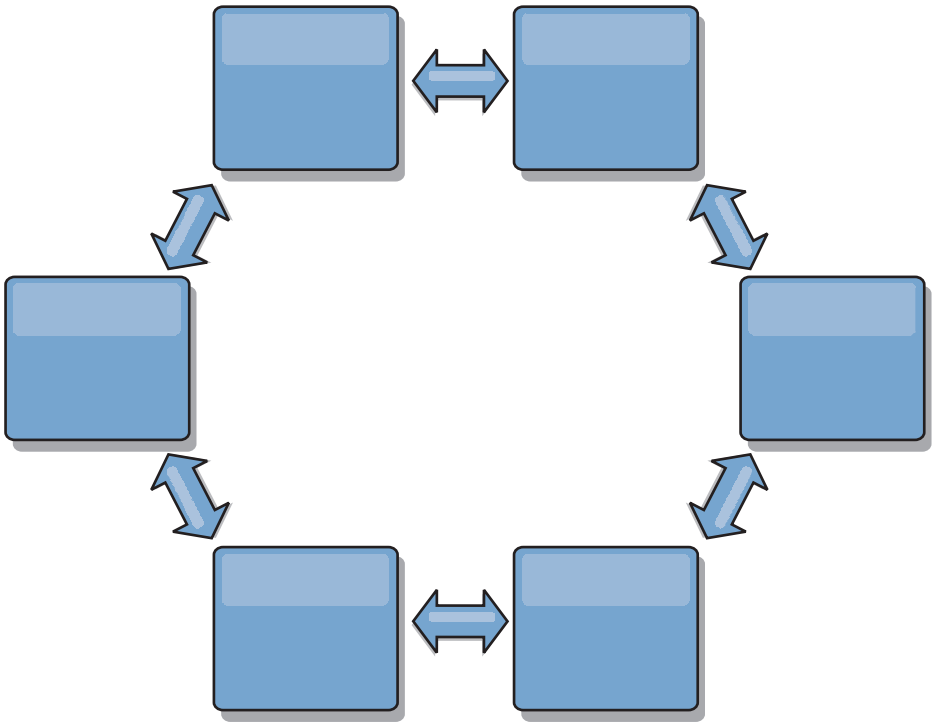
Observe que se o Domínio C falhar, os seguintes eventos poderão ocorrer.

1. O Domínio D pode ficar órfão até o Domínio C ser reiniciado
2. o Domínio C pode se sincronizar com o Domínio B, que é uma cópia do Domínio A
3. O Domínio D pode usar o Domínio C para se sincronizar com as mudanças nos Domínios A e B ocorridas enquanto o Domínio D estava órfão (enquanto o Domínio C estava inativo)

No final, os Domínios A, B, C e D podem se tornar, todos, idênticos uma aos outros novamente.

Topologias em Anel

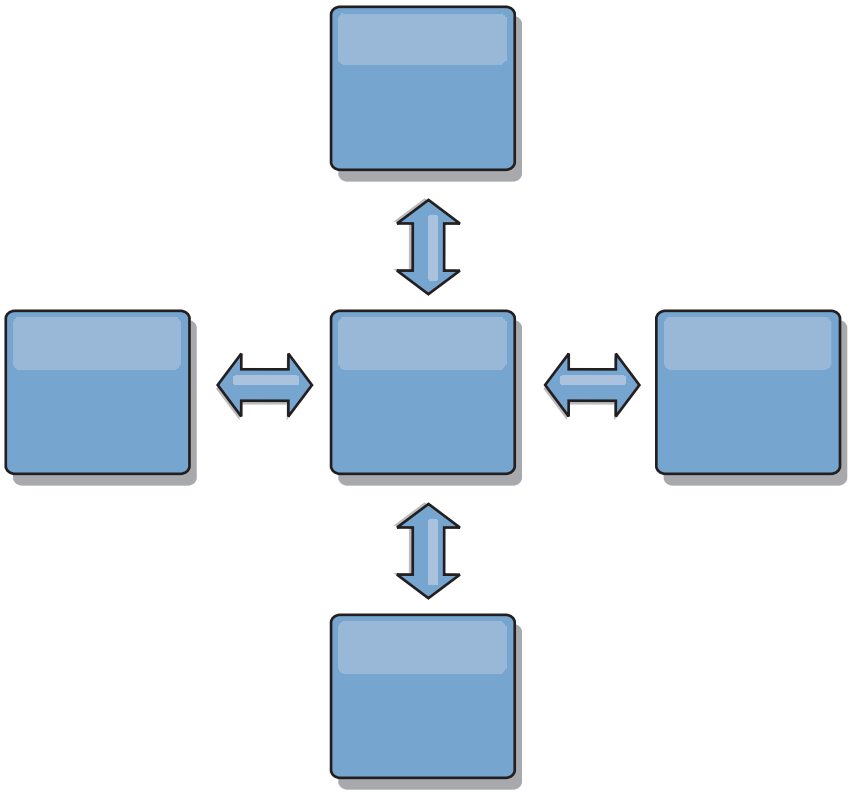
As topologias em anel são um exemplo de uma topologia mais resiliente. Embora um domínio ou um único link possa falhar, os domínios sobreviventes ainda podem obter mudanças viajando ao redor do anel, longe da falha. Cada domínio tem dois links para os outros domínios. Cada domínio tem no máximo dois links, não importa o tamanho de uma topologia em anel. A latência para propagar mudanças pode ser grande, pois as mudanças de um domínio particular podem precisar viajar por meio de vários domínios antes que todos os domínios vejam todas as mudanças. Uma topologia em linha tem o mesmo problema.



Descreva uma topologia em anel mais sofisticada, com um domínio-raiz no centro do anel. O domínio-raiz age como um centro de roteamento central, enquanto os outros domínios agem como centros de roteamento remotos para as mudanças que ocorrem no domínio-raiz. O domínio-raiz pode arbitrar mudanças entre os domínios. Se uma topologia em anel contiver mais de um anel ao redor de um domínio-raiz, o domínio-raiz poderá arbitrar mudanças apenas entre os domínios no anel mais interno. No entanto, os resultados da arbitragem se espalham para os domínios nos outros anéis.

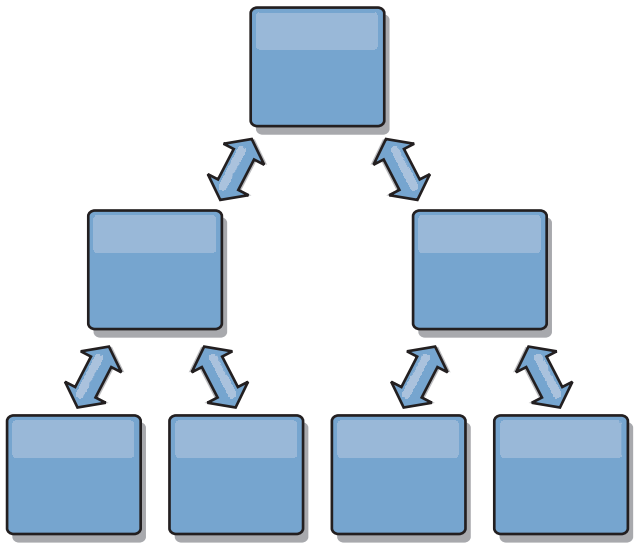
Topologias Hub e Spoke

Uma topologia hub e spoke tem melhor latência, o que significa que as mudanças viajam no máximo por meio de um domínio intermediário (o hub), mas apresenta alguns outros problemas. Ela tem um domínio central que age como um hub. Este "domínio hub" é conectado a cada "domínio spoke" usando um link. Claramente, o encargo de distribuir mudanças entre os domínios fica com o hub. O hub age como um centro de roteamento para colisões, uma configuração que pode ser importante para determinados cenários. Em um ambiente com uma taxa de atualização alta, talvez seja necessário executar o hub em mais hardware que o spoke, para que ele possa continuar ativo. O eXtreme Scale é projetado para escalar linearmente, o que significa que é possível aumentar o hub, conforme necessário, sem dificuldade. No entanto, se o hub falhar, as mudança não serão distribuídas até ele ser reiniciado. As mudanças nos domínios spoke serão distribuídas depois que o hub for reconectado.



Topologias em Árvore

Um último exemplo de topologia é uma árvore direcionada acíclica. Acíclica significa que não há ciclos ou loops. Direcionada significa que existem links apenas entre pais e filhos. Esta configuração pode ser útil para topologias como muitos domínios que não é prático ter um hub central conectado a cada spoke possível ou no qual é necessário ter a habilidade de incluir domínios-filhos sem atualizar o domínio-raiz.



Esta topologia ainda pode ter um centro de roteamento central no domínio-raiz, mas o segundo nível pode agir como centros de roteamento remotos para as mudanças que ocorrem no domínio abaixo deles. O domínio-raiz pode arbitrar as

mudanças entre os domínios no segundo nível apenas. Árvores N-ary também são possíveis. Uma árvore N-ary tem N filhos em cada nível. Cada domínio tem N espalhados.

Considerações sobre Arbitragem no Design de Topologia

Poderão ocorrer colisões de mudanças se os mesmos registros puderem ser alterados simultaneamente em dois locais. Configure cada domínio para ter quase a mesma quantia de CPU, memória, recursos de rede. É necessário observar que os domínios que executam a manipulação da colisão de mudanças (arbitragem) usam mais recursos que outros domínios. As colisões são detectadas automaticamente. Elas são resolvidas usando um entre dois mecanismos:

- **Árbitro de colisão padrão.** O protocolo padrão é usar as mudanças do domínio lexicalmente mais baixo denominado. Por exemplo, se os domínios A e B gerarem um conflito para um registro, a mudança do domínio B será ignorada. O domínio A mantém sua versão e o registro no domínio B é alterado para que corresponda ao registro do domínio A. Isso também se aplica a aplicativos nos quais os usuários ou as sessões são normalmente ligados ou têm afinidade com uma das grades.
- **Árbitro de colisão customizado.** Os aplicativos podem fornecer um árbitro customizado. Quando um domínio detecta uma colisão, ele chama o árbitro. Para obter informações sobre como desenvolver um 'bom' árbitro customizado, consulte Desenvolvendo árbitros customizados para replicação multimestre.

Para topologias nas quais as colisões são possíveis, considere a possibilidade de usar uma topologia hub e spoke ou uma topologia em árvore. Essas duas topologias são úteis para evitar colisões sem fim, que podem acontecer quando:

1. Vários domínios experimentam uma colisão
2. Cada domínio resolve a colisão localmente, produzindo revisões
3. As revisões colidem, resultando em revisões de revisões
4. E assim por diante, uma vez que as revisões são propagadas entre os vários domínios tentando atingir sincronicidade

Para evitar colisões sem fim, escolha um domínio específico – um *domínio de arbitragem* – como o manipulador de colisão para um subconjunto de domínios. Por exemplo, uma topologia hub e spoke pode usar o hub como o manipulador de colisão. O manipulador de colisão spoke ignora quaisquer colisões detectadas pelos domínios spoke. O domínio hub criará revisões, evitando revisões de colisão fora de controle. O domínio designado para tratar colisões deve ser vinculado a todos os domínios para os quais ele é responsável por resolver colisões. Em uma topologia em árvore, os domínios pais internos resolvem colisões para seus filhos imediatos. Em contraste, se usar uma topologia em anel, não será possível designar um domínio no anel como o resolvedor.

A tabela a seguir resume as abordagens de arbitragem que são mais compatíveis com várias topologias.

Tabela 14. Abordagens de Arbitragem. Esta tabela define se a arbitragem de aplicativo é compatível com várias tecnologias.

Topologia	Arbitragem de aplicativo?	Notas
Uma linha de dois domínios	Sim	Escolha um domínio como o árbitro.

Tabela 14. Abordagens de Arbitragem (continuação). Esta tabela define se a arbitragem de aplicativo é compatível com várias tecnologias.

Topologia	Arbitragem de aplicativo?	Notas
Uma linha de três domínios	Sim	O domínio do meio deve ser o árbitro. Pense no domínio do meio como o hub em uma topologia hub e spoke simples.
Uma linha de mais de três domínios	Não	A arbitragem de aplicativo não é suportada.
Um hub com N spokes	Sim	Hub com links para todos os spokes deve ser o domínio de arbitragem.
Um anel de N domínios	Não	A arbitragem de aplicativo não é suportada.
Uma árvore acíclica, direcionada (árvore N-ary)	Sim	Todos os nós-raiz devem arbitrar seus descendentes diretos apenas.

Considerações sobre Links no Design de Topologia

De forma ideal, uma topologia inclui um número mínimo de links enquanto otimiza trade-offs entre latência de mudança, tolerância a falhas e características de desempenho.

- **Latência de mudança**

  A latência de mudança é determinada pelo número de domínios intermediários pelos quais uma mudança deve passar antes de chegar a um domínio específico.

  Uma topologia tem a melhor latência de mudança quando elimina domínios intermediários vinculando cada domínio a outro domínio. No entanto, um domínio deve executar o trabalho de replicação em proporção ao seu número de links. Para topologias grandes, o número completo de links a ser definido pode representar uma carga administrativa.

  A velocidade com que uma mudança é copiada para outros domínios depende de fatores adicionais, como:

  – CPU e largura de banda da rede no domínio de origem
  – O número de domínios e links intermediários entre o domínio de origem e o de destino
  – Os recursos de CPU e de rede disponíveis para os domínios de origem, destino e intermediários

- **Tolerância a Falhas**

  A tolerância a falhas é determinada por quantos caminhos existem entre dois domínios para a replicação de mudança.

  Se existir apenas um único link entre os domínios, se o link falhar, as mudanças não serão propagadas. Se o único link de um domínio para outro domínio passar por domínios intermediários, as mudanças não serão propagadas se qualquer um dos domínios intermediários estiver inativo.

  Considere a topologia em linha com três domínios A, B e C:

  ```
  A <-> B <-> C
  ```

  Se qualquer uma dessas condições for mantida, o Domínio C não verá nenhuma mudança de A:

  – O Domínio A está ativo e o domínio B está inativo
  – O link entre A e B está inativo

– O link entre B e C está inativo

Em contraste, uma topologia em anel permite que cada domínio pegue mudanças de qualquer direção.

```
A <-> B <-> C <-> de volta para A
```

Por exemplo, se o domínio B estiver inativo, o domínio C ainda pode pegar mudanças diretamente do domínio A.

Um design hub e spoke é suscetível a que o hub fique inativo, uma vez que todas as mudanças são pegas por meio do hub. No entanto, vale lembrar que um único domínio ainda é uma grade totalmente tolerante a falhas que pode ter falhas grosseiras como problemas na WAN ou no datacenter físico.

