Caio Vinicius

Engenharia de Software, Arquitetura, Java, JavaScript, NoSQL, e mais sobre desenvolvimento de softwares.

DOMINGO, 22 DE DEZEMBRO DE 2013

Gerenciamento de Transações com EJB - Parte 2

Olá pessoal, continuando a falar sobre transações, agora iremos ver como aplicar transações nos objetos EJB.

Visão Geral sobre EJB

Os EJBs são objetos que tem seu ciclo de vida gerenciados pelo Container , ou seja, ele é o responsável por criar, gerenciar, e encerrar este tipo de objeto, o Container prove vários serviços para objetos EJBs, tais como: transações, segurança, acesso concorrente, acesso remoto, entre outros.

Os EJBs estão divididos basicamente em 2 tipo:

Session Beans: Este tipo expõem métodos para serem acessados local ou remotamente, eles são divididos em 3 subtipos:

- **Stateless:** Este não armazena estado entre as chamadas, ou seja, quando o cliente faz uma chamada a um método de um EJB Stateless, a instância que será disponibilizada pelo Container veem de um pool de objetos, onde a cada chamada um objeto diferente pode atender a solicitação;
- **Stateful:** Este armazena estado entre as chamadas, ou seja, quando um cliente faz uma chamada a um método de um EJB Stateful, a instância que irá atender as solicitações será sempre a mesma, com isso o cliente poderá armazenar estado entre as chamadas;
- **Singleton:** Este foi introduzido a partir do Java EE 6, ele segue o design pattern do GOF, onde é criado apenas uma instância do objeto, ou seja, só existirá uma instância daquele EJB no container.

```
package br.com.cvinicius.ejb;

import javax.ejb.Stateless;

@Stateless
public class ClienteBean implements Cliente{

    @Override
    public void nome(){
        System.out.println(" Cliente EJB....");
    }
}
```

Exemplo de Session Bean Stateless.

Message Driven Beans: Este tipo é responsável por processar mensagens assíncronas, ou seja, ele não é invocado diretamente pelo cliente, e sim acionado pelo Container quando uma fila possui mensagens a serem processadas, em Java a API de fila de mensagens é a JMS (Java Message Service).

Tipos de Transações

Ao trabalhar com transações em objetos EJBs, podemos gerenciar o contexto transacional de 2 formas:

Container (CMT): Este tipo marca a transação para ser gerenciada pelo Container , ou seja, ele será o responsável pelo gerenciamento e execução da transação;

Bean (BMT): Este tipo marca a transação para ser gerenciada manualmente, ou seja, a transação será gerenciada de forma programática.

Por padrão, caso não seja definido nenhum tipo de T ransação, é assumido o tipo **Container (CMT)**.

PESQUISAR

Pesquisar

TRADUTOR

Selecione o idioma

Powered by Google Tradutor

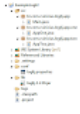
MARCADORES

Java (19)
JavaEE (12)
AngularJS (5)
Spring (5)
SpringBoot (5)
Maven (4)
EJB (3)
FrontEnd (3)
JavaScript (3)
Arquitetura (2)
SpringSecurity (2)
Wildfly (2)
Banco de Dados (1)
CDI (1)
Lombok (1)
PostgreSQL (1)
QueryDSL (1)
VRaptor (1)

ARQUIVOS DO BLOG

► 2017 (2)
► 2016 (9)
► 2015 (2)
► 2014 (5)
▼ 2013 (7)
▼ Dezembro (1)
Gerenciamento de T ransações com EJB - Parte 2
► Agosto (1)
► Junho (1)
► Maio (1)
► Março (1)
► Janeiro (2)
► 2012 (2)

POSTAGENS POPULARES



Criando arquivos de log com Log4J

Olá pessoal, hoje iremos ver como criar logs em aplicações Java com a ferramenta Apache Log4j. O que é um arquivo de log ? Um arquiv...

```java
package br.com.cvinicius.ejb;

import javax.ejb.Stateless;
import javax.ejb.TransactionManagement;
import javax.ejb.TransactionManagementType;

@Stateless
@TransactionManagement(TransactionManagementType.CONTAINER)
public class ClienteBean implements Cliente{

    @Override
    public void nome(){
        System.out.println(" Cliente EJB....");
    }
}
```

Exemplo da configuração do tipo de transação.

Níveis de Propagação

Um ponto importante ao lidarmos com transações, é saber usar corretamente os níveis de propagação, com EJB, ao trabalhar com transações do tipo **CMT**, podemos gerenciar uma mesma transação dentro de várias chamadas de métodos distintos.

