



JBoss Enterprise Application Platform 6.4

Guia de Migração

Para uso com a plataforma Red Hat JBoss Enterprise Application Platform 6

JBoss Enterprise Application Platform 6.4 Guia de Migração

Para uso com a plataforma Red Hat JBoss Enterprise Application Platform 6

Nota Legal

Copyright © 2015 Red Hat, Inc..

This document is licensed by Red Hat under the [Creative Commons Attribution-ShareAlike 3.0 Unported License](#). If you distribute this document, or a modified version of it, you must provide attribution to Red Hat, Inc. and provide a link to the original. If the document is modified, all Red Hat trademarks must be removed.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Resumo

Este livro é um guia para a migração do seu aplicativo a partir das versões anteriores do Red Hat JBoss Enterprise Application Platform.

Índice

CAPÍTULO 1. INTRODUÇÃO	5
1.1. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM 6	5
1.2. GUIA DE MIGRAÇÃO	5
CAPÍTULO 2. PREPARAÇÃO PARA A MIGRAÇÃO	6
2.1. PREPARAÇÃO PARA A MIGRAÇÃO	6
2.2. REVISÃO DOS ASPECTOS NOVOS E DIFERENTES DO JBOSS EAP 6	6
2.3. REVISÃO DA LISTA DE RECURSOS PRETERIDOS E SEM SUPORTE	8
CAPÍTULO 3. MIGRAÇÃO DO SEU APLICATIVO	10
3.1. ALTERAÇÕES EXIGIDAS PELA MAIORIA DOS APLICATIVOS	10
3.1.1. Revisão das Alterações Exigidas pela Maioria dos Aplicativos	10
3.1.2. Alterações do Carregamento de Classe	10
3.1.2.1. Atualização do Aplicativo Devido às Alterações de Carregamento de Classe	10
3.1.2.2. Compreendendo as Dependências de Módulo	10
3.1.2.3. Atualização das Dependências do Aplicativo Devido às Alterações de Carregamento de Classe	11
3.1.3. Alterações do Arquivo de Configuração	12
3.1.3.1. Criação ou Modificação dos Arquivos que Controlam o Carregamento de Classe no EAP 6	12
3.1.3.2. jboss-deployment-structure.xml	15
3.1.3.3. Empacotamento de Recursos para o Novo Sistema de Carregamento de Classe Modular	16
3.1.3.4. Alteração da Localização das Propriedades ResourceBundle	16
3.1.3.5. Criação de um Módulo Personalizado	16
3.1.4. Alterações do Registro em Log	18
3.1.4.1. Modificação das Dependências de Registro em Log	18
3.1.4.2. Atualização do Código de Aplicativo para Estruturas de Registro em Log de Terceiros	19
3.1.4.3. Modificação do Código para Uso da Nova Estrutura de Registro do JBoss	21
3.1.5. Alterações do Empacotamento do Aplicativo	21
3.1.5.1. Modificação do Empacotamento de EARS e WARs	21
3.1.6. Alterações na Configuração do Adaptador de Recursos e Datasource	22
3.1.6.1. Atualização do Aplicativo devido às Alterações de Configuração	22
3.1.6.2. Atualização da Configuração da DataSource	23
3.1.6.3. Instalação e Configuração do Driver JDBC	24
3.1.6.4. Configuração da Fonte de Dados para o Hibernate ou JPA	28
3.1.6.5. Atualização da Configuração do Adaptador de Recurso	29
3.1.7. Alterações de Segurança	30
3.1.7.1. Configuração das Alterações de Segurança do Aplicativo	30
3.1.7.2. Atualização dos Aplicativos que Utilizam PicketLink STS e dos Serviços Web	31
3.1.8. Alterações JNDI	32
3.1.8.1. Atualização dos Nomes do Namespace JNDI do Aplicativo	32
3.1.8.2. Nomes JNDI EJB Portáteis	33
3.1.8.3. Revisão das Regras do Namespace JNDI	33
3.1.8.4. Modificação do Aplicativo para Seguir as Novas Regras do Namespace JNDI	34
3.1.8.5. Exemplos de Namespaces JNDI em Versões Anteriores e a Maneira Como São Especificados no JBoss EAP 6	35
3.1.9. Mapeamento dos Atributos dos Conectores HTTP/HTTPS/AJP	36
3.1.9.1. Mapeamento dos Atributos dos Conectores HTTP/HTTPS/AJP	36
3.2. ALTERAÇÕES DEPENDENTES DOS COMPONENTES E DA ARQUITETURA DO SEU APLICATIVO	41
3.2.1. Revisão das Alterações Dependentes dos Componentes e da Arquitetura do seu Aplicativo	41
3.2.2. Alterações em JPA e Hibernate	42
3.2.2.1. Atualização dos Aplicativos que Utilizam Hibernate e/ou JPA	42
3.2.2.2. Configuração das Alterações para os Aplicativos que Utilizam Hibernate e JPA	42
3.2.2.3. Propriedades da Unidade de Persistência	43

3.2.2.4. Atualização de seu Aplicativo Hibernate 3 para Utilização do Hibernate 4	45
3.2.2.5. Preservação do Comportamento Existente do Valor de Identidade do Hibernate Gerado Automaticamente	46
3.2.2.6. Migração de seu Aplicativo Hibernate 3.3.x para Hibernate 4.x	47
3.2.2.7. Migração de seu Aplicativo Hibernate 3.5.x para Hibernate 4.x	47
3.2.2.8. Modificação das Propriedades de Persistência para os Aplicativos Hibernate e Seam Migrados que Executem em um Ambiente Clusterizado	48
3.2.2.9. Atualização de seu Aplicativo para Ficar de Acordo com a Especificação JPA 2.0	49
3.2.2.10. Substituição do Cache de Segundo Nível JPA/Hibernate com o Infinispan	50
3.2.2.11. Propriedades do Cache Hibernate	51
3.2.2.12. Migração para o Hibernate Validator 4	52
3.2.3. Alterações em JSF	53
3.2.3.1. Habilitação de Aplicativos para Usar Versões Antigas do JSF	53
3.2.4. Alterações dos Serviços Web	54
3.2.4.1. Alterações dos Serviços Web	54
3.2.5. Alterações em JAX-RS e RESTEasy	57
3.2.5.1. Configuração das Alterações em JAX-RS e RESTEasy	57
3.2.6. Alterações no Realm de Segurança LDAP	58
3.2.6.1. Configuração das Alterações no Realm de Segurança LDAP	58
3.2.7. Alterações em HornetQ	60
3.2.7.1. HornetQ e NFS	60
3.2.7.2. Configuração de uma Ponte JMS para a Migração de Mensagens JMS Existentes ao JBoss EAP 6	60
3.2.7.3. Criação de uma Ponte JMS	60
3.2.7.4. Migração do seu Aplicativo para o Uso do HornetQ como o Provedor JMS	65
3.2.7.5. Configuração do Sistema de Mensagens com HornetQ	66
3.2.8. Alterações no Clustering	66
3.2.8.1. Realização de Alterações ao seu Aplicativo para Clustering	66
3.2.8.2. Implantação de um HA Singleton	71
3.2.9. Alterações da Implantação do Estilo de Serviço	77
3.2.9.1. Atualização dos Aplicativos que Usam as Implantações de Estilo de Serviço	77
3.2.10. Alterações de Invocação Remota	77
3.2.10.1. Migração dos Aplicativos Implantados do JBoss EAP 5 que Realizam Invocações Remotas no JBoss EAP 6	77
3.2.10.2. Invocação Remota de um Bean de Sessão usando JNDI	79
3.2.10.3. Referência de Nomeação JNDI EJB	82
3.2.11. Alterações em EJB 2.x	83
3.2.11.1. Atualização dos Aplicativos que usam EJB 2.x	83
3.2.12. Alterações no JBoss AOP	90
3.2.12.1. Atualização dos Aplicativos que Usam o JBoss AOP	90
3.2.13. Migração dos Aplicativos Seam 2.2	91
3.2.13.1. Migração dos Arquivos Seam 2.2 para o JBoss EAP 6	91
3.2.13.2. Problemas de Migração do Arquivo Seam 2.2	94
3.2.14. Migração dos Aplicativos Spring	97
3.2.14.1. Migração de Aplicativos Spring	97
3.2.15. Outras Alterações que Afetam a Migração	97
3.2.15.1. Familiarize-se com Outras Alterações que Podem Afetar sua Migração	97
3.2.15.2. Alteração do Nome Plug-in Maven	97
3.2.15.3. Modificação dos Aplicativos Clientes	98

CAPÍTULO 4. FERRAMENTAS E DICAS 99

4.1. RECURSOS QUE PODEM AUXILIAR NA MIGRAÇÃO	99
4.1.1. Recursos que Podem Assisti-lo na sua Migração	99

4.1.2. Familiarize-se com as Ferramentas que Podem Assisti-lo com a Migração	99
4.1.3. Uso do Tattletale para Encontrar as Dependências dos Aplicativos	100
4.1.4. Baixe e Instale Tattletale	100
4.1.5. Criação e Revisão do Relatório Tattletale	100
4.1.6. Uso da Ferramenta IronJacamar para a Migração das Configurações do Adaptador de Recursos e das Fontes de Dados	101
4.1.7. Baixe e Instale a Ferramenta de Migração IronJacamar	101
4.1.8. Uso da Ferramenta de Migração IronJacamar para Converter um Arquivo de Configuração da Fonte de Dados	102
4.1.9. Uso da Ferramenta de Migração IronJacamar para Converter um Arquivo de Configuração do Adaptador de Recursos	104
4.2. DEPURAÇÃO DE PROBLEMAS RELACIONADOS À MIGRAÇÃO	109
4.2.1. Depuração e Solução de Problemas de Migração	109
4.2.2. Depuração e Solução de ClassNotFoundExceptions e NoClassDefFoundErrors	109
4.2.3. Localização da Dependência de Módulo do JBoss	109
4.2.4. Localização do JAR na Instalação Anterior	110
4.2.5. Depuração e Resolução de ClassCastExceptions	111
4.2.6. Depuração e Resolução de DuplicateServiceExceptions	112
4.2.7. Depuração e Resolução de Erros da Página de Depuração do JBoss Seam	112
4.3. REVISÃO DA MIGRAÇÃO DOS APLICATIVOS DE EXEMPLO	114
4.3.1. Revisão da Migração dos Aplicativos de Exemplo	114
4.3.2. Migração do Exemplo Seam 2.2 JPA para o JBoss EAP 6	114
4.3.3. Migração de Exemplo do Seam 2.2 Booking para o JBoss EAP 6	116
4.3.4. Migração do Arquivo do Seam 2.2 Booking para JBoss EAP 6: Instruções Passo a Passo	119
4.3.5. Construção e Implantação da Versão JBoss EAP 5.X do Aplicativo Seam 2.2 Booking	120
4.3.6. Depuração e Resolução de Exceções e Erros de Implantação do Arquivo do Seam 2.2 Booking	121
4.3.7. Depuração e Resolução de Exceções e Erros de Tempo de Execução do Arquivo do Seam 2.2 Booking	129
4.3.8. Revisão do Sumário das Alterações Feitas Quando Migrando o Aplicativo Seam 2.2 Booking	133
APÊNDICE A. HISTÓRICO DE REVISÃO	136

CAPÍTULO 1. INTRODUÇÃO

1.1. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM 6

O Red Hat JBoss Enterprise Application Platform 6 (JBoss EAP 6) é uma plataforma de middleware construída em padrões abertos e compatível com a especificação Java Enterprise Edition 6. Ele integra o JBoss Application Server 7 com um clustering de alta disponibilidade, um sistema de mensagens, um armazenamento em cache distribuído e outras tecnologias.

O JBoss EAP 6 oferece uma estrutura nova e modular que permite a habilitação do serviço apenas quando necessária, melhorando a velocidade inicial.

O Console de Gerenciamento e a Interface de Linha de Comando de Gerenciamento tornam as edições dos arquivos de configuração XML desnecessárias e agregam a habilidade à realização de script e automatização de tarefas.

Além disso, o JBoss EAP 6 inclui estruturas e APIs para o desenvolvimento rápido de aplicativos Java EE seguros e escaláveis.

[Reportar um erro](#)

1.2. GUIA DE MIGRAÇÃO

O JBoss EAP 6 propicia uma implementação poderosa, leve e rápida da especificação Java Enterprise Edition 6. A arquitetura é construída no Contêiner de Serviço Modular e habilita os serviços por demanda quando o seu aplicativo necessita deles. Devido a esta nova arquitetura, os aplicativos executados no JBoss EAP 5 podem precisar de modificações para serem executados no JBoss EAP 6.

A intenção deste guia é documentar as alterações necessárias para a execução e implantação dos aplicativos JBoss EAP 5.1 no JBoss EAP 6 com êxito. O guia fornece informações sobre como resolver problemas de implantação e de tempo de execução e como prevenir alterações no comportamento do aplicativo. Este é o primeiro passo na mudança para a nova plataforma. Depois que o aplicativo estiver implantado e executando com êxito, os planos para a atualização dos componentes individuais para o uso das novas funções e dos novos recursos do JBoss EAP 6 podem ser feitos.

[Reportar um erro](#)

CAPÍTULO 2. PREPARAÇÃO PARA A MIGRAÇÃO

2.1. PREPARAÇÃO PARA A MIGRAÇÃO

Como o servidor do aplicativo possui uma estrutura diferente das versões anteriores, pode ser que você queira pesquisar e fazer alguns planejamentos antes de tentar migrar o seu aplicativo.

1. Revise os Aspectos Novos e Diferentes do JBoss EAP 6

Um número de coisas que pode impactar a implantação dos aplicativos do JBoss EAP 5 mudou nesta versão. Elas incluem mudanças na estrutura do diretório dos arquivos, nos scripts, na configuração da implantação, no carregamento de classes e nas pesquisas JNDI. Consulte [Seção 2.2, “Revisão dos Aspectos Novos e Diferentes do JBoss EAP 6”](#) para mais detalhes.

2. Revise a Documentação de Introdução

Revise o capítulo intitulado *Get Started Developing Applications* no guia *Development Guide* para o JBoss EAP 6 em https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/. Ele contém informações importantes sobre:

- Java EE 6
- O novo sistema de carregamento de classe modular
- As alterações na estrutura de arquivos
- Como baixar e instalar o JBoss EAP 6
- Como baixar e instalar o JBoss Developer Studio
- Como configurar Maven para o seu ambiente de desenvolvimento
- Como baixar e executar os aplicativos de exemplo de início rápido que são enviados junto com o produto.

3. Aprenda a usar as Dependências do JBoss EAP 6 no seu Projeto Maven

Revise o capítulo intitulado *Maven Guide* no guia *Development Guide* para o JBoss EAP 6 em https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/. A seção *Manage Project Dependencies* contém informações importantes sobre como configurar o seu projeto para utilizar os artefatos JBoss EAP Bill of Material (BOM).

4. Analise e Compreenda o seu Aplicativo

Cada aplicativo é único e os componentes e a arquitetura do aplicativo existente devem ser compreendidos por completo antes da migração ser realizada.



IMPORTANTE

Antes de realizar quaisquer modificações no seu aplicativo, certifique-se de criar uma cópia de backup.

[Reportar um erro](#)

2.2. REVISÃO DOS ASPECTOS NOVOS E DIFERENTES DO JBOSS EAP 6

Introdução

Segue abaixo uma lista das diferenças visíveis no JBoss EAP 6 a partir do lançamento anterior.

Carregamento de classes baseado em módulos

No JBoss EAP 5, a arquitetura de carregamento de classes era hierárquica. No JBoss EAP 6, o carregamento de classes é baseado no JBoss Modules. Isto oferece um verdadeiro isolamento do aplicativo, oculta as classes de implementação do servidor e carrega apenas as classes que seu aplicativo necessita. Além disso, o carregamento de classe é simultâneo proporcionando um melhor desempenho. Os aplicativos gravados para o JBoss EAP 5 precisam ser modificados para a especificação das dependências dos módulos e, em alguns casos, para o reempacotamento dos arquivos. Para mais informações, consulte *Class Loading and Modules* no guia *Development Guide* para o JBoss EAP 6 em

https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

Gerenciamento de Domínio

No JBoss EAP 6, o servidor pode ser executado como um servidor autônomo ou em um domínio gerenciado. Em um domínio gerenciado, você pode configurar grupos inteiros de servidores de uma só vez, mantendo as configurações sincronizadas por toda a sua rede de servidores. Embora isto não deva impactar os aplicativos construídos para os lançamentos anteriores, pode simplificar o gerenciamento de implantações para servidores múltiplos. Para mais informações, consulte *About Managed Domains* no guia *Administration and Configuration Guide* para o JBoss EAP 6 em

https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

Configuração de Implantação

Servidores Autônomos e Domínios Gerenciados

O JBoss EAP 5 usava um perfil baseado na configuração de implantação. Esses perfis estavam localizados no diretório **EAP_HOME/server/**. Os aplicativos continham, com frequência, arquivos múltiplos de configuração para segurança, banco de dados, adaptadores de recursos e outras configurações. No JBoss EAP 6, a configuração de implantação é feita usando um arquivo. Esse arquivo é usado para configurar todos os serviços e subsistemas usados para a implantação. Um servidor autônomo é configurado usando o arquivo **EAP_HOME/standalone/configuration/standalone.xml**. Para os servidores executando em um domínio gerenciado, o servidor é configurado usando o arquivo **EAP_HOME/domain/configuration/domain.xml**. As informações contidas nos arquivos múltiplos de configuração do JBoss EAP 5 devem ser migradas para o novo arquivo de configuração única.

Ordenação de implantações

O JBoss EAP 6 usa uma inicialização rápida e simultânea para implantações, resultando em um desempenho aprimorado e eficiente. Na maioria das vezes, o servidor do aplicativo é capaz de determinar as dependências automaticamente com antecedência e escolher a estratégia de implantação mais eficiente. No entanto, os aplicativos do JBoss EAP 5, que consistem em módulos múltiplos implantados como EARs e usam pesquisas JNDI herdadas ao invés de injeção CDI ou entradas de referência de recurso, podem exigir alterações de configuração.

Estrutura do Diretório e Scripts

Conforme mencionado anteriormente, o JBoss EAP 6 não utiliza mais um perfil baseado na configuração de implantação, portanto não há diretório **EAP_HOME/server/**. Os arquivos de configuração para os servidores autônomos estão agora localizados no diretório **EAP_HOME/standalone/configuration/** e as implantações estão localizadas no diretório

EAP_HOME/standalone/deployments/. Para os servidores executando em um domínio gerenciado, os arquivos de configuração podem ser encontrados no diretório **EAP_HOME/domain/configuration/**.

No JBoss EAP 5, o script do Linux **EAP_HOME/bin/run.sh** ou o script do Windows **EAP_HOME/bin/run.bat** eram usados para iniciar o servidor. No JBoss EAP 6, o script de início do servidor depende de como o seu servidor é executado. O script do Linux **EAP_HOME/bin/standalone.sh** ou o script do Windows **EAP_HOME/bin/standalone.bat** são usados para iniciar o servidor autônomo. O script do Linux **EAP_HOME/bin/domain.sh** ou o script do Windows **EAP_HOME/bin/domain.bat** são usados para iniciar um domínio gerenciado.

Pesquisas JNDI

O JBoss EAP 6 usa agora namespaces JNDI portáteis padronizados. Os aplicativos gravados para o JBoss EAP 5 que usam as pesquisas JNDI devem ser alterados para seguir a nova convenção de namespace JNDI padronizada. Para mais informações sobre a sintaxe de nomeação JNDI, consulte [Seção 3.1.8.2, “Nomes JNDI EJB Portáteis”](#).

Para informações adicionais, consulte *New and Changed Features in JBoss EAP 6* no guia *Development Guide* para o JBoss EAP 6 em https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

[Reportar um erro](#)

2.3. REVISÃO DA LISTA DE RECURSOS PRETERIDOS E SEM SUPORTE

Antes de migrar o seu aplicativo, você deve estar ciente de que alguns recursos que estavam disponíveis nas versões anteriores do JBoss EAP podem estar preteridos ou sem suporte. Para uma lista completa, consulte a seção *Recursos Sem Suporte* das *Notas de Lançamento* para o JBoss EAP 6 localizada no Portal do Consumidor https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

Segue um resumo breve de alguns dos recursos sem suporte.

Argumento de Linha de Comando **-b** para Inicialização do Servidor

Nas versões anteriores, o JBoss EAP usava automaticamente o endereço especificado pelo parâmetro de inicialização **-b**, independente do endereço IP. No JBoss EAP 6, a configuração **<inet-address>** do servidor procura por uma interface de rede que corresponda ao endereço IP. Apesar disto funcionar para **127.0.0.1**, não funciona mais para endereços IP **127.*.*.***. Se você iniciar o servidor JBoss EAP 6 com o argumento de linha de comando **-b** para vincular os endereços IP **127.*.*.***, você deve, agora, mudar primeiro a interface de **<inet-address>** para **<loopback-address>** no arquivo de configuração do servidor.

Para mais informações sobre como configurar o servidor usando o Gerenciamento CLI, consulte a seção intitulada *Management CLI Operations* no guia *Administration and Configuration Guide* para a Plataforma de Aplicação do JBoss Enterprise localizada no Portal do Consumidor https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

Dependências do EJB

Nas versões anteriores do JBoss EAP, as dependências do EJB de um serviço ou serviços, incluindo outros EJBs, poderiam ser especificadas usando uma marcação **<depends>** no descritor de implantação **jboss.xml**. Por exemplo:

```
<depends>jboss.j2ee:jndiName=com/myorg/app/Foo,service=EJB</depends>
<depends>jboss.mq.destination:service=Queue,name=queue/HelloworldQueue</depends>
```

No JBoss EAP 6, você deve usar a anotação **@EJB** para injetar as referências do EJB e a anotação **@Resource** para acessar as fontes de dados ou outros recursos. Por exemplo:

```
@EJB(lookup="java:global/MyApp/FooImpl!com.myorg.app.Foo")
@Resource(mappedName = "java:/queue/HelloworldQueue")
```

As pesquisas JNDI também mudaram. Consulte a seção intitulada *JNDI Changes* neste guia para detalhes.

Para mais informações sobre as referências do EJB, consulte a seção intitulada *EJB Reference Resolution* no guia *Development Guide* para a Plataforma de Aplicação do JBoss Enterprise localizada no Portal do Consumidor https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

HTTPInvoker

Nas versões prévias do JBoss EAP, era possível usar o HTTPInvoker para configurar um EJB, JNDI ou JMS para usar o protocolo HTTP. Isto não é mais possível no JBoss EAP 6.

Implantações de HA Singleton e Serviço BarrierController

O Serviço HA Singleton garante que haja apenas uma instância de um serviço executando dentro do cluster.

O JBoss EAP 5 fornecia suporte para estratégias múltiplas para implantações de HA Singleton, incluindo o serviço Implantador HA Singleton, implantações de POJO utilizando o Controlador HA Singleton e implantações de HA Singleton utilizando o Serviço BarrierController. Todas essas estratégias dependiam da HAPartition para fornecer notificações quando diferentes nós no cluster iniciavam e paravam. Agora, elas não estão mais disponíveis.

No JBoss EAP 6, as implantações de HA Singleton mudaram completamente. O implantador singleton agora opera apenas nos serviços do Contêiner de Serviço Modular (MSC). Quando usando um Serviço Singleton, o serviço de destino é instalado em cada nó no cluster, mas é apenas iniciado em um nó a cada período gerado. Este procedimento simplifica as solicitações de implantação e reduz o período solicitado de relocação do serviço mestre singleton entre os nós. No entanto, ele exige que você escreva um código personalizado para atingir a mesma funcionalidade. Um exemplo de uma implantação de HA Singleton pode ser encontrado nos aplicativos de exemplo de início rápido do JBoss EAP enviados com o produto. Para mais informações sobre HA Singletons, consulte [Seção 3.2.8.2, “Implantação de um HA Singleton”](#).

[Reportar um erro](#)

CAPÍTULO 3. MIGRAÇÃO DO SEU APLICATIVO

3.1. ALTERAÇÕES EXIGIDAS PELA MAIORIA DOS APLICATIVOS

3.1.1. Revisão das Alterações Exigidas pela Maioria dos Aplicativos

As alterações de configuração e carregamento de classe no JBoss EAP 6 irão impactar quase todos os aplicativos. O JBoss EAP 6 também usa uma nova sintaxe de nomeação JNDI portátil padrão. Essas alterações irão impactar a maioria dos aplicativos por isso, sugere-se que, antes de migrar o seu aplicativo, você revise as seguintes informações primeiro:

1. [Seção 3.1.2.1, “Atualização do Aplicativo Devido às Alterações de Carregamento de Classe”](#)
2. [Seção 3.1.6.1, “Atualização do Aplicativo devido às Alterações de Configuração”](#)
3. [Seção 3.1.8.1, “Atualização dos Nomes do Namespace JNDI do Aplicativo”](#)

[Reportar um erro](#)

3.1.2. Alterações do Carregamento de Classe

3.1.2.1. Atualização do Aplicativo Devido às Alterações de Carregamento de Classe

O carregamento de classe modular trata-se de uma mudança significativa no JBoss EAP 6 e irá impactar quase todos os aplicativos. Revise as informações a seguir antes de migrar o seu aplicativo.

1. Primeiro, veja o empacotamento do seu aplicativo e as suas dependências. Para mais informações, consulte: [Seção 3.1.2.3, “Atualização das Dependências do Aplicativo Devido às Alterações de Carregamento de Classe”](#)
2. Caso o seu aplicativo realize o registro em log, será necessário especificar corretamente as dependências do módulo. Para mais informações, consulte: [Seção 3.1.4.1, “Modificação das Dependências de Registro em Log”](#)
3. Devido às alterações do carregamento de classe modular, pode ser que seja necessário alterar a estrutura do empacotamento do seu EAR ou WAR. Para mais informações, consulte: [Seção 3.1.5.1, “Modificação do Empacotamento de EARS e WARs”](#)

[Reportar um erro](#)

3.1.2.2. Compreendendo as Dependências de Módulo

Sumário

Um módulo só é capaz de acessar as suas próprias classes e as classes de qualquer módulo que possua uma dependência explícita ou implícita.

Procedimento 3.1. Compreendendo as Dependências de Módulo

1. Compreendendo as dependências implícitas

Os implantadores dentro do servidor adicionam automaticamente e de forma implícita algumas dependências de módulo comumente usadas, como `javax.api` e `sun.jdk`. Isto permite que as classes sejam visíveis à implantação durante o tempo de execução e reduz a tarefa do desenvolvedor de adicionar explicitamente as dependências. Para obter mais detalhes sobre

como e quando essas dependências implícitas são adicionadas, consulte *Implicit Module Dependencies* no capítulo intitulado *Class Loading and Modules* no guia *Development Guide* para o JBoss EAP 6

https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

2. Compreendendo as dependências explícitas

Para outras classes, os módulos devem ser especificados explicitamente, caso contrário as dependências ausentes resultarão em erros de implantação ou de tempo de execução. Caso esteja faltando uma dependência, você encontrará rastreios como

ClassNotFoundException ou **NoClassDefFoundErrors** no log do servidor. Caso mais de um módulo carregue o mesmo JAR ou um módulo carregue uma classe que estenda uma classe carregada por um módulo diferente, você encontrará rastreios como

ClassCastException no servidor do log. Para especificar as dependências explicitamente, modifique o **MANIFEST.MF** ou crie um arquivo de descritor de implantação específico do JBoss **jboss-deployment-structure.xml**. Para mais informações sobre as dependências de módulos, consulte *Overview of Class Loading and Modules* no capítulo intitulado *Class Loading and Module* no guia *Development Guide* para o JBoss EAP 6 em

https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

[Reportar um erro](#)

3.1.2.3. Atualização das Dependências do Aplicativo Devido às Alterações de Carregamento de Classe

Sumário

O carregamento de classe no JBoss EAP 6 é consideravelmente diferente das versões anteriores do JBoss EAP. O carregamento de classe agora é baseado no projeto JBoss Modules. Ao invés de um carregador de classe único e hierárquico que carrega todos os JARS em um caminho de classe plana, cada biblioteca torna-se um módulo que conecta-se apenas com os módulos exatos dos quais ela depende. As implantações no JBoss EAP 6 também são módulos e elas não têm acesso às classes que são definidas nos JARs no servidor do aplicativo, a não ser que uma dependência explícita seja definida nessas classes. Algumas dependências de módulo definidas pelo servidor do aplicativo são configuradas automaticamente para você. Por exemplo: caso você esteja implantando um aplicativo Java EE, uma dependência no Java EE API é adicionada automaticamente ou implicitamente. Para acesso à lista completa de dependências adicionadas automaticamente pelo servidor, consulte *Implicit Module Dependencies* no capítulo *Class Loading and Modules* do guia *Development Guide* para o JBoss EAP 6 em https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

Tarefas

Quando você migrar o seu aplicativo para JBoss EAP 6, você pode precisar de desempenhar uma ou mais das seguintes tarefas devido às alterações do carregamento de classe modular:

- [Seção 3.1.2.2, “Compreendendo as Dependências de Módulo”](#)
- [Seção 4.1.3, “Uso do Tattletale para Encontrar as Dependências dos Aplicativos”](#)
- [Seção 3.1.3.1, “Criação ou Modificação dos Arquivos que Controlam o Carregamento de Classe no EAP 6”](#)
- [Seção 3.1.3.3, “Empacotamento de Recursos para o Novo Sistema de Carregamento de Classe Modular”](#)

[Reportar um erro](#)

3.1.3. Alterações do Arquivo de Configuração

3.1.3.1. Criação ou Modificação dos Arquivos que Controlam o Carregamento de Classe no EAP 6

Sumário

Devido à alteração no JBoss EAP 6 para o uso do carregamento de classe modular, você pode precisar de criar ou modificar um ou mais arquivos para adicionar dependências ou evitar que dependências automáticas sejam carregadas. Para mais informações sobre o carregamento de classe e precedência do carregamento de classe, consulte o capítulo intitulado *Class Loading and Modules* no guia *Development Guide* para o JBoss EAP 6 em

https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

Os seguintes arquivos são usados para controlar o carregamento de classe no JBoss EAP 6:

jboss-web.xml

Caso tenha definido um elemento **<class-loading>** no arquivo **jboss-web.xml**, será necessário removê-lo. O comportamento que isto causou no JBoss EAP 5 é agora o comportamento do carregamento de classe padrão no JBoss EAP 6, portanto ele não é mais necessário. Caso este elemento não seja removido, você encontrará um `ParseError` e `XMLStreamException` exibidos no log do servidor.

Segue abaixo um exemplo de um elemento **<class-loading>** no arquivo **jboss-web.xml** que foi convertido em comentário.

```
<!DOCTYPE jboss-web PUBLIC
    "-//JBoss//DTD Web Application 4.2//EN"
    "http://www.jboss.org/j2ee/dtd/jboss-web_4_2.dtd">
<jboss-web>
<!--
    <class-loading java2ClassLoadingCompliance="false">
        <loader-repository>
            seam.jboss.org:loader=MyApplication
        </loader-repository>
    </class-loading>
-->
</jboss-web>
```

MANIFEST.MF

Editado manualmente

Dependendo dos componentes ou módulos que o seu aplicativo usar, será necessário adicionar uma ou mais dependências a este arquivo. Você pode adicioná-las tanto como **Dependencies** ou como entradas **Class-Path**.

Segue abaixo um exemplo **MANIFEST.MF** editado por um desenvolvedor:

```
Manifest-Version: 1.0
Dependencies: org.jboss.logmanager
Class-Path: OrderManagerEJB.jar
```


Caso você modifique esse arquivo, certifique-se de incluir um caractere de nova linha no final do arquivo.

Usando o Maven

Se você usar o Maven, será necessário modificar o seu arquivo **pom.xml** para gerar as dependências no arquivo **MANIFEST.MF**. Se o seu aplicativo usar o EJB 3.0, você pode encontrar uma seção no arquivo **pom.xml** parecendo com o seguinte:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-ejb-plugin</artifactId>
  <configuration>
    <ejbVersion>3.0</ejbVersion>
  </configuration>
</plugin>
```

Se o código EJB 3.0 usar **org.apache.commons.log**, você precisará da dependência no arquivo **MANIFEST.MF**. Para gerar essa dependência, adicione o elemento **<plugin>** ao arquivo **pom.xml**, conforme se segue:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-ejb-plugin</artifactId>
  <configuration>
    <ejbVersion>3.0</ejbVersion>
    <archive>
      <manifestFile>src/main/resources/META-INF/MANIFEST.MF</manifestFile>
    </archive>
  </configuration>
</plugin>
```

No exemplo acima, o arquivo **src/main/resources/META-INF/MANIFEST.MF** precisa conter apenas a entrada da dependência:

```
Dependencies: org.apache.commons.logging
```

O Maven gerará o arquivo **MANIFEST.MF** completo:

```
Manifest-Version: 1.0
Dependencies: org.apache.commons.logging
```

jboss-deployment-structure.xml

Esse arquivo é um descritor de implantação específico do JBoss que pode ser usado para controlar o carregamento de classe de forma detalhada. Assim como o **MANIFEST.MF**, esse arquivo pode ser usado para adicionar dependências. Ele também pode evitar que dependências automáticas sejam adicionadas, definir módulos adicionais, alterar o comportamento de carregamentos de classes isoladas de uma implantação EAR e adicionar raízes de recursos adicionais a um módulo.