- **Desempenho**

  O número de links definido em um domínio afeta o desempenho. Mais links usam mais recursos e o desempenho de replicação pode diminuir como resultado. A habilidade de pegar mudanças para um domínio A por meio de outros domínios efetivamente libera o domínio A de replicar suas transações em todo lugar. *O carregamento de distribuição de mudança em um domínio é limitado ao número de links que ele usa. Não tem nenhuma ligação com quantos domínios estão na topologia.* Esta propriedade fornece escalabilidade e permite que a carga de distribuição de mudança seja compartilhada pelos domínios na topologia, em vez de colocar a carga em um único domínio.

  Um domínio pode pegar mudanças indiretamente por meio de outros domínios. Considere uma topologia em linha com cinco domínios.

  ```
  A <=> B <=> C <=> D <=> E
  ```

  – A pega mudanças de B, C, D e E por meio de B
  – B pega mudanças de A e C diretamente e mudanças de D e E por meio de C
  – C pega mudanças de B e D diretamente e mudanças de A por meio de B e E por meio de D
  – D pega mudanças de C e E diretamente e mudanças de A e B por meio de C
  – E pega mudanças de D diretamente e mudanças de A, B e C por meio de D

  O carregamento de distribuição nos domínios A e E é o mais baixo, porque eles têm, cada um, um link apenas para um único domínio. O carregamento de distribuição nos domínios B, C e D é o dobro do carregamento nos domínios A e E porque os domínios B, C e D têm, cada um, um link para dois domínios. Essa distribuição de carregamentos poderia permanecer constante, mesmo que a linha contivesse 1000 domínios, pois o carregamento depende do número de links de cada domínio, não do número geral de domínios na topologia.

Considerações sobre Desempenho

Leve em consideração as seguintes limitações quando usar as topologias de replicação multimestre.

- **Ajuste de distribuição de mudança** (discutido anteriormente)
- **Latência de replicação** (discutido anteriormente)
- **Desempenho do link de replicação** O eXtreme Scale cria um único soquete TCP/IP entre qualquer par de JVMs. Todo tráfego entre essas JVMs ocorre por meio do soquete, incluindo a replicação multimestre. Como os domínios são hospedados em pelo menos N JVMs de contêiner, fornecendo pelo menos N links TCP para os domínios do mesmo nível, os domínios com números maiores de contêineres terão níveis mais altos de desempenho de replicação. Mais contêineres significa mais recursos de CPU e de rede.

- **Ajuste da janela de deslizamento TCP e RFC 1323** A ativação do suporte RFC 1323 em ambos os terminais de um link permite mais dados para um roundtrip, resultando em um maior rendimento. A técnica expande a capacidade da janela em um fator de aproximadamente 16.000.

  A rechamada desses soquetes TCP usa um mecanismo de janela de deslizamento para controlar o fluxo de dados em massa, o que geralmente limita o soquete a 64 KB para um intervalo de roundtrip. Se o intervalo de roundtrip for de 100 ms, a largura de banda será limitada a 640 KB/segundo sem ajuste adicional. Usar totalmente a largura de banda disponível em um link pode exigir um ajuste específico para um sistema operacional. A maioria dos sistemas operacionais inclui parâmetros de ajuste, incluindo opções RFC 1323, para aprimorar o rendimento por meio dos links de alta latência.

  Vários fatores podem afetar o desempenho de replicação:

  – A velocidade com que o eXtreme Scale pode pegar mudanças.
  – A velocidade com que o eXtreme Scale pode atender pedidos de replicação de tomada.
  – A capacidade da janela de deslizamento.
  – Ajuste do buffer de rede em ambos os lados de um link para permitir que o eXtreme Scale pegue mudanças por meio do soquete o mais rápido possível.
- **Serialização de Objeto** Todos os dados devem ser serializáveis. Se um domínio não estiver usando COPY_TO_BYTES, o domínio deverá usar a serialização Java ou ObjectTransformers para otimizar o desempenho da serialização.
- **Compactação** O eXtreme Scale compacta todos os dados enviados entre os domínios por padrão. Não há nenhuma opção para desativar a compactação no release atual.
- **Ajuste de memória** *O uso de memória para uma topologia de replicação multimestre é independente do número de domínios na topologia.*

  A ativação da replicação multimestre inclui uma sobrecarga fixa por Entrada de mapa para tratar a versão. Cada contêiner também rastreia uma quantia fixa de dados para cada domínio na topologia. Uma topologia com dois domínios usa aproximadamente a mesma memória que uma topologia com 50 domínios. O eXtreme Scale não usa logs de reprodução ou filas semelhantes em sua implementação, o que significa que se um link de replicação não estiver disponível por um período substancial de tempo, não haverá nenhuma estrutura de dados aumentando de tamanho, aguardando para retomar a replicação quando o link for reiniciado.

Vários Datacenters com FIXED_PARTITION

Agora, é possível usar uma grade FIXED_PARTITION entre dois ou mais datacenters. Cada datacenter precisa de seu próprio domínio, em termos de replicação multimestre. Cada datacenter pode ler e gravar dados em relação ao domínio local. Essas mudanças serão propagadas para os outros datacenters usando os links que você definiu.

Clientes Totalmente Replicados

Essa variação de topologia envolve um par de servidores do eXtreme Scale sendo executados como um hub. Cada cliente cria uma grade autocontida de contêiner único com um catálogo na JVM do cliente. Um cliente usa sua grade para conectar ao catálogo do hub, fazendo com que o cliente sincronize com o hub assim que obtiver uma conexão com o hub.

Todas as mudanças feitas pelo cliente são locais para o cliente e são replicadas de forma assíncrona para o hub. O hub age como um domínio de arbitragem, distribuindo mudanças para todos os clientes conectados. A topologia de clientes totalmente replicada fornece um bom cache L2 para um mapeador relacional de objeto, como OpenJPA. As mudanças serão distribuídas rapidamente entre JVMs de cliente por meio do hub. Contanto que o tamanho do cache possa estar contido no espaço de heap disponível dos clientes, esta topologia será uma boa arquitetura para este estilo de L2.

Use várias partições para escalar o domínio do hub em várias JVMs, se necessário. Como todos os dados ainda devem se ajustar em uma única JVM de cliente, usar várias partições aumenta a capacidade do hub para distribuir e arbitrar mudanças, mas não altera a capacidade de um único domínio.

Limitações

Leve em consideração as seguintes limitações ao decidir se e como usar as topologias de replicação multimestre.

- **Tenha cuidado com a configuração de carregadores de classe com vários domínios**

  Os domínios devem ter acesso a todas as classes que são usadas como chaves e valores. Todas as dependências devem ser refletidas em todos os caminhos da classe para as JVMs do contêiner da grade para todos os domínios. Se um plug-in CollisionArbiter recuperar o valor para uma entrada de cache, as classes para os valores deverão estar presentes para o domínio que está chamando o árbitro.

- **O uso de carregadores não é recomendado**

  Os carregadores podem ser usados para mudanças de interface entre uma grade e um banco de dados. É improvável que todas as grades (domínios) em uma topologia sejam colocadas geograficamente com o mesmo banco de dados. A latência de WAN e outros fatores podem renderizar este caso de uso não desejável.

  O pré-carregamento da grade é outro problema que requer um design cuidadoso. Geralmente, quando uma grade é reiniciada, ela é pré-carregada novamente. O pré-carregamento não é necessário ou mesmo desejável quando estiver usando a replicação multimestre. Assim que um domínio estiver on-line, ele se recarregará automaticamente com o conteúdo dos domínios aos quais está vinculado. Como resultado, não há necessidade de iniciar um pré-carregamento manual para uma grade que é um domínio em uma topologia de replicação multimestre.

  Os carregadores, geralmente, obedecem as regras de inserção e atualização. Com a replicação multimestre, as inserções devem ser tratadas como mesclagens. Quando os dados estiverem sendo pegos remotamente após o reinício de um domínio, os dados existentes serão ‘inseridos' no domínio local. Como esses dados podem já ter estado no banco de dados local, uma inserção típica falhará com uma exceção de chave duplicada no banco de dados. Em vez disso, a semântica de mesclagem deve ser usada.

  O eXtreme Scale pode ser configurado para fazer um pré-carregamento baseado em shard, usando os métodos de pré-carregamento nos plug-ins do Carregador. Não use esta técnica em uma topologia de replicação multimestre. Em vez disso, use um pré-carregamento baseado no cliente quando a topologia for iniciada (inicialmente). Permita que a topologia multimestre atualize os domínios reiniciados com uma cópia atual do que está armazenado em outros domínios

na topologia. Depois que os domínios tiverem sido iniciados, a topologia multimestre será responsável por mantê-los em sincronia.

- **EntityManager não é suportado**

  Um conjunto de mapas contendo um mapa de entidade não é replicado por meio dos domínios.

- **Mapas de matriz de byte não são suportados**

  Um conjunto de mapas contendo um mapa configurado com COPY_TO_BYTES não é replicado por meio dos domínios.

- **Write-behind não é suportado**

  Um conjunto de mapas contendo um mapa configurado com o suporte write-behind não é replicado por meio dos domínios.

JMS para Distribuição de Mudanças de Transação

Use Java Message Service (JMS) para mudanças de transação distribuída 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

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 mais informações sobre o uso de JMS para mudanças de transação distribuída, consulte as informações sobre mudanças de distribuição entre Java Virtual Machines peer no *Guia de Administração*.

Conjuntos de Mapas para Replicação

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 91 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:

Capítulo 6. Visão Geral do Processamento de Transações

Processamento de Sessões e de Transações

O WebSphere eXtreme Scale usa transações de acordo com seu mecanismo de interação com os dados.

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.

Transações

As transações possuem muitas vantagens para o armazenamento e a manipulação de dados. É possível utilizar transações para proteger a grade contra mudanças simultâneas, aplicar várias mudanças como uma unidade simultânea, replicar dados e implementar um ciclo de vida para bloqueios nas mudanças.

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.

Vantagens das Transações

Utilize as transações pelos seguintes motivos:

Usando as transações, você pode:

- Recuperar alterações se ocorrer uma exceção ou a lógica de negócios precisar desfazer mudanças de estado.
- Para aplicar várias alterações como uma unidade atômica no momento commit.
- 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.

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.

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.

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.

Atributo CopyMode

É possível ajustar o número de cópias definindo o atributo CopyMode do BackingMap ou objetos ObjectMap.

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.

Bloqueio de Entrada de Mapa

Um BackingMap do ObjectGrid suporta várias estratégias de bloqueio para mapas para manter a consistência de entradas de cache.

Cada BackingMap pode ser configurado para utilizar uma das seguintes estratégias de bloqueio:

1. Modo de Bloqueio Otimista
2. Modo de Bloqueio Pessimista
3. Nenhum(a)

A estratégia de bloqueio padrão é OPTIMISTIC. Utilize o bloqueio optimistic quando os dados são alterados de maneira infrequente. Os bloqueios são mantidos apenas por uma curta duração enquanto os dados estão sendo lidos do cache e copiados para a transação. Quando o cache da transação é sincronizado com o cache principal, quaisquer objetos de cache que foram atualizados são verificados junto à versão original. Se a verificação falhar, então, ocorre o rollback da transação e o resultado é uma exceção OptimisticCollisionException.

A estratégia de bloqueio PESSIMISTIC adquire bloqueios para entradas de cache e deve ser utilizada quando os dados são alterados frequentemente. Sempre que uma entrada de cache é lida, um bloqueio é adquirido e mantido condicionalmente até que a transação seja concluída. A duração de alguns bloqueios pode ser ajustada utilizando níveis de isolamento de transação para a sessão.

Se o bloqueio não for necessário porque os dados nunca são atualizados ou são atualizados apenas durante períodos tranquilos, é possível desativar o bloqueio utilizando a estratégia de bloqueio NONE. Esta estratégia é muito rápida porque um gerenciador de bloqueio não é necessário. A estratégia de bloqueio NONE é ideal para tabelas de consulta ou mapas somente leitura.

Para obter mais informações sobre as estratégias de bloqueio, consulte as informações sobre as estratégias de bloqueio no *Visão Geral do Produto*.

Especificando uma Estratégia de Bloqueio

O exemplo a seguir demonstra como a estratégia de bloqueio pode ser configurada nos BackingMaps map1, map2 e map3, em que cada mapa está utilizando uma estratégia de bloqueio diferente. O primeiro fragmento mostra como usar o XML para a configuração de estratégia de bloqueio e o segundo fragmento mostra uma abordagem programática.

Abordagem XML

Configuração de BackingMap - Exemplo 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="test">
            <backingMap name="map1"
                lockStrategy="PESSIMISTIC" numberOfLockBuckets="31"/>
            <backingMap name="map2"
                lockStrategy="OPTIMISTIC" numberOfLockBuckets="409"/>
            <backingMap name="map3"
                lockStrategy="NONE"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

Abordagem Programática

```
Configuração BackingMap - exemplo programático
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.LockStrategy;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
...
ObjectGrid og =
 ObjectGridManagerFactory.getObjectGridManager().createObjectGrid("test");
BackingMap bm = og.defineMap( "map1" );
bm.setLockStrategy( LockStrategy.PESSIMISTIC );
bm.setNumberOfLockBuckets(31);
bm = og.defineMap( "map2" );
bm.setNumberOfLockBuckets(409);
bm.setLockStrategy( LockStrategy.OPTIMISTIC );
bm = og.defineMap("map3");
bm.setLockStrategy( LockStrategy.NONE );
```

Para evitar uma exceção java.lang.IllegalStateException, o método setLockStrategy deve ser chamado antes de usar os métodos initialize ou getSession na instância do ObjectGrid local.

Para obter mais informações, consulte o tópico sobre estratégias de bloqueio em *Visão Geral do Produto*.

Configuração do Gerenciador de Bloqueios

Quando a estratégia de bloqueio pessimistic ou optimistic for utilizada, será criado um gerenciador de bloqueios para o BackingMap. O gerenciador de bloqueios utiliza um mapa hash para controlar entradas bloqueadas por uma ou mais transações. Se existirem muitas entradas de mapa no mapa hash, mais depósitos de bloqueio podem resultar em melhor desempenho. O risco de colisões de sincronização Java é inferior conforme a quantidade de depósitos aumenta. Mais depósitos de bloqueios também resultam em maior simultaneidade. Os exemplos anteriores mostram como um aplicativo pode configurar o número de depósitos de bloqueio para utilizar para uma determinada instância de BackingMap.

Para evitar uma exceção java.lang.IllegalStateException, o método setNumberOfLockBuckets deve ser chamado antes de chamar os métodos initialize ou getSession na instância do ObjectGrid. O parâmetro do método setNumberOfLockBuckets é um inteiro primitivo Java que especifica a quantidade de depósitos de bloqueio para uso. Utilizar um número primo permite uma distribuição uniforme de entradas do mapa sobre os depósitos de bloqueios. Um bom ponto de partida para obter melhor desempenho é configurar o número de depósitos de bloqueios para aproximadamente dez por cento do número esperado de entradas do BackingMap.