Vamos analisar os níveis de propagação disponíveis:

REQUIRED: Se existir uma transação, participa da mesma, senão existir , será criada uma nova transação;

REQUIRES_NEW: Cria uma nova transação para o método mesmo que já exista uma transação corrente;

SUPPORTS: Se existir uma transação, participa da mesma, senão existir , o método é executado normalmente sem transação;

MANDATORY: Se existir uma transação, participa da mesma, senão existir , será lançada uma *javax.ejb.EJBTransactionException*, pois com esse tipo é obrigatório que já exista uma transação corrente;

NOT_SUPPORTED: Se existir uma transação, a mesma será suspensa e o método será executado sem transação, senão existir , o método é executado normalmente sem transação;

NERVER: Se existir uma transação, será lançada uma *javax.ejb.EJBException*, pois com esse tipo não pode haver transação corrente, senão existir , o método é executado normalmente sem transação;

```java
package br.com.cvinicius.ejb;

import javax.ejb.Stateless;
import javax.ejb.TransactionAttribute;
import javax.ejb.TransactionAttributeType;
import javax.ejb.TransactionManagement;
import javax.ejb.TransactionManagementType;

@Stateless
@TransactionManagement(TransactionManagementType.CONTAINER)
public class ClienteBean implements Cliente{

    @Override
    @TransactionAttribute(TransactionAttributeType.REQUIRED)
    public void nome(){
        System.out.println(" Cliente EJB....");
    }
}
```

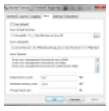
Exemplo de um método definindo o nível de propagação.

Commit e Rollback de Transações

Como sabemos o commit e rollback são as operações mais importantes quando lidamos com transações, o commit é o processo que efetiva com sucesso as operações realizadas dentro de uma transação, e o rollback é o processo que desfaz as operações realizadas dentro de uma transação.

Com transações do tipo **CMT**, o commit é realizado quando a chamada de um método transacionado é encerrada, em transações do tipo **BMT**, o commit deve ser realizado manualmente, invocando o método commit() da interface *javax.transaction.UserTransaction*.

Tratamento de Exceptions em Transações



Monitorando o Tomcat com Java VisualVM

Olá pessoal, hoje iremos ver como monitorar o serviço do Tomcat utilizando o aplicativo Java V isualVM, onde podemos analisar consumos de CPU...

Gerenciamento de Transações com EJB - Parte 2

Olá pessoal, continuando a falar sobre transações, agora iremos ver como aplicar transações nos objetos EJB. Visão Geral sobre EJB Os EJ...

SEGUIR

Email address... Submit

BLOGS

Caelum

Globalcoders

Marcelo DBA

Spring

Thiago Carlos - SQL Server

Os EJB tratam as exceptions em duas categorias:

Application Exceptions: Para o Container, essas são aquelas cujo o cliente precisa saber tratar , ou seja, caso sejam lançadas o cliente deve realizar um tratamento de erro adequado, por padrão, as Applications Exceptions quando lançadas dentro de um método transacionado em beans do tipo **CMT,** não ocorre o rollback automático.

```
import javax.annotation.Resource;
import javax.ejb.SessionContext;
import javax.ejb.Stateless;
import javax.ejb.TransactionAttribute;
import javax.ejb.TransactionAttributeType;
import javax.ejb.TransactionManagement;
import javax.ejb.TransactionManagementType;

@Stateless
@TransactionManagement(TransactionManagementType.CONTAINER)
public class ClienteBean implements Cliente{

    @Resource
    private SessionContext context;

    @Override
    @TransactionAttribute(TransactionAttributeType.REQUIRED)
    public void arquivo()
    throws IOException
    {
        try{
            File file = new File("arquivo.txt");

            file.createNewFile();

        }catch(IOException e){
            context.setRollbackOnly();
        }
    }
}
```

Exemplo de rollback com uma Application Exception.

System Exceptions: Esse tipo de exception são aquelas cujo o cliente não precisa saber realizar o tratamento, ou seja, são todas as exceptions que herdam de *java.lang.RuntimeException* ou *java.rmi.RemoteException* , por padrão, as System Exceptions quando lançadas dentro de um método transacionado em beans do tipo **CMT,** ocorre o rollback automático.

Rollback em Beans CMT

Para mudar o comportamento de uma **Application Exception,** devemos configura lá com a anotação *@javax.ejb.ApplicationException,* setando o atributo *rollback* com o valor *true*, assim quando esta exception for lançada em um método transacionado em um bean do tipo **CMT**, o Container irá realizar o rollback automaticamente, não havendo mais necessidade de realizar o processo manual.

```
package br.com.cvinicius.ejb;

import javax.ejb.ApplicationException;

@ApplicationException(rollback=true)
public class ClienteException extends Exception{

    private static final long serialVersionUID = 1L;

    public ClienteException(Throwable e){
        super(e);
    }
}
```

Exemplo da marcação de uma Application Exception para rollback automático.

Ao desenvolver sistemas corporativos é muito importante saber lidar com transações, quase todos os sistemas acabam utilizando para alguma finalidade e para garantir a integridade das operações.

Até a próxima.

Postado por Caio às 15:58

Marcadores: EJB, JavaEE

Nenhum comentário:

Postar um comentário

Digite seu comentário...

Comentar como: Selecionar perf

Publicar Visualizar

Postagem mais recente Página inicial Postagem mais antiga

Assinar: Postar comentários (Atom)

ENTRE EM CONTATO

Nome

E-mail *

Mensagem *

Enviar