Segue abaixo um exemplo de um arquivo **jboss-deployment-structure.xml** que adiciona uma dependência ao módulo JSF 1.2 e evita o carregamento automático do módulo JSF 2.0.

■

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="com.sun.jsf-impl" slot="1.2" export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
      <module name="com.sun.jsf-impl" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="javax.faces.api" slot="1.2"/>
      <module name="com.sun.jsf-impl" slot="1.2"/>
    </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

Para mais informações sobre este arquivo, consulte: [Seção 3.1.3.2, “jboss-deployment-structure.xml”](#).

application.xml

Nas versões anteriores do JBoss EAP, a ordem das implantações dentro de um EAR era controlada usando o arquivo **jboss-app.xml**. Este não é mais o caso. A especificação Java EE6 fornece o elemento **<initialize-in-order>** no **application.xml**, permitindo o controle da ordem em que os módulos Java EE são implantados dentro de um EAR.

Na maioria das vezes, não é necessário especificar a ordem da implantação. Se o seu aplicativo usa injeções de dependência e referências de recurso para referir-se a componentes em módulos externos, na maioria dos casos, o elemento **<initialize-in-order>** não é necessário, já que o servidor do aplicativo está apto a determinar implicitamente a maneira correta e ideal de ordenar os componentes.

Vamos supor que você possui um aplicativo que contém um **myBeans.jar** e um **myApp.war** que são empacotados dentro de um **myApp.ear**. Um servlet no **myApp.war** usa uma anotação **@EJB** para injetar um bean a partir do **myBeans.jar**. Neste caso, o servidor do aplicativo possui o conhecimento apropriado para garantir que o componente EJB esteja disponível antes do servlet ser iniciado, não havendo a necessidade de usar o elemento **<initialize-in-order>**.

No entanto, se esse servlet usar referências remotas de estilo de pesquisa JNDI de legacia para acesso ao bean, como a seguir, pode ser que a ordem do módulo precise ser especificada.

```
init() {
    Context ctx = new InitialContext();
    ctx.lookup("TheBeanInMyBeansModule");
}
```

Neste caso, o servidor não está apto a determinar que o componente EJB está no **myBeans.jar** e será necessário impor que os componentes no **myBeans.jar** sejam inicializados e ativados antes dos componentes no **myApp.war**. Para fazer isto, é necessário configurar o elemento **<initialize-in-order>** para **true** e especificar a ordem dos módulos **myBeans.jar** e **myApp.war** no arquivo **application.xml**.

Segue abaixo um exemplo que usa o elemento `<initialize-in-order>` para controlar a ordem de implantação. O `myBeans.jar` é implantado antes do arquivo `myApp.war`.

```
<application xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  version="6"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/application_6.xsd">
  <application-name>myApp</application-name>
  <initialize-in-order>true</initialize-in-order>
  <module>
    <ejb>myBeans.jar</ejb>
  </module>
  <module>
    <web>
      <web-uri>myApp.war</web-uri>
      <context-root>myApp</context-root>
    </web>
  </module>
</application>
```

O esquema para o arquivo `application.xml` pode ser encontrado no http://java.sun.com/xml/ns/javaee/application_6.xsd.



NOTA

Observe que a configuração do elemento `<initialize-in-order>` para `true` reduz a velocidade da implantação. É preferível definir as dependências apropriadas usando injeções de dependência ou referências de recurso, já que isto permite ao contêiner maior flexibilidade na otimização das implantações.

jboss-ejb3.xml

O descritor de implantação `jboss-ejb3.xml` ocupa o lugar do descritor de implantação `jboss.xml` substituindo e adicionando os recursos fornecidos pelo descritor de implantação `ejb-jar.xml` do Java Enterprise Edition (EE). O novo arquivo é incompatível com o `jboss.xml`, e o `jboss.xml` é agora ignorado nas implantações.

login-config.xml

O arquivo `login-config.xml` não é mais usado para a configuração de segurança. A segurança é agora configurada no elemento `<security-domain>` no arquivo de configuração do servidor. Para um servidor autônomo, o arquivo deve ser `standalone/configuration/standalone.xml`. Caso esteja executando o seu servidor em um domínio gerenciado, o arquivo deve ser `domain/configuration/domain.xml`.

[Reportar um erro](#)

3.1.3.2. jboss-deployment-structure.xml

`jboss-deployment-structure.xml` é um novo descritor de implantação opcional para o JBoss EAP 6. Este descritor de implantação fornece controle sobre o carregamento de classe na implantação.

O esquema XML para este descritor de implantação está no ***EAP_HOME/docs/schema/jboss-deployment-structure-1_2.xsd***

[Reportar um erro](#)

3.1.3.3. Empacotamento de Recursos para o Novo Sistema de Carregamento de Classe Modular

Sumário

Nas versões anteriores do JBoss EAP, todos os recursos dentro do diretório **WEB-INF/** foram adicionados ao caminho de classe WAR. No JBoss EAP 6, os artefatos do aplicativo web são carregados somente a partir dos diretórios **WEB-INF/classes** e **WEB-INF/lib**. A falta do empacotamento dos artefatos do aplicativo nas localizações especificadas pode resultar em **ClassNotFoundException**, **NoClassDefError** ou em outros erros de tempo de execução.

Para resolver esses erros de carregamento de classe, a estrutura do seu arquivo de aplicativo deve ser modificada ou um módulo personalizado deve ser definido.

Modificação do Empacotamento de Recursos

Para disponibilizar os recursos apenas para o aplicativo, você deve agrupar os arquivos de propriedades, JARs e outros artefatos com WAR movendo-os para o diretório **WEB-INF/classes/** ou **WEB-INF/lib/**. Essa abordagem está descrita em mais detalhes aqui: [Seção 3.1.3.4, “Alteração da Localização das Propriedades ResourceBundle”](#)

Criação de um Módulo Personalizado

Caso queira disponibilizar os recursos personalizados a todos os aplicativos em execução no servidor do JBoss EAP 6, você deve criar um módulo personalizado. Esta abordagem está descrita em mais detalhes aqui: [Seção 3.1.3.5, “Criação de um Módulo Personalizado”](#)

[Reportar um erro](#)

3.1.3.4. Alteração da Localização das Propriedades ResourceBundle

Sumário

Nas versões anteriores do JBoss EAP, o diretório **EAP_HOME/server/SERVER_NAME/conf/** estava no caminho de classe e disponível para o aplicativo. Para disponibilizar as propriedades para o caminho de classe do aplicativo no JBoss EAP 6, você deve empacotá-las dentro do seu aplicativo.

Procedimento 3.2. Alteração da Localização das Propriedades ResourceBundle

1. Caso esteja implantando um arquivo WAR, essas propriedades devem ser empacotadas na pasta **WEB-INF/classes/** do WAR.
2. Caso queira que essas propriedades fiquem acessíveis a todos os componentes em um EAR, elas devem ser empacotadas na raiz de um JAR e, então, o JAR deve ser colocado na pasta **lib/** do EAR.

[Reportar um erro](#)

3.1.3.5. Criação de um Módulo Personalizado

O procedimento a seguir descreve como criar um módulo personalizado para disponibilizar arquivos de propriedades e outros recursos a todos os aplicativos sendo executados no servidor do JBoss EAP.

Procedimento 3.3. Criação de um Módulo Personalizado

1. Crie e preencha a estrutura do diretório **module/**.

- a. Crie uma estrutura de diretório sob o diretório **EAP_HOME/module** para conter os arquivos e JARs. Por exemplo:

```
$ cd EAP_HOME/modules/
$ mkdir -p myorg-conf/main/properties
```

- b. Mova os arquivos de propriedades para o diretório **EAP_HOME/modules/myorg-conf/main/properties/** criado na etapa anterior.

- c. Crie um arquivo **module.xml** no diretório **EAP_HOME/modules/myorg-conf/main/** contendo o seguinte XML:

```
<module xmlns="urn:jboss:module:1.1" name="myorg-conf">
  <resources>
    <resource-root path="properties"/>
  </resources>
</module>
```

2. Modifique o subsistema **ee** no arquivo de configuração do servidor. Você pode usar o JBoss CLI ou pode editar manualmente o arquivo.

- o Siga as seguintes etapas para modificar o arquivo de configuração do servidor usando o JBoss CLI.

- a. Inicie o servidor e conecte-se ao Gerenciamento CLI.

- Para o Linux, insira o seguinte na linha de comando:

```
EAP_HOME/bin/jboss-cli.sh --connect
```

- Para o Windows, insira o seguinte na linha de comando:

```
C:\>EAP_HOME\bin\jboss-cli.bat --connect
```

Você deverá ver a seguinte resposta:

```
Conectado ao controlador autônomo em localhost:9999
```

- b. Para criar o elemento **myorg-conf** <global-modules> no subsistema **ee**, digite o seguinte na linha de comando:

```
/subsystem=ee:write-attribute(name=global-modules, value=
[{"name"=>"myorg-conf", "slot"=>"main"}])
```

Você deverá ver o seguinte resultado:

```
{"outcome" => "success"}
```

- o Siga essas etapas caso prefira editar manualmente o arquivo de configuração do servidor.
 - a. Interrompa o servidor e abra o arquivo de configuração do servidor em um editor de texto. Caso esteja executando um servidor autônomo, o arquivo será **EAP_HOME/standalone/configuration/standalone.xml** ou, caso esteja executando um domínio gerenciado, o arquivo será **EAP_HOME/domain/configuration/domain.xml**.
 - b. Encontre o subsistema **ee** e adicione o módulo global para **myorg-conf**. Segue um exemplo do elemento do subsistema **ee** modificado para a inclusão do elemento **myorg-conf**:

Exemplo 3.1. elemento myorg-conf

```
<subsystem xmlns="urn:jboss:domain:ee:1.0" >
  <global-modules>
    <module name="myorg-conf" slot="main" />
  </global-modules>
</subsystem>
```

3. Presumindo que você copiou um arquivo nomeado **my.properties** à localização de módulo correta, você poderá agora carregar os arquivos de propriedades usando um código semelhante ao seguinte:

Exemplo 3.2. Carregue o arquivo de propriedades

```
Thread.currentThread().getContextClassLoader().getResource("my.pro
perties");
```

[Reportar um erro](#)

3.1.4. Alterações do Registro em Log

3.1.4.1. Modificação das Dependências de Registro em Log

Sumário

O JBoss LogManager suporta front ends para todas as estruturas de registro, portanto você pode manter seu atual código de registro ou mover para a nova infraestrutura de registro do JBoss. Independente da sua decisão, devido às alterações do carregamento de classes modulares, é provável que tenha que modificar seu aplicativo para adicionar as dependências necessárias.

Procedimento 3.4. Atualização do código de registro do aplicativo

1. [Seção 3.1.4.2, “Atualização do Código de Aplicativo para Estruturas de Registro em Log de Terceiros”](#)
2. [Seção 3.1.4.3, “Modificação do Código para Uso da Nova Estrutura de Registro do JBoss”](#)

[Reportar um erro](#)

3.1.4.2. Atualização do Código de Aplicativo para Estruturas de Registro em Log de Terceiros

Sumário

No JBoss EAP 6, as dependências de registro para estruturas de terceiros comuns, como o Apache Commons Logging, Apache log4j, SLF4J e Java Logging, são adicionadas por padrão. Normalmente, é preferível usar a estrutura de registro fornecida pelo contêiner JBoss EAP. No entanto, caso necessite de uma funcionalidade específica fornecida por uma estrutura de terceiros, o módulo do JBoss EAP correspondente deve ser removido de sua implantação. Observe que, embora a sua implantação utilize a estrutura de registro em log de terceiros, os logs do servidor continuam a usar a configuração do subsistema de registro do JBoss EAP.

Os seguintes procedimentos demonstram como remover o módulo **org.apache.log4j** do JBoss EAP 6 de sua implantação. O primeiro procedimento funciona em qualquer versão do JBoss EAP 6. O segundo procedimento é válido apenas no JBoss EAP 6.3 ou em versões posteriores.

Procedimento 3.5. Configuração do JBoss EAP 6 para usar um arquivo log4j.properties ou log4j.xml

Este procedimento funciona em todas as versões do JBoss EAP 6.



NOTA

Já que este método utiliza um arquivo de configuração log4j, você não poderá alterar mais a configuração de registro do log4j durante o tempo de execução.

1. Crie um **jboss-deployment-structure.xml** com o seguinte conteúdo:

```
<jboss-deployment-structure>
  <deployment>
    <!-- Exclusions allow you to prevent the server from
    automatically adding some dependencies -->
    <exclusions>
      <module name="org.apache.log4j" />
    </exclusions>
  </deployment>
</jboss-deployment-structure>
```

2. Posicione o arquivo **jboss-deployment-structure.xml** no diretório **META-INF/** ou no diretório **WEB-INF/**, caso esteja implantando um WAR. Do contrário, posicione-o no diretório **META-INF/** caso esteja implantando um EAR. Se a sua implantação incluir componentes dependentes, o módulo também deve ser removido para cada subimplantação.
3. Adicione o arquivo **log4j.properties** ou **log4j.xml** no diretório **lib/** de seu EAR ou no diretório **WEB-INF/classes/** de sua implantação WAR. Caso prefira posicionar o arquivo no diretório **lib/** de seu WAR, você deve especificar o caminho **<resource-root>** no arquivo **jboss-deployment-structure.xml**.

```
<jboss-deployment-structure>
  <deployment>
    <!-- Exclusions allow you to prevent the server from
```

```

    automatically adding some dependencies -->
    <exclusions>
      <module name="org.apache.log4j" />
    </exclusions>
    <resources>
      <resource-root path="lib" />
    </resources>
  </deployment>
</jboss-deployment-structure>

```

4. Inicie o servidor do JBoss EAP 6 com o seguinte argumento de tempo de execução para evitar que um **ClassCastException** apareça no console no momento de implantação do aplicativo:

```
-Dorg.jboss.as.logging.per-deployment=false
```

5. Implante seu aplicativo.

Procedimento 3.6. Configuração das Dependências de Registro para o JBoss EAP 6.3 ou versões posteriores

No JBoss EAP 6.3 e em versões posteriores, você pode usar o novo atributo do sistema de registro **add-logging-api-dependencies** para remover as dependências da estrutura de registro em log de terceiros. As etapas a seguir demonstram como modificar este atributo de registro em um servidor autônomo do JBoss EAP.

1. Inicie o servidor do JBoss EAP 6 com o seguinte argumento de tempo de execução para evitar que um **ClassCastException** apareça no console no momento de implantação do aplicativo:

```
-Dorg.jboss.as.logging.per-deployment=false
```

2. Abra um terminal e conecte-se ao Gerenciamento CLI.

- o Para o Linux, insira a seguinte linha de comando:

```
$ EAP_HOME/bin/jboss-cli.sh --connect
```

- o Para o Windows, insira a seguinte linha de comando:

```
C:\>EAP_HOME\bin\jboss-cli.bat --connect
```

3. Modifique o atributo **add-logging-api-dependencies** no subsistema de registro.

Este atributo controla se o contêiner deve adicionar ou não as dependências API implícitas de registro às suas implantações.

- o Se definido como **true**, que é o padrão, todas as dependências API implícitas de registro serão adicionadas.
- o Se definido como **false**, as dependências não serão adicionadas às suas implantações.

Para remover as dependências de estrutura de registro em log de terceiros, você deve definir este atributo como **false** usando o seguinte comando:


```
/subsystem=logging:write-attribute(name=add-logging-api-
dependencies, value=false)
```

Este comando adiciona o elemento **<add-logging-api-dependencies>** ao subsistema **logging** do arquivo de configuração **standalone.xml**.

```
<subsystem xmlns="urn:jboss:domain:logging:1.4">
  <add-logging-api-dependencies value="false"/>
  ....
</subsystem>
```

4. Implante seu aplicativo.

[Reportar um erro](#)

3.1.4.3. Modificação do Código para Uso da Nova Estrutura de Registro do JBoss

Sumário

Para usar a nova estrutura, altere suas importações e código conforme abaixo:

Procedimento 3.7. Modificação do Código e das Dependências para Uso da Estrutura de Registro do JBoss

1. Alteração de suas importações e código de registro

Segue abaixo um exemplo do código que usa a nova estrutura de registro do JBoss:

```
import org.jboss.logging.Level;
import org.jboss.logging.Logger;

private static final Logger logger =
    Logger.getLogger(MyClass.class.toString());

if(logger.isTraceEnabled()) {
    logger.tracef("Starting...", subsystem);
}
```

2. Adição da dependência de registro

O JAR contendo as classes de registro do JBoss está localizado no módulo nomeado **org.jboss.logging**. O seu arquivo **MANIFEST-MF** deve parecer com o seguinte:

```
Manifest-Version: 1.0
Dependencies: org.jboss.logging
```

Por favor consulte [Seção 3.1.2.3, “Atualização das Dependências do Aplicativo Devido às Alterações de Carregamento de Classe”](#) e [Seção 4.2.1, “Depuração e Solução de Problemas de Migração”](#) para mais informações sobre como encontrar a dependência do módulo.

[Reportar um erro](#)

3.1.5. Alterações do Empacotamento do Aplicativo

3.1.5.1. Modificação do Empacotamento de EARS e WARs

Sumário

Quando a migração de seu aplicativo for efetuada, você pode ter que alterar a estrutura de empacotamento de seu EAR ou WAR devido à alteração do carregamento de classe modular. As dependências de módulo estão carregadas na seguinte ordem:

1. Dependências do sistema
2. Dependências do Usuário
3. Recursos Locais
4. Dependências inter-implantadas

Procedimento 3.8. Modificação do empacotamento de arquivos

1. Empacotamento do WAR

O WAR é um módulo único e todas as classes no WAR são carregadas com o mesmo carregador de classe. Isto significa que as classes empacotadas no diretório **WEB-INF/lib/** são tratadas da mesma forma que as classes no diretório **WEB-INF/classes**.

2. Empacotamento de um EAR

Um EAR consiste de módulos múltiplos. O diretório **EAR/lib/** é um módulo único e cada WAR ou subimplantação EJB jar dentro do EAR é um módulo separado. As classes não possuem acesso às classes em outros módulos dentro do EAR, a não ser que as dependências explícitas tenham sido definidas. As subimplantações sempre possuem uma dependência automática no módulo pai que fornece-as acesso às classes no diretório **EAR/lib/**. No entanto, as subimplantações nem sempre possuem uma dependência automática para permití-las acessar umas às outras. Este comportamento é controlado pela configuração do elemento **<ear-subdeployments-isolated>** na configuração do subsistema **ee**, conforme abaixo:

```
<subsystem xmlns="urn:jboss:domain:ee:1.0" >
  <ear-subdeployments-isolated>false</ear-subdeployments-isolated>
</subsystem>
```

Por padrão, isto é configurado como falso, permitindo que as subimplantações vejam as classes pertencentes às outras subimplantações dentro do EAR.

Para mais informações sobre o carregamento de classe, consulte o capítulo intitulado *Class Loading and Modules* no guia *Development Guide* para o JBoss EAP 6 em https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

[Reportar um erro](#)

3.1.6. Alterações na Configuração do Adaptador de Recursos e Datasource

3.1.6.1. Atualização do Aplicativo devido às Alterações de Configuração

No JBoss EAP 5, os serviços e subsistemas foram configurados em diversos arquivos diferentes. No JBoss EAP 6, a configuração é agora realizada basicamente em um arquivo. Caso o seu aplicativo utilize qualquer um dos seguintes recursos ou serviços, alterações de configuração podem ser necessárias.

1. Se seu aplicativo usa uma fonte de dados, consulte : [Seção 3.1.6.2, “Atualização da Configuração da DataSource”](#).

2. Se o seu aplicativo usa JPA e atualmente agrupa JARs Hibernate, consulte: [Seção 3.1.6.4, “Configuração da Fonte de Dados para o Hibernate ou JPA”](#) para opções de migração.
3. Se o seu aplicativo usa um adaptador de recursos, consulte: [Seção 3.1.6.5, “Atualização da Configuração do Adaptador de Recurso”](#).
4. Revise as informações a seguir sobre como configurar as alterações para segurança básica: [Seção 3.1.7.1, “Configuração das Alterações de Segurança do Aplicativo”](#).

[Reportar um erro](#)

3.1.6.2. Atualização da Configuração da DataSource

Sumário

Nas versões anteriores do JBoss EAP, a configuração da DataSource JCA foi definida em um arquivo com um sufixo ***-ds.xml**. Esse arquivo foi, então, implantado no diretório **deploy/** do servidor ou empacotado com o aplicativo. O driver JDBC foi copiado ao diretório **server/lib/** ou empacotado no diretório **WEB-INF/lib/** do aplicativo. Apesar desse método de configuração da DataSource ainda possuir suporte para desenvolvimento, ele não é recomendado para produção já que não possui suporte pelas ferramentas administrativas e de gerenciamento do JBoss.

No JBoss EAP 6, a DataSource é configurada no arquivo de configuração do servidor. Se a instância do JBoss EAP 6 estiver em execução em um domínio gerenciado, a DataSource é configurada no arquivo **domain/configuration/domain.xml**. Se a instância do JBoss EAP 6 estiver em execução como um servidor autônomo, a DataSource é configurada no **standalone/configuration/standalone.xml file**. As DataSources configuradas dessa maneira podem ser gerenciadas e controladas usando as interfaces de gerenciamento do JBoss, incluindo o Console de Gerenciamento da Web e a interface de linha de comando (CLI). Essas ferramentas facilitam o gerenciamento de implantações e a configuração dos servidores múltiplos em execução no domínio gerenciado.

A seção seguinte descreve como modificar a configuração da sua DataSource, de forma que ela possa ser gerenciada e suportada pelas ferramentas de gerenciamento disponíveis.

Migração para uma Configuração de DataSource Gerenciável para o JBoss EAP 6

Um driver compatível com JDBC 4.0 pode ser instalado como uma implantação ou como um módulo básico. Um driver que é compatível com o JDBC 4.0 contém um arquivo **META-INF/services/java.sql.Driver** que especifica o nome da classe do driver. Um driver que não é compatível com o JDBC 4.0 requer etapas adicionais. Para mais detalhes sobre como fazer um driver compatível com o JDBC 4.0 e como atualizar a configuração atual da sua DataSource para uma gerenciável pelo Console de Gerenciamento da Web e CLI, consulte [Seção 3.1.6.3, “Instalação e Configuração do Driver JDBC”](#).

Se o seu aplicativo usa Hibernate ou JPA, ele pode requerer alterações adicionais. Consulte [Seção 3.1.6.4, “Configuração da Fonte de Dados para o Hibernate ou JPA”](#) para mais informações.

Usando a Ferramenta de Migração IronJacamar para Converter os Dados de Configuração

Você pode usar a ferramenta IronJacamar para migrar as configurações da DataSource e do ResourceAdapter. Esta ferramenta converte os arquivos de configuração de estilo ***-ds.xml** no formato esperado pelo JBoss EAP 6. Consulte [Seção 4.1.6, “Uso da Ferramenta IronJacamar para a Migração das Configurações do Adaptador de Recursos e das Fontes de Dados”](#) para mais informações.

Migração do Código que Realiza Pesquisas de DataSource Remotas

Nas versões prévias do JBoss EAP, era possível executar uma pesquisa remota JNDI dos objetos da DataSource, no entanto esta prática nunca foi recomendada pelas seguintes razões.

- O controle do cliente de um recurso de servidor não é confiável e pode resultar em conexões vazadas caso o cliente trave ou perca a conexão para o servidor.
- O desempenho é muito lento pois todas as operações do banco de dados possuem proxy através de um **MBean**.
- Não há suporte para a propagação da transação.

Esta funcionalidade foi removida do JBoss EAP 6 e você pode encontrar **NotSerializableException** quando migrar o seu aplicativo. A abordagem recomendada é a criação de um EJB para acesso à DataSource e, então, a solicitação do EJB remotamente. Para mais informações, consulte a seção intitulada *Remote Invocation Changes* neste livro. Informações adicionais podem ser encontradas no guia *Development Guide* para o JBoss 6 em https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

[Reportar um erro](#)

3.1.6.3. Instalação e Configuração do Driver JDBC

Sumário

O driver JDBC pode ser instalado no contêiner em uma das duas seguintes maneiras:

- Como implantação
- Como módulo principal

Os aspectos positivos e negativos de cada abordagem estão descritos abaixo.

No JBoss EAP 6, a fonte de dados é configurada no arquivo de configuração do servidor. Caso a instância do JBoss EAP 6 esteja sendo executada em um domínio gerenciado, a fonte de dados é configurada no arquivo **domain/configuration/domain.xml**. Caso a instância do JBoss EAP 6 esteja sendo executada como um servidor autônomo, a fonte de dados é configurada no arquivo **standalone/configuration/standalone.xml**. As informações de referência de esquema, que são as mesmas em ambos os modos, podem ser encontradas no diretório **doc/** de instalação do JBoss EAP 6. Para fins desta discussão, pressuponha-se que o servidor esteja sendo executado como servidor autônomo e a fonte de dados esteja configurada no arquivo **standalone.xml**.

Procedimento 3.9. Instalação e Configuração do Driver JDBC

1. Instalação do Driver JDBC.

a. Instalação do Driver JDBC como uma implantação.

Esta é a maneira recomendada de instalar o driver. Quando o driver JDBC é instalado como uma implantação, ele é implantado como um JAR regular. Caso a instância do JBoss EAP 6 esteja sendo executada como um servidor autônomo, copie o JAR compatível com o JDBC 4.0 no diretório **EAP_HOME/standalone/deployments/**. Para um domínio gerenciado, você deve usar o Console de Gerenciamento ou Gerenciamento CLI para implantar o JAR nos grupos do servidor.

Segue um exemplo de um driver MySQL JDBC instalado como uma implantação no servidor autônomo:

```
$cp mysql-connector-java-5.1.15.jar
EAP_HOME/standalone/deployments/
```

Qualquer driver compatível com o JDBC 4.0 é automaticamente reconhecido e instalado no sistema por nome e versão. Um JAR compatível com o JDBC 4.0 contém um arquivo de texto nomeado **META-INF/services/java.sql.Driver** que especifica o(s) nome(s) da classe do driver. Caso o driver não seja compatível com o JDBC 4.0, ele pode ser implantável em uma das seguintes maneiras:

- Crie e adicione o arquivo **java.sql.Driver** ao JAR sob o caminho **META-INF/services/**. Esse arquivo deve conter o nome de classe do driver, por exemplo:

```
com.mysql.jdbc.Driver
```

- Crie um arquivo **java.sql.Driver** no diretório de implantação. Para uma instância do JBoss EAP 6 executando como um servidor autônomo, o arquivo deve ser posicionado aqui: **EAP_HOME/standalone/deployments/META-INF/services/java.sql.Driver**. Caso o servidor esteja em um domínio gerenciado, você deve usar o Console de Gerenciamento ou o Gerenciamento CLI para implantar o arquivo.

Os aspectos positivos desta abordagem são:

- Este é o método mais simples já que não há necessidade de definir um módulo.
- Quando o servidor executa em um domínio gerenciado, as implantações que usam esta abordagem são propagadas automaticamente a todos os servidores no domínio. Isto significa que o administrador não precisa distribuir o driver JAR manualmente.

Os aspectos negativos desta abordagem são:

- Se o driver JDBC consiste em mais de um JAR, por exemplo o driver JAR mais um JAR de licença dependente ou um JAR de localização, você não pode instalar o driver como uma implantação. O JDBC deve ser instalado como um módulo principal.
- Caso o driver não seja compatível com o JDBC 4.0, um arquivo deve ser criado contendo o(s) nome(s) da classe do driver e deve ser importado para um JAR ou sobreposto no diretório **deployments/**.

b. Instalação do Driver JDBC como um módulo principal.

Para instalar um driver JDBC como um módulo principal, uma estrutura do caminho do arquivo sob o diretório **EAP_HOME/modules/** deve ser criada. Esta estrutura contém o JAR do driver JDBC, JARs de localização ou licenças adicionais do fornecedor e um arquivo **module.xml** para a definição do módulo.

■ Instalação do Driver MySQL JDBC como um módulo principal

- i. Crie uma estrutura de diretório **EAP_HOME/modules/com/mysql/main/**
- ii. No subdiretório **main/**, crie um arquivo **module.xml** contendo a seguinte definição do módulo para o driver MySQL JDBC:

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.0" name="com.mysql">
  <resources>
    <resource-root path="mysql-connector-java-5.1.15.jar"/>
  </resources>
</module>
```

```

</resources>
<dependencies>
  <module name="javax.api"/>
</dependencies>
</module>

```

O nome do módulo, "com.mysql", coincide com a estrutura do diretório para esse módulo. O elemento **<dependencies>** é usado para especificar as dependências do módulo em outros módulos. Neste caso, como no caso de todas as fontes de dados JDBC, ele é dependente do Java JDBC APIs que é definido em outro módulo nomeado **javax.api**. Este módulo está localizado sob o diretório **modules/system/layers/base/javax/api/main/**.



NOTA

Certifique-se de que você NÃO possua espaço no início do arquivo **module.xml** ou obterá um erro "Novas dependências ausentes/não satisfatórias" para este driver.

- iii. Copie MySQL JDBC driver JAR no diretório **EAP_HOME/modules/com/mysql/main/**:

```

$ cp mysql-connector-java-5.1.15.jar
EAP_HOME/modules/com/mysql/main/

```

■ Instalação do driver IBM DB2 JDBC e do JAR de licença como um módulo principal.

Este exemplo é fornecido apenas para demonstrar a implantação de drivers que requerem JARs além do JAR do Driver JDBC.

- i. Crie a estrutura do diretório **EAP_HOME/modules/com/ibm/db2/main/**.
- ii. No subdiretório **main/** crie um arquivo **module.xml** contendo a seguinte definição do módulo para a licença e o driver IBM DB2 JDBC:

```

<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.1" name="com.ibm.db2">
  <resources>
    <resource-root path="db2jcc.jar"/>
    <resource-root path="db2jcc_license_cisuz.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>

```



NOTA

Certifique-se de que você NÃO possua espaço no início do arquivo **module.xml** ou obterá um erro "Novas dependências ausentes/não satisfatórias" para este driver.

- iii. Copie o driver JDBC e o JAR de licença no diretório **EAP_HOME/modules/com/ibm/db2/main/**.