LockDeadlockException

Este exemplo de código que mostra como capturar a exceção e a mensagem resultante que será exibida.

```
try {
...
} catch (ObjectGridException oe) {
System.out.println(oe);
}
```

O resultado é:

```
com.ibm.websphere.objectgrid.plugins.LockDeadlockException: _Message
```

Essa mensagem representa a cadeia que é transmitida como um parâmetro quando a exceção é criada e emitida.

Causa da Exceção

O tipo mais comum de exceção de conflito ocorre ao utilizar uma estratégia de bloqueio pessimista e dois clientes separados, cada qual possuindo um bloqueio compartilhado sobre um determinado objeto. Em seguida, ambos tentam evoluir para um bloqueio exclusivo sobre tal objeto. O diagrama a seguir ilustra tal situação, incluindo os blocos de transação que fazem com que a exceção seja emitida.

O trecho de código Java a seguir demonstra como transmitir um arquivo de configuração XML para criar um ObjectGrid.

Esta é uma visualização abstrata do que está ocorrendo em seu programa quando ocorre a exceção. Em um aplicativo com muitos encadeamentos atualizando o mesmo ObjectMap, é possível encontrar esta situação. Este é um exemplo de dois clientes que executam blocos de código de transação, conforme ilustrado na figura anterior.

Soluções Possíveis

É possível encontrar a situação ilustrada na Figura 1 quando muitos encadeamentos iniciam transações em um determinado mapa. Nesse caso, uma exceção será emitida para evitar a interrupção do seu programa. É possível notificar a si mesmo e acrescentar código no bloco de captura para obter mais detalhes sobre a causa. Como essa exceção só ocorre numa estratégia de bloqueio pessimista, uma solução simples é apenas utilizar a estratégia de bloqueio otimista. Se precisar de uma estratégia de bloqueio pessimista, no entanto, poderá utilizar o método getForUpdate em vez de o método get. Isso elimina o recebimento de exceções para a situação descrita anteriormente.

Estratégias de Bloqueio

As estratégias de bloqueio incluem pessimista, otimista e nenhum. Para escolher uma estratégia de bloqueio, é necessário considerar questões como a porcentagem de cada tipo de operações que você tem, se você utiliza um utilitário de carga, entre outras.

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" />
```

Bloqueio Pessimista

Use a estratégia de bloqueio pessimista para ler e gravar mapas quando outras estratégias de bloqueio não foram 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 otimista assume que nenhuma transação em execução simultânea pode tentar atualizar a mesma entrada de mapa. 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 usar nenhuma estratégia de bloqueio, nenhum bloqueio de transação para uma entrada de mapa é 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 mais informações 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, é possível 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 159.

Para obter mais informações, consulte o tópico sobre manipulação de bloqueios no *Guia de Programação* ou bloqueio de entrada de mapa no *Guia de Administração*.

JMS para Distribuição de Mudanças de Transação

Use Java Message Service (JMS) para mudanças de transação distribuída 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

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 mais informações sobre o uso de JMS para mudanças de transação distribuída, consulte as informações sobre mudanças de distribuição entre Java Virtual Machines peer no *Guia de Administração*.

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 construir 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 mudança 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 7. Visão Geral de Segurança

WebSphere eXtreme Scale pode proteger o acesso a dados, incluindo permissão para integração 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 8. Visão Geral do Serviço de Dados REST

O serviço de dados REST WebSphere eXtreme Scale é um serviço HTTP Java compatível com Microsoft WCF Data Services (formalmente, ADO.NET Data Services) e implementa o Open Data Protocol (OData). O Microsoft WCF Data Services é compatível com essa especificação quando utiliza Visual Studio 2008 SP1 e .NET Framework 3.5 SP1.

Requisitos de Compatibilidade

O serviço de dados REST permite que qualquer cliente HTTP acesse uma grade de dados. O serviço de dados REST é compatível com o suporte do WCF Data Services fornecido com o Microsoft .NET Framework 3.5 SP1. Aplicativos RESTful podem ser desenvolvidos com um rico conjunto de ferramentas fornecido pelo Microsoft Visual Studio 2008 SP1. A figura fornece uma visão geral de como o WCF Data Services interage com clientes e bancos de dados.

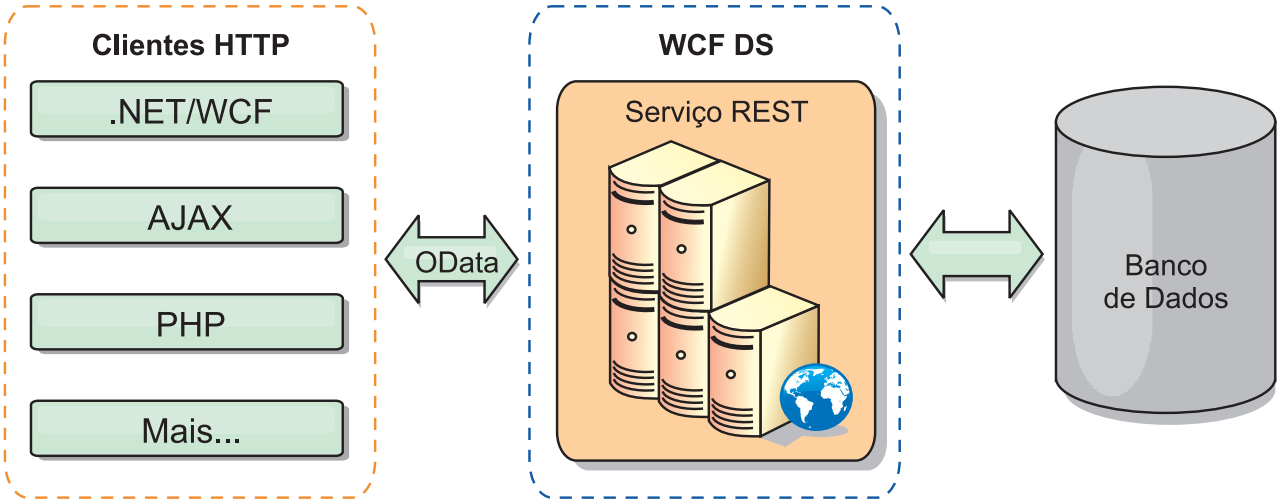


Figura 41. Microsoft WCF Data Services

O WebSphere eXtreme Scale inclui um conjunto de APIs com várias funções para clientes Java. Conforme mostrado na figura a seguir, o serviço de dados REST é um gateway entre clientes HTTP e a grade do WebSphere eXtreme Scale, comunicando-se com a grade por meio de um cliente do WebSphere eXtreme Scale. O serviço de dados REST é um servlet Java, que permite implementações flexíveis para Plataforma Java comum, plataformas Enterprise Edition (JEE), como o WebSphere Application Server. O serviço de dados REST se comunica com a grade do WebSphere eXtreme Scale usando as APIs Java do WebSphere eXtreme Scale. Ele permite clientes do WCF Data Services ou qualquer outro cliente que possa se comunicar com HTTP e XML.

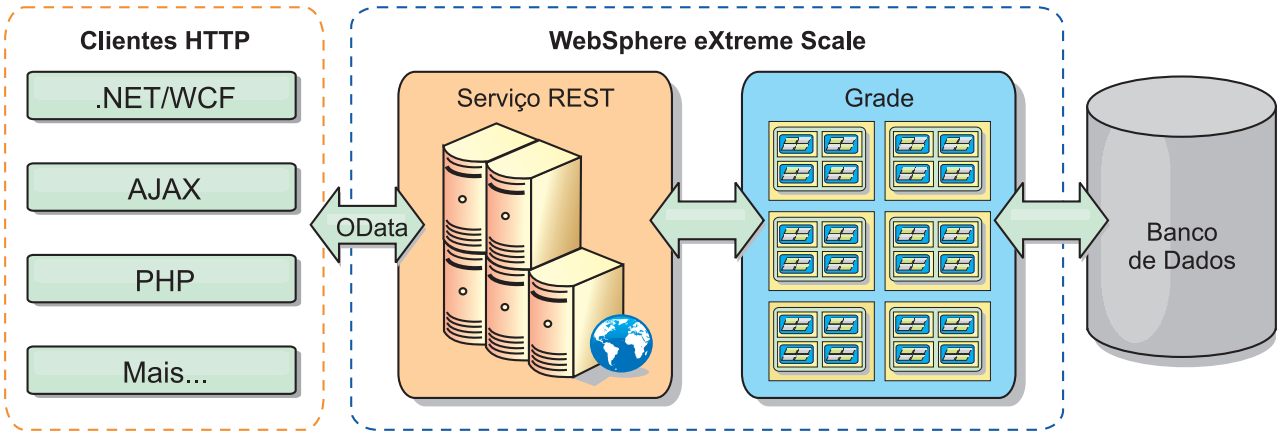


Figura 42. Serviço de Dados REST do WebSphere eXtreme Scale

Consulte o "Amostra e Tutorial de Serviços de Dados REST" na página 214 ou use os seguintes links para aprender mais sobre o WCF Data Services.

- Microsoft WCF Data Services Developer Center
- Visão geral do ADO.NET Data Services no MSDN
- White Paper: Utilizando ADO.NET Data Services
- Atom Publish Protocol: Data Services URI and Payload Extensions
- Conceptual Schema Definition File Format
- Entity Data Model for Data Services Packaging Format
- Open Data Protocol
- FAQ do Open Data Protocol

Características

Esta versão do serviço de dados REST do eXtreme Scale suporta os seguintes recursos:

- Modelagem automática de entidades de API EntityManager do eXtreme Scale como entidades do WCF Data Services, que inclui o seguinte suporte:
  - Tipo de dados Java para conversão de tipo do Entity Data Model
  - Suporte de associação de entidade
  - Suporte à raiz do esquema e à associação de chave, que é requerido para grades de dados particionadas

  Consulte Modelo de entidade para obter mais informações.
- Formato de carga útil de dados Atom Publish Protocol (AtomPub ou APP) XML e JavaScript™ Object Notation (JSON).
- Operações Create, Read, Update and Delete (CRUD) utilizando os respectivos métodos de pedido de HTTP: POST, GET, PUT e DELETE. Além disso, a extensão da Microsoft MERGE é suportada.
- Consultas simples, usando filtros
- Pedidos de recuperação de lote e do conjunto de mudanças
- Suporte à grade particionada para alta disponibilidade
- Interoperabilidade com clientes de API EntityManager do eXtreme Scale
- Suporte para servidores da Web JEE padrão
- Simultaneidade otimista

- Autorização e autenticação de usuário entre o serviço de dados REST e a grade de dados do eXtreme Scale

Limitações e Problemas Conhecidos

- Pedidos em túnel não são suportados.

Capítulo 9. Visão Geral de Integração da Estrutura Spring

O Spring é uma estrutura popular para desenvolvimento de aplicativos Java. O WebSphere eXtreme Scale fornece suporte para permitir que o Spring gerencie as transações do eXtreme Scale e configure clientes e servidores que compõem a grade de dados de memória implementada.

Transações Nativas Gerenciadas do Spring

O Spring fornece transações gerenciadas por contêiner que são similares a um servidor de aplicativos do Java Platform, Enterprise Edition. Porém, o mecanismo do Spring pode se conectar em diferentes implementações. O WebSphere eXtreme Scale fornece a integração do gerenciador de transações que permite ao Spring para gerenciar os ciclos de vida da transação do ObjectGrid. Consulte as informações sobre transações nativas no *Guia de Programação* para obter detalhes.

Beans de Extensão Gerenciados do Spring e Suporte a Espaço de Nomes

Além disso, o eXtreme Scale se integra ao Spring para permitir que os beans de estilo do Spring definidos para pontos de extensão ou plug-ins. Este recurso fornece configurações mais sofisticadas e mais flexíveis para configuração dos pontos de extensão.

Além dos beans de extensão gerenciados do Spring, o eXtreme Scale fornece um espaço de nomes Spring chamado "objectgrid". Beans e implementações integradas são predefinidos neste espaço de nomes, o que facilita aos usuários configurar o eXtreme Scale. Consulte Beans de extensão Spring e suporte a espaço de nomes para obter mais detalhes sobre esses tópicos e uma amostra de como iniciar um servidor de contêiner do eXtreme Scale usando configurações Spring.

Suporte ao Escopo Shard

Com a configuração do Spring estilo tradicional, um bean ObjectGrid pode se do tipo singleton ou prototype. O ObjectGrid também suporta um novo escopo chamado de escopo "shard". Se um bean for definido como escopo shard, então somente um bean será criado por shard. Todos os pedidos para beans com um ID ou IDs correspondentes a essa definição de bean no mesmo shard resultarão no retorno dessa instância de bean específica pelo contêiner Spring.

O exemplo a seguir mostra que um bean com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl é definido com o escopo configurado para shard. Portanto, apenas uma instância da classe JPAPropFactoryImpl é criada por shard.

```
<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl" scope="shard" />
```

Fluxo da Web do Spring

O Fluxo da Web do Spring armazena o estado da sua sessão na Sessão HTTP pelo padrão. Se um aplicativo da Web for configurado para usar o eXtreme Scale para o gerenciamento de sessão, então ele será usado automaticamente pelo Spring para armazenar este estado e se tornará tolerante a falhas da mesma forma que a sessão.

compactando

As extensões Spring do eXtreme Scale estão no arquivo `ogspring.jar`. Este arquivo Java archive (JAR) deve estar no caminho de classe para o suporte ao Spring funcionar. Se um aplicativo JEE que está em execução em um WebSphere Extended Deployment alterado WebSphere Application Server Network Deployment, então o aplicativo deve colocar o arquivo `spring.jar` e seus arquivos associados nos módulos EAR (Enterprise Archive). Você também deve colocar o arquivo `ogspring.jar` no mesmo local.

Capítulo 10. Tutoriais, Exemplos e Amostras

Vários tutoriais, exemplos e amostras do WebSphere eXtreme Scale estão disponíveis.

Tutoriais

Os seguintes tutoriais estão disponíveis atualmente.

- Tutorial do ObjectMap
- "Tutorial do Entity Manager: Visão Geral"
-
-

Exemplos

Os tópicos abaixo ilustram recursos-chave do WebSphere eXtreme Scale.

- Consulte o exemplo da API de grade de dados em *Guia de Programação*
- Consulte os detalhes sobre como configurar implementações locais no *Guia de Administração*

Amostras

Amostras para ilustrar como utilizar as APIs do ObjectGrid em diversos ambientes são fornecidas com o produto WebSphere eXtreme Scale.

Artigos com Tutoriais e Exemplos

Tabela 15. Artigos Disponíveis por Recurso

Artigo	Características
Criando aplicativos prontos para grade	API de ObjectMap, API de EntityManager, Consulta, Agentes, Java SE e EE, Estatísticas, Particionamento, Administração/Operações, Eclipse
Computação em estilo de grade e processamento de dados	API do EntityManager, Agentes
Criando uma alternativa de banco de dados escalável, resiliente e de alto desempenho	API do ObjectMap, Replicação, Particionamento, Administração/Operações, Eclipse
Melhorando xsadmin para WebSphere eXtreme Scale	Administração
Redbook: Guia do Usuário	Todos os tópicos

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.

Sobre Esta Tarefa

Procedimento

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.

Procedimento

1. Crie a entidade customer, que é usada para armazenar detalhes do cliente independentemente do objeto Order. Um exemplo da entidade customer é apresentado a seguir:

```
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 do "Tutorial do Entity Manager: Criando uma Classe de Entidade" na página 182. 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 gerenciador de entidade persistir o objeto Order, ele também deverá 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.