```
$ cp db2jcc.jar EAP_HOME/modules/com/ibm/db2/main/
$ cp db2jcc_license_cisuz.jar
EAP_HOME/modules/com/ibm/db2/main/
```

Os aspectos positivos desta abordagem são:

- Esta é a única abordagem que funciona quando o driver JDBC consiste em mais de um JAR.
- Com esta abordagem, os drivers que não são compatíveis com o JDBC 4.0 podem ser instalados sem modificar o driver JAR ou criar uma sobreposição de arquivo.

Os aspectos negativos desta abordagem são:

- É mais difícil configurar um módulo.
- O módulo deve ser manualmente copiado para cada servidor em execução em um domínio gerenciado.

2. Configuração da fonte de dados.

a. Adição do driver do banco de dados.

Adicione o elemento **<driver>** ao elemento **<drivers>** do mesmo arquivo. Mais uma vez, isto contém algumas das mesmas informações sobre a fonte de dados que foram definidas anteriormente no arquivo ***-ds.xml**.

Primeiro, determine se o driver JAR é compatível com o JDBC 4.0. Um JAR que é compatível com o JDBC 4.0 contém um arquivo **META-INF/services/java.sql.Driver** que especifica o nome de classe do driver. O

servidor usa este arquivo para encontrar o nome da(s) classe(s) do driver no JAR. Um driver que é compatível com o JDBC 4.0 não requer um elemento **<driver-class>**, desde que ele já esteja especificado no JAR. Segue um exemplo do elemento do driver para um driver MySQL compatível com o JDBC 4.0:

```
<driver name="mysql-connector-java-5.1.15.jar"
module="com.mysql"/>
```

Um driver que não seja compatível com o JDBC 4.0 requer um atributo **<driver-class>** para identificar a classe do driver já que não há um arquivo **META-INF/services/java.sql.Driver** que especifique o nome da classe do driver. Segue

um exemplo do elemento do driver para um driver não compatível com o JDBC 4.0:

```
<driver name="mysql-connector-java-5.1.15.jar"
module="com.mysql">
<driver-class>com.mysql.jdbc.Driver</driver-class></driver>
```

b. Criação da fonte de dados.

Crie um elemento **<datasource>** na seção **<datasources>** do arquivo **standalone.xml**. Este arquivo contém muitas das mesmas informações sobre a fonte de dados anteriormente definida no arquivo ***-ds.xml**.



IMPORTANTE

Você deve interromper o servidor antes de editar o arquivo de configuração do servidor para que a sua alteração seja efetivada ao reiniciar o servidor.

Segue um exemplo de um elemento da fonte de dados MySQL no arquivo `standalone.xml`:

```
<datasource jndi-name="java:/YourDatasourceName" pool-
name="YourDatasourceName">
  <connection-
url>jdbc:mysql://localhost:3306/YourApplicationURL</connection-
url>
  <driver>mysql-connector-java-5.1.15.jar</driver>
  <transaction-isolation>TRANSACTION_READ_COMMITTED</transaction-
isolation>
  <pool>
    <min-pool-size>100</min-pool-size>
    <max-pool-size>200</max-pool-size>
  </pool>
  <security>
    <user-name>USERID</user-name>
    <password>PASSWORD</password>
  </security>
  <statement>
    <prepared-statement-cache-size>100</prepared-statement-cache-
size>
    <share-prepared-statements/>
  </statement>
</datasource>
```

3. Atualização das referências JNDI no código do aplicativo.

Você deve substituir os antigos nomes de pesquisa JNDI no código fonte do aplicativo para usar os novos nomes padronizados da fonte de dados JNDI que você definiu anteriormente. Consulte [Seção 3.1.8.4, “Modificação do Aplicativo para Seguir as Novas Regras do Namespace JNDI”](#) para mais informações.

Você deve substituir também quaisquer anotações `@Resource` existentes que acessam a fonte de dados para utilizar o novo nome JNDI. Por exemplo:

```
@Resource(name = "java:/YourDatasourceName").
```

[Reportar um erro](#)

3.1.6.4. Configuração da Fonte de Dados para o Hibernate ou JPA

Se o seu aplicativo usa JPA e agrupa atualmente JARs Hibernate, pode ser que você queira usar o Hibernate que está incluído no JBoss EAP 6. Para usar essa versão do Hibernate, você deve remover o pacote antigo do Hibernate do seu aplicativo.

Procedimento 3.10. Remoção do pacote Hibernate

1. Remova os JARs Hibernate das pastas da biblioteca do seu aplicativo.

2. Remova ou converta em comentário o elemento `<hibernate.transaction.manager_lookup_class>` no seu arquivo `persistence.xml`, já que este elemento não é necessário.

[Reportar um erro](#)

3.1.6.5. Atualização da Configuração do Adaptador de Recurso

Sumário

Nas versões anteriores do servidor do aplicativo, a configuração do adaptador de recursos era definida em um arquivo com um sufixo `*-ds.xml`. No JBoss EAP 6, um adaptador de recursos é configurado no arquivo de configuração do servidor. Se estiver executando em um domínio gerenciado, o arquivo de configuração é o arquivo `EAP_HOME/domain/configuration/domain.xml`. Se estiver executando em um servidor autônomo, configure o adaptador de recursos no arquivo `EAP_HOME/standalone/configuration/standalone.xml`. A informação de referência do esquema, que é a mesma para ambos os módulos, pode ser encontrada sob *Schemas* no site do IronJacamar aqui: <http://www.ironjacamar.org/documentation.html>.



IMPORTANTE

Você deve interromper o servidor antes de editar o arquivo de configuração do servidor para que a sua alteração permaneça ao reiniciar o servidor.

Definição do adaptador de recursos

As informações do descritor do adaptador de recursos estão definidas sob o elemento do subsistema a seguir no arquivo de configuração do servidor:

```
<subsystem xmlns="urn:jboss:domain:resource-adapters:1.1"/>
```

Algumas das mesmas informações que foram definidas anteriormente no arquivo `*-ds.xml` do adaptador de recursos serão usadas.

Segue abaixo um exemplo do elemento do adaptador de recursos no arquivo de configuração do servidor:

```
<resource-adapters>
  <resource-adapter>
    <archive>multiple-full.rar</archive>
    <config-property name="Name">ResourceAdapterValue</config-property>
    <transaction-support>NoTransaction</transaction-support>
    <connection-definitions>
      <connection-definition
        class-
name="org.jboss.jca.test.deployers.spec.rars.multiple.MultipleManagedConne
ctionFactory1"
        enabled="true" jndi-name="java:/eis/MultipleConnectionFactory1"
        pool-name="MultipleConnectionFactory1">
      <config-property name="Name">MultipleConnectionFactory1Value</config-
property>
      </connection-definition>
      <connection-definition
        class-
name="org.jboss.jca.test.deployers.spec.rars.multiple.MultipleManagedConne
```

```

ctionFactory2"
    enabled="true" jndi-name="java:/eis/MultipleConnectionFactory2"
    pool-name="MultipleConnectionFactory2">
    <config-property name="Name">MultipleConnectionFactory2Value</config-
property>
    </connection-definition>
</connection-definitions>
<admin-objects>
    <admin-object
        class-
name="org.jboss.jca.test.deployers.spec.rars.multiple.MultipleAdminObject1
Impl"
        jndi-name="java:/eis/MultipleAdminObject1">
        <config-property name="Name">MultipleAdminObject1Value</config-
property>
        </admin-object>
        <admin-object class-
name="org.jboss.jca.test.deployers.spec.rars.multiple.MultipleAdminObject2
Impl"
        jndi-name="java:/eis/MultipleAdminObject2">
        <config-property name="Name">MultipleAdminObject2Value</config-
property>
        </admin-object>
    </admin-objects>
</resource-adapter>
</resource-adapters>

```

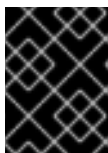
[Reportar um erro](#)

3.1.7. Alterações de Segurança

3.1.7.1. Configuração das Alterações de Segurança do Aplicativo

Configuração da segurança para a autenticação básica

Nas versões anteriores do JBoss EAP, os arquivos de propriedades localizados no diretório **EAP_HOME/server/SERVER_NAME/conf/** estavam no caminho de classe e poderiam ser facilmente encontrados pelo **UsersRolesLoginModule**. No JBoss EAP 6, a estrutura do diretório foi alterada. Os arquivos de propriedades devem ser empacotados dentro do aplicativo para disponibilizá-los no caminho de classe.



IMPORTANTE

O servidor deve ser interrompido antes de editar o arquivo de configuração do servidor para que a sua alteração permaneça ao reiniciar o servidor.

Para configurar a segurança para uma autenticação básica, adicione um novo domínio de segurança sob **security-domains** para **standalone/configuration/standalone.xml** ou o arquivo de configuração do servidor **domain/configuration/domain.xml**:

```

<security-domain name="example">
    <authentication>
        <login-module code="UsersRoles" flag="required">
            <module-option name="usersProperties"

```

```

        value="${jboss.server.config.dir}/example-
users.properties"/>
    <module-option name="rolesProperties"
        value="${jboss.server.config.dir}/example-
roles.properties"/>
    </login-module>
</authentication>
</security-domain>

```

Se a instância do JBoss EAP 6 estiver executando como um servidor autônomo,

`${jboss.server.config.dir}` refere-se ao diretório

`EAP_HOME/standalone/configuration/`. Se a instância estiver executando em um domínio

gerenciado, **`${jboss.server.config.dir}`** refere-se ao diretório

`EAP_HOME/domain/configuration/`.

Modificação dos nomes do domínio de segurança

No JBoss EAP 6, os domínios de segurança não usam mais o prefixo **`java:/jaas/`** nos seus nomes.

- Para os aplicativos Web, você deve remover este prefixo das configurações do domínio de segurança em **`jboss-web.xml`**.
- Para os aplicativos Empresariais (Enterprise), você deve remover este prefixo das configurações do domínio de segurança no arquivo **`jboss-ejb3.xml`**. Esse arquivo substituiu **`jboss.xml`** no JBoss EAP 6.

[Reportar um erro](#)

3.1.7.2. Atualização dos Aplicativos que Utilizam PicketLink STS e dos Serviços Web

Sumário

Se o seu aplicativo JBoss EAP 6.1 usa PicketLink STS e serviços Web, você pode precisar de fazer alterações quando migrar para o JBoss EAP 6.2 ou versões mais recentes. Uma correção aplicada ao JBoss EAP para solucionar [CVE-2013-2133](#) reforça as verificações de autorização pelo contêiner antes da execução de quaisquer manipuladores JAXWS anexos aos pontos de extremidade WS baseados em EJB3. Consequentemente, algumas das funcionalidades do PicketLink podem ser afetadas já que o PicketLink **`SAML2Handler`** estabelece uma entidade de segurança que deve ser utilizada mais tarde no processo. Pode ser que você encontre **`NullPointerException`** no log do servidor já que a entidade é **`NULL`** quando **`HandlerAuthInterceptor`** acessa **`SAML2Handler`**. Esta verificação de segurança deve ser desabilitada para a correção deste problema.

Procedimento 3.11. Desabilitação das Verificações de Autorização Adicionais

- Você pode desabilitar as verificações de autorização adicionais e continuar usando as implantações PicketLink existentes usando um dos seguintes métodos.
 - **Definição de uma Propriedade em Todo o Sistema**
As verificações de autorização adicionais podem ser desabilitadas ao nível do servidor definindo o valor da propriedade do sistema **`org.jboss.ws.cxf.disableHandlerAuthChecks`** para **`true`**. Este método afeta qualquer implantação realizada ao servidor do aplicativo.

Consulte o tópico intitulado *Configure System Properties Using the Management CLI* no guia *Administration and Configuration Guide* do JBoss EAP para mais informações sobre como definir uma propriedade de sistema.

o Criação de uma Propriedade no Arquivo de Descritor de Serviços Web da Implantação

As verificações de autorização adicionais podem ser desabilitadas ao nível da implantação definindo o valor da propriedade **org.jboss.ws.cxf.disableHandlerAuthChecks** para **true** no arquivo **jboss-webservices.xml**. Este método impacta somente a implantação específica.

- Crie um arquivo **jboss-webservices.xml** no diretório **META-INF/** da implantação que deseja desabilitar as verificações da autorização adicionais.
- Adicione o seguinte conteúdo:

```
<?xml version="1.1" encoding="UTF-8"?>
<webservices xmlns="http://www.jboss.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  version="1.2"
  xsi:schemaLocation="http://www.jboss.com/xml/ns/javaee">
  <property>
    <name>org.jboss.ws.cxf.disableHandlerAuthChecks</name>
    <value>true</value>
  </property>
</webservices>
```



NOTA

A habilitação da propriedade **org.jboss.ws.cxf.disableHandlerAuthChecks** renderiza um sistema vulnerável ao [CVE-2013-2133](#). Se o aplicativo espera que as restrições de segurança declaradas nos métodos EJB sejam aplicadas e não as aplica de maneira independente ao manipulador JAX-WS, então a propriedade não deve ser habilitada. A propriedade deve ser apenas usada para fins de compatibilidade com versões anteriores, quando precisa-se evitar a interrupção do aplicativo.

[Reportar um erro](#)

3.1.8. Alterações JNDI

3.1.8.1. Atualização dos Nomes do Namespace JNDI do Aplicativo

Sumário

O EJB 3.1 introduziu um namespace JNDI global padronizado e uma série de namespaces relacionados que mapeiam vários escopos do aplicativo Java EE. Os nomes EJB Portáteis estão vinculados a apenas três deles: **java:global**, **java:module** e **java:app**. Os aplicativos com EJBs que usam o JNDI devem ser alterados para seguirem a nova convenção padronizada do JNDI namespace.

Procedimento 3.12. Modificação das pesquisas JNDI

- Saiba mais [Seção 3.1.8.2, “Nomes JNDI EJB Portáteis ”](#)
- [Seção 3.1.8.3, “Revisão das Regras do Namespace JNDI”](#)
- [Seção 3.1.8.4, “Modificação do Aplicativo para Seguir as Novas Regras do Namespace JNDI”](#)

Exemplos de Mapeamentos JNDI

Exemplos de namespaces JNDI em versões anteriores e de como são especificados no JBoss EAP 6 podem ser encontrados aqui: [Seção 3.1.8.5, “Exemplos de Namespaces JNDI em Versões Anteriores e a Maneira Como São Especificados no JBoss EAP 6”](#)

[Reportar um erro](#)

3.1.8.2. Nomes JNDI EJB Portáteis

Sumário

A especificação Java EE 6 define quatro namespaces lógicos, cada um com o seu próprio escopo. No entanto, os nomes EJB portáteis estão vinculados a apenas três deles. A tabela a seguir mostra detalhes de quando e como usar cada namespace.

Tabela 3.1. Namespaces JNDI Portáteis

Namespace JNDI	Descrição
java:global	<p>Os nomes neste namespace são compartilhados por todos os aplicativos implantados em uma instância do servidor do aplicativo. Use nomes neste namespace para encontrar os arquivos EJBs externos implantados no mesmo servidor.</p> <p>Exemplo de um namespace java:global: java:global/jboss-seam-booking/jboss-seam-booking-jar/HotelBookingAction</p>
java:module	<p>Os nomes neste namespace são compartilhados por todos os componentes em um módulo, por exemplo, todos os beans enterprise em um módulo EJB único ou todos os componentes em um módulo web.</p> <p>Exemplo de um namespace java:module: java:module/HotelBookingAction!org.jboss.seam.example.booking.HotelBooking</p>
java:app	<p>Os nomes neste namespace são compartilhados por todos os componentes em um único aplicativo. Por exemplo, um arquivo jar EJB e um WAR no mesmo arquivo EAR teriam acesso aos recursos no namespace java:app.</p> <p>Exemplo de um namespace java:app: java:app/jboss-seam-booking-jar/HotelBookingAction</p>

Você pode encontrar mais informações sobre os contextos de nomeação JNDI na seção EE.5.2.2, "Application Component Environment Namespaces" em "JSR 316: Java™ Platform, Enterprise Edition (Java EE) Specification, v6". Você pode baixar a especificação aqui: <http://jcp.org/en/jsr/detail?id=316>

[Reportar um erro](#)

3.1.8.3. Revisão das Regras do Namespace JNDI

Sumário

O JBoss EAP 6 aprimorou os nomes do namespace JNDI, não apenas para fornecer regras consistentes e previsíveis para cada nome vinculado ao servidor do aplicativo, mas também para evitar problemas futuros de compatibilidade. Isto significa que você pode vir a ter problemas com os

namespaces atuais no seu aplicativo, caso eles não sigam as novas regras.

Os namespaces devem seguir as seguintes regras:

1. Os nomes relativos não qualificados, como **DefaultDS** ou **jdbc/DefaultDS**, devem ser qualificados relativos a **java:comp/env**, **java:module/env** ou **java:jboss/env**, dependendo do contexto.
2. Os nomes **absolute** não qualificados, como **/jdbc/DefaultDS**, devem ser qualificados relativos a um nome **java:jboss/root**.
3. Os nomes **absolute** qualificados, como **java:/jdbc/DefaultDS**, devem ser qualificados da mesma forma que os nomes **absolute** não qualificados acima.
4. O namespace **java:jboss** especial é compartilhado por toda a instância do servidor AS.
5. Qualquer nome **relative** com um prefixo **java:** deve estar em um dos cinco namespaces: **comp**, **module**, **app**, **global** ou o proprietário **jboss**. Qualquer nome começando com **java:xxx**, onde xxx não coincide com nenhum dos cinco acima, resultaria em um erro de nome inválido.

[Reportar um erro](#)

3.1.8.4. Modificação do Aplicativo para Seguir as Novas Regras do Namespace JNDI

- Segue abaixo um exemplo de um pesquisa JNDI no JBoss EAP 5.1. Este código é geralmente encontrado em um método de inicialização.

```
private ProductManager productManager;
try {
    context = new InitialContext();
    productManager = (ProductManager)
context.lookup("OrderManagerApp/ProductManagerBean/local");
} catch(Exception lookupError) {
    throw new ServletException("Unable to find the ProductManager
bean", lookupError);
}
```

Observe que o nome de pesquisa é **OrderManagerApp/ProductManagerBean/local**.

- Segue abaixo um exemplo de como a mesma pesquisa seria codificada no JBoss EAP 6, usando injeção de dependência.

```
@EJB(lookup="java:app/OrderManagerEJB/ProductManagerBean!services.ej
b.ProductManager")
private ProductManager productManager;
```

Os valores de pesquisa são agora definidos como variáveis de membro e usam o novo nome **java:app** do namespace JNDI portátil

java:app/OrderManagerEJB/ProductManagerBean!services.ejb.ProductManager.

- Caso prefira não usar a injeção de dependência, você pode continuar a criar o novo InitialContext, conforme acima, e a modificar a busca para usar o novo nome do namespace JNDI.

```
private ProductManager productManager;
try {
    context = new InitialContext();
    productManager = (ProductManager)
context.lookup("java:app/OrderManagerEJB/ProductManagerBean!services
.ejb.ProductManager");
} catch (Exception lookupError) {
    throw new ServletException("Unable to find the ProductManager
bean", lookupError);
}
```

[Reportar um erro](#)

3.1.8.5. Exemplos de Namespaces JNDI em Versões Anteriores e a Maneira Como São Especificados no JBoss EAP 6

Tabela 3.2. Tabela de Mapeamento de Namespaces JNDI

Namespace no JBoss EAP 5.x	Namespace no JBoss EAP 6	Comentários Adicionais
OrderManagerApp/ProductManagerBean/local	java:module/ProductManagerBean!services.ejb.ProductManager	Associação padrão Java EE 6. Com escopo para o módulo atual e acessível somente dentro do mesmo módulo.
OrderManagerApp/ProductManagerBean/local	java:app/OrderManagerEJB/ProductManagerBean!services.ejb.ProductManager	Associação padrão Java EE 6. Com escopo para o aplicativo atual e acessível somente dentro do mesmo aplicativo.
OrderManagerApp/ProductManagerBean/local	java:global/OrderManagerApp/OrderManagerEJB/ProductManagerBean!services.ejb.ProductManager	Associação padrão Java EE 6. Com escopo para o servidor do aplicativo e acessível globalmente.
java:comp/UserTransaction	java:comp/UserTransaction	O namespace tem como escopo o componente atual. Não é acessível para threads que não são Java EE 6, por exemplo, os threads criados diretamente pelo seu aplicativo.
java:comp/UserTransaction	java:jboss/UserTransaction	Acessível globalmente. Use este, caso java:comp/UserTransaction não esteja disponível.
java:/TransactionManager	java:jboss/TransactionManager	

Namespace no JBoss EAP 5.x	Namespace no JBoss EAP 6	Comentários Adicionais
java:/TransactionSynchronizationRegistry	java:jboss/TransactionSynchronizationRegistry	

[Reportar um erro](#)

3.1.9. Mapeamento dos Atributos dos Conectores HTTP/HTTPS/AJP

3.1.9.1. Mapeamento dos Atributos dos Conectores HTTP/HTTPS/AJP

A tabela a seguir mostra como mapear os atributos dos Conectores HTTP, HTTPS e AJP das versões anteriores para os novos atributos na Red Hat JBoss Enterprise Application Platform 6.

Tabela 3.3. Mapeamento dos Atributos dos Conectores

Nome do Atributo na Versão Anterior	Equivalente ao JBoss EAP 6	Detalhes
maxThreads	max-connections	<p>Este atributo dimensiona o pool de conexões/threads do JBossWeb. Ele é definido nos conectores no subsistema web. O padrão é 512 por núcleo da CPU.</p> <pre><connector name="http" protocol="HTTP/1.1" scheme="http" socket-binding="http" enabled="true" max-connections="200" /></pre>
minSpareThreads maxSpareThreads	N/D	<p>Não há um equivalente para minSpareThreads ou maxSpareThreads já que há pouco incentivo para a recuperação de threads. Caso use um executor para o conector, ao invés do pool de thread do operador padrão, a coisa mais próxima disto seria o tamanho core-threads e os atributos keepalive-time.</p>
proxyName proxyPort	proxy-name proxy-port	<p>Este atributo é definido no conector no subsistema web.</p> <pre><connector name="http" protocol="HTTP/1.1" scheme="http" socket-binding="http" enabled="true" proxy-name="proxy.com" proxy- port="80"/></pre>

Nome do Atributo na Versão Anterior	Equivalente ao JBoss EAP 6	Detalhes
redirectPort	redirect-port	<p>Este atributo é definido no conector no subsistema web .</p> <pre>connector name="http" protocol="HTTP/1.1" scheme="https" secure="true" socket- binding="http" redirect-port="8443" proxy- name="loadbalancer.hostname.com" proxy- port="443"</pre>
enableLookups	enable-lookups	<p>Este atributo é definido no conector no subsistema web .</p> <pre><connector name="http" protocol="HTTP/1.1" scheme="http" socket-binding="http" enable- lookups="true"/></pre>
MaxHttpHeaderSize	Propriedade do Sistema	<p>Este atributo é definido usando as Propriedades do Sistema. O valor padrão é 8 KB.</p> <pre><system-properties> <property name="org.apache.coyote.http11.Http11Protocol. MAX_HEADER_SIZE" value="8192"/> </system-properties></pre>
maxKeepAliveRequests	Propriedade do Sistema	<p>Este atributo é definido usando as Propriedades do Sistema.</p> <pre><system-properties> <property name="org.apache.coyote.http11.Http11Protocol. MAX_KEEP_ALIVE_REQUESTS" value="1"/> </system-properties></pre>

Nome do Atributo na Versão Anterior	Equivalente ao JBoss EAP 6	Detalhes
connectionTimeout	Propriedade do Sistema	<p>Este atributo é definido usando as Propriedades do Sistema. As configurações a seguir definem AJP connectionTimeout para 600000 milissegundos (10 minutos) e HTTP connectionTimeout para 120000 milissegundos (2 minutos).</p> <pre> <system-properties> <!-- connectionTimeout for AJP connector. Default value is "-1" (no timeout). --> <property name="org.apache.coyote ajp.DEFAULT_CONNECTION_TIMEOUT" value="600000"/> <!-- connectionTimeout for HTTP connector. Default value is "60000". --> <property name="org.apache.coyote.http11.DEFAULT_CONNECTION_TIMEOUT" value="120000"/> </system-properties> </pre>
compactação	Propriedade do Sistema	<p>Este atributo habilita compactação. Você pode especificar o tipo de conteúdo, cujo padrão é text/html, text/xml, text/plain. Você também pode especificar o tamanho mínimo do conteúdo a ser compactado, cujo padrão é 2048 bytes. A compactação é definida usando as Propriedades do Sistema.</p> <pre> <system-properties> <property name="org.apache.coyote.http11.Http11Protocol.COMPRESSION" value="on"/> <property name="org.apache.coyote.http11.Http11Protocol.COMPRESSION_MIN_SIZE" value="4096"/> <property name="org.apache.coyote.http11.Http11Protocol.COMPRESSION_MIME_TYPES" value="text/javascript,text/css,text/html"/> </system-properties> </pre>
URIEncoding	Propriedade do Sistema	<p>Este atributo é definido usando as Propriedades do Sistema.</p> <pre> <system-properties> <property name="org.apache.catalina.connector.URI_ENCODING" value="UTF-8"/> </system-properties> </pre>

Nome do Atributo na Versão Anterior	Equivalente ao JBoss EAP 6	Detalhes
useBodyEncodingForURI	Propriedade do Sistema	<p>Este atributo é definido usando as Propriedades do Sistema.</p> <pre><system-properties> <property name="org.apache.catalina.connector.USE_BODY_ENCODING_FOR_QUERY_STRING" value="true"/> </system-properties></pre>
servidor	Propriedade do Sistema	<p>Este atributo é definido usando as Propriedades do Sistema.</p> <pre><system-properties> <property name="org.apache.coyote.http11.Http11Protocol.SERVER" value="NewServerHeader"/> </system-properties></pre>
allowTrace	Propriedade do Sistema	<p>Este atributo é definido usando as Propriedades do Sistema.</p> <pre><system-properties> <property name="org.apache.catalina.connector.ALLOW_TRACE " value="true"/> </system-properties></pre>
xpoweredby	Propriedade do Sistema	<p>Este atributo é definido usando as Propriedades do Sistema.</p> <pre><system-properties> <property name="org.apache.catalina.connector.X_POWERED_BY " value="true"/> </system-properties></pre>
keepAliveTimeout	N/D	<p>Antes do JBoss 6.4, não havia um parâmetro equivalente no JBoss EAP 6. Internamente, foi padronizado para o valor connectionTimeout.</p> <p>No JBoss EAP 6.4, uma nova propriedade de sistema, org.apache.coyote.http11.DEFAULT_KEEP_ALIVE_TIMEOUT, foi introduzida para controlar o keepAliveTimeout. O argumento - Dorg.apache.coyote.http11.DEFAULT_KEEP_ALIVE_TIMEOUT passa a fazer efeito quando usado com o argumento - Dorg.apache.coyote.http11.DEFAULT_DISABLE_UPLOAD_TIMEOUT=true.</p>

Nome do Atributo na Versão Anterior	Equivalente ao JBoss EAP 6	Detalhes
disableUploadTimeout connectionUploadTimeout	N/D	Atualmente, não há parâmetros equivalentes no JBoss EAP 6. O disableUploadTimeout é verdadeiro por padrão e o connectionUploadTimeout usa internamente o valor connectionTimeout .
packetSize	Propriedade do Sistema	<p>Este atributo é definido usando as Propriedades do Sistema. As configurações a seguir definem AJP packetSize para 20000</p> <pre><system-properties> <property name="org.apache.coyote ajp.MAX_PACKET_SIZE" value="20000"/> </system-properties></pre>
maxPostSize maxSavePostSize	max-post-size max-save-post-size	<p>Um valor de 0 significa ilimitado. Observe que este parâmetro pode limitar o tamanho dos dados apenas quando Content-Type é application/x-www-form-urlencoded. Para mais informações, consulte esta solução no Portal do Consumidor: How to limit data size of HTTP POST method from a client to JBoss</p>
tomcatAuthentication	Propriedade do Sistema	<p>Dependendo da versão do JBoss EAP 6, este atributo é definido usando a Propriedade de Sistema org.apache.coyote.ajp.AprProcessor.TOMCATAUTHENTICATION ou org.apache.coyote.ajp.DEFAULT_TOMCAT_AUTHENTICATION. É necessário corrigir ou atualizar todas as versões do servidor.</p> <p>No JBoss EAP 6.0.1, tomcatAuthentication é configurado usando a seguinte propriedade.</p> <pre><system-properties> <property name="org.apache.coyote.ajp.AprProcessor.TOMCATAUTHENTICATION" value="false"/> </system-properties></pre> <p>No JBoss EAP 6.1 e nas versões mais recentes, tomcatAuthentication é configurado usando a seguinte propriedade.</p> <pre><system-properties> <property name="org.apache.coyote.ajp.DEFAULT_TOMCAT_AUTHENTICATION" value="false"/> </system-properties></pre> <p>Para mais informações, consulte esta solução no Portal do Consumidor How to configure tomcatAuthentication in JBoss EAP 6</p>

Para mais informações sobre os atributos de conectores, consulte esta solução no Portal do Consumidor: [Equivalent HTTP/HTTPS/AJP connector attributes mapping between JBoss EAP 5.x and JBoss EAP 6.x](#)

[Reportar um erro](#)

3.2. ALTERAÇÕES DEPENDENTES DOS COMPONENTES E DA ARQUITETURA DO SEU APLICATIVO

3.2.1. Revisão das Alterações Dependentes dos Componentes e da Arquitetura do seu Aplicativo

Se o seu aplicativo utiliza qualquer uma das seguintes tecnologias ou componentes, você pode precisar fazer modificações no seu aplicativo quando migrar para o JBoss EAP 6.

Hibernate e JPA

O seu aplicativo pode precisar de algumas modificações, caso use o Hibernate ou JPA. Consulte [Seção 3.2.2.1, “Atualização dos Aplicativos que Utilizam Hibernate e/ou JPA”](#) para mais informações.

REST

Se o seu aplicativo utiliza JAX-RS, você deve estar ciente de que o JBoss EAP 6 instala automaticamente o RESTEasy, portanto você não precisa mais configurá-lo. Consulte [Seção 3.2.5.1, “Configuração das Alterações em JAX-RS e RESTEasy”](#) para mais informações.

LDAP

O realm de segurança LDAP é configurado de forma diferente no JBoss EAP 6. Se o seu aplicativo utiliza LDAP, refira-se ao seguinte tópico para mais informações [Seção 3.2.6.1, “Configuração das Alterações no Realm de Segurança LDAP”](#).

Sistema de Mensagem

O JBoss Messaging não está mais incluído no JBoss EAP 6. Se o seu aplicativo utiliza o JBoss Messaging, como provedor de mensagem, você precisa substituir o código do JBoss Messaging por HornetQ. O tópico [Seção 3.2.7.4, “Migração do seu Aplicativo para o Uso do HornetQ como o Provedor JMS”](#) explica o que você precisa fazer.

Clustering

A maneira como você habilita o clustering foi alterada no JBoss EAP 6. Consulte [Seção 3.2.8.1, “Realização de Alterações ao seu Aplicativo para Clustering”](#) para mais detalhes.

Implantação de Estilo de Serviço

Embora o JBoss EAP 6 não utilize mais descritores de estilo de serviço, o contêiner suporta, sempre que possível, essas implantações de estilo de serviço sem alterações. Consulte [Seção 3.2.9.1, “Atualização dos Aplicativos que Usam as Implantações de Estilo de Serviço”](#) para mais informações sobre implantação.

Invocação remota

Se o seu aplicativo realiza invocações remotas, você ainda pode utilizar JNDI para pesquisar um proxy para o seu bean e invocar esse proxy retornado. Para mais informações sobre as alterações de namespaces e a sintaxe necessária, consulte [Seção 3.2.10.1, “Migração dos Aplicativos Implantados do JBoss EAP 5 que Realizam Invocações Remotas no JBoss EAP 6”](#).

Seam 2.2

Se o seu aplicativo utiliza Seam 2.2, consulte o tópico [Seção 3.2.13.1, “Migração dos Arquivos Seam 2.2 para o JBoss EAP 6”](#) para melhor entendimento das alterações necessárias que você precisa realizar.

Spring

Se o seu aplicativo utiliza Spring, consulte [Seção 3.2.14.1, “Migração de Aplicativos Spring”](#).

Outras alterações que podem afetar sua migração

Para alterações adicionais no JBoss EAP 6 que podem impactar o seu aplicativo, consulte [Seção 3.2.15.1, “Familiarize-se com Outras Alterações que Podem Afetar sua Migração”](#).

[Reportar um erro](#)

3.2.2. Alterações em JPA e Hibernate

3.2.2.1. Atualização dos Aplicativos que Utilizam Hibernate e/ou JPA

Sumário

Se o seu aplicativo utiliza Hibernate ou JPA, leia as seções a seguir e faça as alterações necessárias para migrar para o JBoss EAP 6.

- [Seção 3.2.2.2, “Configuração das Alterações para os Aplicativos que Utilizam Hibernate e JPA”](#)
- [Seção 3.2.2.4, “Atualização de seu Aplicativo Hibernate 3 para Utilização do Hibernate 4”](#)
- [Seção 3.2.2.9, “Atualização de seu Aplicativo para Ficar de Acordo com a Especificação JPA 2.0”](#)
- [Seção 3.2.2.10, “Substituição do Cache de Segundo Nível JPA/Hibernate com o Infinispan”](#)
- [Seção 3.2.2.12, “Migração para o Hibernate Validator 4”](#)

[Reportar um erro](#)

3.2.2.2. Configuração das Alterações para os Aplicativos que Utilizam Hibernate e JPA

Sumário

Se o seu aplicativo possui um arquivo `persistence.xml` ou o código utiliza as anotações `@PersistenceContext` ou `@PersistenceUnit`, o JBoss EAP 6 detectará isto durante a implantação e considerará que o aplicativo está usando JPA. Ele adiciona implicitamente o Hibernate 4, mais algumas outras dependências, ao caminho de classe do seu aplicativo.

Se o seu aplicativo utiliza atualmente as bibliotecas do Hibernate 3, é possível, na maioria das vezes, trocar para o Hibernate 4 e executar o aplicativo com êxito. No entanto, caso encontre **ClassNotFoundException** quando estiver implantando seu aplicativo, pode ser possível resolver esse problema usando uma das seguintes abordagens.



IMPORTANTE

Os aplicativos que utilizam Hibernate diretamente com Seam 2.2 podem usar uma versão do Hibernate 3 empacotada dentro do aplicativo. O Hibernate 4, que é fornecido através do módulo `org.hibernate` do JBoss EAP 6, não é suportado pelo Seam 2.2. Este exemplo tem a intenção de ajudá-lo a executar seu aplicativo no JBoss EAP 6 como uma primeira etapa. Observe que o empacotamento do Hibernate 3 com um aplicativo Seam 2.2 não é uma configuração com suporte.

Procedimento 3.13. Configuração do Aplicativo

1. Copie os JARs necessários do Hibernate 3 para a biblioteca do seu aplicativo.

Você pode conseguir resolver o problema copiando os JARs específicos do Hibernate 3 que contêm as classes faltantes no diretório `lib/` do aplicativo ou adicionando-os ao caminho de classe usando algum outro método. Em alguns casos, isto pode resultar em **ClassCastException** ou em outros problemas de carregamento de classe devido ao uso misto das versões Hibernate. Caso isto aconteça, você precisará usar a próxima abordagem.

2. Instrua o servidor para usar apenas as bibliotecas do Hibernate 3.

O JBoss EAP 6 permite que você empacote os jars do provedor de persistência Hibernate 3.5 (ou posterior) com o aplicativo. Para direcionar o servidor para o uso apenas das bibliotecas do Hibernate 3 e excluir as bibliotecas do Hibernate 4, você precisa determinar `jboss.as.jpa.providerModule` como `hibernate3-bundled` no `persistence.xml`, conforme a seguir:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
version="1.0">
  <persistence-unit name="plannerdatasource_pu">
    <description>Hibernate 3 Persistence Unit.</description>
    <jta-data-source>java:jboss/datasources/PlannerDS</jta-data-
source>
    <properties>
      <property name="hibernate.show_sql" value="false" />
      <property name="jboss.as.jpa.providerModule"
value="hibernate3-bundled" />
    </properties>
  </persistence-unit>
</persistence>
```

O implantador Java Persistence API (JPA) detectará a presença de um provedor de persistência no aplicativo e usará as bibliotecas do Hibernate 3. Consulte [Seção 3.2.2.3, “Propriedades da Unidade de Persistência”](#) para mais informações sobre as propriedades de persistência JPA.