Sobre Esta Tarefa

Figura 43. 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.

Procedimento

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.

Procedimento

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, por meio 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.

Procedimento

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.

Procedimento

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();
     // Verify that the order wasn't updated by someone else.
     // 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) {
        // Add the item back to the inventory.
        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.

Sobre 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.

Procedimento

1. "Tutorial do ObjectQuery - Etapa 1" na página 191
   - 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 192
   - Como criar um índice que a consulta pode utilizar
3. "Tutorial do ObjectQuery - Etapa 3" na página 193
   - 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 195
   - 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, será possível 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.

Procedimento

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");

       // Define the schema
       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, será possível 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 191 antes de continuar com esta etapa do tutorial.

Procedimento

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 uma consulta executa esses filtros no campo itemName, o mecanismo de consulta automaticamente usa o índice definido. Usar o índice permite que a consulta seja executada muito mais rapidamente 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 192 antes de continuar com esta etapa.

Sobre 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.

Procedimento

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");

        // Define the schema
        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, será possível 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 193 antes de continuar com a etapa atual.

Procedimento

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");

        // Define the schema
        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: Visão Geral

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.

Sobre 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

Procedimento

1. "Tutorial de Segurança do Java SE - Etapa 1" na página 198
   - 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 201
   - 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 207
   - Uso da política de autorização do JAAS
4. "Tutorial de Segurança do Java SE - Etapa 4" na página 211
   - 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.

Procedimento

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 198.

Sobre 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.

Procedimento

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

Após autenticar um cliente, na etapa anterior, é possível fornecer privilégios de segurança por meio 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 201 antes de continuar com esta tarefa.

Sobre Esta Tarefa

A etapa anterior deste tutorial demonstrou como ativar a autenticação em uma grade 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 usuário autenticado.

Semelhante a vários outros sistemas, o eXtreme Scale adota um mecanismo de autorização baseado em permissão. WebSphere eXtreme Scale tem categorias de permissão diferentes que são representadas por diferentes classes de permissão. Este tópico descreve o MapPermission. Para conhecer a categoria completa de permissões, consulte a referência de autorização do cliente no *Guia de Programação*..

No WebSphere eXtreme Scale, a classe com.ibm.websphere.objectgrid.security.MapPermission representa permissões para os recursos do eXtreme Scale, especificamente os métodos das interfaces ObjectMap ou JavaMap. O WebSphere eXtreme Scale define as seguintes cadeias de permissões para acesso aos métodos de ObjectMap e 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 permissões de mapa diferentes para métodos diferentes. Se as permissões requeridas não forem concedidas ao cliente, isso resultará em um AccessControlException.

Este tutorial demonstra como utilizar a autorização Java Authentication and Authorization Service (JAAS) para conceder acessos do mapa de autorização para diferentes usuários.

Procedimento

1. **Ative a autorização do eXtreme Scale**. Para ativar a autorização no ObjectGrid, você precisa configurar o atributo securityEnabled como `true` para esse ObjectGrid específico no arquivo XML. A ativação da segurança no ObjectGrid significa que você está ativando a autorização. Utilize os seguintes comandos para criar um novo arquivo XML do ObjectGrid com a segurança ativada.
   a. Navegue para o diretório `bin`.
      ```
      cd objectgridRoot/bin
      ```
   b. Copie o arquivo `SimpleApp.xml` no arquivo `SecureSimpleApp.xml`.
      ```
      cp SimpleApp.xml SecureSimpleApp.xml
      ```
   c. Abra o arquivo `SecureSimpleApp.xml` e 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. **Defina a política de autorização.** Na seção de autenticação pré-cliente, você criou três usuários no keystore: cashier, manager e administrator. Neste exemplo, o usuário "cashier" só tem permissões de leitura a todos os mapas, e o usuário "manager" tem 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 é definido no diretório de segurança:

```
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:

   - A `codebase "http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction"` é uma URL reservada especialmente para ObjectGrid. Todas as permissões do ObjectGrid concedidas a principals devem utilizar esse código base especial.
   - A primeira instrução de concessão concede permissão de mapa "read" ao principal `"CN=cashier,O=acme,OU=OGSample"`, de modo que cashier tenha apenas permissão de leitura ao mapa para todos os mapas na contabilidade do ObjectGrid.
   - A segunda instrução de concessão concede permissão de mapa "all" ao principal `"CN=manager,O=acme,OU=OGSample"`, de modo que manager tenha todas as permissões para os mapas na contabilidade do ObjectGrid.

   Agora é possível 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.

c. É possível iniciar um servidor de contêiner seguro utilizando o seguinte script. Execute o seguinte script a partir do diretório bin:

   - 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:

- Utilize o arquivo `SecureSimpleApp.xml` em vez de o arquivo `SimpleApp.xml`.
- Inclua outro argumento `-Djava.security.auth.policy` para configurar o arquivo de política de autorização JAAS para o processo do servidor de contêiner.

Utilize o mesmo comando da etapa anterior do tutorial:

a. Navegue até o diretório bin.

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, em vez de utilizar o usuário "manager", utilize o usuário "cashier" para ativar o aplicativo cliente.

c. Navegue até o diretório bin.

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
```

Essa exceção ocorre porque o usuário "cashier" não tem permissão de gravação, portanto, 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*.

O que Fazer Depois

Conclua a próxima etapa do tutorial. Consulte “Tutorial de Segurança do Java SE - Etapa 4”.

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 207 antes de continuar com esta tarefa.

Sobre 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.

Procedimento

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 207.

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 201.

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`.

Amostra e Tutorial de Serviços de Dados REST

Este tópico descreve como introduzir rapidamente o serviço de dados REST do WebSphere eXtreme Scale. São fornecidas instruções para o WebSphere Application Server versão 7.0, WebSphere Application Server Community Edition e Apache Tomcat.

Sobre Esta Tarefa

A amostra incluída possui código de origem e binários compilados para executar uma grade do eXtreme Scale particionada. Esta amostra demonstra como criar uma grade simples, modela os dados usando entidades do eXtreme Scale e fornece dois aplicativos cliente de linha de comandos que permitem incluir e consultar entidades usando Java ou C# (veja a Figura 1).

O cliente Java de amostra usa a API EntityManager Java do eXtreme Scale para persistir e consultar dados na grade. Esse cliente pode ser executado no Eclipse ou utilizando um script de linha de comandos. Observe que o cliente Java de amostra não demonstra o serviço de dados REST, mas permite a atualização de dados na grade, portanto, um navegador da Web ou outros clientes podem ler os dados. O cliente Java de amostra e o navegador da Web, conforme mostrado na Figura 1, ilustram clientes HTTP usando o serviço de dados REST e clientes Java do eXtreme Scale usando a mesma grade e dados do eXtreme Scale contidos aí.

O cliente C# Microsoft WCF Data Services se comunica com a grade do eXtreme Scale por meio do serviço de dados REST usando a estrutura .NET. O cliente WCF Data Services pode ser utilizado para atualizar e consultar a grade.

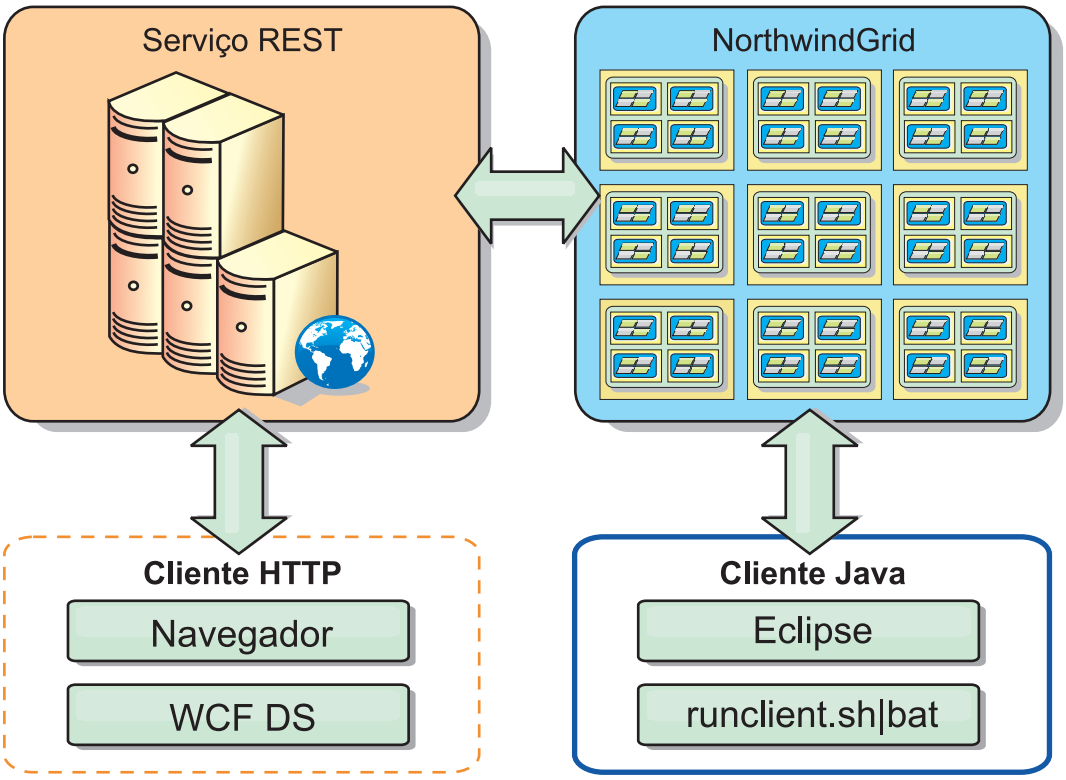


Figura 44. Topologia de Amostra de Introdução

Procedimento

1. Configure e inicie a grade do eXtreme Scale. Consulte “Ativando o Serviço de Dados REST” na página 217.
2. Configure e inicie o serviço de dados REST em um servidor da Web. Consulte “Configurando Servidores de Aplicativos para o Serviço de Dados REST” na página 226.

3. Execute um cliente para interagir com o serviço de dados REST. Duas opções estão disponíveis:
   a. Execute o cliente Java de amostra para preencher a grade com dados usando a API EntityManager e consulte os dados na grade usando um navegador da Web e o serviço de dados REST do eXtreme Scale. Consulte "Utilizando um Cliente Java com Serviço de Dados REST" na página 235.
   b. Execute o cliente WCF Data Services C# de amostra. Consulte "Cliente Visual Studio 2008 WCF com Serviço de Dados REST" na página 237.

Convenções de Diretório

Este tópico descreve muitos exemplos e a sintaxe da linha de comandos que deve fazer referência a diretórios especiais como *wxs_install_root* e *wxs_home*. Esses diretórios são definidos da seguinte maneira.

wxs_install_root
O diretório *wxs_install_root* é o diretório-raiz no qual arquivos do produto WebSphere eXtreme Scale são instalados. Este pode ser o diretório no qual o arquivo zip de teste é extraído ou o diretório no qual o produto eXtreme Scale é instalado.

- Exemplo ao extrair o teste:

  ```
  /opt/IBM/WebSphere/eXtremeScale
  ```

- Exemplo quando o eXtreme Scale é instalado em um diretório independente:

  ```
  /opt/IBM/eXtremeScale
  ```

- Exemplo quando o eXtreme Scale é integrado ao WebSphere Application Server:

  ```
  /opt/IBM/WebSphere/AppServer
  ```

wxs_home
O diretório *wxs_home* é o diretório-raiz de bibliotecas, amostras e componentes do produto WebSphere eXtreme Scale. É o mesmo que o diretório *wxs_install_root* quando o teste é extraído. Para instalações independentes, esse é o subdiretório de ObjectGrid dentro de *wxs_install_root*. Para instalações integradas ao WebSphere Application Server, esse diretório está no diretório `optionalLibraries/ObjectGrid` dentro de *wxs_install_root*.

- Exemplo ao extrair o teste:

  ```
  /opt/IBM/WebSphere/eXtremeScale
  ```

- Exemplo quando o eXtreme Scale é instalado em um diretório independente:

  ```
  /opt/IBM/eXtremeScale/ObjectGrid
  ```

- Exemplo quando o eXtreme Scale é integrado ao WebSphere Application Server:

  ```
  /opt/IBM/WebSphere/AppServer/optionalLibraries/ObjectGrid
  ```

was_root
O diretório *was_root* é o diretório-raiz de uma instalação do WebSphere Application Server:

```
/opt/IBM/WebSphere/AppServer
```

restservice_home
O diretório *restservice_home* é o diretório no qual as bibliotecas e amostras do serviço de dados REST do eXtreme Scale estão localizadas. Este diretório é denominado *restservice* e é um subdiretório no diretório *wxs_home*.

- Exemplo para implementações independentes:

  ```
  /opt/IBM/WebSphere/eXtremeScale/ObjectGrid/restservice
  ```

- Exemplo para implementações integradas do WebSphere Application Server:
  `/opt/IBM/WebSphere/AppServer/optionalLibraries/ObjectGrid/restservice`

tomcat_root
O *tomcat_root* é o diretório-raiz da instalação do Apache Tomcat.

`/opt/tomcat5.5`

wasce_root
O *wasce_root* é o diretório-raiz da instalação do WebSphere Application Server Community Edition.

`/opt/IBM/WebSphere/AppServerCE`

java_home
java_home é o diretório-raiz de uma instalação de Java Runtime Environment (JRE).

`/opt/IBM/WebSphere/eXtremeScale/java`

Ativando o Serviço de Dados REST

O serviço de dados REST pode representar metadados de entidade do WebSphere eXtreme Scale para representar cada entidade como um EntitySet.

Iniciando uma Grade do eXtreme Scale de Amostra

No geral, antes de iniciar o serviço de dados REST, inicie a grade do eXtreme Scale. As etapas a seguir iniciarão um único processo do serviço de catálogo do eXtreme Scale e dois processos do servidor de contêiner.

O WebSphere eXtreme Scale pode ser instalado usando três métodos diferentes:

- Instalação por tentativa
- Implementação independente
- Implementação integrada do WebSphere Application Server

Modelo de Dados Escalável no eXtreme Scale

A amostra Northwind da Microsoft utiliza a tabela Detalhes do Pedido para estabelecer uma associação muitos-para-muitos entre Pedidos e Produtos.