3. Desabilite o cache de segundo nível do Hibernate

O cache de segundo nível para o Hibernate 3 não exibe o mesmo comportamento com o JBoss EAP 6, como fazia nas versões anteriores. Caso você esteja usando o cache de segundo nível do Hibernate com o seu aplicativo, você deve desativá-lo até que ele seja atualizado para o Hibernate 4. Para desativar o cache de segundo nível, configure o `<hibernate.cache.use_second_level_cache>` como `false` no arquivo `persistence.xml`.

[Reportar um erro](#)

3.2.2.3. Propriedades da Unidade de Persistência

Propriedades de Configuração do Hibernate 4.x

O JBoss EAP 6 automaticamente define as seguintes propriedades de configuração do Hibernate 4.x:

Tabela 3.4. Propriedades da Unidade de Persistência do Hibernate

Nome da Propriedade	Valor Padrão	Propósito
hibernate.id.new_generator_mappings	verdadeiro	Esta configuração é relevante caso esteja usando @GeneratedValue(AUTO) para gerar valores de chave de índice únicos para novas entidades. Os novos aplicativos devem manter o valor padrão true . Os aplicativos existentes que usavam Hibernate 3.3.x podem precisar alterá-lo para false para continuarem usando um objeto de sequência ou um gerador baseado em tabelas e manterem a compatibilidade com versões anteriores. O aplicativo pode substituir esse valor no arquivo persistence.xml . Mais informações sobre este comportamento são fornecidas abaixo.
hibernate.transaction.jta.platform	Instância de Interface org.hibernate.service.jta.platform.spi.JtaPlatform	Esta classe passa o gerenciador de transação, a transação do usuário e o registro de sincronização da transação no Hibernate.
hibernate.ejb.resource_scanner	Instância da Interface org.hibernate.ejb.packaging.Scanner	Essa classe sabe como utilizar o indexador de anotação do JBoss EAP 6 para fornecer uma implantação mais rápida.
hibernate.transaction.manager_lookup_class		Essa propriedade é removida se encontrada no persistence.xml, já que pode entrar em conflito com hibernate.transaction.jta.platform
hibernate.session_factory_name	<i>QUALIFIED_PERSISTENCE_UNIT_NAME</i>	Configurado como o nome do aplicativo + o nome da unidade de persistência. O aplicativo pode especificar um valor diferente, mas ele deve ser único por todas as implantações do aplicativo na instância do JBoss EAP 6.
hibernate.session_factory_name_is_jndi	falso	Configurado somente se o aplicativo não tiver especificado um valor para hibernate.session_factory_name .
hibernate.ejb.entitymanager_factory_name	<i>QUALIFIED_PERSISTENCE_UNIT_NAME</i>	Configurado como o nome do aplicativo + o nome da unidade da persistência. O aplicativo pode especificar um valor diferente, mas ele deve ser único por todas as implantações do aplicativo na instância do JBoss EAP 6.

No Hibernate 4.x, caso `new_generator_mappings` esteja configurado como `true`:

- `@GeneratedValue(AUTO)` mapeia para `org.hibernate.id.enhanced.SequenceStyleGenerator`.
- `@GeneratedValue(TABLE)` mapeia para `org.hibernate.id.enhanced.TableGenerator`.
- `@GeneratedValue(SEQUENCE)` mapeia para `org.hibernate.id.enhanced.SequenceStyleGenerator`.

No Hibernate 4.x, caso `new_generator_mappings` esteja configurado como `false`:

- `@GeneratedValue(AUTO)` mapeia para Hibernate "native".
- `@GeneratedValue(TABLE)` mapeia para `org.hibernate.id.MultipleHiLoPerTableGenerator`.
- `@GeneratedValue(SEQUENCE)` mapeia para Hibernate "seqhilo".

Para mais informações sobre essas propriedades, acesse <http://www.hibernate.org/docs> e consulte [Hibernate 4.1 Developer Guide](#).

Propriedades de Persistência JPA

As propriedades JPA a seguir são suportadas na definição da unidade de persistência no arquivo `persistence.xml`:

Tabela 3.5. Propriedades da Unidade de Persistência JPA

Nome da Propriedade	Valor Padrão	Propósito
<code>org.jboss.as.jpa.providerModule</code>	<code>org.hibernate</code>	<p>O nome do módulo do provedor de persistência.</p> <p>O valor deve ser hibernate3-bundled, caso os JARs do Hibernate 3 estiverem no arquivo do aplicativo.</p> <p>Se o provedor de persistência estiver empacotado com o aplicativo, este valor deve ser application.</p>
<code>org.jboss.as.jpa.adapterModule</code>	<code>org.jboss.as.jpa.hibernate:4</code>	<p>O nome das classes de integração que ajudam o JBoss EAP 6 a funcionar com o provedor de persistência.</p> <p>Os valores atuais válidos são:</p> <ul style="list-style-type: none"> • <code>org.jboss.as.jpa.hibernate:4</code>: destinado para as classes de integração do Hibernate 4 • <code>org.jboss.as.jpa.hibernate:3</code>: destinado para as classes de integração do Hibernate 3

[Reportar um erro](#)

3.2.2.4. Atualização de seu Aplicativo Hibernate 3 para Utilização do Hibernate 4

Sumário

Quando você atualizar o seu aplicativo para utilizar Hibernate 4, algumas atualizações são gerais e válidas independente da versão do Hibernate sendo usada pelo aplicativo no momento. Para as demais atualizações, você precisa estabelecer qual versão o aplicativo está usando.

Procedimento 3.14. Atualização do aplicativo para a utilização do Hibernate 4

1. O comportamento padrão do gerador de sequência de incremento automático foi alterado no JBoss EAP 6. Consulte [Seção 3.2.2.5, “Preservação do Comportamento Existente do Valor de Identidade do Hibernate Gerado Automaticamente”](#) para mais informações.
2. Estabeleça a versão do Hibernate sendo utilizada pelo aplicativo e escolha o procedimento de atualização correto.
 - o [Seção 3.2.2.6, “Migração de seu Aplicativo Hibernate 3.3.x para Hibernate 4.x”](#)
 - o [Seção 3.2.2.7, “Migração de seu Aplicativo Hibernate 3.5.x para Hibernate 4.x”](#)
3. Consulte [Seção 3.2.2.8, “Modificação das Propriedades de Persistência para os Aplicativos Hibernate e Seam Migrados que Executam em um Ambiente Clusterizado”](#) caso planeja executar o seu aplicativo em um ambiente com cluster.

[Reportar um erro](#)

3.2.2.5. Preservação do Comportamento Existente do Valor de Identidade do Hibernate Gerado Automaticamente

O Hibernate 3.5 introduziu uma propriedade básica nomeada **hibernate.id.new_generator_mappings** que direciona a maneira como as colunas de sequência e identidade são geradas quando usando **@GeneratedValue**. No JBoss EAP 6, o valor padrão para essa propriedade é configurado como a seguir:

- Quando você implanta um aplicativo Hibernate nativo, o valor padrão é **false**.
- Quando você implanta um aplicativo JPA, o valor padrão é **true**.

Diretrizes para Novos Aplicativos

Os novos aplicativos que usam a anotação **@GeneratedValue** devem definir o valor para a propriedade **hibernate.id.new_generator_mappings** como **true**. Esta é a configuração preferida já que é mais portátil entre os diferentes bancos de dados. Na maioria das vezes, ela é mais eficiente e, em alguns casos, ela trata as questões de compatibilidade com a especificação JPA 2.

- Para os novos aplicativos JPA, o JBoss EAP 6 define por padrão a propriedade **hibernate.id.new_generator_mappings** como **true** e isto não deve ser alterado.
- Para os novos aplicativos Hibernate nativos, o JBoss EAP 6 define por padrão a propriedade **hibernate.id.new_generator_mappings** como **false**. Você deve definir essa propriedade como **true**.

Diretrizes para os Aplicativos JBoss EAP 5 Existentes

Os aplicativos existentes que utilizam a anotação **@GeneratedValue** devem certificar-se de que o mesmo gerador seja usado para criar os valores da chave primária para novas entidades, quando o aplicativo é migrado para o JBoss EAP 6.

- Para os aplicativos JPA existentes, o JBoss EAP 6 define por padrão a propriedade **hibernate.id.new_generator_mappings** como **true**. Você deve definir esta propriedade como **false** no arquivo **persistence.xml**.
- Para os aplicativos Hibernate nativos existentes, o JBoss EAP 6 define por padrão **hibernate.id.new_generator_mappings** como **false** e isto não deve ser alterado.

Consulte [Seção 3.2.2.3, “Propriedades da Unidade de Persistência”](#) para mais informações sobre essas configurações de propriedades.

[Reportar um erro](#)

3.2.2.6. Migração de seu Aplicativo Hibernate 3.3.x para Hibernate 4.x

1. Mapeie os tipos de Hibernate text para JDBC LONGVARCHAR

Nas versões anteriores ao Hibernate 3.5, o tipo **text** era mapeado para **JDBC CLOB**. Um novo tipo de Hibernate, **materialized_clob**, foi adicionado ao Hibernate 4 para mapear as propriedades Java **String** para **JDBC CLOB**. Caso o seu aplicativo possua propriedades configuradas como **type="text"** que devem ser mapeadas para **JDBC CLOB**, você precisa seguir uma das opções abaixo:

- a. Se o seu aplicativo utiliza arquivos de mapeamento hbm, altere a propriedade para **type="materialized_clob"**.
- b. Se o seu aplicativo utiliza anotações, você deve substituir **@Type(type = "text")** por **@Lob**.

2. Revise o código para encontrar alterações nos tipos de valores retornados

As projeções de critérios de agregação numérica agora retornam o mesmo tipo de valor que seus equivalentes HQL. Como consequência, os tipos retornados das seguintes projeções em **org.hibernate.criterion** foram alterados.

- a. Devido às alterações em **CountProjection**, **Projections.rowCount()**, **Projections.count(propertyName)** e **Projections.countDistinct(propertyName)**, as projeções **count** e **count distinct** agora retornam um valor **Long**.
- b. Devido às alterações em **Projections.sum(propertyName)**, as projeções **sum** retornam agora um tipo de valor que depende do tipo de propriedade.



NOTA

Falha em modificar o código do seu aplicativo pode resultar em **java.lang.ClassCastException**.

- i. Para as propriedades mapeadas como **Long**, **Short**, **Integer** ou tipos **integer** primitivos, o valor **Long** é retornado;
- ii. Para as propriedades mapeadas como **Float**, **Double** ou tipos de ponto flutuante primitivo, o valor **Double** é retornado.

[Reportar um erro](#)

3.2.2.7. Migração de seu Aplicativo Hibernate 3.5.x para Hibernate 4.x

1. Mescle AnnotationConfiguration na Configuração.

Embora **AnnotationConfiguration** tenha sido preterido, ele não deve afetar a sua migração.

Caso você ainda esteja usando um arquivo **hbm.xml**, você deve estar ciente que o JBoss EAP 6 usa o **org.hibernate.cfg.EJB3NamingStrategy** no **AnnotationConfiguration** ao invés do **org.hibernate.cfg.DefaultNamingStrategy** que foi usado em versões anteriores. Isto pode resultar em incompatibilidade de nomeação. Se você se basear na estratégia de nomeação para usar como padrão o nome de uma tabela de associação (muitos-para-muitos e coleções de elementos), você pode encontrar esse problema. Para resolver isto, você pode solicitar ao Hibernate que utilize **org.hibernate.cfg.DefaultNamingStrategy** de legacia chamando **Configuration#setNamingStrategy** e passando-o **org.hibernate.cfg.DefaultNamingStrategy#INSTANCE**.

2. Modifique os namespaces para estarem de acordo com os novos nomes dos arquivos Hibernate DTD, conforme descrito na tabela abaixo.

Tabela 3.6. Tabela de Mapeamento de Namespace DTD

Namespace DTD Anterior	Namespace DTD Novo
<code>http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd</code>	<code>http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd</code>
<code>http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd</code>	<code>http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd</code>

3. Modifique as variáveis do ambiente.

- Caso você esteja usando Oracle e as propriedades **materialized_clob** ou **materialized_blob**, a variável de ambiente global **hibernate.jdbc.use_streams_for_binary** deve ser configurada como verdadeira.
- Caso você esteja usando PostgreSQL e as propriedades **CLOB** ou **BLOB**, a variável de ambiente global **hibernate.jdbc.use_streams_for_binary** deve ser configurada como falsa.

[Reportar um erro](#)

3.2.2.8. Modificação das Propriedades de Persistência para os Aplicativos Hibernate e Seam Migrados que Executam em um Ambiente Clusterizado

Caso o aplicativo gerenciado pelo contêiner JPA seja migrado, as propriedades que influenciam a serialização dos contextos de persistência estendidos são automaticamente passadas para o contêiner.

No entanto, devido às alterações no Hibernate, você pode executar com problemas de serialização, caso execute seu aplicativo Hibernate ou Seam migrado em um ambiente clusterizado. Você pode encontrar mensagens de log de erro semelhantes ao seguinte:

```
javax.ejb.EJBTransactionRolledbackException: JBAS010361: Failed to
deserialize
....
```

```
Caused by: java.io.InvalidObjectException: could not resolve session
factory during session deserialization
[uuid=8aa29e74373ce3a301373ce3a44b0000, name=null]
```

Para solucionar esses erros, você precisa modificar as propriedades no arquivo de configuração. Na maioria das vezes, o arquivo é **persistence.xml**. Para os aplicativos Hibernate API nativos, o arquivo é **hibernate.cfg.xml**.

Procedimento 3.15. Configuração das propriedades de persistência para executar em um ambiente clusterizado

1. Configure o valor **hibernate.session_factory_name** como um nome único. Este nome deve ser único por todas as implantações do aplicativo na instância do JBoss EAP 6. Por exemplo:

```
<property name="hibernate.session_factory_name" value="jboss-seam-
booking.ear_session_factory"/>
```

2. Configure o valor **hibernate.ejb.entitymanager_factory_name** como um nome único. Este nome deve ser único para todas as implantações do aplicativo na instância do JBoss EAP 6. Por exemplo:

```
<property name="hibernate.ejb.entitymanager_factory_name"
value="seam-booking.ear_PersistenceUnitName"/>
```

Para mais informações sobre as configurações das Propriedades da Unidade de Persistência Hibernate JPA, consulte [Seção 3.2.2.3, “Propriedades da Unidade de Persistência”](#).

[Reportar um erro](#)

3.2.2.9. Atualização de seu Aplicativo para Ficar de Acordo com a Especificação JPA 2.0

Sumário

A especificação JPA 2.0 requer que um contexto de persistência não possa ser propagado fora da transação JTA. Caso o seu aplicativo utilize apenas contextos de persistência de transação com escopo, o comportamento é o mesmo que no JBoss EAP 6, assim como era nas versões anteriores do servidor do aplicativo, e não requer alterações. No entanto, se o seu aplicativo utiliza um contexto de persistência estendido (XPC) para permitir o enfileiramento ou o agrupamento das alterações de dados, é possível que necessite fazer alterações no seu aplicativo.

Comportamento de propagação do contexto de persistência

Caso o seu aplicativo possua um bean de sessão com monitorização de estado (stateful), **Bean1**, que usa um contexto de persistência estendido e demanda um bean de sessão sem monitorização de estado (stateless), **Bean2**, que usa um contexto de persistência de transação com escopo, espera-se que ocorra o seguinte comportamento:

- Se **Bean1** inicia uma transação JTA e realiza a invocação de método **Bean2** com a transação JTA ativa, o comportamento no JBoss EAP 6 é o mesmo das versões anteriores e não requer alterações.
- Se **Bean1** não inicia uma transação JTA e realiza a invocação de método **Bean2**, o JBoss EAP 6 não propaga o contexto de persistência estendido ao **Bean2**. Esse comportamento é diferente das versões anteriores que propagavam o contexto de persistência estendido ao

Bean2. Se o seu aplicativo espera que o contexto de persistência estendido seja propagado ao bean com o gerenciador de entidade transacional, o seu aplicativo precisará ser alterado para realizar a invocação dentro de uma transação JTA ativa.

[Reportar um erro](#)

3.2.2.10. Substituição do Cache de Segundo Nível JPA/Hibernate com o Infinispan

Sumário

O JBoss Cache foi substituído pelo Infinispan para o cache de segundo nível (2LC). Isso requer uma alteração ao arquivo **persistence.xml**. A sintaxe é um pouco diferente e depende se você estiver usando JPA ou o cache de segundo nível Hibernate. Esses exemplos presumem que você esteja usando Hibernate.

Este é um exemplo de como as propriedades para o cache de segundo nível eram especificadas no arquivo **persistence.xml** no JBoss EAP 5.x.

```
<property name="hibernate.cache.region.factory_class"
value="org.hibernate.cache.jbc2.JndiMultiplexedJBossCacheRegionFactory"/>
<property name="hibernate.cache.region.jbc2.cachefactory"
value="java:CacheManager"/>
<property name="hibernate.cache.use_second_level_cache" value="true"/>
<property name="hibernate.cache.region.jbc2.cfg.entity" value="mvcc-
entity"/>
<property name="hibernate.cache.region_prefix" value="services"/>
```

As etapas a seguir usarão este exemplo para configurar Infinispan no JBoss EAP 6.

Procedimento 3.16. Modificação do arquivo **persistence.xml** para usar Infinispan

1. Configure Infinispan para um aplicativo JPA no JBoss EAP 6

Esta é a forma como você especifica as propriedades para atingir a mesma configuração para um aplicativo JPA usando Infinispan no JBoss EAP 6:

```
<property name="hibernate.cache.use_second_level_cache"
value="true"/>
```

Além disso, você precisa especificar um **shared-cache-mode** com um valor de **ENABLE_SELECTIVE** ou **ALL** conforme abaixo:

- **ENABLE_SELECTIVE** é o valor padrão e recomendado. Isto significa que as entidades não estão armazenadas em cache, a não ser que você marque-as, explicitamente, como armazenáveis em cache.

```
<shared-cache-mode>ENABLE_SELECTIVE</shared-cache-mode>
```

- **ALL** significa que as entidades são sempre armazenadas em cache, mesmo que você marque-as como não armazenáveis em cache.

```
<shared-cache-mode>ALL</shared-cache-mode>
```

2. Configure Infinispan para um aplicativo Hibernate native no JBoss EAP 6

Esta é a forma como você pode especificar a mesma configuração para um aplicativo Hibernate nativo usando Infinispan com o JBoss EAP 6:

```
<property name="hibernate.cache.region.factory_class"
value="org.jboss.as.jpa.hibernate4.infinispan.InfinispanRegionFactor
y"/>
<property name="hibernate.cache.infinispan.cachemanager"
value="java:jboss/infinispan/container/hibernate"/>
<property name="hibernate.transaction.manager_lookup_class"
value="org.hibernate.transaction.JBossTransactionManagerLookup"/>
<property name="hibernate.cache.use_second_level_cache"
value="true"/>
```

Você deve adicionar também as seguintes dependências ao arquivo **MANIFEST.MF**:

```
Manifest-Version: 1.0
Dependencies: org.infinispan, org.hibernate
```

Para mais informações sobre as propriedades do cache Hibernate, consulte [Seção 3.2.2.11](#), “Propriedades do Cache Hibernate”.

[Reportar um erro](#)

3.2.2.11. Propriedades do Cache Hibernate

Tabela 3.7. Propriedades

Nome da Propriedade	Descrição
hibernate.cache.region.factory_class	O nome da classe de um CacheProvider personalizado.
hibernate.cache.use_minimal_puts	Booleano. Otimiza a operação do cache de segundo nível para minimizar as gravações, ao custo de leituras mais frequentes. Essa configuração é mais útil para caches com cluster, sendo que o Hibernate3 é habilitado por padrão para as implementações do cache com cluster.
hibernate.cache.use_query_cache	Booleano. Habilita o cache de consulta. Consultas individuais ainda precisam ser configuradas como armazenáveis em cache.
hibernate.cache.use_second_level_cache	Booleano. Usado para desativar completamente o cache de segundo nível, que é habilitado por padrão para classes que especificam um mapeamento <cache> .
hibernate.cache.query_cache_factory	O nome da classe de uma interface QueryCache personalizada. O valor padrão é o StandardQueryCache interno.

Nome da Propriedade	Descrição
<code>hibernate.cache.region_prefix</code>	Um prefixo para usar para os nomes de região de cache de segundo nível.
<code>hibernate.cache.use_structured_entries</code>	Booleano. Força o Hibernate a armazenar dados no cache de segundo nível em um formato mais fácil para o usuário.
<code>hibernate.cache.default_cache_concurrency_strategy</code>	Configuração usada para fornecer o nome do <code>org.hibernate.annotations.CacheConcurrencyStrategy</code> padrão quando <code>@Cacheable</code> ou <code>@Cache</code> forem usados. <code>@Cache(strategy="...")</code> é usado para substituir este padrão.

[Reportar um erro](#)

3.2.2.12. Migração para o Hibernate Validator 4

Sumário

O Hibernate Validator 4.x é uma base de código completamente nova que implementa [JSR 303 - Bean Validation](#). O processo de migração do Validator 3.x para 4.x é bastante simples, mas existem algumas alterações que devem ser realizadas quando migrar seu aplicativo.

Procedimento 3.17. É possível que tenha que executar uma ou mais das seguintes tarefas

1. Acesse o ValidatorFactory padrão

O JBoss EAP 6 conecta um ValidatorFactory padrão ao contexto JNDI sob o nome `java:comp/ValidatorFactory`.

2. Compreenda a validação disparada pelo ciclo de vida

Quando usado em combinação com o Hibernate Core 4, a validação baseada no ciclo de vida é automaticamente habilitada pelo Hibernate Core.

- a. A validação ocorre nas operações **INSERT**, **UPDATE** e **DELETE** de entidade.
- b. É possível configurar os grupos a serem validados pelo tipo de evento usando as seguintes propriedades:
 - `javax.persistence.validation.group.pre-persist`,
 - `javax.persistence.validation.group.pre-update`
 - `javax.persistence.validation.group.pre-remove`.

Os valores dessas propriedades são os nomes de classe inteiramente qualificados, separados por vírgula dos grupos a serem validados.

Os grupos de validação são um novo recurso de especificação Bean Validation. Caso não queira tirar proveito das vantagens desse novo recurso, você não precisa fazer alterações quando migrar para o Hibernate Validator 4.

- c. É possível desativar a validação baseada no ciclo de vida configurando a propriedade **`javax.persistence.validation.mode`** como **`none`**. Outros valores válidos para esta propriedade são **`auto`** (padrão), **`callback`** e **`ddl`**.

3. Configure seu aplicativo para usar validação manual

- a. Caso queira controlar manualmente a validação, é possível criar um **`Validator`** segundo uma das seguintes maneiras:
 - Crie uma instância **`Validator`** a partir de **`ValidatorFactory`** usando o método **`getValidator()`**.
 - Injete as instâncias do **`Validator`** em seu EJB, bean CDI ou qualquer outro recurso Java EE injetável.
- b. É possível usar o **`ValidatorContext`** devolvido pelo **`ValidatorFactory.usingContext()`** para personalizar a instância do seu Validador. Ao usar esta API, você pode configurar um **`MessageInterpolator`**, **`TraversableResolver`** e **`ConstraintValidatorFactory`** personalizados. Essas interfaces estão especificadas na especificação Bean Validation e são novas para o Hibernate Validator 4.

4. Modifique o código para usar as novas restrições do Bean Validation

As novas restrições de validação de nível Bean exigem alterações de código, quando você migra para o Hibernate Validator 4.

- a. Para fazer a atualização para o Hibernate Validator 4, você precisa usar as restrições nos seguintes pacotes:
 - **`javax.validation.constraints`**
 - **`org.hibernate.validator.constraints`**
- b. Todas as restrições que existiam no Hibernate Validator 3 continuam disponíveis no Hibernate Validator 4. Para usá-las, é necessário importar a classe especificada e, em alguns casos, alterar o nome ou o tipo de parâmetro de restrição.

5. Use restrições personalizadas

No Hibernate Validator 3, uma restrição personalizada necessitava de implementar a interface **`org.hibernate.validator.Validator`**. No Hibernate Validator 4, é necessário implementar a interface **`javax.validation.ConstraintValidator`**. Essa interface contém os mesmos métodos **`initialize()`** e **`isValid()`** da interface anterior, no entanto, o método de assinatura foi alterado. Além disso, a alteração **`DDL`** não possui mais suporte no Hibernate Validator 4.

[Reportar um erro](#)

3.2.3. Alterações em JSF

3.2.3.1. Habilitação de Aplicativos para Usar Versões Antigas do JSF

Sumário

Se o seu aplicativo usa uma versão mais antiga do JSF, não será necessária a atualização para a versão JSF 2.0. Ao invés disso, é possível criar um arquivo **`jboss-deployment-structure.xml`** para solicitar que o JBoss EAP 6 use o JSF 1.2 ao invés do JSF 2.0 com a implantação do seu

aplicativo. Este descritor de implantação específico do JBoss é usado para controlar o carregador de classe e está localizado no diretório **META-INF/** ou **WEB-INF/** do seu WAR, ou no diretório **META-INF/** do seu EAR.

Segue abaixo um exemplo de um arquivo **jboss-deployment-structure.xml** que adiciona uma dependência para o módulo JSF 1.2 e remove ou previne o carregamento automático do módulo JSF 2.0.

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="com.sun.jsf-impl" slot="1.2" export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
      <module name="com.sun.jsf-impl" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="javax.faces.api" slot="1.2"/>
      <module name="com.sun.jsf-impl" slot="1.2"/>
    </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

[Reportar um erro](#)

3.2.4. Alterações dos Serviços Web

3.2.4.1. Alterações dos Serviços Web

O JBoss EAP 6 inclui suporte para a implantação dos pontos de extremidade do Serviço Web JAX-WS. Este suporte é fornecido pelo JBossWS. Para mais informações sobre os Serviços Web, consulte o capítulo *Serviços Web JAX-WS* no *Guia de Desenvolvimento* para o JBoss EAP 6 em https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

O JBossWS 4 inclui as seguintes alterações que podem impactar a sua migração.

Alterações JBossWS API

Os componentes Common e SPI foram reformulados no JBossWS 4. A tabela a seguir lista a API e as alterações de empacotamento que podem afetar a migração do seu aplicativo.

Tabela 3.8. Alterações JBossWS API

JAR Antigo	Pacote Antigo	Novo JAR	Novo Pacote
JBossWS SPI	org.jboss.wsf.spi.annotation.*	JBossWS API	org.jboss.ws.api.annotation.*
JBossWS SPI	org.jboss.wsf.spi.binding.*	JBossWS API	org.jboss.ws.api.binding.*

JAR Antigo	Pacote Antigo	Novo JAR	Novo Pacote
JBossWS SPI	org.jboss.wsf.spi.management.recording.*	JBossWS API	org.jboss.ws.api.monitoring.*
JBossWS SPI	org.jboss.wsf.spi.tools.*	JBossWS API	org.jboss.ws.api.tools.*
JBossWS SPI	org.jboss.wsf.spi.tools.ant.*	JBossWS API	org.jboss.ws.tools.ant.*
JBossWS SPI	org.jboss.wsf.spi.tools.cmd.*	JBossWS API	org.jboss.ws.tools.cmd.*
JBossWS SPI	org.jboss.wsf.spi.util.ServiceLoader	JBossWS API	org.jboss.ws.api.util.ServiceLoader
JBossWS Common	org.jboss.wsf.common.*	JBossWS API	org.jboss.ws.common.*
JBossWS Common	org.jboss.wsf.common.handler.*	JBossWS API	org.jboss.ws.api.handler.*
JBossWS Common	org.jboss.wsf.common.addressing.*	JBossWS API	org.jboss.ws.api.addressing.*
JBossWS Common	org.jboss.wsf.common.DOMUtils	JBossWS API	org.jboss.ws.api.util.DOMUtils
JBossWS Native	org.jboss.ws.annotation.EndpointConfig	JBossWS API	org.jboss.ws.api.annotation.EndpointConfig
JBossWS Framework	org.jboss.wsf.framework.invocation.RecordingServerHandler	JBossWS Common	org.jboss.ws.common.invocation.RecordingServerHandler

Anotação @WebContext

No JBossWS 3.4.x, esta anotação foi empacotada como **org.jboss.wsf.spi.annotation.WebContext** no JBossWS SPI JAR. No JBossWS 4.0, esta anotação foi movida para **org.jboss.ws.api.annotation.WebContext** no JBossWS API JAR. Caso o seu aplicativo inclua a dependência obsoleta, você deve substituir as importações e as dependências no código fonte do seu aplicativo e compilá-lo ao novo JBossWS API JAR.

Há também uma alteração de um atributo que não é compatível com versões anteriores. O atributo **String[] virtualHosts** foi alterado para **String virtualHost**. No JBoss EAP 6, você pode especificar apenas um host virtual por implantação. Se diversos serviços web usarem a anotação **@WebContext**, o valor do virtualHost deve ser idêntico a todos os pontos de extremidade definidos no arquivo de implantação.

Configuração do Ponto de Extremidade

O JBossWS 4.0 fornece uma integração da pilha dos Serviços Web JBoss com a maioria dos módulos do Apache CXF. A camada de integração permite o uso de APIs de serviços web padrões, incluindo JAX-WS. Ela também permite o uso dos recursos avançados do Apache CX na parte

superior do contêiner do JBoss EAP 6 sem exigir instalação ou configuração complexa.

O subsistema **webservice** na configuração de domínio do JBoss EAP 6 inclui configurações do ponto de extremidade predefinidas. Você pode definir também as suas próprias configurações adicionais do ponto de extremidade. A anotação

@org.jboss.ws.api.annotation.EndpointConfig é usada para fazer referência a uma dada configuração do ponto de extremidade.

Consulte o capítulo *JAX-WS Web Services* no Guia de Desenvolvimento (em inglês, Development Guide) para o JBoss EAP 6 em

https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/ para mais informações sobre como configurar os pontos de extremidade do serviço web no servidor JBoss.

jboss-webservices.xml Descritor de Implantação

O JBossWS 4.0 introduz um novo descritor de implantação para configurar os serviços web. O arquivo **jboss-webservices.xml** fornece informações adicionais para a implantação dada e substitui parcialmente o arquivo **jboss.xml** obsoleto.

Para as implantações do serviço web EJB, o local esperado do arquivo do descritor **jboss-webservices.xml** está no diretório **META-INF/**. Para os pontos de extremidade dos serviços web EJB e POJO agrupados no arquivo WAR, o local esperado do arquivo **jboss-webservices.xml** está no diretório **WEB-INF/**.

Segue um exemplo do arquivo de um descritor **jboss-webservices.xml** e uma tabela descrevendo os elementos.

```
<webservicess>
  <context-root>foo</context-root>
  <config-name>Standard WSSecurity Endpoint</config-name>
  <config-file>META-INF/custom.xml</config-file>
  <property>
    <name>prop.name</name>
    <value>prop.value</value>
  </property>
  <port-component>
    <ejb-name>TestService</ejb-name>
    <port-component-name>TestServicePort</port-component-name>
    <port-component-uri>/*</port-component-uri>
    <auth-method>BASIC</auth-method>
    <transport-guarantee>NONE</transport-guarantee>
    <secure-wsdl-access>true</secure-wsdl-access>
  </port-component>
  <webservice-description>
    <webservice-description-name>TestService</webservice-
description-name>
    <wsdl-publish-location>file:///bar/foo.wsdl</wsdl-publish-
location>
  </webservice-description>
</webservicess>
```

Tabela 3.9. Descrição do Elemento do Arquivo jboss-webservice.xml

Nome do Elemento	Descrição
contexto-raiz	Usado para personalizar a raiz de contexto da implantação dos serviços web.
nome de configuração arquivo de configuração	Usado para associar uma implantação do ponto de extremidade com uma dada configuração do ponto de extremidade. As configurações do ponto de extremidade são especificadas no arquivo de configuração referenciado ou no subsistema webservices da configuração de domínio.
propriedade	Usado para configurar pares de valor de nome de propriedade simples para configurar o comportamento de pilha do serviço web.
porta-componente	Usado para personalizar o URI de destino do ponto de extremidade EJB ou para configurar as propriedades relacionadas com a segurança.
descrição do serviço web	Usado para personalizar ou substituir o local publicado WSDL do serviço web.

[Reportar um erro](#)

3.2.5. Alterações em JAX-RS e RESTEasy

3.2.5.1. Configuração das Alterações em JAX-RS e RESTEasy

O JBoss EAP 6 configura automaticamente o RESTEasy para que você não precise configurá-lo. Portanto, toda a configuração RESTEasy existente de seu arquivo **web.xml** deve ser removida por completo e substituída por uma das três opções abaixo:

1. Subclassifique **javax.ws.rs.core.Application** e use a anotação **@ApplicationPath**.

Esta é a opção mais fácil e não requer qualquer configuração xml. Simplesmente crie uma subclasse **javax.ws.rs.core.Application** em seu aplicativo e anote-a com o caminho onde deseja disponibilizar as suas classes JAX-RS. Por exemplo:

```
@ApplicationPath("/mypath")
public class MyApplication extends Application {
}
```

No exemplo acima, os seus recursos JAX-RS estão disponíveis no caminho **/MY_WEB_APP_CONTEXT/mypath/**.



NOTA

Observe que o caminho deve ser especificado como **/mypath** e não, **/mypath/***, sem barra ou asterisco.

2. Subclassifique **javax.ws.rs.core.Application** e use o arquivo **web.xml** para configurar o mapeamento JAX-RS.

Caso não deseja usar a anotação `@ApplicationPath`, ainda será necessário subclassificar `javax.ws.rs.core.Application`. E, então, será necessário configurar o mapeamento JAX-RS no arquivo `web.xml`. Por exemplo:

```
public class MyApplication extends Application {
}

<servlet-mapping>
  <servlet-name>com.acme.MyApplication</servlet-name>
  <url-pattern>/hello/*</url-pattern>
</servlet-mapping>
```

No exemplo acima, os seus recursos JAX-RS estão disponíveis no caminho `/MY_WEB_APP_CONTEXT/hello`.



NOTA

É possível usar também esta abordagem para substituir um caminho de aplicativo que foi configurado usando a anotação `@ApplicationPath`.

3. Modifique o arquivo `web.xml`.

Caso deseja subclassificar o `Application`, é possível configurar o mapeamento JAX-RS no arquivo `web.xml`, como a seguir:

```
<servlet-mapping>
  <servlet-name>javax.ws.rs.core.Application</servlet-name>
  <url-pattern>/hello/*</url-pattern>
</servlet-mapping>
```

No exemplo acima, os seus recursos JAX-RS estão disponíveis no caminho `/MY_WEB_APP_CONTEXT/hello`.



NOTA

Quando esta opção é escolhida, é necessário apenas adicionar o mapeamento. Não é necessário adicionar o servlet correspondente. O servidor é responsável por adicionar o servlet correspondente automaticamente.

[Reportar um erro](#)

3.2.6. Alterações no Realm de Segurança LDAP

3.2.6.1. Configuração das Alterações no Realm de Segurança LDAP

No JBoss EAP 5, o realm de segurança LDAP foi configurado em um elemento `<application-policy>` no arquivo `login-config.xml`. No JBoss EAP 6, o realm de segurança LDA é configurado no elemento `<security-domain>` no arquivo de configuração do servidor. Para um servidor autônomo, este é o arquivo `standalone/configuration/standalone.xml`. Caso você esteja executando o seu servidor em um domínio gerenciado, este é o arquivo `domain/configuration/domain.xml`.

Segue abaixo um exemplo da configuração de realm de segurança LDAP no arquivo **login-config.xml** do JBoss EAP 5:

```
<application-policy name="mcp_ldap_domain">
  <authentication>
    <login-module code="org.jboss.security.auth.spi.LdapExtLoginModule"
flag="required">
      <module-option
name="java.naming.factory.initial">com.sun.jndi.ldap.LdapCtxFactory</modul
e-option>
      <module-option
name="java.naming.security.authentication">simple</module-option>
      ....
    </login-module>
  </authentication>
</application-policy>
```

Segue abaixo um exemplo da configuração LDAP no arquivo de configuração do servidor no JBoss EAP 6:

```
<subsystem xmlns="urn:jboss:domain:security:1.2">
  <security-domains>
    <security-domain name="mcp_ldap_domain" cache-type="default">
      <authentication>
        <login-module code="org.jboss.security.auth.spi.LdapLoginModule"
flag="required">
          <module-option name="java.naming.factory.initial"
value="com.sun.jndi.ldap.LdapCtxFactory"/>
          <module-option name="java.naming.security.authentication"
value="simple"/>
          ...
        </login-module>
      </authentication>
    </security-domain>
  </security-domains>
</subsystem>
```

NOTA

O analisador XML foi alterado no JBoss EAP 6. No JBoss EAP 5, as opções de módulo eram especificadas como conteúdo de elemento, como por exemplo:

```
<module-option
name="java.naming.factory.initial">com.sun.jndi.ldap.LdapCtxFac
tory</module-option>
```

Agora, as opções de módulo devem ser especificadas como atributos de elemento com "value=", conforme abaixo:

```
<module-option name="java.naming.factory.initial"
value="com.sun.jndi.ldap.LdapCtxFactory"/>
```

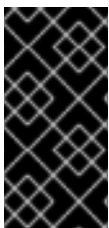
3.2.7. Alterações em HornetQ

3.2.7.1. HornetQ e NFS

Na maioria das vezes, NFS não é um método apropriado para o armazenamento de dados JMS para uso com HornetQ, ao utilizar NIO como um tipo de diário, devido à maneira como o mecanismo de bloqueio síncrono funciona. No entanto, NFS pode ser usado em determinadas circunstâncias, apenas nos servidores do Red Hat Enterprise Linux. Isto é devido à implantação NFS usada pelo Red Hat Enterprise Linux.

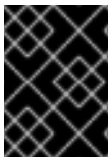
A implementação NFS do Red Hat Enterprise Linux suporta ambas E/S direta (abrindo arquivos com o sinalizador definido `O_DIRECT`) e E/S assíncrona baseada em kernel. Com esses dois recursos presentes, é possível usar NFS como uma opção de armazenamento compartilhado, sob regras de configuração estritas:

- O cache do cliente NFS do Red Hat Enterprise Linux deve ser desabilitado.



IMPORTANTE

O log do servidor deve ser verificado após o JBoss EAP 6 ser iniciado para certificar-se de que a biblioteca nativa foi carregada com sucesso e que o tipo de diário `ASYNCIO` está sendo usado. Caso a biblioteca nativa não carregue, HornetQ falhará com o tipo de diário `NIO` e isto será mencionado no log do servidor.



IMPORTANTE

A biblioteca nativa que implementa a E/S assíncrona requer que **libaio** seja instalado no sistema do Red Hat Enterprise Linux, onde o JBoss EAP 6 está sendo executado.

[Reportar um erro](#)

3.2.7.2. Configuração de uma Ponte JMS para a Migração de Mensagens JMS Existentes ao JBoss EAP 6

O JBoss EAP 6 substituiu o JBoss Messaging pelo HornetQ como a implementação JMS padrão. A maneira mais fácil de migrar as mensagens JMS de um ambiente para outro é usando uma ponte JMS. A função da ponte JMS é consumir mensagens de um destino JMS fonte e enviá-las a um destino JMS. Você pode configurar e implantar uma ponte JMS ao servidor JBoss EAP 5.x ou JBoss EAP 6.1 ou a um servidor mais recente.

Consulte [Seção 3.2.7.3, “Criação de uma Ponte JMS”](#) para mais detalhes sobre como migrar mensagens JMS do JBoss EAP 5.x para o JBoss EAP 6.x.

[Reportar um erro](#)

3.2.7.3. Criação de uma Ponte JMS

Sumário

Uma ponte JMS consome mensagens a partir de um tópico ou fila JMS fonte e as envia a um tópico ou fila JMS destino, que está normalmente em um servidor diferente. Isto pode ser usado como ponte para as mensagens entre quaisquer servidores JMS, contando que elas sejam compatíveis com o JMS 1.1.

Os recursos JMS de destino e fonte são pesquisados usando o JNDI e as classes de cliente para a pesquisa JNDI devem ser empacotadas em um módulo. O nome do módulo é, então, declarado na configuração da ponte JMS.

Procedimento 3.18. Criação de uma Ponte JMS

Este procedimento demonstra como configurar uma ponte JMS para a migração de mensagens de um servidor do JBoss EAP 5.x para um servidor do JBoss EAP 6.

1. Configuração da Ponte no Servidor do JBoss EAP 5.x Fonte

Para evitar conflitos nas classes entre as versões, você deve seguir as seguintes etapas para configurar a ponte JMS no JBoss EAP 5.x. Os nomes da ponte e do diretório SAR são arbitrários e podem ser alterados, caso prefira.

- a. Crie um subdiretório no diretório de implantação do JBoss EAP 5 para conter o SAR, por exemplo: **EAP5_HOME/server/PROFILE_NAME/deploy/myBridge.sar/**.
- b. Crie um subdiretório nomeado **META-INF** no **EAP5_HOME/server/PROFILE_NAME/deploy/myBridge.sar/**.
- c. Crie um arquivo **jboss-service.xml** no diretório **EAP5_HOME/server/PROFILE_NAME/deploy/myBridge.sar/META-INF/**. Ele deve conter informações semelhantes ao exemplo a seguir.

```
<server>
  <loader-repository>
    com.example:archive=unique-archive-name
    <loader-repository-
config>java2ParentDelegation=false</loader-repository-config>
  </loader-repository>

  <!-- JBoss EAP 6 JMS Provider -->
  <mbean code="org.jboss.jms.jndi.JMSProviderLoader"
name="jboss.messaging:service=JMSProviderLoader,name=EnterpriseAp
plicationPlatform6JMSProvider">
    <attribute
name="ProviderName">EnterpriseApplicationPlatform6JMSProvider</at
tribute>
    <attribute
name="ProviderAdapterClass">org.jboss.jms.jndi.JNDIProviderAdapte
r</attribute>
    <attribute
name="FactoryRef">jms/RemoteConnectionFactory</attribute>
    <attribute
name="QueueFactoryRef">jms/RemoteConnectionFactory</attribute>
    <attribute
name="TopicFactoryRef">jms/RemoteConnectionFactory</attribute>
    <attribute name="Properties">

    java.naming.factory.initial=org.jboss.naming.remote.client.Initia
lContextFactory

    java.naming.provider.url=remote://EnterpriseApplicationPlatform6h
ost:4447
    java.naming.security.principal=jbossuser
    java.naming.security.credentials=jbosspass
```

```

        </attribute>
    </mbean>

    <mbean code="org.jboss.jms.server.bridge.BridgeService"
name="jboss.jms:service=Bridge,name=MyBridgeName" xmbean-
dd="xmdesc/Bridge-xmbean.xml">
        <depends optional-attribute-
name="SourceProviderLoader">jboss.messaging:service=JMSProviderLo
ader,name=JMSProvider</depends>
        <depends optional-attribute-
name="TargetProviderLoader">jboss.messaging:service=JMSProviderLo
ader,name=EnterpriseApplicationPlatform6JMSProvider</depends>
        <attribute
name="SourceDestinationLookup">/queue/A</attribute>
        <attribute
name="TargetDestinationLookup">jms/queue/test</attribute>
        <attribute name="QualityOfServiceMode">1</attribute>
        <attribute name="MaxBatchSize">1</attribute>
        <attribute name="MaxBatchTime">-1</attribute>
        <attribute name="FailureRetryInterval">60000</attribute>
        <attribute name="MaxRetries">-1</attribute>
        <attribute name="AddMessageIDInHeader">false</attribute>
        <attribute name="TargetUsername">jbossuser</attribute>
        <attribute name="TargetPassword">jbosspass</attribute>
    </mbean>
</server>

```



NOTA

O elemento **load-repository** está presente para garantir que o SAR tenha um carregador de classe isolado. Além disso, observe que tanto a pesquisa JNDI e quanto a ponte "destino" incluem credenciais de segurança para o usuário "jbossuser" com a senha "jbosspass". Isto acontece pelo fato do JBoss EAP 6 ser protegido por padrão. O usuário nomeado "jbossuser" com a senha "jbosspass" foi criado em **ApplicationRealm** com a função **guest** usando o script **EAP_HOME/bin/add_user.sh**.

- d. Copie os seguintes JARs a partir do diretório

EAP_HOME/modules/system/layers/base/ no diretório

EAP5_HOME/server/PROFILE_NAME/deploy/myBridge.sar/. Substitua cada **VERSION_NUMBER** pelo número da versão vigente na sua distribuição do JBoss EAP 6.

- **org/hornetq/main/hornetq-core-VERSION_NUMBER.jar**
- **org/hornetq/main/hornetq-jms-VERSION_NUMBER.jar**
- **org/jboss/ejb-client/main/jboss-ejb-client-VERSION_NUMBER.jar**
- **org/jboss/logging/main/jboss-logging-VERSION_NUMBER.jar**
- **org/jboss/logmanager/main/jboss-logmanager-VERSION_NUMBER.jar**
- **org/jboss/marshalling/main/jboss-marshalling-VERSION_NUMBER.jar**

- `org/jboss/marshalling/river/main/jboss-marshalling-river-VERSION_NUMBER.jar`
- `org/jboss/remote-naming/main/jboss-remote-naming-VERSION_NUMBER.jar`
- `org/jboss/remoting3/main/jboss-remoting-VERSION_NUMBER.jar`
- `org/jboss/sasl/main/jboss-sasl-VERSION_NUMBER.jar`
- `org/jboss/netty/main/netty-VERSION_NUMBER.jar`
- `org/jboss/remoting3/remote-jmx/main/remoting-jmx-VERSION_NUMBER.jar`
- `org/jboss/xnio/main/xnio-api-VERSION_NUMBER.jar`
- `org/jboss/xnio/nio/main.xnio-nio-VERSION_NUMBER.jar`



NOTA

Não copie simplesmente `EAP_HOME/bin/client/jboss-client.jar`, pois as classes API javax entrarão em conflito com aquelas no JBoss EAP 5.x.

2. Configuração da Ponte no Servidor JBoss EAP 6 de Destino

No JBoss EAP 6.1 e versões mais recentes, a ponte JMS pode ser usada para transferir as mensagens a partir de qualquer servidor compatível com o JMS 1.1. Uma vez que os recursos JMS fonte e destino são pesquisados usando JNDI, as classes de pesquisa JNDI do provedor do sistema de mensagem fonte, ou agente de mensagem, devem ser empacotadas em um módulo JBoss. As etapas a seguir usam o agente de mensagem fictício 'MyCustomMQ' como um exemplo.

- a. Criação do módulo JBoss para o provedor do sistema de mensagem.
 - i. Crie uma estrutura de diretório sob `EAP_HOME/modules/system/layers/base/` para o novo módulo. O subdiretório `main/` conterá o cliente JARs e o arquivo `module.xml`. Segue abaixo um exemplo da estrutura do diretório criada para o provedor do sistema de mensagem MyCustomMQ:
`EAP_HOME/modules/system/layers/base/org/mycustommq/main/`
 - ii. No subdiretório `main/`, crie um arquivo `module.xml` contendo a definição do módulo para o provedor do sistema de mensagem. Segue abaixo um exemplo do `module.xml` criado para o provedor do sistema de mensagem MyCustomMQ.

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.1" name="org.mycustommq">
  <properties>
    <property name="jboss.api" value="private"/>
  </properties>

  <resources>
    <!-- Insert resources required to connect to the
    source or target -->
    <resource-root path="mycustommq-1.2.3.jar" />
  </resources>
</module>
```

```

        <resource-root path="mylogapi-0.0.1.jar" />
    </resources>

    <dependencies>
        <!-- Add the dependencies required by JMS Bridge code
-->
        <module name="javax.api" />
        <module name="javax.jms.api" />
        <module name="javax.transaction.api"/>
        <!-- Add a dependency on the org.hornetq module since
we send -->
        <!-- messages to the HornetQ server embedded in the
local EAP instance -->
        <module name="org.hornetq" />
    </dependencies>
</module>

```

- iii. Copie os JARs do provedor do sistema de mensagem necessários para a pesquisa JNDI dos recursos fonte para o subdiretório **main/** do módulo. A estrutura do diretório para o módulo MyCustomMQ deve parecer-se com o seguinte.

```

modules/
  -- system
    -- layers
      -- base
        -- org
          -- mycustommq
            -- main
              -- mycustommq-1.2.3.jar
              -- mylogapi-0.0.1.jar
              -- module.xml

```

- b. Configuração da ponte JMS no subsistema **messaging** do servidor JBoss EAP 6.

- i. Antes de começar, interrompa o servidor e realize o backup dos arquivos de configuração do servidor atual. Caso você esteja executando um servidor autônomo, este é o arquivo **EAP_HOME/standalone/configuration/standalone-full-ha.xml**. Caso você esteja executando um domínio gerenciado, realize o backup de ambos os arquivos **EAP_HOME/domain/configuration/domain.xml** e **EAP_HOME/domain/configuration/host.xml**.
- ii. Adicione o elemento **jms-bridge** ao subsistema **messaging** no arquivo de configuração do servidor. Os elementos **source** e **target** fornecem os nomes dos recursos JMS usados para as pesquisas JNDI. Se as credenciais **user** e **password** estiverem especificadas, elas serão passadas como argumentos quando a conexão JMS for criada.

Segue abaixo um exemplo do elemento **jms-bridge** configurado para o fornecedor do sistema de mensagem MyCustomMQ:

```

<subsystem xmlns="urn:jboss:domain:messaging:1.3">
    ...
    <jms-bridge name="myBridge" module="org.mycustommq">
        <source>
            <connection-factory name="ConnectionFactory"/>

```

```

        <destination name="sourceQ"/>
        <user>user1</user>
        <password>pwd1</password>
        <context>
            <property key="java.naming.factory.initial"
value="org.mycustommq.jndi.MyCustomMQInitialContextFactory"/>
            <property key="java.naming.provider.url"
value="tcp://127.0.0.1:9292"/>
        </context>
    </source>
    <target>
        <connection-factory name="java:/ConnectionFactory"/>
        <destination name="/jms/targetQ"/>
    </target>
    <quality-of-service>DUPLICATES_OK</quality-of-service>
    <failure-retry-interval>500</failure-retry-interval>
    <max-retries>1</max-retries>
    <max-batch-size>500</max-batch-size>
    <max-batch-time>500</max-batch-time>
    <add-messageID-in-header>true</add-messageID-in-header>
</jms-bridge>
</subsystem>

```

No exemplo acima, as propriedades JNDI são definidas no elemento **context** para **source**. Caso o elemento **context** esteja omitido, como no exemplo acima **target**, os recursos JMS serão pesquisados na instância local.

[Reportar um erro](#)

3.2.7.4. Migração do seu Aplicativo para o Uso do HornetQ como o Provedor JMS

O JBoss Messaging não está mais incluído no JBoss EAP 6. Se o seu aplicativo utiliza JBoss Messaging como um fornecedor do sistema de mensagem, você precisará substituir o código do JBoss Messaging pelo HornetQ.

Procedimento 3.19. Antes de começar

1. Desligue o cliente e o servidor
2. Realize uma cópia de backup dos dados do JBoss Messaging. Os dados da mensagem são armazenados em um banco de dados em tabelas com o prefixo **JBM_**.

Procedimento 3.20. Alteração do seu provedor para HornetQ

1. Transferência de configurações

Transfira as configurações existentes do JBoss Messaging para configuração do JBoss EAP 6. As configurações a seguir podem ser encontradas nos descritores de implantação localizados no servidor JBoss Messaging:

- o Configuração do Serviço de Fábricas de Conexões

Esta configuração descreve as fábricas de conexões JMS implantadas com o servidor JBoss Messaging. O JBoss Messaging configura as fábricas de conexões em um arquivo nomeado **connection-factories-service.xml**, que está localizado no diretório de implantação do servidor do aplicativo.

- Configuração de Destino

Esta configuração descreve os tópicos e as filas JMS implantadas com o servidor JBoss Messaging. Por padrão, o JBoss Messaging configura os destinos em um arquivo nomeado **destinations-service.xml** que está localizado no diretório de implantação do servidor do aplicativo.

- Configuração do Serviço de Ponte de Mensagens

Esta configuração inclui os serviços de ponte implantados com o servidor JBoss Messaging. Nenhuma ponte é implantada por padrão, portanto o nome do arquivo de implantação varia, dependendo da instalação do seu JBoss Messaging.

2. Modificação do código do seu aplicativo

Se o código do aplicativo usa o JMS padrão, não são necessárias alterações ao código. No entanto, se o aplicativo for conectar a um cluster, você deve revisar cuidadosamente a documentação HornetQ sobre a semântica do clustering. O clustering está fora do escopo da especificação JMS e HornetQ e JBoss Messaging adotaram abordagens bastante diferentes em suas respectivas implementações da funcionalidade do clustering.

Caso o aplicativo utilize recursos específicos ao JBoss Messaging, você deve modificar o código para usar os recursos equivalentes disponíveis no HornetQ.

Para mais informações sobre como configurar o sistema de mensagens com HornetQ, consulte [Seção 3.2.7.5, “Configuração do Sistema de Mensagens com HornetQ”](#).

3. Migração de mensagens existentes

Mova quaisquer mensagens do banco de dados do JBoss Messaging para o diário HornetQ usando a ponte JMS. As instruções para a configuração da ponte JMS podem ser encontradas aqui: [Seção 3.2.7.2, “Configuração de uma Ponte JMS para a Migração de Mensagens JMS Existentes ao JBoss EAP 6”](#).

[Reportar um erro](#)

3.2.7.5. Configuração do Sistema de Mensagens com HornetQ

O método recomendado de configuração do sistema de mensagens no JBoss EAP 6 é o Console de Gerenciamento ou o Gerenciamento CLI. É possível fazer alterações persistentes com qualquer uma dessas ferramentas de gerenciamento sem precisar editar manualmente os arquivos de configuração **standalone.xml** ou **domain.xml**. É útil, no entanto, familiarizar-se com os componentes do sistema de mensagens dos arquivos de configuração padrão, onde exemplos de documentação usando as ferramentas de gerenciamento fornecem trechos do arquivo de configuração para referência.

[Reportar um erro](#)

3.2.8. Alterações no Clustering

3.2.8.1. Realização de Alterações ao seu Aplicativo para Clustering

1. Inicie o JBoss EAP 6 com o clustering habilitado

Para habilitar o clustering no JBoss EAP 5.x, era preciso iniciar as instâncias do seu servidor usando o perfil **all** ou alguma derivação disto, como por exemplo:

```
$ EAP5_HOME/bin/run.sh -c all
```

No JBoss EAP 6, o método para a habilitação do clustering depende se os servidores estão executando em um domínio gerenciado ou se são autônomos.

a. **Habilitação do clustering para os servidores executando em um domínio gerenciado**

Para habilitar o clustering para os servidores iniciados usando o controlador de domínio, atualize o seu **domain.xml** e designe um grupo de servidor para usar o perfil **ha** e o grupo de associação de soquete **ha-sockets**. Por exemplo:

```
<server-groups>
  <server-group name="main-server-group" profile="ha">
    <jvm name="default">
      <heap size="64m" max-size="512m"/>
    </jvm>
    <socket-binding-group ref="ha-sockets"/>
  </server-group>
</server-groups>
```

b. **Habilitação do cluster para servidores autônomos**

Para habilitar o clustering para os servidores autônomos, inicie o servidor usando o arquivo de configuração apropriado como a seguir:

```
$ EAP_HOME/bin/standalone.sh --server-config=standalone-ha.xml -
Djboss.node.name=UNIQUE_NODE_NAME
```

2. **Especifique o endereço vinculado**

No JBoss EAP 5.x, você normalmente indicaria o endereço vinculado usado para clustering utilizando o argumento da linha de comando **-b**, conforme abaixo:

```
$ EAP5_HOME/bin/run.sh -c all -b 192.168.0.2
```

O JBoss EAP 6 associa os soquetes às interfaces e aos endereços IP contidos nos elementos **<interfaces>** nos arquivos **standalone.xml**, **domain.xml** e **host.xml**. As configurações padrão que são enviadas junto com o JBoss EAP incluem duas configurações de interface:

```
<interfaces>
  <interface name="management">
    <inet-address
value="${jboss.bind.address.management:127.0.0.1}"/>
  </interface>
  <interface name="public">
    <inet-address value="${jboss.bind.address:127.0.0.1}"/>
  </interface>
</interfaces>
```

Essas configurações de interface usam os valores das propriedades do sistema **jboss.bind.address.management** e **jboss.bind.address**. Caso essas propriedades do sistema não sejam determinadas, o padrão **127.0.0.1** é usado para cada valor.

O endereço vinculado também pode ser especificado como um argumento da linha de comando quando você inicia o servidor ou pode ser explicitamente definido dentro do arquivo de configuração do servidor JBoss EAP 6.

- o Especifique o argumento vinculado na linha de comando quando iniciar o servidor autônomo do JBoss EAP.

Segue abaixo um exemplo de como especificar o endereço vinculado na linha de comando para um servidor autônomo:

```
EAP_HOME/bin/standalone.sh -Djboss.bind.address=127.0.0.1
```



NOTA

É possível usar o argumento **-b**, que é um atalho para **-Djboss.bind.address=127.0.0.1**:

```
EAP_HOME/bin/standalone.sh -b=127.0.0.1
```

O formato da sintaxe do JBoss EAP 5 ainda possui suporte também:

```
EAP_HOME/bin/standalone.sh -b 127.0.0.1
```

Observe que o argumento **-b** apenas altera a interface **public**. Isto não afeta a interface **management**.

- o Especifique o endereço vinculado no arquivo de configuração do servidor.

Para os servidores executando em um domínio gerenciado, especifique os endereços vinculados no arquivo **domain/configuration/host.xml**. Para os servidores autônomos, especifique os endereços vinculados no arquivo **standalone-ha.xml**.

No exemplo a seguir, a interface **public** é especificada como a interface padrão para todos os soquetes dentro do grupo de associação de soquete **ha-sockets**.

```
<interfaces>
  <interface name="management">
    <inet-address value="192.168.0.2"/>
  </interface>
  <interface name="public">
    <inet-address value="192.168.0.2"/>
  </interface>
</interfaces>

<socket-binding-groups>
  <socket-binding-group name="ha-sockets" default-
interface="public">
    <!-- ... -->
  </socket-binding-group>
</socket-binding-groups>
```



NOTA

Caso o endereço vinculado seja especificado como um valor codificado, ao invés de uma propriedade do sistema no arquivo de configuração, não será possível substituí-lo por um argumento da linha de comando.

3. Configure `jvmRoute` para suportar `mod_jk` e `mod_proxy`

No JBoss EAP 5, o servidor web **jvmRoute** foi configurado usando uma propriedade no arquivo **server.xml**. No JBoss EAP 6, o atributo **jvmRoute** é configurado no subsistema da web do arquivo de combinação do servidor usando o atributo **instance-id**, conforme o seguinte:

```
<subsystem xmlns="urn:jboss:domain:web:1.1" default-virtual-
server="default-host" native="false" instance-id="
{JVM_ROUTE_SERVER}">
```

O **{JVM_ROUTE_SERVER}** acima deve ser substituído pela ID do servidor **jvmRoute**.

O **instance-id** pode ser determinado usando o Console de Gerenciamento.

4. Especifique a porta e o endereço multicast

No JBoss EAP 5.x, você poderia especificar a porta e o endereço multicast usados para a comunicação intra-cluster utilizando os argumentos da linha de comando **-u** e **-m**, como a seguir:

```
$ EAP5_HOME/bin/run.sh -c all -u 228.11.11.11 -m 45688
```

No JBoss EAP 6, a porta e o endereço multicast usados para a comunicação intra-cluster são definidos pela associação de soquete referenciada pela pilha do protocolo JGroups relevante, como a seguir:

```
<subsystem xmlns="urn:jboss:domain:jgroups:1.0" default-stack="udp">
  <stack name="udp">
    <transport type="UDP" socket-binding="jgroups-udp"/>
    <!-- ... -->
  </stack>
</subsystem>
```

```
<socket-binding-groups>
  <socket-binding-group name="ha-sockets" default-
interface="public">
    <!-- ... -->
    <socket-binding name="jgroups-udp" port="55200" multicast-
address="228.11.11.11" multicast-port="45688"/>
    <!-- ... -->
  </socket-binding-group>
</socket-binding-groups>
```

Se você preferir especificar a porta e o endereço multicast na linha de comando, você pode definir as portas e o endereço multicast como propriedades de sistema e, então, usar essas propriedades na linha de comando, quando você iniciar o servidor. No exemplo a seguir, **jboss.mcast.addr** é o nome da variável para o endereço multicast e **jboss.mcast.port** é o nome da variável para a porta.

```
<socket-binding name="jgroups-udp" port="55200"
multicast-address="{jboss.mcast.addr:230.0.0.4}" multicast-
port="{jboss.mcast.port:45688}"/>
```

Você pode, então, iniciar o seu servidor usando os seguintes argumentos da linha de comando:

```
$ EAP_HOME/bin/domain.sh -Djboss.mcast.addr=228.11.11.11 -
Djboss.mcast.port=45688
```

5. Utilize uma pilha de protocolo alternativa

No JBoss EAP 5.x, você poderia manipular a pilha do protocolo padrão usada para todos os serviços de clustering utilizando a propriedade do sistema `jboss.default.jgroups.stack`.