Object to relational mapping specifications (ORMs), como ADO.NET Entity Framework e Java Persistence API (JPA), podem mapear as tabelas e os relacionamentos utilizando entidades. No entanto, esta arquitetura não é escalada. Tudo deve estar localizado na mesma máquina ou em um cluster de máquinas caro para uma boa execução.

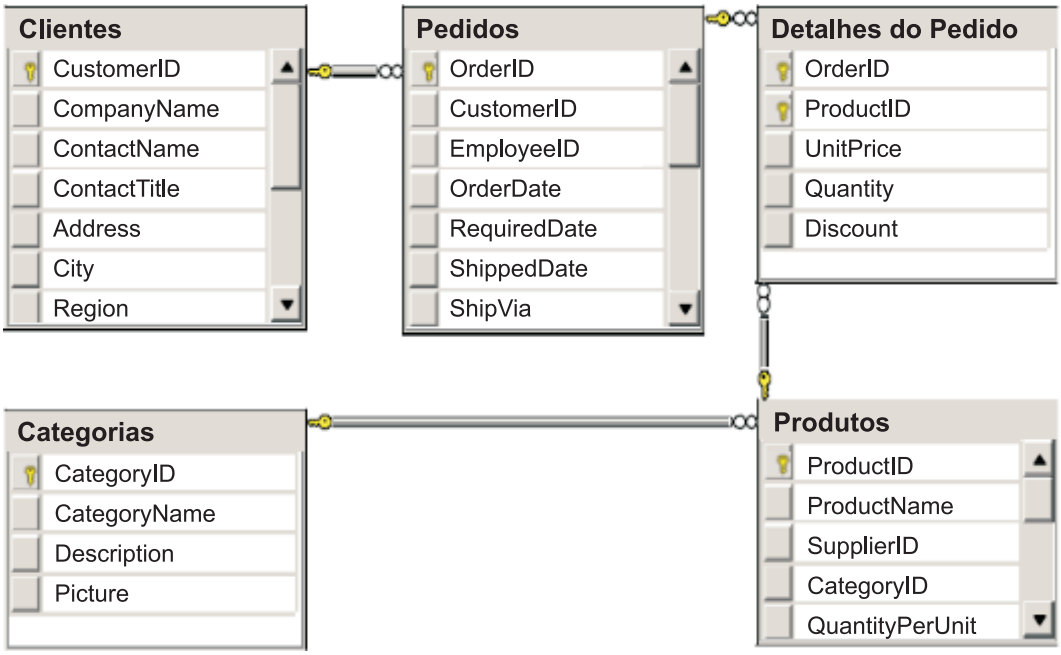


Figura 45. Diagrama do Esquema da Amostra Northwind do Microsoft SQL Server

Para criar uma versão escalável da amostra, as entidades devem ser modeladas de forma que cada entidade ou grupo de entidades relacionadas possa ser particionado com base em uma única chave. Ao criar partições em uma única chave, os pedidos podem ser espalhados entre vários servidores independentes. Para atingir essa configuração, as entidades foram divididas em duas árvores: a árvore Cliente e Ordem e a árvore Produto e Categoria. Neste modelo, cada árvore pode ser particionada de forma independente e, portanto, pode crescer a taxas diferentes, aumentando a escalabilidade.

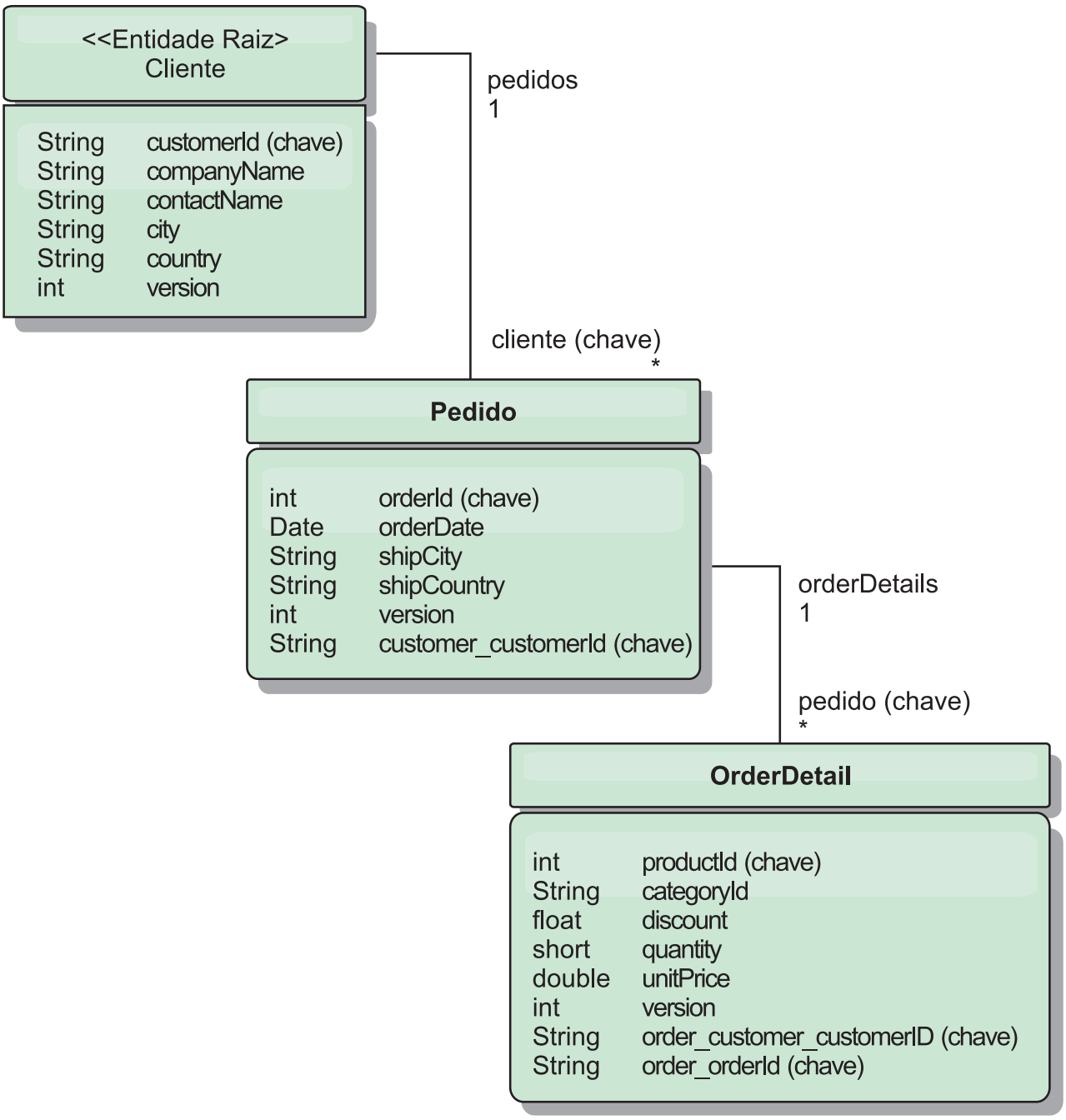


Figura 46. Diagrama do Esquema de Entidade Cliente e Ordem

Por exemplo, Order e Product têm números inteiros separados exclusivos como chaves. De fato, as tabelas Order e Product são realmente independentes uma da outra. Por exemplo, considere o efeito do tamanho de um catálogo (número de produtos que você vende) com o número total de pedidos. Intuitivamente, pode parecer que ter muitos produtos implica também em ter muitas ordens, mas este não é necessariamente o caso. Se isso fosse verdade, você poderia facilmente aumentar as vendas incluindo mais produtos em seu catálogo. Orders e Products têm suas próprias tabelas independentes. É possível estender mais este conceito para que as ordens e os produtos tenham, cada um, suas próprias grades de dados separadas. Com grades independentes, é possível controlar o número de partições e servidores, além do tamanho de cada grade separadamente, para que seu

aplicativo possa escalar. Se você dobrar o tamanho de seu catálogo, será necessário dobrar a grade de produtos, mas a grade de ordens pode permanecer inalterada. O contrário é verdade para um pico de pedidos ou pico de pedidos esperado.

No esquema, um Customer tem zero ou mais Orders, e um Order tem itens de linha (OrderDetail), cada um com um produto específico. Um Produto é identificado pelo ID (a chave do Produto) em cada OrderDetail. Uma única grade armazena Clientes, Ordens e OrderDetails com o Cliente como a entidade raiz da grade. É possível recuperar os Clientes pelo ID, mas é necessário obter as Ordens que começam com o ID do Cliente. Então, o ID do cliente é incluído na Ordem como parte de sua chave. Da mesma forma, o ID do cliente e o ID da ordem fazem parte do ID do OrderDetail.

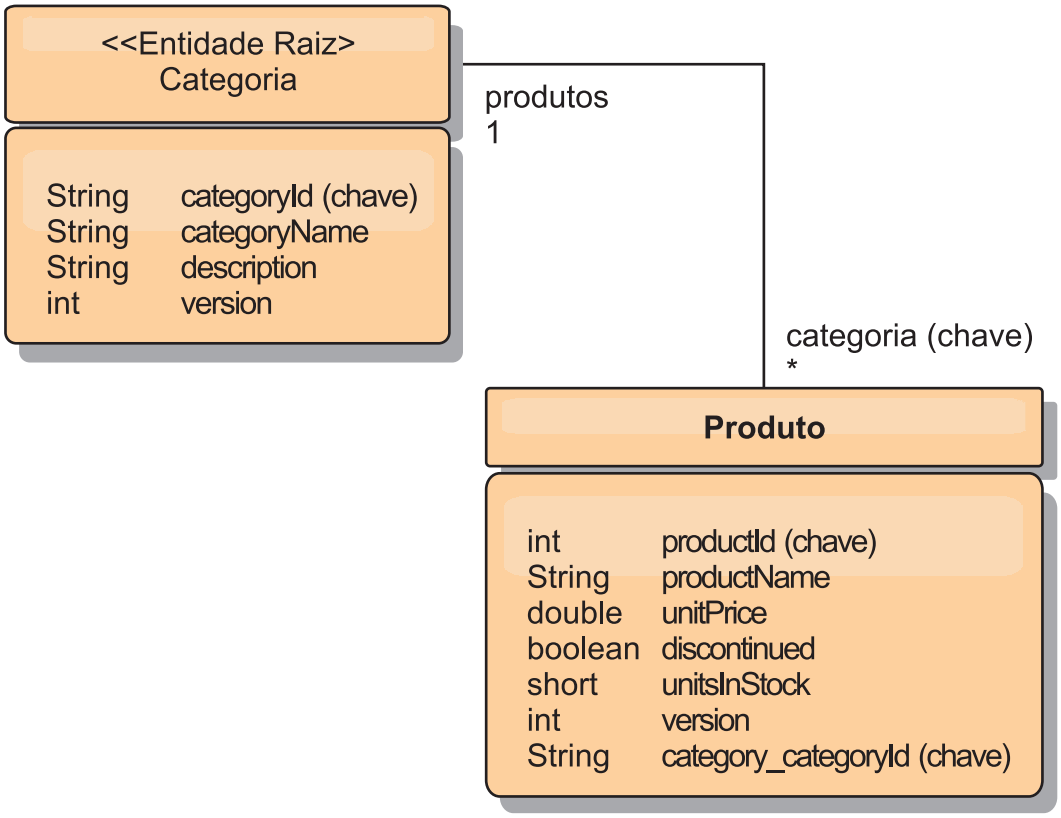


Figura 47. Diagrama do Esquema de Entidade Categoria e Produto

No esquema Category e Product, Category é a raiz do esquema. Com este esquema, os clientes podem consultar os produtos por categoria. Consulte "Recuperando e Atualizando Dados com o REST" para obter detalhes adicionais sobre as principais associações e sua importância.

Recuperando e Atualizando Dados com o REST

O protocolo OData requer que todas as entidades possam ser endereçadas por sua forma canônica. Isso significa que cada entidade deve incluir a chave da entidade raiz particionada (o esquema raiz).

A seguir está um exemplo de como utilizar a associação de uma entidade raiz para endereçar um filho em:

```
/Customer('ACME')/order(100)
```

No WCF Data Services, a entidade-filha deve ser diretamente endereçável, o que significa que a chave na raiz do esquema deve fazer parte da chave da filha: /Order(customer_customerId='ACME', orderId=100). Isso é conseguido por meio da criação de uma associação com a entidade raiz, em que a associação um-para-um ou muitos-para-um com a entidade raiz também é identificada como uma chave. Quando entidades são incluídas como parte da chave, os atributos da entidade pai são expostos como propriedades-chave.

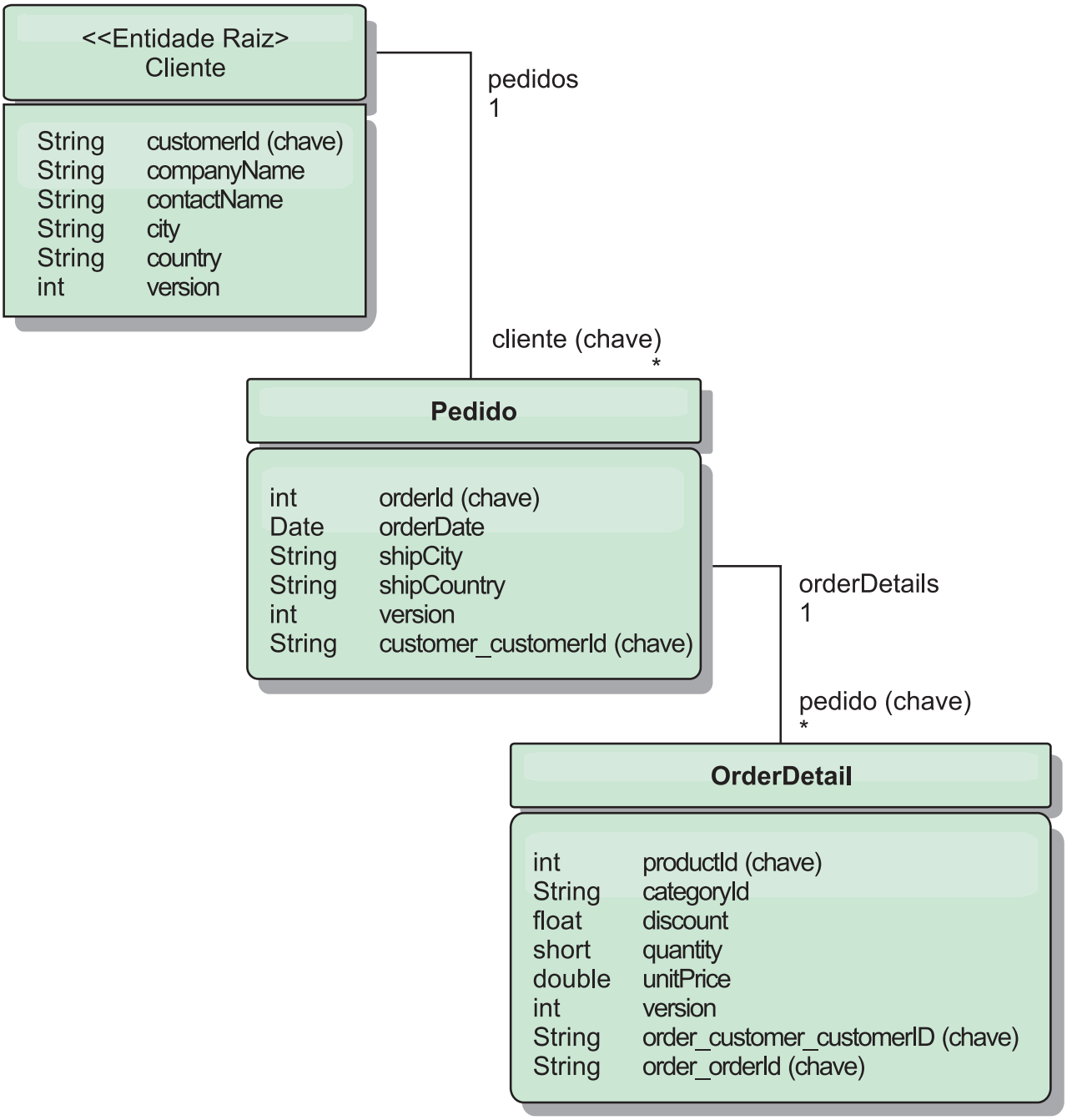


Figura 48. Diagrama do Esquema de Entidade Cliente e Ordem

O diagrama do esquema de entidade Cliente/Ordem ilustra como cada entidade é particionada usando o Cliente. A entidade Order inclui Customer como parte de sua chave e é, portanto, diretamente acessível. O serviço de dados REST expõe

todas as associações chave como propriedades individuais: Order tem customer_customerId e OrderDetail tem order_customer_customerId e order_orderId.