```
$ EAP5_HOME/bin/run.sh -c all -Djboss.default.jgroups.stack=tcp
```

No JBoss EAP 6, a pilha do protocolo padrão é definida pelo subsistema JGroups dentro do `domain.xml` ou `standalone-ha.xml`:

```
<subsystem xmlns="urn:jboss:domain:jgroups:1.0" default-stack="udp">
  <stack name="udp">
    <!-- ... -->
  </stack>
</subsystem>
```

6. Substitua a Replicação Buddy

O JBoss EAP 5.x usava a JBoss Cache Buddy Replication para suprimir a replicação de dados em todas as instâncias em um cluster.

No JBoss EAP 6, a Replicação Buddy foi substituída pelo cache distribuído do Infinispan, também conhecido como modo **DIST**. A distribuição é um modo de clustering poderoso que permite o Infinispan escalar linearmente à medida que mais servidores são adicionados ao cluster. Segue abaixo um exemplo de como configurar o servidor para usar o modo de cache DIST.

- a. Abra uma linha de comando e inicie o servidor com o Perfil HA ou o Perfil Completo. Por exemplo:

```
EAP_HOME/bin/standalone.sh -c standalone-ha.xml
```

- b. Abra outra linha de comando e conecte-se ao Gerenciamento CLI.

- Para o Linux, insira a seguinte linha de comando:

```
$ EAP_HOME/bin/jboss-cli.sh --connect
```

- Para o Windows, insira a seguinte linha de comando:

```
C:\>EAP_HOME\bin\jboss-cli.bat --connect
```

Você deverá encontrar a seguinte resposta:

```
Conectado ao controlador autônomo em localhost:9999
```

- c. Emita os seguintes comandos:

```
/subsystem=infinispan/cache-container=web/:write-attribute(name=default-cache,value=dist)
/subsystem=infinispan/cache-container=web/distributed-cache=dist/:write-attribute(name=owners,value=3)
:reload
```

Você deverá encontrar a seguinte resposta após cada comando:

```
"outcome" => "success"
```

Esses comandos modificam o elemento **dist** <**distributed-cache**> na configuração **web** <**cache-container**> no subsistema **infinispan** do arquivo **standalone-ha.xml**, como a seguir:

```
<cache-container name="web" aliases="standard-session-cache"
  default-cache="dist"
  module="org.jboss.as.clustering.web.infinispan">
  <transport lock-timeout="60000"/>
  <replicated-cache name="repl" mode="ASYNC" batching="true">
    <file-store/>
  </replicated-cache>
  <replicated-cache name="sso" mode="SYNC" batching="true"/>
  <distributed-cache name="dist" owners="3" l1-lifespan="0"
    mode="ASYNC" batching="true">
    <file-store/>
  </distributed-cache>
</cache-container>
```

Para mais informações, consulte o capítulo intitulado *Clustering in Web Applications* no guia *Development Guide* do JBoss EAP 6 localizado no Portal do Consumidor https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

[Reportar um erro](#)

3.2.8.2. Implantação de um HA Singleton

Sumário

O procedimento a seguir demonstra como implantar um Serviço que está encapsulado com o decorador `SingletonService` e é usado como um serviço singleton no cluster inteiro. Este serviço ativa um timer programado, que é iniciado apenas uma vez no cluster.

Procedimento 3.21. Implantação do Serviço HA Singleton

1. Grave o aplicativo do serviço HA singleton.

Segue abaixo um exemplo de um **Service** que está encapsulado com o decorador **SingletonService** a ser implantado como um serviço singleton. Um exemplo completo pode ser encontrado na seção início rápido **cluster-ha-singleton** que é enviada junto com o Red Hat JBoss Enterprise Application Platform 6. O início rápido contém todas as instruções para a construção e implantação do aplicativo.

a. Criação de um serviço.

A lista abaixo é um exemplo de um serviço:

```
package org.jboss.as.quickstarts.cluster.hasingleton.service.ejb;

import java.util.Date;
import java.util.concurrent.atomic.AtomicBoolean;

import javax.naming.InitialContext;
import javax.naming.NamingException;

import org.jboss.logging.Logger;
```

```

import org.jboss.msc.service.Service;
import org.jboss.msc.service.ServiceName;
import org.jboss.msc.service.StartContext;
import org.jboss.msc.service.StartException;
import org.jboss.msc.service.StopContext;

/**
 * @author <a href="mailto:wfink@redhat.com">Wolf-Dieter Fink</a>
 */
public class HATimerService implements Service<String> {
    private static final Logger LOG =
        Logger.getLogger(HATimerService.class);
    public static final ServiceName SINGLETON_SERVICE_NAME =
        ServiceName.JBOSS.append("quickstart", "ha", "singleton",
            "timer");

    /**
     * A flag whether the service is started.
     */
    private final AtomicBoolean started = new
        AtomicBoolean(false);

    /**
     * @return the name of the server node
     */
    public String getValue() throws IllegalStateException,
        IllegalArgumentException {
        LOG.infof("%s is %s at %s",
            HATimerService.class.getSimpleName(), (started.get() ? "started"
                : "not started"), System.getProperty("jboss.node.name"));
        return "";
    }

    public void start(StartContext arg0) throws StartException {
        if (!started.compareAndSet(false, true)) {
            throw new StartException("The service is still
started!");
        }
        LOG.info("Start HASingleton timer service '" +
            this.getClass().getName() + "'");

        final String node =
            System.getProperty("jboss.node.name");
        try {
            InitialContext ic = new InitialContext();
            ((Scheduler) ic.lookup("global/jboss-cluster-ha-
singleton-
service/SchedulerBean!org.jboss.as.quickstarts.cluster.hasingleton.
service.ejb.Scheduler")).initialize("HASingleton timer @" +
            node + " " + new Date());
        } catch (NamingException e) {
            throw new StartException("Could not initialize
timer", e);
        }
    }
}

```

```

        public void stop(StopContext arg0) {
            if (!started.compareAndSet(true, false)) {
                LOGGER.warn("The service '" +
this.getClass().getName() + "' is not active!");
            } else {
                LOGGER.info("Stop HASingleton timer service '" +
this.getClass().getName() + "'");
                try {
                    InitialContext ic = new InitialContext();
                    ((Scheduler) ic.lookup("global/jboss-cluster-ha-
singleton-
service/SchedulerBean!org.jboss.as.quickstarts.cluster.hasingleto
n.service.ejb.Scheduler")).stop();
                } catch (NamingException e) {
                    LOGGER.error("Could not stop timer", e);
                }
            }
        }
    }
}

```

b. Criação de um ativador que instale Service como um singleton clusterizado.

A lista abaixo é um exemplo de um ativador de Serviço que instala **HATimerService** como um serviço de singleton clusterizado:

```

package org.jboss.as.quickstarts.cluster.hasingleton.service.ejb;

import org.jboss.as.clustering.singleton.SingletonService;
import org.jboss.logging.Logger;
import org.jboss.msc.service.DelegatingServiceContainer;
import org.jboss.msc.service.ServiceActivator;
import org.jboss.msc.service.ServiceActivatorContext;
import org.jboss.msc.service.ServiceController;

/**
 * Service activator that installs the HATimerService as a
 * clustered singleton service
 * during deployment.
 *
 * @author Paul Ferraro
 */
public class HATimerServiceActivator implements ServiceActivator
{
    private final Logger log = Logger.getLogger(this.getClass());

    @Override
    public void activate(ServiceActivatorContext context) {
        log.info("HATimerService will be installed!");

        HATimerService service = new HATimerService();
        SingletonService<String> singleton = new
SingletonService<String>(service,
HATimerService.SINGLETON_SERVICE_NAME);
        /*
         * To pass a chain of election policies to the singleton,

```

```

for example,
    * to tell JGroups to prefer running the singleton on a
    node with a
    * particular name, uncomment the following line:
    */
    // singleton.setElectionPolicy(new
    PreferredSingletonElectionPolicy(new
    SimpleSingletonElectionPolicy(), new
    NamePreference("node2/cluster")));

    singleton.build(new
    DelegatingServiceContainer(context.getServiceTarget(),
    context.getServiceRegistry()))
        .setInitialMode(ServiceController.Mode.ACTIVE)
        .install()
    ;
}
}

```



NOTA

O exemplo de código acima utiliza uma classe, **org.jboss.as.clustering.singleton.SingletonService**, que faz parte da API privada do JBoss EAP. Uma API pública estará disponível na versão EAP 7 e a classe privada será preterida, porém estas classes serão mantidas e estarão disponíveis durante o ciclo de lançamento do EAP 6.x.

c. Criação de um Arquivo ServiceActivator

Crie um arquivo nomeado **org.jboss.msc.service.ServiceActivator** no diretório do arquivo **resources/META-INF/services/**. Adicione uma linha contendo um nome inteiramente qualificado da classe ServiceActivator criada na etapa anterior.

```

org.jboss.as.quickstarts.cluster.hasingleton.service.ejb.HATimerServiceActivator

```

d. Criação de um bean Singleton que implementa um timer a ser usado como um timer singleton no cluster inteiro.

Este bean Singleton não deve possuir uma interface remota e não deve haver referência à sua interface local por outro EJB em qualquer aplicativo. Isto evita uma pesquisa por parte de um cliente ou outro componente e garante que o SingletonService tenha controle total do Singleton.

i. Criação da interface do Agendador

```

package
org.jboss.as.quickstarts.cluster.hasingleton.service.ejb;

/**
 * @author <a href="mailto:wfink@redhat.com">Wolf-Dieter
 Fink</a>
 */
public interface Scheduler {

    void initialize(String info);
}

```

```

        void stop();
    }

```

- ii. Criação de um bean do Singleton que implementa o timer do singleton amplo com cluster.

```

package
org.jboss.as.quickstarts.cluster.hasingleton.service.ejb;

import javax.annotation.Resource;
import javax.ejb.ScheduleExpression;
import javax.ejb.Singleton;
import javax.ejb.Timeout;
import javax.ejb.Timer;
import javax.ejb.TimerConfig;
import javax.ejb.TimerService;

import org.jboss.logging.Logger;

/**
 * A simple example to demonstrate a implementation of a
 * cluster-wide singleton timer.
 *
 * @author <a href="mailto:wfink@redhat.com">Wolf-Dieter
 * Fink</a>
 */
@Singleton
public class SchedulerBean implements Scheduler {
    private static Logger LOGGER =
        Logger.getLogger(SchedulerBean.class);
    @Resource
    private TimerService timerService;

    @Timeout
    public void scheduler(Timer timer) {
        LOGGER.info("HASingletonTimer: Info=" +
            timer.getInfo());
    }

    @Override
    public void initialize(String info) {
        ScheduleExpression sexpr = new ScheduleExpression();
        // set schedule to every 10 seconds for demonstration
        sexpr.hour("*").minute("*").second("0/10");
        // persistent must be false because the timer is
        started by the HASingleton service
        timerService.createCalendarTimer(sexpr, new
            TimerConfig(info, false));
    }

    @Override
    public void stop() {
        LOGGER.info("Stop all existing HASingleton timers");
    }
}

```

```

        for (Timer timer : timerService.getTimers()) {
            LOGGER.trace("Stop HASingleton timer: " +
timer.getInfo());
            timer.cancel();
        }
    }
}

```

2. Inicie cada instância do JBoss EAP 6 com o clustering habilitado.

Para habilitar o clustering para os servidores autônomos, você deve iniciar cada servidor com o perfil **HA**, usando um nome de nó único e uma porta de deslocamento para cada instância.

- Para o Linux, use a seguinte sintaxe de comando para iniciar os servidores:

```

EAP_HOME/bin/standalone.sh --server-config=standalone-ha.xml -
Djboss.node.name=UNIQUE_NODE_NAME -Djboss.socket.binding.port-
offset=PORT_OFFSET

```

Exemplo 3.3. Inicie múltiplos servidores autônomos no Linux

```

$ EAP_HOME/bin/standalone.sh --server-config=standalone-ha.xml
-Djboss.node.name=node1
$ EAP_HOME/bin/standalone.sh --server-config=standalone-ha.xml
-Djboss.node.name=node2 -Djboss.socket.binding.port-offset=100

```

- Para o Microsoft Windows, use a seguinte sintaxe de comando para iniciar os servidores:

```

EAP_HOME\bin\standalone.bat --server-config=standalone-ha.xml -
Djboss.node.name=UNIQUE_NODE_NAME -Djboss.socket.binding.port-
offset=PORT_OFFSET

```

Exemplo 3.4. Inicie múltiplos servidores autônomos no Microsoft Windows

```

C:> EAP_HOME\bin\standalone.bat --server-config=standalone-
ha.xml -Djboss.node.name=node1
C:> EAP_HOME\bin\standalone.bat --server-config=standalone-
ha.xml -Djboss.node.name=node2 -Djboss.socket.binding.port-
offset=100

```



NOTA

Caso decida por não usar os argumentos da linha de comando, o arquivo **standalone-ha.xml** pode ser configurado para cada instância do servidor para vincular a uma interface separada.

3. Implante o aplicativo nos servidores

O comando Maven a seguir implanta o aplicativo em um servidor autônomo executando nas portas padrões.

■


```
mvn clean install jboss-as:deploy
```

Para a implantação de servidores adicionais, passe o nome do servidor. Caso esteja em um host diferente, passe o nome do host e o número da porta na linha de comando:

```
mvn clean package jboss-as:deploy -Djboss-as.hostname=localhost -  
Djboss-as.port=10099
```

Consulte a seção início rápido **cluster-ha-singleton**, que é enviada junto com o JBoss EAP 6, para mais detalhes sobre a configuração e implantação do Maven.

[Reportar um erro](#)

3.2.9. Alterações da Implantação do Estilo de Serviço

3.2.9.1. Atualização dos Aplicativos que Usam as Implantações de Estilo de Serviço

Sumário

Os MBeans faziam parte da arquitetura básica nas versões anteriores do Red Hat JBoss Enterprise Application Platform. As implantações do Arquivo de Serviço JBoss (SAR) usando os descritores de estilo de serviço específicos **jboss-service.xml** e **jboss-beans.xml** eram usadas pelo servidor do aplicativo para a criação do MBeans baseada no JBoss Beans. A arquitetura interna foi alterada no JBoss EAP 6 e não é mais baseada na arquitetura MBean JMX. Os MBeans não fazem parte da arquitetura básica mais. Eles são agora um invólucro para a API de gerenciamento.

Se o seu aplicativo utiliza descritores de implantação de estilo de serviço, ele pode continuar a funcionar no JBoss EAP 6 desde que ele dependa apenas do MBeans que o seu aplicativo definiu e não dependa dos invólucros do JBoss Management API MBean. No JBoss EAP 6, os SARs podem declarar apenas as dependências do MBean nos MBeans que foram criados por outra implantação SAR. Isto significa que, se o seu aplicativo dependia dos MBeans que o JBoss EAP criou, tais como um MBean para um EJB ou um componente de mensagem, eles deixarão de funcionar. Os únicos MBeans que você pode ficar na dependência são os MBeans que você definiu em outros **jboss-service.xml** files.

O Arquivo de Serviço JBoss (SAR) e os descritores de estilo de serviço usados em versões anteriores do JBoss EAP não fazem parte da especificação Java EE 6 e o seu uso não é recomendado no JBoss EAP 6. Recomenda-se que você modifique o seu aplicativo para a especificação Java EE 6. Para os singletons MBeans, você deve modificar o código para utilizar o Java EE6 **@Singleton**. Para mais informações sobre a criação e a implantação de serviços MBean, consulte o capítulo entitulado *JBoss MBean Services* no guia *Development Guide* para a Plataforma do Aplicativo JBoss Enterprise 6 localizada no Portal do Consumidor https://access.redhat.com/documentation/JBoss_Enterprise_Application_Platform/.

[Reportar um erro](#)

3.2.10. Alterações de Invocação Remota

3.2.10.1. Migração dos Aplicativos Implantados do JBoss EAP 5 que Realizam Invocações Remotas no JBoss EAP 6

Sumário

No JBoss EAP 5, a interface remota do EJB era vinculada ao JNDI, por padrão, sob o nome "ejbName/local" para as interfaces locais e "ejbName/remote" para as interfaces remotas. O aplicativo do cliente, então, pesquisava o bean usando "ejbName/remote".

No JBoss EAP 6, uma nova e específica API do cliente EJB foi introduzida para realizar a invocação. No entanto, caso não deseja regravar o seu código para usar a nova API, você pode modificar o código existente para uso do `ejb:BEAN_REFERENCE` para o acesso remoto aos EJBs com a seguinte sintaxe.

Para os beans sem monitorização de estado, a sintaxe `ejb:BEAN_REFERENCE` é:

```
ejb:<app-name>/<module-name>/<distinct-name>/<bean-name>!<fully-qualified-
classname-of-the-remote-interface>
```

Para beans com monitorização de estado (stateful), a sintaxe `ejb:BEAN_REFERENCE` é:

```
ejb:<app-name>/<module-name>/<distinct-name>/<bean-name>!<fully-qualified-
classname-of-the-remote-interface>?stateful
```

Os valores a serem substituídos na sintaxe acima são:

- **<app-name>** - o nome do aplicativo dos EJBs implantados. Este é tipicamente o nome ear sem o sufixo .ear, no entanto, o nome pode ser substituído no arquivo application.xml. Se o aplicativo não é implantado como um .ear, esse valor é uma cadeia vazia. Presume-se que este exemplo não foi implantado como um EAR.
- **<module-name>** - o nome do módulo dos EJBs implantados no servidor. Isto é tipicamente o nome jar da implantação EJB, sem o sufixo .jar, porém pode ser substituído usando o ejb-jar.xml. Neste exemplo, presume que os EJBs foram implantados em um jboss-ejb-remote-app.jar, portanto o nome do módulo é jboss-ejb-remote-app.
- **<distinct-name>** - um nome distinto opcional para o EJB. Este exemplo não usa um nome distinto, portanto usa-se uma cadeia vazia.
- **<bean-name>** - por padrão, é o nome de classe simples da classe de implantação do bean.
- **<fully-qualified-classname-of-the-remote-interface>** - o nome da classe inteiramente qualificado de visualização remota.

Atualização do código do cliente

Suponha que você tenha implantado o seguinte EJB sem monitorização de estado em um servidor do JBoss EAP 6. Observe que isto expõe uma visualização remota para o bean.

```
@Stateless
@Remote(RemoteCalculator.class)
public class CalculatorBean implements RemoteCalculator {

    @Override
    public int add(int a, int b) {
        return a + b;
    }

    @Override
    public int subtract(int a, int b) {
```

```

        return a - b;
    }
}

```

No JBoss EAP 5, a invocação e a pesquisa EJB do cliente foi codificada de forma parecida com isto:

```

InitialContext ctx = new InitialContext();
RemoteCalculator calculator = (RemoteCalculator)
ctx.lookup("CalculatorBean/remote");
int a = 204;
int b = 340;
int sum = calculator.add(a, b);

```

No JBoss EAP 6, usando a informação descrita acima, a invocação e a pesquisa do cliente estão codificadas da seguinte maneira:

```

final Hashtable jndiProperties = new Hashtable();
jndiProperties.put(Context.URL_PKG_PREFIXES,
"org.jboss.ejb.client.naming");
final Context context = new InitialContext(jndiProperties);
final String appName = "";
final String moduleName = "jboss-ejb-remote-app";
final String distinctName = "";
final String beanName = CalculatorBean.class.getSimpleName();
final String viewClassName = RemoteCalculator.class.getName();
final RemoteCalculator statelessRemoteCalculator = (RemoteCalculator)
context.lookup("ejb:" + appName + "/" + moduleName + "/" + distinctName +
"/" + beanName + "!" + viewClassName);

int a = 204;
int b = 340;
int sum = statelessRemoteCalculator.add(a, b);

```

Caso o seu cliente esteja acessando um EJB com monitorização de estado, você deve acrescentar "?stateful" ao final da pesquisa de contexto, como abaixo:

```

final RemoteCalculator statefulRemoteCalculator = (RemoteCalculator)
context.lookup("ejb:" + appName + "/" + moduleName + "/" + distinctName +
"/" + beanName + "!" + viewClassName + "?stateful")

```

Um exemplo completo, incluindo ambos o código do cliente e do servidor, pode ser encontrado no Início Rápido. Para mais informações, consulte *Review the Quickstart Tutorials* no capítulo intitulado *Get Started Developing Applications* no guia *Development Guide* para o JBoss EAP 6 em https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

Para mais informações sobre as invocações remotas utilizando JNDI, consulte [Seção 3.2.10.2, "Invocação Remota de um Bean de Sessão usando JNDI"](#).

[Reportar um erro](#)

3.2.10.2. Invocação Remota de um Bean de Sessão usando JNDI

Essa tarefa descreve como adicionar suporte a um cliente remoto para a invocação dos beans de sessão usando JNDI. Essa tarefa presume que o projeto esteja sendo construído usando Maven.

A seção início rápido **ejb-remote** contém os projetos Maven operantes que demonstram essa funcionalidade. Esta seção contém projetos para ambos os beans de sessão para implantação e o cliente remoto. As amostras de código abaixo foram tiradas do projeto de cliente remoto.

Essa tarefa presume que os beans de sessão não requerem autenticação.



ATENÇÃO

A Red Hat recomenda que você explicitamente desabilite o SSL a favor do TLSv1.1 ou TLSv1.2 em todos os pacotes afetados.

Pré-requisitos

Os pré-requisitos a seguir devem ser cumpridos antes da inicialização:

- Você já deve possuir um projeto Maven criado e pronto para ser usado.
- A configuração para o repositório Maven do JBoss EAP 6 já foi adicionada.
- Os beans de sessão que você deseja invocar já estão implantados.
- Os beans de sessão implantados implementam as interfaces comerciais remotas.
- As interfaces comerciais remotas dos beans de sessão estão disponíveis como uma dependência Maven. Caso as interfaces comerciais remotas estejam disponíveis apenas como um arquivo JAR, recomenda-se a adição do JAR ao seu repositório Maven como um artefato. Consulte a documentação Maven para orientações **install:install-file** sobre as direções, <http://maven.apache.org/plugins/maven-install-plugin/usage.html>
- Você precisa saber o nome do host e a porta JNDI do servidor hospedando os beans de sessão.

Para invocar um bean de sessão de um cliente remoto, você precisa primeiro configurar o projeto corretamente.

Procedimento 3.22. Adição da Configuração do Projeto Maven para Invocação Remota dos Beans de Sessão

1. Adicione as dependências do projeto necessárias

O `pom.xml` para o projeto deve ser atualizado para incluir as dependências necessárias.

2. Adicione o arquivo `jboss-ejb-client.properties`

A API do cliente EJB JBoss espera encontrar um arquivo na raiz do projeto nomeado **jboss-ejb-client.properties** que contenha a informação da conexão para o serviço JNDI. Adicione este arquivo ao diretório `src/main/resources/` do seu projeto com o seguinte conteúdo.

```
# In the following line, set SSL_ENABLED to true for SSL
remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false
remote.connections=default
```

```
# Uncomment the following line to set SSL_STARTTLS to true for SSL
#
remote.connection.default.connect.options.org.xnio.Options.SSL_START
TLS=true
remote.connection.default.host=localhost
remote.connection.default.port = 4447
remote.connection.default.connect.options.org.xnio.Options.SASL_POLI
CY_NOANONYMOUS=false
# Add any of the following SASL options if required
#
remote.connection.default.connect.options.org.xnio.Options.SASL_POLI
CY_NOANONYMOUS=false
#
remote.connection.default.connect.options.org.xnio.Options.SASL_POLI
CY_NOPLAINTEXT=false
#
remote.connection.default.connect.options.org.xnio.Options.SASL_DISA
LLOWED_MECHANISMS=JBoss-LOCAL-USER
```

Altere a porta e o nome do host para coincidir com o seu servidor. **4447** é o número de porta padrão. Para uma conexão segura, configure a linha **SSL_ENABLED** como **true** e remova os comentários da linha **SSL_STARTTLS**. A interface remota no contêiner suporta as conexões seguras e não seguras usando a mesma porta.

3. Adicione dependências para as interfaces comerciais remotas

Adicione as dependências Maven ao **pom.xml** para as interfaces comerciais remotas dos beans de sessão.

```
<dependency>
  <groupId>org.jboss.as.quickstarts</groupId>
  <artifactId>jboss-ejb-remote-server-side</artifactId>
  <type>ejb-client</type>
  <version>${project.version}</version>
</dependency>
```

Agora que o projeto foi configurado corretamente, você pode adicionar o código para acessar e invocar os beans de sessão.

Procedimento 3.23. Obtenção de um Proxy Bean usando JNDI e Invocação de Métodos do Bean

1. Maneje as exceções verificadas

Dois dos métodos usados nos códigos a seguir (**InitialContext()** e **lookup()**) possuem uma exceção verificada do tipo **javax.naming.NamingException**. Essas chamadas de método devem estar inseridas em um bloco de tentativa/captura que capture **NamingException** ou em um método que seja declarado para lançar **NamingException**. O início rápido **ejb-remote** utiliza a segunda técnica.

2. Crie um Contexto JNDI

Um objeto do Contexto JNDI fornece o mecanismo para solicitar recursos do servidor. Crie um contexto JNDI usando o seguinte código:

```
final Hashtable jndiProperties = new Hashtable();
jndiProperties.put(Context.URL_PKG_PREFIXES,
"org.jboss.ejb.client.naming");
```

```
final Context context = new InitialContext(jndiProperties);
```

As propriedades de conexão para o serviço JNDI são lidas a partir do arquivo **jboss-ejb-client.properties**.

3. Use o método `lookup()` do Contexto JNDI para obter um proxy bean

Invoque o método `lookup()` do proxy bean e passe-o o nome JNDI do bean de sessão que você necessita. Isto retornará um objeto que deve ser convertido para o tipo da interface comercial remota que contém os métodos que você deseja invocar.

```
final RemoteCalculator statelessRemoteCalculator =
(RemoteCalculator) context.lookup(
    "ejb:/jboss-ejb-remote-server-side//CalculatorBean!" +
    RemoteCalculator.class.getName());
```

Os nomes JNDI de bean de sessão são definidos usando uma sintaxe especial. Consulte [Seção 3.2.10.3, “Referência de Nomeação JNDI EJB”](#) para mais informações.

4. Invocação de Métodos

Agora que você possui um objeto bean proxy, você pode invocar qualquer um dos métodos contidos na interface comercial remota.

```
int a = 204;
int b = 340;
System.out.println("Adding " + a + " and " + b + " via the remote
stateless calculator deployed on the server");
int sum = statelessRemoteCalculator.add(a, b);
System.out.println("Remote calculator returned sum = " + sum);
```

O bean proxy passa a solicitação de invocação do método para o bean de sessão no servidor, onde é executado. O resultado é retornado ao bean proxy que, então, o retorna ao chamador. A comunicação entre o bean proxy e o bean de sessão remoto é transparente ao chamador.

Agora você deve conseguir configurar um projeto Maven para suportar a invocação dos beans de sessão em um servidor remoto e gravar o código para invocar os métodos dos beans de sessão usando um bean proxy recuperado do servidor usando JNDI.

[Reportar um erro](#)

3.2.10.3. Referência de Nomeação JNDI EJB

O nome de pesquisa JNDI para um bean de sessão possui a seguinte sintaxe:

```
ejb:<appName>/<moduleName>/<distinctName>/<beanName>!<viewClassName>?
stateful
```

<appName>

Se o arquivo JAR do bean de sessão foi implantado dentro de um arquivo empresarial (enterprise) (EAR), então, este é o nome deste EAR. Por padrão, o nome de um EAR é o seu nome de arquivo sem o sufixo **.ear**. O nome do aplicativo pode ser também substituído em seu arquivo **application.xml**. Se um bean de sessão não está implantado em um EAR, deixe em branco.

<moduleName>

O nome do módulo é o nome do arquivo JAR que o bean de sessão está implantado. Por padrão, o nome do arquivo JAR é o seu próprio nome de arquivo sem o sufixo **.jar**. O nome do módulo também pode ser substituído no arquivo **ejb-jar.xml** do JAR.

<distinctName>

O JBoss EAP 6 permite que cada implantação especifique um nome distinto opcional. Caso a implantação não possua um nome distinto, então, deixe em branco.

<beanName>

O nome do bean é o nome da classe do bean de sessão a ser invocado.

<viewClassName>

O nome da classe de exibição é o nome de classe inteiramente qualificado da interface remota. Isto inclui o nome do pacote da interface.

?stateful

O sufixo **?stateful** é necessário quando o nome JNDI refere-se a um bean de sessão com monitorização de estado. Não é incluído para outros tipos de bean.

[Reportar um erro](#)

3.2.11. Alterações em EJB 2.x

3.2.11.1. Atualização dos Aplicativos que usam EJB 2.x

O JBoss EAP 6 foi construído nos padrões abertos e é compatível com a especificação do Java Enterprise Edition 6. Embora o servidor do aplicativo forneça suporte para o EJB 2.x, ele não suporta mais os recursos que vão além da especificação. Lembre-se de que a especificação Java EE 7 marcou o EJB 2.x como uma opção, portanto é altamente recomendável que você regrave o código do seu aplicativo na especificação EJB 3.x.

Caso ainda queira migrar o seu código EJB 2.x, será necessário, na maioria das vezes, fazer modificações para executar no JBoss EAP 6. O tópico a seguir descreve algumas alterações que possam ser necessárias para executar o EJB 2.x no JBoss EAP 6.

Alterações de Configuração Necessárias para Executar o EJB 2.x no JBoss EAP 6

Inicie o Servidor com o Perfil Completo

Os beans de Persistência Gerenciada pelo Contêiner (do inglês, Container Managed Persistence - CMP) EJB 2.x requerem o perfil completo da Edição 6 do Java Enterprise. Este perfil contém elementos de configuração que são necessários para executar CMP EJBs.

Este perfil de configuração contém o módulo de extensão **org.jboss.as.cmp**:

```
<extensions>
...
<extension module="org.jboss.as.cmp"/>
...
</extensions>
```

Ele contém também o subsistema **cmp**:


```
<profiles>
    ...
    <subsystem xmlns="urn:jboss:domain:cmp:1.1"/>
    ...
</profiles>
```

Para iniciar um servidor autônomo do JBoss EAP 6 com o perfil completo, passe o argumento **-c standalone-full.xml** ou **-c standalone-full-ha.xml** na linha de comando quando iniciar o servidor.

A Configuração do Contêiner Não É Mais Suportada

Nas versões anteriores do JBoss EAP, era possível configurar um contêiner diferente para a entidade CMP e outros beans e usá-lo estabelendo referências dentro do arquivo do descritor de implantação do aplicativo **jboss.xml**. Por exemplo, haviam configurações diferentes, no geral, para SLSB para beans de sessão.

No JBoss EAP 6.x, é possível usar os beans de Entidade EJB 2 com um contêiner padrão. No entanto, as configurações de contêiner diferentes não possuem mais suporte. A abordagem recomendada é migrar os Beans de Sessão com Monitorização de Estado (do inglês, Stateful Session Beans - SFSB), os Beans de Sessão sem Monitorização de Estado (do inglês, Stateless Session Beans - SLSB), os Beans Controlados por Mensagem (do inglês, Message Driven Beans - MDB) EJB2 para EJB 3, e quanto à Persistência Gerenciada Pelo Contêiner (CMP) e aos Beans de Entidade de Persistência Gerenciada pelo Bean (do inglês, Bean-Managed Persistence - BMP) usar a API de Persistência Java (do inglês, Java Persistence API - JPA) de acordo com a especificação EJB 3.

A configuração do contêiner padrão no JBoss EAP 6 contém algumas alterações para os beans CMP EJB 2:

- O bloqueio pessimista é ativo por padrão. Isto pode resultar em travamentos.
- O código de detecção de travamento que estava na camada CMP no JBoss EAP 5.x não faz mais parte do JBoss EAP 6.

No JBoss EAP 5.x, era possível personalizar o cache, o pool, **commit-options** e a pilha do interceptor. No JBoss EAP 6, isto não é mais possível. Existe apenas uma implementação, que é parecida com a política **Instância por transação** com **commit-option C**. Caso você migre um aplicativo que utiliza a configuração do contêiner do bean de entidade **cmp2.x jdbc2 pm**, que usa um gerenciador de persistência baseado em JDBC compatível com CMP2.x, haverá um impacto no desempenho. Este contêiner foi otimizado para melhor desempenho. Recomenda-se a migração destas entidades ao EJB 3 antes de migrar o aplicativo.

Configuração do Interceptor ao Lado do Servidor

O JBoss EAP 6 suporta o Java EE **Interceptor** padrão usando as anotações **@Interceptors** e **@AroundInvoke**. No entanto, isto não permite a manipulação fora da Segurança ou Transação.

Nas versões anteriores do JBoss EAP, era possível modificar a pilha do interceptor para possuir interceptores personalizados para cada invocação EJB. Isto era normalmente usado para implementar uma segurança personalizada ou repetir mecanismos, antes das verificações de segurança ou verificações de transação ou criação. O JBoss EAP 6.1 introduziu os interceptores de

contêiner para fornecer uma funcionalidade semelhante. Para mais informações sobre os interceptores de contêiner, consulte o capítulo intitulado *Container Interceptor* no guia *Development Guide* do JBoss EAP.

Uma outra abordagem para fornecer mais controle antes, durante ou após a fase de confirmação de uma transação, enquanto mantendo a especificação Java EE, é usar o Registro de Sincronização de Transação para a adição de um ouvinte.

O recurso pode ser recuperado usando um dos seguintes métodos:

- Use **InitialContext**

```
TransactionSynchronizationRegistry tsr =
    (TransactionSynchronizationRegistry)
        new
    InitialContext().lookup("java:jboss/TransactionSynchronizationRegi
        stry");
    tsr.registerInterposedSynchronization(new MyTxCallback());
```

- Use Injeção

```
@Resource(mappedName =
    "java:comp/TransactionSynchronizationRegistry")
TransactionSynchronizationRegistry tsr;
...
tsr.registerInterposedSynchronization(new MyTxCallback());
```

A rotina de retorno de chamada deve implementar a Interface **javax.transaction.Synchronization**. Use o método **beforeCompletion()** para desempenhar quaisquer verificações antes que a transação seja confirmada ou revertida. Se um **RuntimeException** for lançado a partir deste método, a transação é revertida e o cliente é notificado com um **EJBTransactionRolledbackException**. No caso de um XA-Transaction, todos os recursos serão revertidos de acordo com o contrato XA. É possível também habilitar a lógica comercial para depender do estado da transação usando o método **afterCompletion(int txStatus)**. Caso um **RuntimeException** seja lançado a partir deste método, a transação permanece no estado anterior, confirmada ou revertida, e o cliente não é notificado. Apenas o gerenciador da transação apresenta um aviso dentro dos arquivos de log do servidor.

Configuração ao Lado do Servidor para Interceptores ao Lado do Cliente

Nas versões anteriores do JBoss EAP, era possível configurar os interceptores do cliente com a configuração do servidor e fornecer apenas as classes com a API do cliente.

No JBoss EAP 6, isto não é mais possível já que o Proxy do cliente não é mais criado ao lado do servidor e transmitido ao cliente após a pesquisa. O proxy é agora gerado ao lado do cliente. Esta otimização evita uma invocação do servidor para os carregamentos de classe e pesquisa.

Configuração do Pool de Beans de Entidades

A configuração do pool de beans de entidades não é recomendada no JBoss EAP 6. Isto deve-se a ela ser limitada à configuração do elemento **<strict-max-pool>**, travamentos e outros problemas que podem ocorrer caso o pool seja muito pequeno para carregar todas as entidades no conjunto de resultado. Os beans de entidade não possuem amplos métodos de ciclo de vida durante a inicialização, portanto criar a instância e cercar o contêiner não é mais lento do que quando se utiliza uma instância de bean de entidade em pool.

Substituição do Arquivo do Descritor de Implantação `jboss.xml`

O descritor de implementação `jboss-ejb3.xml` substitui o arquivo do descritor de implantação `jboss.xml`. Este arquivo é usado para substituir e adicionar os recursos fornecidos pelo Java Enterprise Edition (EE), definidos no descritor de implantação `ejb-jar.xml`. O novo arquivo, incompatível com `jboss.xml` e `jboss.xml`, é agora ignorado nas implantações.

Por exemplo, nas versões anteriores do JBoss EAP, caso fosse definido um `<resource-ref>` no arquivo `ejb-jar.xml`, você necessitaria de uma definição de recurso correspondente para o nome JNDI no arquivo `jboss.xml`. XDoclet gerava automaticamente ambos os arquivos do descritor de implantação. No JBoss EAP 6, a informação de mapeamento JNDI é agora definida no arquivo `jboss-ejb3.xml`. Suponha que a fonte de dados esteja definida no código de fonte Java, como segue.

```
DataSource ds1 = (DataSource) new
InitialContext().lookup("java:comp/env/jdbc/Resource1");
DataSource ds2 = (DataSource) new
InitialContext().lookup("java:comp/env/jdbc/Resource2");
```

`ejb-jar.xml` define as seguintes referências de recurso.

```
<resource-ref >
  <res-ref-name>jdbc/Resource1</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
</resource-ref>
<resource-ref >
  <res-ref-name>java:comp/env/jdbc/Resource2</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
</resource-ref>
```

O arquivo `jboss-ejb3.xml` mapeia os nomes JNDI às referências usando a seguinte sintaxe XML.

```
<resource-ref>
  <res-ref-name>jdbc/Resource1</res-ref-name>
  <jndi-name>java:jboss/datasources/ExampleDS</jndi-name>
</resource-ref>
<resource-ref>
  <res-ref-name>java:comp/env/jdbc/Resource2</res-ref-name>
  <jndi-name>java:jboss/datasources/ExampleDS</jndi-name>
</resource-ref>
```

Algumas das opções de configuração que estavam disponíveis no JBoss EAP 5.x do arquivo `jboss.xml` não foram implementadas no JBoss EAP 6. A lista a seguir descreve alguns dos atributos comumente usados no arquivo `jboss.xml` e se há uma maneira alternativa de arquivá-los no JBoss EAP 6.

- O elemento **method-attribute** era usado para configurar entidades individuais e métodos de beans de sessão.
 - As opções de configuração **read-only** e **idempotent** não foram transportadas para o JBoss EAP 6.

- A opção **transaction-timeout** está agora configurada no arquivo **jboss-ejb3.xml**.
- O atributo **missing-method-permission-exclude-mode** alterou o comportamento dos métodos sem implementar metadados de segurança explícitos em um bean protegido. No JBoss EAP 6, a ausência de uma anotação **@RolesAllowed** é tratada atualmente de forma semelhante ao **@PermitAll**

Configuração de Mapeamento do Tipo de Fonte de Dados

Nas versões anteriores do JBoss EAP, era possível configurar o mapeamento do tipo de fonte de dados dentro do arquivo de configuração de implantação da fonte de dados ***-ds.xml**.

No JBoss EAP 6, isto deve ser realizado agora no arquivo do descritor de implantação **jbossCMP-jdbc.xml**.

```
<defaults>
  <datasource-mapping>mySQL</datasource-mapping>
  <create-table>true</create-table>
  ....
</defaults>
```

Nas versões anteriores do JBoss EAP, o mapeamento personalizado era feito no arquivo **standardjbossCMP-jdbc.xml**. Este arquivo não está mais disponível e o mapeamento é agora realizado no arquivo do descritor de implantação **jbossCMP-jdbc.xml**.

Alterações Adicionais da Persistência Gerenciada pelo Contêiner (CMP) e do Relacionamento Gerenciado pelo Contêiner (CMR)

Alterações de Coleção e do Interador do Relacionamento Gerenciado pelo Contêiner (CMR)

Nas versões anteriores do JBoss EAP, era possível para alguns contêineres, como por exemplo, o contêiner **cmp2.x jdbc2 pm**, interar as coleções CMR, além de remover e adicionar relações. Isto não é mais possível no JBoss EAP 6, já que a configuração do contêiner não é suportada. Consulte [EJB2.1 Finder for CMP entities with relations \(CMR\) returns duplicates in EAP6](#) na seção de Soluções da Base de Conhecimento de Suporte do Portal do Consumidor, para mais informações sobre como atingir esta mesma funcionalidade no código do aplicativo.

Entradas Duplicadas do Relacionamento Gerenciado pelo Contêiner (CMR) para Localizadores

Nas versões anteriores do JBoss EAP, era possível selecionar contêineres CMP diferentes que usavam estratégias de persistência diferentes. O contêiner **cmp2.x jdbc2 pm** no JBoss EAP 5.x usava **SQL-92** otimizado para gerar a sintaxe LEFT OUTER JOIN otimizada para localizadores. A implementação não contém essas otimizações, já que o Boss EAP 6.x suporta apenas o contêiner padrão para CMP e CMR. O localizador deve incluir a palavra-chave **DISTINCT** na declaração **SELECT** para evitar um produto cartesiano no conjunto de resultado. Consulte [EJB2.1 Finder for CMP entities with relations \(CMR\) returns duplicates in EAP6](#) na seção de Soluções da Base de Conhecimento de Suporte do Portal do Consumidor para mais informações.

Alteração Padrão de Exclusão em Cascata para os Beans de Entidade CMP

O valor padrão de exclusão em cascata foi alterado para **false**. Isto pode resultar em falhas de exclusão no JBoss EAP 6. Se as relações de entidades estiverem marcadas como **cascade-delete**, **batch-cascade-delete** deve ser explicitamente configurado como **true** no arquivo

`jbosscmp-jdbc.xml`. Para mais informações, consulte [cascade delete fail for EJB2 CMP Entities after migration to EAP6](#) na seção de Soluções da Base de Conhecimento de Suporte do Portal do Consumidor.

Mapeadores Personalizados de CMP para Campos Personalizados

Se você utilizou classes de mapeador do cliente, tais como `JDBCParameterSetter`, `JDBCResultSetReader` e `Mapper` em seu aplicativo JBoss EAP 5.x, você pode encontrar `java.lang.ClassNotFoundException` quando implantar seu aplicativo ao JBoss EAP 6. Isto é devido aos nomes de pacotes para as interfaces terem sido alterados de `org.jboss.ejb.plugins.cmp.jdbc.Mapper` para `org.jboss.as.cmp.jdbc.Mapper`. Para mais informações, consulte [How to use Field mapping for custom classes in an EJB2 CMP application in EAP6](#) na seção de Soluções da Base de Conhecimento de Suporte do Portal do Consumidor.

Geração de Chaves Primárias Usando comandos de entidade

Caso o seu aplicativo JBoss EAP 5 use **entity-commands** para gerar chaves primárias, por exemplo **Sequence** ou **Auto-increment**, você pode encontrar um `ClassNotFoundException` para a classe `JDBCOracleSequenceCreateCommand` quando migrar o seu aplicativo para JBoss EAP 6. Isto é devido ao pacote de classe ter sido alterado de `org.jboss.ejb.plugins.cmp.jdbc` para `org.jboss.as.cmp.jdbc.keygen`. Caso utilize esta classe no seu aplicativo JBoss EAP 6, uma dependência deve ser adicionada no módulo `EAP_HOME/modules/system/layers/base/org.jboss.as/cmp`.

Alterações no Aplicativo

Modifique o Código para Usar as Novas Regras do Namespace JNDI.

Assim como EJB 3.0, você deve usar o prefixo JNDI completo com EJB 2.x. Consulte [Seção 3.1.8.1, “Atualização dos Nomes do Namespace JNDI do Aplicativo”](#) para mais informações sobre as novas regras do namespace JNDI e exemplos de código.

Exemplos demonstrando como atualizar os namespaces JNDI de versões anteriores podem ser encontradas aqui: [Seção 3.1.8.5, “Exemplos de Namespaces JNDI em Versões Anteriores e a Maneira Como São Especificados no JBoss EAP 6”](#).

Modifique o Descritor do Arquivo `jboss-web.xml`

Modifique `<jndi-name>` para cada `<ejb-ref>` para usar o novo formato de pesquisa totalmente qualificado JNDI.

Use XDoclet para Mapear o Nome JNDI de Interfaces Locais Internas

No EJB 2, era bastante comum usar o padrão **Locator** para pesquisar Beans. Se você usava este padrão em seu aplicativo, ao invés de modificar o código do aplicativo, você pode usar [XDoclet](#) para gerar um mapa para os novos nomes JNDI.

Uma típica anotação XDoclet parece com o seguinte:

```
@ejb.bean name="UserAttribute" display-name="UserAttribute" local-jndi-name="ejb21/UserAttributeEntity" view-type="local" type="CMP" cmp-version="2.x" primkey-field="id"
```

O nome JNDI **ejb21/UserAttributeEntity** no exemplo acima não é mais válido no JBoss EAP 6. Você pode mapear este nome para um nome JNDI usando o subsistema **naming** na configuração do servidor e um patch para XDoclet.

Mapeadores personalizados podem ser criados conforme mencionado no parágrafo acima intitulado *CMP Customized Mappers for Custom Fields* ou você pode modificar o código conforme descrito no procedimento a seguir.

Procedimento 3.24. Alteração do Código XDoclet Gerado e Uso do Subsistema de Nomeação

1. Extraia o template XDoclet **lookup.xdt** localizado no **ejb-module.jar** e modifique o **lookup()** no **lookupHome** como a seguir:

```
private static Object lookupHome(java.util.Hashtable environment,
String jndiName, Class narrowTo) throws
javax.naming.NamingException {
    // Obtain initial context
    javax.naming.InitialContext initialContext = new
javax.naming.InitialContext(environment);
    try {
        // Replace the existing lookup
        // Object objRef = initialContext.lookup(jndiName);
        // This is the new mapped lookup
        Object objRef;
        try {
            // try JBoss EAP mapping
            objRef = initialContext.lookup("global/"+jndiName);
        } catch (java.lang.Exception e) {
            objRef = initialContext.lookup(jndiName);
        }
        // only narrow if necessary
        if (java.rmi.Remote.class.isAssignableFrom(narrowTo))
            return javax.rmi.PortableRemoteObject.narrow(objRef,
narrowTo);
        else
            return objRef;
    } finally {
        initialContext.close();
    }
}
```

2. Execute Ant, configurando o atributo do template para usar o **lookup.xdt** modificado para a tarefa **ejbdoclet**.
3. Modifique o subsistema **naming** no arquivo de configuração do servidor para mapear o antigo nome JNDI para o novo nome JNDI válido.

```
<subsystem xmlns="urn:jboss:domain:naming:1.2">
    <bindings>
        <lookup name="java:global/ejb21/UserAttributeEntity"
lookup="java:global/ejb2CMP/ejb/UserAttribute!de.wfink.ejb21.cmp.c
mr.UserAttributeLocalHome"/>
    </bindings>
    <remote-naming/>
</subsystem>
```

Sumário dos Arquivos Obsoletos

Os seguintes arquivos não possuem mais suporte no JBoss EAP 6.

jboss.xml

O arquivo do descritor de implantação **jboss.xml** não possui mais suporte e será ignorado caso seja incluído no arquivo implantado.

standardjbosscmp-jdbc.xml

O arquivo de configuração **standardjbosscmp-jdbc.xml** não possui mais suporte. Esta informação de configuração é agora incluída no módulo **org.jboss.as.cmp** e não pode mais ser personalizada.

standardjboss.xml

O arquivo de configuração **standardjboss.xml** não possui mais suporte. A informação de configuração é agora incluída no arquivo **standalone.xml**, quando executando um servidor autônomo, ou no arquivo **domain.xml**, quando executando em um domínio gerenciado.

[Reportar um erro](#)

3.2.12. Alterações no JBoss AOP

3.2.12.1. Atualização dos Aplicativos que Usam o JBoss AOP

O JBoss AOP (Aspect Orientated Programing - Programação Orientada do Aspecto) não está mais incluído no JBoss EAP 6. Nas versões anteriores, o JBoss AOP era usado pelo contêiner EJB. No JBoss EAP 6, o contêiner EJB usa um novo mecanismo. Se o seu aplicativo usa o JBoss AOP, você deve modificar o código do seu aplicativo conforme abaixo.

Refatore o Aplicativo

- As configurações EJB3 padrão que eram realizadas anteriormente no arquivo **ejb3-interceptors-aop.xml** são agora configuradas no arquivo de configuração. Para um servidor autônomo, este é o arquivo **standalone/configuration/standalone-full.xml**. Caso você esteja executando o seu servidor em um domínio gerenciado, o arquivo é **domain/configuration/domain.xml**.
- Os Interceptores AOP ao lado do servidor devem ser modificados para o uso do Java EE padrão **Interceptor**. Para mais informações sobre os interceptores de contêiner e como utilizar um interceptor ao lado do cliente em um aplicativo, consulte o capítulo entitulado *Container Interceptors* no guia *Development Guide* do JBoss EAP 6 localizado no Portal do Consumidor
https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

Utilize as Bibliotecas do JBoss AOP

- Caso você não possa refatorar o código, você pode obter uma cópia das bibliotecas do JBoss AOP e empacotá-las com o aplicativo. As bibliotecas AOP podem funcionar no JBoss EAP 6, mas não são implantadas. Você pode implantá-las manualmente usando o seguinte argumento da linha de comando, quando inicia o seu servidor: -
Djboss.aop.path=PATH_TO_AOP_CONFIG



NOTA

Embora as bibliotecas do JBoss AOP possam funcionar no JBoss EAP 6, esta não é uma configuração com suporte.

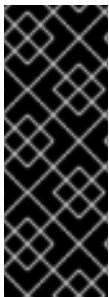
[Reportar um erro](#)

3.2.13. Migração dos Aplicativos Seam 2.2

3.2.13.1. Migração dos Arquivos Seam 2.2 para o JBoss EAP 6

Visão Geral

Quando você migra um aplicativo Seam 2.2, você precisa configurar a fonte de dados e especificar quaisquer dependências de módulo. Você precisa determinar também se o aplicativo possui alguma dependência nos arquivos que não são enviados juntos com o JBoss EAP 6 e copiar os JARs dependentes no diretório **lib/** do aplicativo.



IMPORTANTE

Os aplicativos que usam o Hibernate diretamente com o Seam 2.2 podem usar uma versão do Hibernate 3 empacotada dentro do aplicativo. O Hibernate 4, que é fornecido através do módulo org.hibernate do JBoss EAP 6, não é suportado pelo Seam 2.2. Este exemplo tem a intenção de ajudá-lo a executar o seu aplicativo no JBoss EAP 6 como uma primeira etapa. Você deve estar ciente de que o empacotamento do Hibernate 3 com o aplicativo Seam 2.2 não possui uma configuração com suporte.

Procedimento 3.25. Migração dos Arquivos Seam 2.2

1. Atualize a configuração da fonte de dados

Alguns exemplos do Seam 2.2 usam a fonte de dados JDBC padrão nomeada **java:/ExampleDS**. Essa fonte de dados padrão foi alterada no JBoss EAP 6 para **java:jboss/datasources/ExampleDS**. Caso seu aplicativo utilize o banco de dados do exemplo, você pode realizar uma das seguintes opções:

- Se você quiser usar o banco de dados do exemplo que é enviado junto com o JBoss EAP 6, modifique o arquivo **META-INF/persistence.xml** para substituir o elemento **jta-data-source** existente com o nome JNDI da fonte de dados do banco de dados do exemplo:

```
<!-- <jta-data-source>java:/ExampleDS</jta-data-source> -->
<jta-data-source>java:jboss/datasources/ExampleDS</jta-data-source>
```

- Caso você prefira manter o seu banco de dados existente, você pode adicionar a definição da fonte de dados ao arquivo **EAP_HOME/standalone/configuration/standalone.xml**.



IMPORTANTE

Você deve interromper o servidor antes de editar o arquivo de configuração do servidor para que a sua alteração seja efetivada ao reiniciar o servidor.

A seguinte definição é uma cópia da fonte de dados HSQL padrão definida no JBoss EAP 6:

```
<datasource name="ExampleDS" jndi-name="java:/ExampleDS"
enabled="true" jta="true" use-java-context="true" use-ccm="true">
  <connection-url>jdbc:h2:mem:test;DB_CLOSE_DELAY=-
1</connection-url>
  <driver>h2</driver>
  <security>
    <user-name>sa</user-name>
    <password>sa</password>
  </security>
</datasource>
```

- Você pode também adicionar a definição da fonte de dados usando a interface da linha de comando do Gerenciamento CLI. Segue abaixo um exemplo da sintaxe que você deve usar para adicionar uma fonte de dados. O "\ " no final da linha indica a continuação do comando na linha seguinte.

Exemplo 3.5. Exemplo da sintaxe para a adição da definição da fonte de dados

```
$ EAP_HOME/bin/jboss-cli --connect
[standalone@localhost:9999 /] data-source add --name=ExampleDS
--jndi-name=java:/ExampleDS \
  --connection-url=jdbc:h2:mem:test;DB_CLOSE_DELAY=-1 --
driver-name=h2 \
  --user-name=sa --password=sa
```

Consulte [Seção 3.1.6.2, “Atualização da Configuração da DataSource”](#) para mais informações sobre como configurar uma fonte de dados.

2. Adicione quaisquer dependências necessárias

Já que os aplicativos Seam 2.2 utilizam JSF 1.2, você precisa adicionar dependências para os módulos JSF 1.2 e excluir os módulos JSF 2.0. Para realizar isto, você precisa criar um arquivo **jboss-deployment-structure.xml** no diretório **META-INF/** do EAR que contém os seguintes dados:

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="com.sun.jsf-impl" slot="1.2"
export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
      <module name="com.sun.jsf-impl" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="javax.faces.api" slot="1.2"/>
      <module name="com.sun.jsf-impl" slot="1.2"/>
    </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```


Se o seu aplicativo utiliza estruturas de registro em log de terceiros, você precisa adicionar essas dependências conforme descrito aqui: [Seção 3.1.4.1, “Modificação das Dependências de Registro em Log”](#).

3. Se o seu aplicativo utiliza Hibernate 3.x, tente executar primeiro o aplicativo usando as bibliotecas do Hibernate 4

Se o seu aplicativo não utiliza o Contexto de Persistência Gerenciado Seam, a busca Hibernate, a validação ou quaisquer outros recursos que mudaram com o Hibernate 4, é possível que consiga executar com as bibliotecas do Hibernate 4. No entanto, se você encontrar **ClassNotFoundException** ou **ClassCastException** que apontam para as classes Hibernate, ou ver erros semelhantes aos seguintes, é possível que tenha que seguir as instruções da próxima etapa e modificar o seu aplicativo para usar as bibliotecas do Hibernate 3.3.

```
Caused by: java.lang.LinkageError: loader constraint
violation in interface itable initialization: when resolving method
"org.jboss.seam.persistence.HibernateSessionProxy.getSession(Lorg/hibernate/EntityMode;)Lorg/hibernate/Session;" the class loader
(instance of org/jboss/modules/ModuleClassLoader) of the current
class, org/jboss/seam/persistence/HibernateSessionProxy, and the
class loader (instance of org/jboss/modules/ModuleClassLoader) for
interface org/hibernate/Session have different Class objects for the
type org/hibernate/Session used in the signature
```

4. Copie os arquivos dependentes das estruturas externas ou de outras localizações

Se o seu aplicativo utiliza Hibernate 3.x e você não consegue usar Hibernate 4 no seu aplicativo com sucesso, você precisará copiar os JARs do Hibernate 3.x no diretório **/lib** e excluir o módulo Hibernate na seção de implantações de **META-INF/jboss-deployment-structure.xml**, como a seguir:

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
    <exclusions>
      <module name="org.hibernate"/>
    </exclusions>
  </deployment>
</jboss-deployment-structure>
```

Você deve tomar algumas medidas adicionais quando agrupar Hibernate 3.x ao seu aplicativo. Consulte [Seção 3.2.2.2, “Configuração das Alterações para os Aplicativos que Utilizam Hibernate e JPA”](#) para mais informações.

5. Depure e resolva os erros JNDI do Seam 2.2

Ao migrar o aplicativo Seam 2.2, você pode encontrar erros **javax.naming.NameNotFoundException** no log, como a seguir:

```
javax.naming.NameNotFoundException: Name 'jboss-seam-booking' not
found in context ''
```

Se você não deseja modificar as pesquisas JNDI através do código, você pode modificar o arquivo **components.xml** do aplicativo, como a seguir:

a. Substitua o elemento core-init existente

Primeiro, você precisa substituir o elemento `core:init` existente, como a seguir:

```
<!-- <core:init jndi-pattern="jboss-seam-booking/#
{ejbName}/local" debug="true" distributable="false"/> -->
<core:init debug="true" distributable="false"/>
```

b. Localize as mensagens INFO associadas ao JNDI no log do servidor

A seguir, localize as mensagens INFO associadas ao JNDI que estão impressas no log do servidor quando o aplicativo é implantado. As mensagens vinculadas ao JNDI devem parecer-se com o seguinte:

```
INFO
org.jboss.as.ejb3.deployment.processors.EjbJndiBindingsDeployment
UnitProcessor (MSC service thread 1-1) JNDI bindings for session
bean
named AuthenticatorAction in deployment unit subdeployment
"jboss-seam-booking.jar" of deployment "jboss-seam-booking.ear"
are as follows:
    java:global/jboss-seam-booking/jboss-seam-
booking.jar/AuthenticatorAction!org.jboss.seam.example.booking.Au
thenticator
    java:app/jboss-seam-
booking.jar/AuthenticatorAction!org.jboss.seam.example.booking.Au
thenticator

    java:module/AuthenticatorAction!org.jboss.seam.example.booking.Au
thenticator
    java:global/jboss-seam-booking/jboss-seam-
booking.jar/AuthenticatorAction
    java:app/jboss-seam-booking.jar/AuthenticatorAction
    java:module/AuthenticatorAction
```

c. Adicione os elementos do componente

Para cada mensagem INFO vinculada ao JNDI no log, adicione um elemento **component** correspondente ao arquivo **components.xml**:

```
<component
class="org.jboss.seam.example.booking.AuthenticatorAction" jndi-
name="java:app/jboss-seam-booking.jar/AuthenticatorAction" />
```

Para mais informações sobre como depurar e resolver problemas de migração, consulte [Seção 4.2.1, “Depuração e Solução de Problemas de Migração”](#).

Para uma lista de problemas de migração conhecidos com os arquivos Seam 2, consulte [Seção 3.2.13.2, “Problemas de Migração do Arquivo Seam 2.2”](#).

Resultado

O arquivo Seam 2.2 implanta e executa com êxito no JBoss EAP 6.

[Reportar um erro](#)

3.2.13.2. Problemas de Migração do Arquivo Seam 2.2

Seam 2.2 Drools e Java 7 não são compatíveis

Seam 2.2 Drools e Java 7 são incompatíveis e falham com `org.drools.RuntimeDroolsException`: valor '1.7' não é um erro de nível de mensagem válido.

Seam 2.2.5 assinado como `cglib.jar` evita o funcionamento do exemplo Spring

Quando o exemplo Spring é executado usando o `cglib.jar` assinado enviado junto com Seam 2.2.5 no JBoss EAP 5, uma falha com a seguinte causa raiz ocorre:

```
java.lang.SecurityException: class
"org.jboss.seam.example.spring.UserService$$EnhancerByCGLIB$$7d6c3d12"'s
signer information does not match signer information of other classes in
the same package
```

A solução para este problema é remover a assinatura de `cglib.jar`, como a seguir:

```
zip -d $SEAM_DIR/lib/cglib.jar META-INF/JBOSSCOD\*
```

O exemplo Seambay falha com `NotLoggedInException`

A causa deste problema é o cabeçalho da mensagem SOAP ficando nulo durante o processamento da mensagem em `SOAPRequestHandler` e, como consequência, a ID de conversa não é definida.

A solução para este problema é substituir

`org.jboss.seam.webservice.SOAPRequestHandler.handleOutbound`, conforme descrito em <https://issues.jboss.org/browse/JBPAPP-8376>.

O exemplo Seambay falha com `UnsupportedOperationException`: nenhuma transação

Este erro é causado por alterações no nome JNDI do `UserTransaction` no JBoss EAP 6.

A solução para este problema é substituir

`org.jboss.seam.transaction.Transaction.getUserTransaction`, conforme descrito em <https://issues.jboss.org/browse/JBPAPP-8322>.

O exemplo de tarefas lança `org.jboss.resteasy.spi.UnhandledException`: não foi possível cancelar o marshall do corpo da solicitação

Este erro é causado pela incompatibilidade entre `seam-resteasy-2.2.5`, incluído no JBoss EAP 5.1.2), e `RESEasy 2.3.1.GA`, incluído no JBoss EAP 6.

Para solucionar este problema, use `jboss-deployment-structure.xml` para excluir `resteasy-jaxrs`, `resteasy-jettison-provider` e `resteasy-jaxb-provider` da implantação principal e `resteasy-jaxrs`, `resteasy-jettison-provider`, `resteasy-jaxb-provider` e `resteasy-yaml-provider` do `jboss-seam-tasks.war`, conforme descrito em <https://issues.jboss.org/browse/JBPAPP-8315>. Depois, é necessário incluir as bibliotecas `RESEasy` agrupadas com Seam 2.2 no EAR.

Deadlock entre `org.jboss.seam.core.SynchronizationInterceptor` e o bloqueio EJB da instância de componente com monitorização de estado (stateful) durante uma solicitação AJAX

Uma página de erro com a mensagem a seguir ou uma mensagem semelhante é exibida: "Caused by `javax.servlet.ServletException` with message: "javax.el.ELException: /main.xhtml @36,71 value="#{hotelSearch.pageSize}": `org.jboss.seam.core.LockTimeoutException`: could not acquire lock on @Synchronized component: hotelSearch".

O problema é que Seam 2 realiza seu próprio bloqueio fora do bean de sessão com monitorização de estado (stateful) (SFSB) e com um escopo diferente. Isto significa que se um thread acessa um EJB duas vezes na mesma transação, após a primeira invocação, ele terá o bloqueio SFSB, mas

não o bloqueio seam. Assim, um segundo thread pode adquirir o bloqueio seam, o qual atingirá, então, o bloqueio EJB e aguardará. Quando o primeiro thread tenta sua segunda invocação, ele bloqueará o deadlock e o interceptor seam 2. No Java EE 5, os EJBs lançariam imediatamente uma exceção aos acessos concomitantes. Esse comportamento foi alterado no Java EE 6.

Para solucionar este problema é preciso adicionar `@AccessTimeout(0)` ao EJB. Isto fará com que um **ConcurrentAccessException** seja lançado imediatamente quando esta situação ocorrer.

A criação de pedido do exemplo Dvdstore falha com `javax.ejb.EJBTransactionRolledbackException`

O exemplo dvdstore exibe o seguinte erro:

```
JBAS011437: Found extended persistence context in SFSB invocation call
stack but that cannot be used because the transaction already has a
transactional context associated with it. This can be avoided by
changing application code, either eliminate the extended persistence
context or the transactional context. See JPA spec 2.0 section 7.6.3.1.
```

Este problema é devido às alterações na especificação JPA.

A solução para este problema é alterar o contexto de persistência para **transactional** nas classes **CheckoutAction** e **ShowOrdersAction**, além de usar a operação de mesclagem do gerenciador de entidade nos métodos **cancelOrder** e **detailOrder**.