Utilizando a API EntityManager, é possível localizar Order utilizando o id de Customer e Order:

```
transaction.begin();
// Look-up the Order using the Customer.  We only include the Id
// in the Customer class when building the OrderId key instance.
Order order = (Order) em.find(Order.class,
    new OrderId(100, new Customer('ACME')));
...
transaction.commit();
```

Ao utilizar o serviço de dados REST, Order pode ser recuperada com qualquer uma das seguintes URLs:

- `/Order(orderId=100, customer_customerId='ACME')`
- `/Customer('ACME')/orders?$filter=orderId eq 100`

A chave do cliente é endereçada utilizando o nome do atributo da entidade Customer, um caractere sublinhado e o nome do atributo do id de Customer: customer_customerId.

Uma entidade também pode incluir uma entidade não raiz como parte de sua chave se todos os ancestrais para a entidade não raiz tiverem associações de chave com a raiz. Neste exemplo, OrderDetail tem uma associação de chave com Order, e Order tem uma associação de chave com a entidade Customer raiz. Utilizando a API EntityManager:

```
transaction.begin();
// Construct an OrderDetailId key instance.  It includes
// The Order and Customer with only the keys set.
Customer customerACME = new Customer("ACME");
Order order100 = new Order(100, customerACME);
OrderDetailId orderDetailKey =
    new OrderDetailId(order100, "COMP");
OrderDetail orderDetail = (OrderDetail)
    em.find(OrderDetail.class, orderDetailKey);
...
```

O serviço de dados REST permite endereçar OrderDetail diretamente:

```
/OrderDetail(productId=500,
order_customer_customerId='ACME', order_orderId =100)
```

A associação da entidade OrderDetail com a entidade Product foi quebrada para permitir o particionamento de Orders e Product Inventory independentemente. A entidade OrderDetail armazena o ID do produto e categoria de um relacionamento sólido. Ao desacoplar os dois esquemas de entidade, apenas uma partição é acessada por vez.

O esquema Categoria e Produto ilustrado no diagrama mostra que a entidade-raiz é Categoria e cada Produto tem uma associação com uma entidade Categoria. A entidade Category está incluída na identidade de Product. O serviço de dados REST expõe uma propriedade-chave: category_categoryId, que permite o endereçamento direto de Product.

Como Categoria é a entidade-raiz, em um ambiente particionado, a Categoria deve ser conhecida para localizar o Produto. Utilizando a API EntityManager, a transação deve ser retida na entidade Category antes de localizar Product.

Utilizando a API EntityManager:

```
transaction.begin();
// Create the Category root entity with only the key.  This
// allows us to construct a ProductId without needing to find
// The Category first.  The transaction is now pinned to the
// partition where Category "COMP" is stored.
Category cat = new Category("COMP");
Product product = (Product) em.find(Product.class,
    new ProductId(500, cat));
...
```

O serviço de dados REST permite endereçar Product diretamente:

```
/Product(productId=500,
category_categoryId='COMP')
```

Iniciando uma Grade Independente para Serviços de Dados REST

Siga estas etapas para iniciar a grade de amostra do serviço REST do WebSphere eXtreme Scale para uma implementação do eXtreme Scale independente.

Antes de Iniciar

Instale o produto WebSphere eXtreme Scale de Teste ou completo:

- Instale a versão independente do produto WebSphere eXtreme Scale 7.1 e aplique as correções subsequentes.
- Faça download e extraia o WebSphere eXtreme Scale Versão 7.1 de teste, que inclui o serviço de dados REST do WebSphere eXtreme Scale.

Sobre Esta Tarefa

Inicie a grade de amostra do WebSphere eXtreme Scale.

Procedimento

1. Inicie o processo do serviço de catálogo. Abra uma janela do terminal ou da linha de comandos e configure a variável de ambiente JAVA_HOME:
   - Linux UNIX `export JAVA_HOME=java_home`
   - Windows `set JAVA_HOME=java_home`
2. `cd restservice_home/gettingstarted`
3. Inicie o processo do serviço de catálogo. Para iniciar o serviço *sem* a segurança do eXtreme Scale, use os comandos a seguir.
   - Linux UNIX `./runcat.sh`
   - Windows `runcat.bat`

   Para iniciar o serviço com a segurança do eXtreme Scale, use os comandos a seguir.
   - Linux UNIX `./runcat_secure.sh`
   - Windows `runcat_secure.bat`
4. Inicie dois processos do servidor de contêiner. Abra uma janela do terminal ou de linha de comandos e configure a variável de ambiente JAVA_HOME:
   - Linux UNIX `export JAVA_HOME=java_home`
   - Windows `set JAVA_HOME=java_home`
5. `cd restservice_home/gettingstarted`

6. Inicie um processo do servidor de contêiner:

   Para iniciar o servidor sem a segurança do eXtreme Scale, use os comandos a seguir:

   - Linux UNIX `./runcontainer.sh container0`
   - Windows `runcontainer.bat container0`

   Para iniciar o servidor com a segurança do eXtreme Scale, use os comandos a seguir.

   - Linux UNIX `./runcontainer_secure.sh container0`
   - Windows `runcontainer_secure.bat container0`
7. Abra uma janela do terminal ou de linha de comandos e configure a variável de ambiente JAVA_HOME:
   - Linux UNIX `export JAVA_HOME=java_home`
   - Windows `set JAVA_HOME=java_home`
8. `cd restservice_home/gettingstarted`
9. Inicie um segundo processo do servidor de contêiner.

   Para iniciar o servidor sem a segurança do eXtreme Scale, use os comandos a seguir.

   - Linux UNIX `./runcontainer.sh container1`
   - Windows `runcontainer.bat container1`

   Para iniciar o servidor com a segurança do eXtreme Scale, use os comandos a seguir.

   - Linux UNIX `./runcontainer_secure.sh container1`
   - Windows `runcontainer_secure.bat container1`

Resultados

Aguarde até que os contêineres do eXtreme Scale estejam prontos antes de continuar com as próximas etapas. Os servidores de contêiner estão prontos quando a seguinte mensagem é exibida na janela do terminal:

`CWOBJ1001I: O Servidor de ObjectGrid container_name está pronto para processar pedidos.`

Em que *container_name* é o nome do contêiner que foi iniciado.

Iniciando uma Grade para os Serviços de Dados REST no WebSphere Application Server

Siga estas etapas para iniciar uma grade de dados de amostra do serviço REST do WebSphere eXtreme Scale independente para uma implementação do WebSphere eXtreme Scale que está integrada ao WebSphere Application Server. Embora o WebSphere eXtreme Scale esteja integrado ao WebSphere Application Server, estas etapas iniciam um processo e um contêiner do serviço de catálogo do WebSphere eXtreme Scale independente.

Antes de Iniciar

Instale o produto WebSphere eXtreme Scale Versão 7.1 em um diretório de instalação do WebSphere Application Server Versão 7.0.0.5 ou posterior (com a segurança desativada), aumente pelo menos um perfil do Servidor de Aplicativos.

Sobre Esta Tarefa

Inicie a grade de amostra do WebSphere eXtreme Scale.

Procedimento

1. Inicie o processo do serviço de catálogo. Abra uma janela do terminal ou da linha de comandos e configure a variável de ambiente JAVA_HOME:
   - Linux UNIX `export JAVA_HOME=java_home`
   - Windows `set JAVA_HOME=java_home`

   `cd restservice_home/gettingstarted`
2. Inicie o processo do serviço de catálogo.

   Para iniciar o servidor sem a segurança do eXtreme Scale, use os comandos a seguir.
   - Linux UNIX `./runcat.sh`
   - Windows `runcat.bat`

   Para iniciar o servidor com a segurança do eXtreme Scale, use os comandos a seguir.
   - Linux UNIX `./runcat_secure.sh`
   - Windows `runcat_secure.bat`
3. Inicie dois processos do servidor de contêiner. Abra uma janela do terminal ou de linha de comandos e configure a variável de ambiente JAVA_HOME:
   - Linux UNIX `export JAVA_HOME=java_home`
   - Windows `set JAVA_HOME=java_home`
4. Inicie um processo do servidor de contêiner.

   Para iniciar o servidor sem a segurança do eXtreme Scale, use os comandos a seguir.
   a. Abra uma janela de linha de comandos.
   b. `cd restservice_home/gettingstarted`
   c. Para iniciar o servidor *sem* a segurança do eXtreme Scale, use os comandos a seguir.
      - Linux UNIX `./runcontainer.sh container0`
      - Windows `runcontainer.bat container0`
   d. Para iniciar o servidor com a segurança do eXtreme Scale, use os comandos a seguir.
      - Linux UNIX `./runcontainer_secure.sh container0`
      - Windows `runcontainer_secure.bat container0`
5. Inicie um segundo processo do servidor de contêiner.
   a. Abra uma janela de linha de comandos.
   b. `cd restservice_home/gettingstarted`
   c. Para iniciar o servidor *sem* a segurança do eXtreme Scale, use os comandos a seguir.
      - Linux UNIX `./runcontainer.sh container1`
      - Windows `runcontainer.bat container1`

d. Para iniciar o servidor *com* a segurança do eXtreme Scale, use os comandos a seguir.
   - Linux UNIX `./runcontainer_secure.sh container1`
   - Windows `runcontainer_secure.bat container1`

Resultados

Aguarde até que os servidores de contêiner estejam prontos antes de continuar com as próximas etapas. Os servidores de contêiner estarão prontos quando a seguinte mensagem for exibida:

`CWOBJ1001I: O Servidor de ObjectGrid container_name está pronto para processar pedidos.`

Em que *container_name* é o nome do contêiner que foi iniciado na etapa anterior.

Configurando Servidores de Aplicativos para o Serviço de Dados REST

Iniciando Serviços de Dados REST no WebSphere Application Server Versão 7.0

Este tópico descreve como configurar e iniciar o serviço de dados REST do eXtreme Scale usando o WebSphere Application Server Versão 7.0.

Antes de Iniciar

Verifique se a grade do eXtreme Scale de amostra foi iniciada. Consulte "Ativando o Serviço de Dados REST" na página 217 para obter detalhes sobre como iniciar a grade.

Procedimento

1. Faça download e instale o WebSphere Application Server Versão 7.0 for Developers.

   Restrição: Não ative a segurança.
2. Faça download e instale o WebSphere Application Server Versão 7.0 Fix Pack 5 ou posterior.
3. Inclua o JAR de tempo de execução do cliente do WebSphere eXtreme Scale, o arquivo `wsogclient.jar` e o JAR ou diretório de configuração do serviço de dados REST no caminho de classe do servidor de aplicativos:
   a. Abra o console administrativo do WebSphere Application Server.
   b. Navegue para **Ambiente** → **Bibliotecas Compartilhadas**
   c. Clique em **Novo**
   d. Inclua as seguintes entradas nos campos:
      1) Nome: extremescale_client_v71
      2) Caminho da classe: `wxs_home/lib/wsogclient.jar`
   e. Clique em **OK**
   f. Clique em **Novo**
   g. Inclua as seguintes entradas nos campos apropriados:
      1) Nome: `extremescale_gettingstarted_config`
      2) Caminho de classe:

- `restservice_home/gettingstarted/restclient/bin`
- `restservice_home/gettingstarted/common/bin`

Lembre-se: Inclua cada caminho em uma linha separada.

h. Clique em **OK**

i. Salve as mudanças na configuração principal

4. Instale o arquivo EAR do serviço de dados REST, `wxsrestservice.ear`, no WebSphere Application Server usando o console administrativo:
   a. Abra o console administrativo do WebSphere Application Server
   b. Navegue para **Aplicativos → Novo Aplicativo**
   c. Navegue para `restservice_home/lib/wxsrestservice.ear`, selecione o arquivo e clique em **Avançar**
   d. Escolha as opções de instalação detalhadas e clique em **Avançar**
   e. Na tela de avisos de segurança do aplicativo, clique em **Continuar**
   f. Escolha as opções de instalação padrão e clique em **Avançar**
   g. Escolha um servidor para o qual mapear o aplicativo e clique em **Avançar**
   h. Na página de recarregamento de JSP, utilize os padrões e clique em **Avançar**
   i. Na página de bibliotecas compartilhadas, mapeie o módulo `wxsrestservice.war` para as seguintes bibliotecas compartilhadas definidas:
      - extremescale_client_v71
      - extremescale_ gettingstarted _config
   j. Na página de relacionamentos da biblioteca compartilhada do mapa, utilize os padrões e clique em **Avançar**
   k. Na página hosts virtuais do mapa, clique nos padrões e em **Avançar**
   l. Na página de raízes de contexto do mapa, configure a raiz de contexto para: `wxsrestservice`.
   m. Na tela Resumo, clique em **Concluir** para concluir a instalação
   n. Salve as mudanças na configuração principal
5. Inicie o servidor de aplicativos e o aplicativo de serviço de dados REST do eXtreme Scale wxsrestservice. Após o aplicativo ser iniciado, revise o arquivo `SystemOut.log` para o servidor de aplicativos e verifique se a seguinte mensagem é exibida: `CWOBJ4000I: O serviço de dados REST do WebSphere eXtreme Scale foi iniciado.`
6. Verifique se o serviço de dados REST está funcionando
   a. Abra um navegador e navegue para: `http://localhost:9080/wxsrestservice/restservice/NorthwindGrid`. O documento de serviço para NorthwindGrid é exibido.
   b. Navegue para: `http://localhost:9080/wxsrestservice/restservice/NorthwindGrid/$metadata`. O documento Entity Model Data Extensions (EDMX) é exibido.
7. Para parar os processos da grade, use `CTRL+C` na respectiva janela de comando.

Iniciando Serviços de Dados REST com o WebSphere eXtreme Scale integrado ao WebSphere Application Server 7.0

Este tópico descreve como configurar e iniciar o serviço de dados REST do eXtreme Scale usando o WebSphere Application Server versão 7.0 que foi integrado e aumentado com o WebSphere eXtreme Scale.

Antes de Iniciar

Verifique se a grade do eXtreme Scale independente de amostra foi iniciada. Consulte “Ativando o Serviço de Dados REST” na página 217 para obter detalhes sobre como iniciar a grade.

Sobre Esta Tarefa

Para introduzir o serviço de dados REST do WebSphere eXtreme Scale usando o WebSphere Application Server, siga estas etapas:

Procedimento

1. Inclua o JAR de configuração de amostra do serviço de dados REST do WebSphere eXtreme Scale no caminho de classe:
   a. Abra o Console Administrativo do WebSphere
   b. Navegue para Ambiente -> Bibliotecas Compartilhadas
   c. Clique em Novo
   d. Inclua as seguintes entradas nos campos apropriados:
      1) Nome: extremescale_gettingstarted_config
      2) Caminho de Classe
         - `restservice_home/gettingstarted/restclient/bin`
         - `restservice_home/gettingstarted/common/bin`

         Lembre-se: Cada caminho deve aparecer em uma linha diferente.
   e. Clique em **OK**
   f. Salve as mudanças na configuração principal
2. Instale o arquivo EAR do serviço de dados REST, wxsrestservice.ear, no WebSphere Application Server utilizando o console administrativo do WebSphere:
   a. Abra o console administrativo do WebSphere
   b. Navegue para Aplicativos -> Novo Aplicativo
   c. Navegue para o arquivo `restservice_home/lib/wxsrestservice.ear` no sistema de arquivos. Selecione o arquivo e clique em **Avançar**.
   d. Escolha as opções de instalação detalhadas e clique em **Avançar**.
   e. Na tela de avisos de segurança do aplicativo, clique em **Continuar**.
   f. Escolha as opções de instalação padrão e clique em **Avançar**.
   g. Escolha um servidor para o qual mapear o módulo wxsrestservice.war e clique em **Avançar**.
   h. Na página de recarregamento de JSP, utilize os padrões e clique em **Avançar**.
   i. Na página bibliotecas compartilhadas, mapeie o módulo "wxsrestservice.war" para as seguintes bibliotecas compartilhadas que foram definidas durante a etapa 1: `extremescale_ gettingstarted _config`
   j. Na página de relacionamentos da biblioteca compartilhada do mapa, utilize os padrões e clique em **Avançar**.
   k. Na página hosts virtuais do mapa, clique nos padrões e em **Avançar**.
   l. Na página de raízes de contexto do mapa, configure a raiz de contexto como: wxsrestservice.
   m. Na tela Resumo, clique em **Concluir** para concluir a instalação.

n. Salve as alterações na configuração principal.

3. Se a grade do eXtreme Scale tiver sido iniciada com a segurança do eXtreme Scale ativada, configure a seguinte propriedade no arquivo `restservice_home/gettingstarted/restclient/bin/wxsRestService.properties`.

```
ogClientPropertyFile=restservice_home/gettingstarted/security/security.ogclient.properties
```

4. Inicie o servidor de aplicativos e o aplicativo do serviço de dados REST do eXtreme Scale "wxsrestservice".

   Após o aplicativo ser iniciado, revise o SystemOut.log para o servidor de aplicativos e verifique se a seguinte mensagem aparece: `CWOBJ4000I: O serviço de dados REST do WebSphere eXtreme Scale foi iniciado.`

5. Verifique se o serviço de dados REST está funcionando:
   a. Abra um navegador e navegue para:

      http://localhost:9080/wxsrestservice/restservice/NorthwindGrid

      O documento de serviço para NorthwindGrid é exibido.

   b. Navegue para:

      http://localhost:9080/wxsrestservice/restservice/NorthwindGrid/$metadata

      O documento Entity Model Data Extensions (EDMX) é exibido

6. Para parar os processos da grade, utilize CTRL+C na respectiva janela de comando para parar o processo.

Iniciando o Serviço de Dados REST no WebSphere Application Server Community Edition

Este tópico descreve como configurar e iniciar o serviço de dados REST do eXtreme Scale usando o WebSphere Application Server Community Edition.

Antes de Iniciar

Verifique se a grade de dados de amostra foi iniciada. Consulte “Ativando o Serviço de Dados REST” na página 217 para obter detalhes sobre como iniciar a grade.

Procedimento

1. Faça download e instale o WebSphere Application Server Community Edition Versão 2.1.1.3 ou posterior para `wasce_root`, como: `/opt/IBM/wasce`
2. Inicie o servidor WebSphere Application Server Community Edition executando o seguinte comando:
   - Linux UNIX `wasce_root/bin/startup.sh`
   - Windows `wasce_root/bin/startup.bat`
3. Se a grade do eXtreme Scale tiver sido iniciada com a segurança do eXtreme Scale ativada, configure as seguintes propriedades no arquivo `restservice_home/gettingstarted/restclient/bin/wxsRestService.properties`.

```
ogClientPropertyFile=restservice_home/gettingstarted/security/security.ogclient.properties
loginType=none
```

4. Instale o serviço de dados REST do eXtreme Scale e a amostra fornecida no servidor WebSphere Application Server Community Edition:
   a. Inclua o JAR de tempo de execução do cliente ObjectGrid ao repositório WebSphere Application Server Community Edition:

1) Abra o console administrativo WebSphere Application Server Community Edition e efetue login.

   Dica: A URL padrão é: `http://localhost:8080/console`. O ID de usuário padrão é `system` e a senha é `manager`.
2) Clique no **Repositório**, na pasta Serviços.
3) Na seção **Incluir Archive no Repositório**, preencha o seguinte nas caixas de texto de entrada:

Tabela 16. Archive para Repositório

Caixa de texto	Valor
File	wxs_home/lib/ogclient.jar
Grupo	com.ibm.websphere.xs
Artefato	ogclient
Versão	7.0
Tipo	jar

4) Clique no botão Instalar.

   Dica: Consulte a seguinte nota técnica para obter detalhes sobre os diferentes métodos de dependências de classe e de biblioteca de configuração: Especificando dependências externas para aplicativos em execução no WebSphere Application Server Community Edition.

b. Implemente o módulo do serviço de dados REST, que é o arquivo `wxsrestservice.war`, para o servidor WebSphere Application Server Community Edition.
   1) Edite o arquivo XML de implementação `restservice_home/gettingstarted/wasce/geronimo-web.xml` de amostra para incluir dependências do caminho nos diretórios do caminho de classe de amostra de introdução:

      Altere os caminhos classesDirs para os dois GBeans do cliente de introdução:
      - O caminho "classesDirs" para o GBean GettingStarted_Client_SharedLib deve ser configurado para: `restservice_home/gettingstarted/restclient/bin`
      - O caminho "classesDirs" para o GBean GettingStarted_Common_SharedLib deve ser configurado para: `restservice_home/gettingstarted/common/bin`
   2) Abra o console administrativo do WebSphere Application Server Community Edition e efetue login.

      Dica: A URL padrão é: `http://localhost:8080/console`. O ID de usuário padrão é `system` e a senha é `manager`.
   3) Clique em **Implementar Novo**.
   4) Na página **Instalar Novos Aplicativos**, insira os seguintes valores nas caixas de texto:

Tabela 17. Valores de Instalação

Caixa de texto	Valor
Archive	`restservice_home/lib/wxsrestservice.war`

Tabela 17. Valores de Instalação (continuação)

Caixa de texto	Valor
Plano	`restservice_home/gettingstarted/wasce/geronimo-web.xml`

5) Clique no botão Instalar.

   A página do console deve indicar que o aplicativo foi instalado e iniciado com sucesso.

6) Examine o log de saída do sistema ou o console do WebSphere Application Server Community Edition para verificar se o serviço de dados REST foi iniciado com êxito verificando se a seguinte mensagem está presente:

   ```
   CWOBJ4000I: O serviço de dados REST do WebSphere eXtreme Scale
   foi iniciado.
   ```

5. Verifique se o serviço de dados REST está funcionando:
   a. Abra o seguinte link em uma janela do navegador: `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid`. O documento de serviço para a grade NorthwindGrid é exibido.
   b. Abra o seguinte link em uma janela do navegador: `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/$metadata`. O documento Entity Model Data Extensions (EDMX) é exibido.
6. Para parar os processos da grade, use `CTRL+C` na respectiva janela de comando para parar o processo.
7. Para parar o WebSphere Application Server Community Edition, use o seguinte comando:
   - UNIX Linux `wasce_root/bin/shutdown.sh`
   - Windows `wasce_root\bin\shutdown.bat`

   Dica: O ID de usuário padrão é `system` e a senha é `manager`. Se você estiver usando uma porta customizada, use a opção `-port`.

Iniciando Serviços de Dados REST no Apache Tomcat

Este tópico descreve como configurar e iniciar o serviço de dados REST do eXtreme Scale usando o Apache Tomcat, versão 5.5 ou posterior.

Antes de Iniciar

Verifique se a grade do eXtreme Scale de amostra foi iniciada. Consulte “Ativando o Serviço de Dados REST” na página 217 para obter detalhes sobre como iniciar a grade.

Procedimento

1. Faça download e instale o Apache Tomcat Versão 5.5 ou posterior em tomcat_root. Por exemplo: `/opt/tomcat`
2. Instale o serviço de dados REST do eXtreme Scale e a amostra fornecida no servidor Tomcat da seguinte maneira:
   a. Se estiver usando um Sun JRE ou JDK, você deve instalar o IBM ORB no Tomcat:
      - Para Tomcat versão 5.5

        Copie todos os arquivos JAR de:

        `wxs_home/lib/endorsed`

para

`tomcat_root/common/endorsed`

- Para Tomcat versão 6.0
  1) Crie um diretório "endorsed"
     - UNIX Linux `mkdir tomcat_root/endorsed`
     - Windows `md tomcat_root/endorsed`
  2) Copie todos os arquivos JAR de:

     `wxs_home/lib/endorsed`

     para

     `tomcat_root/endorsed`

b. Implemente o módulo de serviço de dados REST: wxsrestservice.war para o servidor Tomcat.

   Copie o arquivo wxsrestservice.war de:

   `restservice_home/lib`

   para:

   `tomcat_root/webapps`

c. Inclua o JAR de tempo de execução do cliente ObjectGrid e o JAR do aplicativo no caminho de classe compartilhado no Tomcat:
   1) Edite o arquivo `tomcat_root/conf/catalina.properties`
   2) Anexe os seguintes nomes de caminho no final da propriedade shared.loader no formato de lista separada por vírgula:
      - wxs_home/lib/ogclient.jar
      - restservice_home/gettingstarted/restclient/bin
      - restservice_home/gettingstarted/common/bin

      Importante: O separador de caminho deve ser uma **barra**.

3. Se a grade do eXtreme Scale tiver sido iniciada com a segurança do eXtreme Scale ativada, configure as seguintes propriedades no arquivo `restservice_home/gettingstarted/restclient/bin/wxsRestService.properties`.

```
ogClientPropertyFile=restservice_home/gettingstarted/security/security.ogclient.properties
loginType=none
```

4. Inicie o servidor Tomcat com o serviço de dados REST:
   - Se estiver utilizando Tomcat 5.5 no UNIX® ou Windows®, ou Tomcat 6.0 no UNIX:
     a. `cd tomcat_root/bin`
     b. Inicie o servidor:
        - UNIX Linux `./catalina.sh run`
        - Windows `catalina.bat run`

     c. O console exibirá logs do Apache Tomcat. Quando o serviço de dados REST for iniciado com sucesso, a seguinte mensagem será exibida no console administrativo:

        `CWOBJ4000I: O serviço de dados REST do WebSphere eXtreme Scale foi iniciado.`
   - Se estiver utilizando Tomcat 6.0 no Windows:
     a. `cd tomcat_root/bin`

b. Inicie a ferramenta de configuração do Apache Tomcat 6 com o seguinte comando: `tomcat6w.exe`
c. Clique no botão Iniciar na janela de propriedades do Apache Tomcat 6 para iniciar o servidor Tomcat.
d. Revise os seguintes logs para verificar se o servidor Tomcat foi iniciado com sucesso:
   - `tomcat_root/bin/catalina.log`

     Exibe o status do mecanismo do servidor Tomcat
   - `tomcat_root/bin/stdout.log`

     Exibe o log de saída do sistema.
e. Quando o serviço de dados REST for iniciado com sucesso, a seguinte mensagem será exibida no log de saída do sistema: `CWOBJ4000I: O serviço de dados REST do WebSphere eXtreme Scale foi iniciado.`

5. Verifique se o serviço de dados REST está funcionando:
   a. Abra um navegador e navegue para:

      http://localhost:8080/wxsrestservice/restservice/NorthwindGrid

      O documento de serviço para NorthwindGrid é exibido.
   b. Navegue para:

      http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/$metadata

      O documento Entity Model Data Extensions (EDMX) é exibido.
6. Para parar os processos da grade, utilize CTRL+C na respectiva janela de comando.
7. Para parar o Tomcat, utilize CTRL +C na janela onde o iniciou.

Utilizando um Navegador com Serviços de Dados REST

O serviço de dados REST do eXtreme Scale cria feeds ATOM por padrão ao usar um navegador da Web. O formato feed ATOM pode não ser compatível com navegadores antigos ou pode ser interpretado de modo que os dados não possam ser visualizados como XML. Os tópicos a seguir fornecem detalhes sobre como configurar o Internet Explorer Versão 8 e o Firefox Versão 3 para exibir os feeds ATOM e o XML dentro do navegador.

Sobre Esta Tarefa

O serviço de dados REST do eXtreme Scale cria feeds ATOM por padrão ao usar um navegador da Web. O formato feed ATOM pode não ser compatível com navegadores antigos ou pode ser interpretado de modo que os dados não possam ser visualizados como XML. Para os navegadores antigos, será solicitado que você salve os arquivos no disco. Após os arquivos serem transferidos por download, utilize seu leitor XML favorito para examinar os arquivos. O XML gerado não é formatado para ser exibido, portanto, tudo será impresso em uma linha. A maioria dos programas de leitura XML, como Eclipse, suporta reformatação de XML em um formato legível.

Para navegadores modernos, como Microsoft Internet Explorer Versão 8 e Firefox Versão 3, os arquivos XML ATOM podem ser exibidos nativamente no navegador. Os tópicos a seguir fornecem detalhes sobre como configurar o Internet Explorer Versão 8 e o Firefox Versão 3 para exibir os feeds ATOM e o XML dentro do navegador.

Procedimento

Configurando o Internet Explorer Versão 8

- Para permitir que o Internet Explorer leia os feeds ATOM que o serviço de dados REST gera, use as seguintes etapas:
  1. Clique em **Ferramentas** → **Opções da Internet**
  2. Selecione a guia **Conteúdo**
  3. Clique no botão **Configurações** na seção **Feeds e Web Slices**
  4. Desmarque a caixa: "Ativar visualização de leitura de feed"
  5. Clique em **OK** para retornar ao navegador.
  6. Reinicie o Internet Explorer.

Configurando o Firefox Versão 3

- O Firefox não exibe automaticamente páginas com tipo de conteúdo: application/atom+xml. Na primeira vez que uma página é exibida, o Firefox solicita o salvamento do arquivo. Para exibir a página, abra o próprio arquivo com o Firefox da seguinte forma:
  1. Na caixa de diálogo do seletor de aplicativo, selecione o botão de opções "Abrir com" e clique no botão **Navegar**.
  2. Navegue para o diretório de instalação do Firefox. Por exemplo: `C:\Program Files\Mozilla Firefox`
  3. Selecione `firefox.exe` e pressione o botão **OK**.
  4. Marque “Fazer isto automaticamente para arquivos como este...” .
  5. Clique no botão **OK**.
  6. Em seguida, o Firefox exibe a página XML ATOM em uma nova janela do navegador ou guia
- O Firefox renderiza automaticamente feeds ATOM em formato legível. Entretanto, os feeds que o serviço de dados REST cria incluem XML. O Firefox não pode exibir XML, a menos que você desative o renderizador de feed. Ao contrário do Internet Explorer, no Firefox, o plug-in de renderização de feed ATOM deve ser editado explicitamente. Para configurar o Firefox para ler ATOM feeds como arquivos XML, siga estas etapas:
  1. Abra o seguinte arquivo em um editor de texto: `<firefoxInstallRoot>\components\FeedConverter.js`. No caminho, `<firefoxInstallRoot>` é o diretório-raiz onde o Firefox está instalado.

     Para sistemas operacionais Windows, o diretório padrão é: `C:\Program Files\Mozilla Firefox`.
  2. Procure o fragmento semelhante ao seguinte:

     ```
     // show the feed page if it wasn't sniffed and we have a document,
     // or we have a document, title, and link or id
     if (result.doc && (!this._sniffed ||
         (result.doc.title && (result.doc.link ||
     result.doc.id)))) {
     ```
  3. Comente as duas linhas que começam com `if` e `result` colocando `//` (duas barras) na frente delas.
  4. Anexe a seguinte instrução ao fragmento: `if(0) {`.
  5. O texto resultante deve ser como o seguinte:

```
// show the feed page if it wasn't sniffed and we have a document,
// or we have a document, title, and link or id
//if (result.doc && (!this._sniffed ||
//    (result.doc.title && (result.doc.link ||
result.doc.id)))) {
if(0) {
```

6. Salve o arquivo.
7. Reinicie o Firefox
8. Agora o Firefox pode exibir automaticamente todos os feeds no navegador.

- Teste sua configuração tentando algumas URLs.

Exemplo

Esta seção descreve algumas URLs de exemplo que podem ser usadas para visualizar os dados que foram incluídos pela amostra de introdução fornecida com o serviço de dados REST do eXtreme Scale. Antes de usar as seguintes URLs, inclua o conjunto de dados padrão na grade de amostra do eXtreme Scale usando o cliente Java de amostra ou o cliente do Visual Studio WCF Data Services de amostra.

Os exemplos a seguir assumem que a porta seja 8080, o que pode variar. Consulte a seção para obter detalhes sobre como configurar o serviço de dados REST em servidores de aplicativos diferentes.

- Visualizar um único cliente com o ID de "ACME":

  ```
  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')
  ```

- Visualizar todas as ordens para o cliente "ACME":

  ```
  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')/orders
  ```

- Visualizar o cliente "ACME" e as ordens:

  ```
  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')?$expand=orders
  ```

- Visualizar a ordem 1000 para o cliente "ACME":

  ```
  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=1000,customer_customerId='ACME')
  ```

- Visualizar a ordem 1000 para o cliente "ACME" e seu Cliente associado:

  ```
  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
  Order(orderId=1000,customer_customerId='ACME')?$expand=customer
  ```

- Visualizar a ordem 1000 para o cliente "ACME" e seu Cliente e OrderDetails associados:

  ```
  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
  Order(orderId=1000,customer_customerId='ACME')?$expand=customer,orderDetails
  ```

- Visualizar todas as ordens para o cliente "ACME" para o mês de outubro de 2009 (GMT):

  ```
  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
  Customer(customerId='ACME')/orders?$filter=orderDate
  ge datetime'2009-10-01T00:00:00'
  and orderDate lt datetime'2009-11-01T00:00:00'
  ```

- Visualizar todas as 3 primeiras ordens e orderDetails para o cliente "ACME" para o mês de outubro de 2009 (GMT):

  ```
  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
  Customer(customerId='ACME')/orders?$filter=orderDate
  ge datetime'2009-10-01T00:00:00'
  and orderDate lt datetime'2009-11-01T00:00:00'
  &$orderby=orderDate&$top=3&$expand=orderDetails
  ```

Utilizando um Cliente Java com Serviço de Dados REST

O aplicativo cliente Java usa a API EntityManager do eXtreme Scale para inserir dados na grade.

Sobre Esta Tarefa

As seções anteriores descreveram como criar uma grade do eXtreme Scale e configurar e iniciar o serviço de dados REST do eXtreme Scale. O aplicativo cliente Java usa a API EntityManager do eXtreme Scale para inserir dados na grade. Ele não demonstra como utilizar interfaces REST. O objetivo desse cliente é demonstrar como a API EntityManager é usada para interagir com a grade do eXtreme Scale e permitir a modificação de dados na grade. Para visualizar dados na grade usando o serviço de dados REST, use um navegador da Web ou use o aplicativo cliente Visual Studio 2008.

Procedimento

Para incluir rapidamente conteúdo na grade do eXtreme Scale, execute o seguinte comando:

1. Abra uma janela do terminal ou da linha de comandos e configure a variável de ambiente JAVA_HOME:
   - Linux UNIX `export JAVA_HOME=java_home`
   - Windows `set JAVA_HOME=java_home`
2. `cd restservice_home/gettingstarted`
3. Insira alguns dados na grade. Os dados que são inseridos serão recuperados depois com o uso de um navegador da Web e do serviço de dados REST.

   Se a grade tiver sido iniciada *sem* a segurança do eXtreme Scale, use os comandos a seguir.
   - UNIX Linux `./runclient.sh load default`
   - Windows `runclient.bat load default`

   Se a grade tiver sido iniciada *com* a segurança do eXtreme Scale, use os comandos a seguir.
   - UNIX Linux `./runclient_secure.sh load default`
   - Windows `runclient_secure.bat load default`

   Para um cliente Java, use a seguinte sintaxe de comando:
   - UNIX Linux `runclient.sh comando`
   - Windows `runclient.bat comando`

   Os comandos a seguir estão disponíveis:
   - `load default`

     Carrega um conjunto predefinido de entidades Customer, Category e Product na grade e cria um conjunto aleatório de Orders para cada cliente.
   - `load category categoryId categoryName firstProductId num_products`

     Cria uma Category de produto e um número fixo de entidades Product na grade. O parâmetro firstProductId identifica o número do ID do primeiro produto, e a cada produto subsequente é designado o próximo id até o número especificado de produtos ser criado.
   - `load customer companyCode contactNamecompanyName numOrders firstOrderIdshipCity maxItems discountPct`

     Carrega um novo Customer na grade e cria um conjunto fixo de entidades Order para qualquer produto aleatório atualmente carregado na grade. O

número de Orders é determinado pela configuração do parâmetro <numOrders>. Cada Order terá um número aleatório de entidades OrderDetail até <maxItems>

- `display customer companyCode`

  Exibe uma entidade Customer e as entidades Order e OrderDetail associadas.

- `display category categoryId`

  Exibe uma entidade Category do produto e as entidades Product associadas.

Resultados

- `runclient.bat load default`
- `runclient.bat load customer IBM "John Doe" "IBM Corporation" 5 5000 Rochester 5 0.05`
- `runclient.bat load category 5 "Household Items" 100 5`
- `runclient.bat display customer IBM`
- `runclient.bat display category 5`

Executando e Construindo a Grade de Amostra e o Cliente Java com o Eclipse

A amostra de introdução do serviço de dados REST pode ser atualizada e aprimorada com uso do Eclipse. Para obter detalhes sobre como configurar seu ambiente Eclipse, consulte o documento de texto: `restservice_home/gettingstarted/ECLIPSE_README.txt`.

Depois que o projeto WXSRestGettingStarted for importado para o Eclipse e estiver sendo construído com êxito, a amostra automaticamente recompilará e os arquivos de script usados par iniciar o servidor e o cliente de contêiner automaticamente selecionarão os arquivos de classe e os arquivos XML. O serviço de dados REST detectará automaticamente quaisquer mudanças, já que o servidor da Web está configurado para ler os diretórios de compilação do Eclipse automaticamente.

Importante: Ao alterar arquivos de origem ou de configuração, o servidor de contêiner do eXtreme Scale e o aplicativo do serviço de dados REST devem ser reiniciados. O servidor de contêiner do eXtreme Scale deve ser iniciado antes do aplicativo da Web do serviço de dados REST.

Cliente Visual Studio 2008 WCF com Serviço de Dados REST

A amostra de introdução ao serviço de dados REST do eXtreme Scale inclui um cliente de Serviços de Dados WCF que pode interagir com o serviço de dados REST do eXtreme Scale. A amostra é gravada como um aplicativo de linha de comandos em C#.

Requisitos de Software

O cliente da amostra C# do WCF Data Services requer o seguinte:

- Sistema Operacional
  - Microsoft Windows XP
  - Microsoft Windows Server 2003
  - Microsoft Windows Server 2008
  - Microsoft Windows Vista
- Microsoft Visual Studio 2008 com Service Pack 1

Dica: Consulte o link anterior para conhecer os requisitos adicionais de hardware e software.

- Microsoft .NET Framework 3.5 Service Pack 1
- Microsoft Support: Uma atualização para o .NET Framework 3.5 Service Pack 1 está disponível

Construindo e Executando o Cliente de Introdução

O cliente de amostra do WCF Data Services inclui um projeto e uma solução do Visual Studio 2008 e o código de origem para a execução da amostra. A amostra deve ser carregada no Visual Studio 2008 e compilada em um programa executável Windows antes de poder ser executada. Para construir e executar a amostra, consulte o documento de texto: `restservice_home/gettingstarted/VS2008_README.txt`.

Sintaxe de Comando C# do Cliente WCF Data Services

Windows `WXSRESTGettingStarted.exe <service URL> <command>`

<service URL> é a URL do serviço de dados REST do eXtreme Scale configurada na seção.

Os comandos a seguir estão disponíveis:

- `load default`

  Carrega um conjunto predefinido de entidades Customer, Category e Product na grade e cria um conjunto aleatório de Orders para cada cliente.
- `load category <categoryId> <categoryName> <firstProductId> <numProducts>`

  Cria uma Category de produto e um número fixo de entidades Product na grade. O parâmetro firstProductId identifica o número do ID do primeiro produto, e a cada produto subsequente é designado o próximo id até o número especificado de produtos ser criado.
- `load customer <companyCode> <contactName> <companyName> <numOrders> <firstOrderId> <shipCity> <maxItems> <discountPct>`

  Carrega um novo Customer na grade e cria um conjunto fixo de entidades Order para qualquer produto aleatório atualmente carregado na grade. O número de Orders é determinado pela configuração do parâmetro <numOrders>. Cada Order terá um número aleatório de entidades OrderDetail até <maxItems>
- `display customer <companyCode>`

  Exibe uma entidade Customer e as entidades Order e OrderDetail associadas.
- `display category <categoryId>`

  Exibe uma entidade Category do produto e as entidades Product associadas.
- `unload`

  Remove todas as entidades que foram carregadas utilizando o comando "default load".

Os exemplos a seguir ilustram vários comandos.

- `WXSRestGettingStarted.exe http://localhost:8080/wxsrestservice/restservice/NorthwindGrid load default`
- `WXSRestGettingStarted.exe http://localhost:8080/wxsrestservice/restservice/NorthwindGrid load customer`

- IBM "John Doe" "IBM Corporation" 5 5000 Rochester 5 0.05
- WXSRestGettingStarted.exe http://localhost:8080/wxsrestservice/restservice/NorthwindGrid load category 5 "Household Items" 100 5
- WXSRestGettingStarted.exe http://localhost:8080/wxsrestservice/restservice/NorthwindGrid display customer IBM
- WXSRestGettingStarted.exe http://localhost:8080/wxsrestservice/restservice/NorthwindGrid display category 5

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 produtos, programas ou serviços IBM não significa que apenas produtos, programas ou serviços IBM possam ser utilizados. Qualquer produto, programa ou serviço funcionalmente equivalente, que não infrinja nenhum direito de propriedade intelectual da IBM poderá ser utilizado em substituição a este produto, programa ou serviço. A avaliação e verificação da operação em conjunto com outros produtos, exceto aqueles expressamente designados pela IBM, são de inteira responsabilidade do Cliente.

A IBM pode ter patentes ou solicitações de patentes pendentes relativas a assuntos tratados nesta publicação. O fornecimento desta publicação não lhe garante direito algum sobre tais patentes. Pedidos de licença devem ser enviados, por escrito, para:

Gerência de Relações Comerciais e Industriais da IBM Brasil
Av. Pasteur, 138-146
Botafogo
Rio de Janeiro, RJ
CEP 22290-240

Licenciados deste programa que desejam obter 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:

Gerência de Relações Comerciais e Industriais da IBM Brasil
Av. Pasteur, 138-146
Botafogo
Rio de Janeiro, RJ
CEP 22290-240

Tais informações podem estar disponíveis, sujeitas a termos e condições apropriadas, 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 do 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 serviços de terceiros.

Índice Remissivo

T

U

V

W

Impresso no Brasil