O provedor do JBoss Cache Seam Cache não pode ser usado no JBoss EAP 6

JBoss Cache não é suportado no JBoss EAP 6. Isto faz com que o provedor do JBoss Cache Seam falhe em um aplicativo Seam no servidor do aplicativo com:

```
java.lang.NoClassDefFoundError: org/jboss/util/xml/JBossEntityResolver
```

Hibernate 3.3.x Auto examina por problemas de entidades JPA com o JBoss EAP 6

A solução para este problema é listar manualmente todas as classes de entidade no arquivo `persistence.xml`. Por exemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
version="1.0">
  <persistence-unit name="example_pu">
    <description>Hibernate 3 Persistence Unit.</description>
    <jta-data-source>java:jboss/datasources/ExampleDS</jta-data-
source>
    <properties>
      <property name="jboss.as.jpa.providerModule"
value="hibernate3-bundled" />
    </properties>
    <class>com.acme.Foo</class>
    <class>com.acme.Bar</class>
  </persistence-unit>
</persistence>
```

Chamar componentes EJB Seam a partir de Threads não pertencentes a EJB gera `javax.naming.NameNotFoundException`

Este problema é o resultado de alterações no JBoss EAP 6 para implementar o novo sistema de carregamento de classe modular e adotar as novas convenções do namespace JNDI padronizadas. O namespace `java:app` é designado para os nomes compartilhados por todos os componentes em um aplicativo único. Threads não pertencentes a EE, tais como threads assíncronos Quartz, devem usar o namespace `java:global`, o qual é compartilhado por todos os aplicativos implantados em uma instância do servidor do aplicativo.

Caso você receba `javax.naming.NameNotFoundException`, quando tentar chamar os componentes EJB Seam a partir dos métodos assíncronos Quartz, você deve modificar o arquivo `components.xml` para usar o nome JNDI global, por exemplo:

```
<component class="org.jboss.seam.example.quartz.MyBean" jndi-
name="java:global/seam-quartz/quartz-ejb/myBean"/>
```

Para mais informações sobre as alterações JNDI, consulte [Seção 3.1.8.1, “Atualização dos Nomes do Namespace JNDI do Aplicativo”](#). Para mais informações sobre este problema, em particular, consulte [BZ#948215 - Seam2.3 javax.naming.NameNotFoundException trying to call EJB Seam components from quartz asynchronous methods](#) nas notas de lançamento *2.2.0 Release Notes* para *Red Hat JBoss Web Framework Kit* no Portal do Consumidor Red Hat.

[Reportar um erro](#)

3.2.14. Migração dos Aplicativos Spring

3.2.14.1. Migração de Aplicativos Spring

Informações sobre a migração de aplicativos Spring podem ser encontradas na documentação *Red Hat JBoss Web Framework Kit* localizada no Portal do Consumidor <https://access.redhat.com/site/documentation/>. Pesquise por **Red Hat JBoss Middleware** e clique no link *Red Hat JBoss Web Framework Kit*. Os guias *Spring Installation Guide* e *Spring Developer Guide* estão disponíveis em múltiplos formatos.

[Reportar um erro](#)

3.2.15. Outras Alterações que Afetam a Migração

3.2.15.1. Familiarize-se com Outras Alterações que Podem Afetar sua Migração

Segue abaixo uma lista de outras alterações no JBoss EAP 6 que poderiam impactar os esforços de sua migração.

- [Seção 3.2.15.2, “Alteração do Nome Plug-in Maven”](#)
- [Seção 3.2.15.3, “Modificação dos Aplicativos Clientes”](#)

[Reportar um erro](#)

3.2.15.2. Alteração do Nome Plug-in Maven

jboss-maven-plugin não foi atualizado e não funciona no JBoss EAP 6. Você deve usar **org.jboss.as.plugins:jboss-as-maven-plugin** para implantar o diretório correto.

[Reportar um erro](#)

3.2.15.3. Modificação dos Aplicativos Clientes

Se planeja migrar um aplicativo cliente que conectará ao JBoss EAP 6, você deve estar ciente de que o nome e a localização do JAR que agrupam as bibliotecas clientes foram alterados. Esse JAR é agora nomeado **jboss-client.jar** e está localizado no diretório **EAP_HOME/bin/client/**. Ele substitui **EAP_HOME/client/jbossall-client.jar** e contém todas as dependências necessárias para conectar-se ao JBoss EAP 6 a partir de um cliente remoto.

[Reportar um erro](#)

CAPÍTULO 4. FERRAMENTAS E DICAS

4.1. RECURSOS QUE PODEM AUXILIAR NA MIGRAÇÃO

4.1.1. Recursos que Podem Assisti-lo na sua Migração

Segue abaixo uma lista de recursos que podem ajudá-lo na migração de um aplicativo para o JBoss EAP 6.

Ferramentas

Existem diversas ferramentas que ajudam a automatizar algumas das alterações de configuração. Consulte [Seção 4.1.2, “Familiarize-se com as Ferramentas que Podem Assisti-lo com a Migração”](#) para mais informações.

Dicas de Depuração

Consulte [Seção 4.2.1, “Depuração e Solução de Problemas de Migração”](#) para uma lista dos erros e das causas e resoluções de problemas mais comuns que você pode encontrar quando migrar seu aplicativo.

Exemplo de Migrações

Consulte [Seção 4.3.1, “Revisão da Migração dos Aplicativos de Exemplo”](#) para exemplos de aplicativos que foram migrados para o JBoss EAP 6.

[Reportar um erro](#)

4.1.2. Familiarize-se com as Ferramentas que Podem Assisti-lo com a Migração

Sumário

Existem algumas ferramentas que podem assisti-lo no processo de sua migração. Segue abaixo uma lista dessas ferramentas juntamente com a descrição do que elas fazem.

Tattletale

Com a alteração para o carregamento de classe modular, você precisa encontrar e retificar as dependências dos aplicativos. Tattletale pode ajudá-lo a identificar os nomes dos módulos dependentes e gerar a configuração XML para seu aplicativo.

[Seção 4.1.3, “Uso do Tattletale para Encontrar as Dependências dos Aplicativos”](#)

Ferramenta de Migração IronJacamar

No JBoss EAP 6, as fontes de dados e os adaptadores de recurso não são mais configurados em um arquivo separado. Eles são agora definidos no arquivo de configuração do servidor e usam novos esquemas. A Ferramenta de Migração IronJacamar pode ajudar a converter a configuração antiga no formato esperado pelo JBoss EAP 6.

[Seção 4.1.6, “Uso da Ferramenta IronJacamar para a Migração das Configurações do Adaptador de Recursos e das Fontes de Dados”](#)

[Reportar um erro](#)

4.1.3. Uso do Tattletale para Encontrar as Dependências dos Aplicativos

Sumário

Devido às alterações para o carregamento de classe modular no JBoss EAP 6, você pode encontrar os rastreamentos **ClassNotFoundException** ou **ClassCastException** no log do JBoss quando migrar o seu aplicativo. Para resolver esses erros, você precisa encontrar os JARs que contêm as classes especificadas pelas exceções.

Tattletale é uma excelente ferramenta de terceiros que escaneia recursivamente seu aplicativo e fornece relatórios detalhados sobre seus conteúdos. Tattletale 1.2.0.Beta2, ou versões posteriores, contém um suporte adicional para ajudar no novo carregamento de classe do JBoss Modules usado no JBoss EAP 6. O relatório "JBoss AS 7" do Tattletale pode ser usado para identificar e gerar automaticamente os nomes dos módulos dependentes para incluir o arquivo **jboss-deployment-structure.xml** do seu aplicativo.

Procedimento 4.1. Instale e execute Tattletale para encontrar as dependências do aplicativo

1. [Seção 4.1.4, "Baixe e Instale Tattletale"](#)
2. [Seção 4.1.5, "Criação e Revisão do Relatório Tattletale"](#)



NOTA

Tattletale é uma ferramenta de terceiros e não é suportada como parte do JBoss EAP 6. Para a documentação atual sobre como instalar e usar Tattletale, visite o website do Tattletale <http://tattletale.jboss.org/>.

[Reportar um erro](#)

4.1.4. Baixe e Instale Tattletale

Procedimento 4.2. Baixe e Instale Tattletale

1. Baixe Tattletale versão 1.2.0.Beta2, ou a mais recente, em <http://sourceforge.net/projects/jboss/files/JBoss%20Tattletale>.
2. Descomprima o arquivo no diretório de sua escolha.
3. Modifique o arquivo **TATTLETALE_HOME/jboss-tattletale.properties** realizando o seguinte:
 - a. Adicione **ee6** e **as7** à propriedade **profiles**.


```
profiles=java5, java6, ee6, as7
```
 - b. Remova os comentários das propriedades **scan** e **reports**.

[Reportar um erro](#)

4.1.5. Criação e Revisão do Relatório Tattletale

1. Crie o relatório Tattletale emitindo o comando: **java -jar TATTLETALE_HOME/tattletale.jar APPLICATION_ARCHIVE OUTPUT_DIRECTORY**

Por exemplo: `java -jar tattletale-1.2.0.Beta2/tattletale.jar
~/applications/jboss-seam-booking.ear ~/output-results/`

2. Em um navegador, abra o arquivo **OUTPUT_DIRECTORY/index.html** e clique em "JBoss AS 7" sob a seção "Reports" (Relatórios).
 - a. A coluna da esquerda lista os arquivos usados pelo aplicativo. Clique no link **ARCHIVE_NAME** para visualizar detalhes sobre o arquivo, tais como sua localização, informações de manifesto e as classes que contém.
 - b. O link **jboss-deployment-structure.xml** na coluna da direita mostra como especificar a dependência do módulo para o arquivo nomeado na coluna da esquerda. Clique neste link para verificar como definir as informações de módulos de dependência da implantação para este arquivo.

[Reportar um erro](#)

4.1.6. Uso da Ferramenta IronJacamar para a Migração das Configurações do Adaptador de Recursos e das Fontes de Dados

Sumário

Nas versões anteriores do servidor do aplicativo, as fontes de dados e os adaptadores de recursos eram configurados e implantados usando um arquivo com um sufixo ***-ds.xml**. A distribuição IronJacamar 1.1 contém uma ferramenta de migração que pode ser usada para converter esses arquivos de configuração no formato esperado pelo JBoss EAP 6. A ferramenta analisa o arquivo de configuração da fonte a partir da versão anterior, cria e grava a configuração XML a um arquivo de saída no novo formato. Esse XML pode, então, ser copiado e colado sob o subsistema correto no arquivo de configuração do servidor do JBoss EAP 6. Essa ferramenta envia todos os esforços para converter atributos e elementos antigos no novo formato. No entanto, pode ser necessário realizar modificações adicionais no arquivo gerado.

Procedimento 4.3. Instalação e execução da ferramenta de Migração IronJacamar

1. [Seção 4.1.7, "Baixe e Instale a Ferramenta de Migração IronJacamar"](#)
2. [Seção 4.1.8, "Uso da Ferramenta de Migração IronJacamar para Converter um Arquivo de Configuração da Fonte de Dados"](#)
3. [Seção 4.1.9, "Uso da Ferramenta de Migração IronJacamar para Converter um Arquivo de Configuração do Adaptador de Recursos"](#)



NOTA

A ferramenta de Migração IronJacamar é uma ferramenta de terceiros e não possui suporte como parte do JBoss EAP 6. Consulte <http://www.ironjacamar.org/> para mais informações sobre IronJacamar. Consulte <http://www.ironjacamar.org/documentation.html> para informações atuais sobre como instalar e usar esta ferramenta.

[Reportar um erro](#)

4.1.7. Baixe e Instale a Ferramenta de Migração IronJacamar

**NOTA**

A ferramenta de migração está disponível apenas na versão Jacamar 1.1, ou posteriores, e requer Java 7 ou versão posterior.

1. Baixe a distribuição mais recente do IronJacamar aqui:
<http://www.ironjacamar.org/download.html>
2. Descomprima o arquivo baixado em um diretório de sua escolha.
3. Localize o script conversor na distribuição IronJacamar.
 - O script Linux está localizado aqui: **`IRONJACAMAR_HOME/doc/as/converter.sh`**
 - O arquivo em lotes do Windows está localizado aqui:
`IRONJACAMAR_HOME/doc/as/converter.bat`

[Reportar um erro](#)

4.1.8. Uso da Ferramenta de Migração IronJacamar para Converter um Arquivo de Configuração da Fonte de Dados

**NOTA**

O script convertor IronJacamar requer Java 7 ou versão posterior.

Procedimento 4.4. Conversão de um Arquivo de Configuração da Fonte de Dados

1. Abra uma linha de comando e navegue até o diretório **`IRONJACAMAR_HOME/doc/as/`**.
2. Execute o script conversor digitando o seguinte comando:
 - Para Linux: **`./converter.sh -ds SOURCE_FILE TARGET_FILE`**
 - Para Microsoft Windows: **`./converter.bat -ds SOURCE_FILE TARGET_FILE`**

`SOURCE_FILE` é o arquivo `-ds.xml` da fonte de dados da versão anterior. **`TARGET_FILE`** contém a nova configuração.

Por exemplo, para converter o arquivo de configuração da fonte de dados **`jboss-seam-booking-ds.xml`** localizado no diretório atual, você digitaria:

- Para Linux: **`./converter.sh -ds jboss-seam-booking-ds.xml new-datasource-config.xml`**
- Para Microsoft Windows: **`./converter.bat -ds jboss-seam-booking-ds.xml new-datasource-config.xml`**

Perceba que o parâmetro para a conversão da fonte de dados é **`-ds`**.

3. Copie o elemento **`<datasource>`** do arquivo de destino e cole-o no arquivo de configuração do servidor sob o elemento **`<subsystem`**
`xmlns="urn:jboss:domain:datasources:1.1"><datasources>`.



IMPORTANTE

Você deve interromper o servidor antes de editar o arquivo de configuração do servidor para que sua alteração seja efetivada ao reiniciar o servidor.

- o Caso você esteja executando em um domínio gerenciado, copie o XML no arquivo **EAP_HOME/domain/configuration/domain.xml**.
- o Caso você esteja executando como um servidor autônomo, copie o XML no arquivo **EAP_HOME/standalone/configuration/standalone.xml**.

4. Modifique o XML gerado no novo arquivo de configuração.

Segue abaixo um exemplo do arquivo de configuração da fonte de dados **jboss-seam-booking-ds.xml** para exemplo do Seam 2.2 Booking que foi enviado junto com o JBoss EAP 5.x:

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
  <local-tx-datasource>
    <jndi-name>bookingDataSource</jndi-name>
    <connection-url>jdbc:hsqldb:./</connection-url>
    <driver-class>org.hsqldb.jdbcDriver</driver-class>
    <user-name>sa</user-name>
    <password></password>
  </local-tx-datasource>
</datasources>
```

Segue abaixo o arquivo de configuração que foi gerado pela execução do script conversor. O arquivo gerado contém um elemento **<driver-class>**. A melhor maneira de definir a classe de driver no JBoss EAP 6 é usando um elemento **<driver>**. Segue abaixo o XML resultante no arquivo de configuração do JBoss EAP 6 com modificações para converter em comentário o elemento **<driver-class>** e adicionar o elemento correspondente **<driver>**:

```
<subsystem xmlns="urn:jboss:domain:datasources:1.1">
  <datasources>
    <datasource enabled="true" jndi-
name="java:jboss/datasources/bookingDataSource" jta="true"
      pool-name="bookingDataSource" use-ccm="true" use-java-
context="true">
      <connection-url>jdbc:hsqldb:./</connection-url>
      <!-- Comment out the following driver-class element
      since it is not the preferred way to define this.
      <driver-class>org.hsqldb.jdbcDriver</driver-class>
      -->
      <!-- Specify the driver, which is defined later in the
      datasource -->
      <driver>h2</driver>
      <transaction-isolation>TRANSACTION_NONE</transaction-
isolation>
      <pool>
        <prefill>false</prefill>
        <use-strict-min>false</use-strict-min>
        <flush-strategy>FailingConnectionOnly</flush-strategy>
      </pool>
```

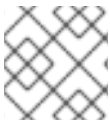
```

<security>
  <user-name>sa</user-name>
  <password/>
</security>
<validation>
  <validate-on-match>>false</validate-on-match>
  <background-validation>>false</background-validation>
  <use-fast-fail>>false</use-fast-fail>
</validation>
<timeout/>
<statement>
  <track-statements>>false</track-statements>
</statement>
</datasource>
<drivers>
  <!-- The following driver element was not in the
XML target file. It was created manually. -->
  <driver name="h2" module="com.h2database.h2">
    <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-
datasource-class>
  </driver>
</drivers>
</datasources>
</subsystem>

```

[Reportar um erro](#)

4.1.9. Uso da Ferramenta de Migração IronJacamar para Converter um Arquivo de Configuração do Adaptador de Recursos



NOTA

O script convertor IronJacamar requer Java 7 ou versão posterior.

1. Abra uma linha de comando e navegue até o diretório **IRONJACAMAR_HOME/docs/as/**.
2. Execute o script conversor digitando o seguinte comando:

- Para Linux: **./converter.sh -ra SOURCE_FILE TARGET_FILE**
- Para Microsoft Windows: **./converter.bat -ra SOURCE_FILE TARGET_FILE**

SOURCE_FILE é o arquivo -ds.xml do adaptador de recursos da versão anterior. **TARGET_FILE** contém a nova configuração.

Por exemplo, para converter o arquivo de configuração do adaptador de recursos **mttestadapter-ds.xml** localizado no diretório atual, você digitaria:

- Para Linux: **./converter.sh -ra mttestadapter-ds.xml new-adapter-config.xml**
- Para Microsoft Windows: **./converter.bat -ra mttestadapter-ds.xml new-adapter-config.xml**

Perceba que o parâmetro para a conversão do adaptador de recursos é **-ra**.

3. Copie o elemento inteiro **<resource-adapters>** do arquivo de destino e cole-o no arquivo de configuração do servidor sob o elemento **<subsystem**
xmlns="urn:jboss:domain:resource-adapters:1.1">.



IMPORTANTE

Você deve interromper o servidor antes de editar o arquivo de configuração do servidor para que sua alteração seja efetivada ao reiniciar o servidor.

- o Caso você esteja executando em um domínio gerenciado, copie o XML no arquivo **EAP_HOME/domain/configuration/domain.xml**.
 - o Caso você esteja executando como um servidor autônomo, copie o XML no arquivo **EAP_HOME/standalone/configuration/standalone.xml**.
4. Modifique o XML gerado no novo arquivo de configuração.

Segue abaixo um exemplo do arquivo de configuração do adaptador de recursos **mttestadapter-ds.xml** a partir do JBoss EAP 5.x TestSuite:

```
<?xml version="1.0" encoding="UTF-8"?>
  <!--
=====
-->
  <!-- ConnectionManager setup for jboss test adapter
-->
  <!-- Build jmx-api (build/build.sh all) and view for config
documentation -->
  <!--
=====
-->
<connection-factories>
  <tx-connection-factory>
    <jndi-name>JBossTestCF</jndi-name>
    <xa-transaction/>
    <rar-name>jbosstestadapter.rar</rar-name>
    <connection-
definition>javax.resource.cci.ConnectionFactory</connection-
definition>
    <config-property name="IntegerProperty"
type="java.lang.Integer">2</config-property>
    <config-property name="BooleanProperty"
type="java.lang.Boolean">>false</config-property>
    <config-property name="DoubleProperty"
type="java.lang.Double">5.5</config-property>
    <config-property name="UrlProperty"
type="java.net.URL">http://www.jboss.org</config-property>
    <config-property name="sleepInStart" type="long">200</config-
property>
    <config-property name="sleepInStop" type="long">200</config-
property>
  </tx-connection-factory>
  <tx-connection-factory>
```

```

    <jndi-name>JBossTestCF2</jndi-name>
    <xa-transaction/>
    <rar-name>jbosstestadapter.rar</rar-name>
    <connection-
definition>javax.resource.cci.ConnectionFactory</connection-
definition>
    <config-property name="IntegerProperty"
type="java.lang.Integer">2</config-property>
    <config-property name="BooleanProperty"
type="java.lang.Boolean">>false</config-property>
    <config-property name="DoubleProperty"
type="java.lang.Double">5.5</config-property>
    <config-property name="UrlProperty"
type="java.net.URL">http://www.jboss.org</config-property>
    <config-property name="sleepInStart" type="long">200</config-
property>
    <config-property name="sleepInStop" type="long">200</config-
property>
  </tx-connection-factory>
  <tx-connection-factory>
    <jndi-name>JBossTestCFByTx</jndi-name>
    <xa-transaction/>
    <track-connection-by-tx>true</track-connection-by-tx>
    <rar-name>jbosstestadapter.rar</rar-name>
    <connection-
definition>javax.resource.cci.ConnectionFactory</connection-
definition>
    <config-property name="IntegerProperty"
type="java.lang.Integer">2</config-property>
    <config-property name="BooleanProperty"
type="java.lang.Boolean">>false</config-property>
    <config-property name="DoubleProperty"
type="java.lang.Double">5.5</config-property>
    <config-property name="UrlProperty"
type="java.net.URL">http://www.jboss.org</config-property>
    <config-property name="sleepInStart" type="long">200</config-
property>
    <config-property name="sleepInStop" type="long">200</config-
property>
  </tx-connection-factory>
</connection-factories>

```

Segue abaixo o arquivo de configuração que foi gerado pela execução do script conversor. Substitua o valor do atributo do nome da classe "FIXME_MCF_CLASS_NAME" no XML gerado com o nome de classe correto da fábrica de conexão gerenciada, neste caso "org.jboss.test.jca.adapter.TestManagedConnectionFactory". Este é o XML resultante no arquivo de configuração do JBoss EAP 6 com modificações no valor do elemento **<class-name>**:

```

<subsystem xmlns="urn:jboss:domain:resource-adapters:1.1">
  <resource-adapters>
    <resource-adapter>
      <archive>jbosstestadapter.rar</archive>
      <transaction-support>XATransaction</transaction-support>
      <connection-definitions>
        <!-- Replace the "FIXME_MCF_CLASS_NAME" class-name value with the

```

```

correct class name
  <connection-definition class-name="FIXME_MCF_CLASS_NAME"
enabled="true"
  jndi-name="java:jboss/JBossTestCF" pool-name="JBossTestCF"
  use-ccm="true" use-java-context="true"> -->
<connection-definition
  class-
name="org.jboss.test.jca.adapter.TestManagedConnectionFactory"
  enabled="true"
  jndi-name="java:jboss/JBossTestCF" pool-name="JBossTestCF"
  use-ccm="true" use-java-context="true">
  <config-property name="IntegerProperty">2</config-property>
  <config-property name="sleepInStart">200</config-property>
  <config-property name="sleepInStop">200</config-property>
  <config-property name="BooleanProperty">>false</config-property>
  <config-property
name="UrlProperty">http://www.jboss.org</config-property>
  <config-property name="DoubleProperty">5.5</config-property>
  <pool>
    <prefill>>false</prefill>
    <use-strict-min>>false</use-strict-min>
    <flush-strategy>FailingConnectionOnly</flush-strategy>
  </pool>
  <security>
    <application/>
  </security>
  <timeout/>
  <validation>
    <background-validation>>false</background-validation>
    <use-fast-fail>>false</use-fast-fail>
  </validation>
</connection-definition>
  </connection-definitions>
</resource-adapter>
<resource-adapter>
  <archive>jbosstestadapter.rar</archive>
  <transaction-support>XATransaction</transaction-support>
  <connection-definitions>
    <!-- Replace the "FIXME_MCF_CLASS_NAME" class-name value with the
correct class name
    <connection-definition class-name="FIXME_MCF_CLASS_NAME"
enabled="true"
    jndi-name="java:jboss/JBossTestCF2" pool-name="JBossTestCF2"
    use-ccm="true" use-java-context="true"> -->
  <connection-definition
    class-
name="org.jboss.test.jca.adapter.TestManagedConnectionFactory"
    enabled="true"
    jndi-name="java:jboss/JBossTestCF2" pool-name="JBossTestCF2"
    use-ccm="true" use-java-context="true">
    <config-property name="IntegerProperty">2</config-property>
    <config-property name="sleepInStart">200</config-property>
    <config-property name="sleepInStop">200</config-property>
    <config-property name="BooleanProperty">>false</config-property>
    <config-property
name="UrlProperty">http://www.jboss.org</config-property>

```



```

<config-property name="DoubleProperty">5.5</config-property>
<pool>
  <prefill>false</prefill>
  <use-strict-min>false</use-strict-min>
  <flush-strategy>FailingConnectionOnly</flush-strategy>
</pool>
<security>
  <application/>
</security>
<timeout/>
<validation>
  <background-validation>false</background-validation>
  <use-fast-fail>false</use-fast-fail>
</validation>
</connection-definition>
</connection-definitions>
</resource-adapter>
<resource-adapter>
  <archive>jbosstestadapter.rar</archive>
  <transaction-support>XATransaction</transaction-support>
  <connection-definitions>
    <!-- Replace the "FIXME_MCF_CLASS_NAME" class-name value with the
correct class name
    <connection-definition class-name="FIXME_MCF_CLASS_NAME"
enabled="true"
      jndi-name="java:jboss/JBossTestCFByTx" pool-
name="JBossTestCFByTx"
      use-ccm="true" use-java-context="true"> -->
    <connection-definition
      class-
name="org.jboss.test.jca.adapter.TestManagedConnectionFactory"
      enabled="true"
      jndi-name="java:jboss/JBossTestCFByTx" pool-
name="JBossTestCFByTx"
      use-ccm="true" use-java-context="true">
      <config-property name="IntegerProperty">2</config-property>
      <config-property name="sleepInStart">200</config-property>
      <config-property name="sleepInStop">200</config-property>
      <config-property name="BooleanProperty">false</config-property>
      <config-property
name="UrlProperty">http://www.jboss.org</config-property>
      <config-property name="DoubleProperty">5.5</config-property>
      <pool>
        <prefill>false</prefill>
        <use-strict-min>false</use-strict-min>
        <flush-strategy>FailingConnectionOnly</flush-strategy>
      </pool>
      <security>
        <application/>
      </security>
      <timeout/>
      <validation>
        <background-validation>false</background-validation>
        <use-fast-fail>false</use-fast-fail>
      </validation>
    </connection-definition>
  </connection-definitions>
</resource-adapter>

```



```

        </connection-definitions>
    </resource-adapter>
</resource-adapters>
</subsystem>

```

[Reportar um erro](#)

4.2. DEPURAÇÃO DE PROBLEMAS RELACIONADOS À MIGRAÇÃO

4.2.1. Depuração e Solução de Problemas de Migração

Devido ao carregamento de classe, às regras de nomeação JNDI e outras alterações no servidor do aplicativo, você pode encontrar exceções ou outros erros caso tente implantar seu aplicativo "como-está" (as-is). Segue abaixo a descrição de como resolver as exceções e os erros mais comuns que você pode encontrar.

- [Seção 4.2.2, “Depuração e Solução de ClassNotFoundExceptions e NoClassDefFoundErrors”](#)
- [Seção 4.2.5, “Depuração e Resolução de ClassCastExceptions”](#)
- [Seção 4.2.6, “Depuração e Resolução de DuplicateServiceExceptions”](#)
- [Seção 4.2.7, “Depuração e Resolução de Erros da Página de Depuração do JBoss Seam”](#)

[Reportar um erro](#)

4.2.2. Depuração e Solução de ClassNotFoundExceptions e NoClassDefFoundErrors

Sumário

ClassNotFoundExceptions ocorre normalmente devido a uma dependência não-resolvida. Isto significa que você deve definir explicitamente as dependências em outros módulos ou copiar os JARs de fontes externas.

1. Primeiro, tente encontrar a dependência ausente. Mais informações podem ser encontradas aqui [Seção 4.2.3, “Localização da Dependência de Módulo do JBoss ”](#)
2. Caso não exista um módulo para a classe ausente, localize o JAR na instalação anterior. Para mais informações, consulte [Seção 4.2.4, “Localização do JAR na Instalação Anterior”](#)

[Reportar um erro](#)

4.2.3. Localização da Dependência de Módulo do JBoss

Para resolver a dependência, primeiro tente localizar o módulo que contém a classe especificada pelo **ClassNotFoundException** pesquisando no diretório

EAP_HOME/modules/system/layers/base/. Se você encontrar um módulo para a classe, você deve adicionar uma dependência à entrada do manifesto.

Por exemplo, se você encontrar este rastreamento ClassNotFoundException no log:

```

Caused by: java.lang.ClassNotFoundException:
org.apache.commons.logging.Log
    from [Module "deployment.TopicIndex.war:main" from Service Module

```

```
Loader]
    at
    org.jboss.modules.ModuleClassLoader.findClass(ModuleClassLoader.java:188)
```

Localize o módulo do JBoss contendo essa classe fazendo o seguinte:

Procedimento 4.5. Localização da Dependência

1. Primeiro, determine se existe um módulo óbvio para a classe.
 - a. Navegue até o diretório **EAP_HOME/modules/system/layers/base/** e procure pela classe correspondendo com o caminho do módulo nomeada em **ClassNotFoundException**.

Você encontrará o caminho do módulo **org/apache/commons/logging/**.
 - b. Abra o arquivo **EAP_HOME/modules/system/layers/base/org/apache/commons/logging/main/module.xml** e localize o nome do módulo que é, neste caso, "org.apache.commons.logging".
 - c. Adicione o nome do módulo às Dependências no arquivo **MANIFEST.MF**:

```
Manifest-Version: 1.0
Dependencies: org.apache.commons.logging
```

2. Caso não haja um caminho de módulo óbvio para a classe, é possível que tenha que encontrar a dependência em outra localização.
 - a. Localize a classe nomeada pela **ClassNotFoundException** no relatório Tattletale.
 - b. Localize o módulo contendo o JAR no diretório **EAP_HOME/modules** e localize o nome do módulo como na etapa anterior.

[Reportar um erro](#)

4.2.4. Localização do JAR na Instalação Anterior

Caso a classe não seja encontrada em um JAR empacotado em um módulo definido pelo servidor, procure o JAR na sua instalação **EAP5_HOME** ou no diretório **lib/** do seu servidor anterior.

Por exemplo, caso você encontre este rastreamento **ClassNotFoundException** no log:

```
Caused by: java.lang.NoClassDefFoundError:
org.hibernate.validator.ClassValidator at
java.lang.Class.getDeclaredMethods0(Native Method)
```

Procure pelo JAR contendo essa classe fazendo o seguinte:

1. Abra um terminal e navegue até o diretório **EAP5_HOME/**.
2. Emita o comando:

```
grep 'org.hibernate.validator.ClassValidator' `find . -name '*.jar'`
```

3. Você pode encontrar mais de um resultado. Neste caso, o resultado a seguir é o JAR que precisamos:

```
Binary file ./jboss-eap-5.1/seam/lib/hibernate-validator.jar matches
```

4. Copie esse JAR ao diretório **lib/** do aplicativo.

Se achar que precisa de um número grande de JARs, pode ser mais fácil definir um módulo para as classes. Para mais informações, consulte *Modules* no capítulo nomeado *Get Started Developing Applications* no *Development Guide* para o JBoss EAP 6 em https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

5. Recompile e implante o aplicativo novamente.

[Reportar um erro](#)

4.2.5. Depuração e Resolução de ClassCastExceptions

ClassCastExceptions acontece com frequência quando uma classe é carregada por um carregador de classe diferente da classe que estende. Também pode ser o resultado da mesma classe existindo em múltiplos JARs.

1. Procure pelo aplicativo para encontrar todo(s) JAR(s) que contém a classe nomeada pela **ClassCastException**. Caso exista um módulo definido para a classe, encontre e remova os JAR(s) duplicados do WAR ou EAR do aplicativo.
2. Localize o módulo do JBoss contendo a classe e defina explicitamente a dependência no arquivo **MANIFEST.MF** ou no arquivo **jboss-deployment-structure.xml**. Para mais informações, consulte *Class Loading and Subdeployments* no capítulo entitulado *Class Loading and Modules* no *Development Guide* para o JBoss EAP 6 em https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.
3. Caso não consiga resolver isto seguindo as etapas acima, geralmente é possível determinar a causa do problema imprimindo a informação do carregador de classe no log. Por exemplo, você encontrará o seguinte **ClassCastException** no log:

```
java.lang.ClassCastException: com.example1.CustomClass1 não foi
possível converter ao com.example2.CustomClass2
```

- a. No seu código, imprima a informação do carregador de classe para as classes nomeadas pela **ClassCastException** no log, por exemplo:

```
logger.info("Class loader for CustomClass1: " +
com.example1.CustomClass1.getClass().getClassLoader().toString())
;
logger.info("Class loader for CustomClass2: " +
com.example2.CustomClass2.getClass().getClassLoader().toString())
;
```

- b. A informação no log mostra quais módulos estão carregando as classes e, baseado no seu aplicativo, será necessário remover ou mover o(s) JAR(s) em conflito.

[Reportar um erro](#)

4.2.6. Depuração e Resolução de DuplicateServiceExceptions

Caso obtenha uma DuplicateServiceException para uma subimplantação de um JAR ou uma mensagem que o aplicativo WAR já foi instalado, quando implantar seu EAR no JBoss EAP 6, pode ser devido às alterações na maneira como JBossWS lida com a implantação.

A versão JBossWS 3.3.0 introduziu um novo novo Algoritmo de Mapeamento de Raiz de Contexto para os pontos de extremidade baseados no servlet para torná-lo compatível com TCK6 sem dificuldades. Se o arquivo EAR do aplicativo conter um JAR com o mesmo nome, JBossWS pode criar um contexto WAR e um contexto web com o mesmo nome. O contexto web entra em conflito com o contexto WAR e isto resulta em erros de implantação. Resolva os problemas de implantação de uma das seguintes maneiras:

- Renomeie o arquivo JAR com um nome diferente do WAR, de forma que os contextos web e WAR gerados sejam únicos.
- Forneça um elemento `<context-root>` no arquivo `jboss-web.xml`.
- Forneça um elemento `<context-root>` no arquivo `jboss-webservices.xml`.
- Personalize o elemento `<context-root>` para o WAR no arquivo `application.xml`.

[Reportar um erro](#)

4.2.7. Depuração e Resolução de Erros da Página de Depuração do JBoss Seam

Após migrar e implantar o seu aplicativo com sucesso, é possível encontrar um erro de tempo de execução que o redirecione à página "Depuração do JBoss Seam". O URL para esta página é "http://localhost:8080/APPLICATION_CONTEXT/debug.seam". Esta página permite que você visualize e inspecione os componentes Seam em qualquer contexto Seam associado com a sua sessão de login atual.

JBoss Seam Debug Page

This page allows you to browse and inspect components in any of the Seam contexts associated with the current session. It also shows a list of active, long-running conversations. You can select a conversation to view its contents or destroy it.

Conversations

Conversation ID	Nested?	Activity	Description	View ID	Action
15	false	12:26:58 PM - 12:27:01 PM	Book hotel: Hilton Diagonal Mar	/book.xhtml	Select Destroy

- + Component
- + Conversation Context (None selected)
- + Business Process Context
- + Session Context
- + Application Context

Figura 4.1. Página de Depuração do JBoss Seam

A razão mais provável de você ter sido redirecionado para esta página é o Seam ter capturado uma

Exceção que não foi tratada no código do aplicativo. A causa principal da exceção é encontrada, normalmente, em um dos links na página "Boss Seam Debug Page" (Página de Depuração do JBoss Seam).

1. Amplie a seção **Component** na página e procure pelo componente **org.jboss.seam.caughtException**.
2. A causa e o rastreamento da pilha devem levá-lo às dependências ausentes.

JBoss Seam Debug Page

This page allows you to browse and inspect components in any of the Seam contexts associated with the current session. It also shows a list of active, long-running conversations. You can select a conversation to view its contents or destroy it.

Conversations

Conversation ID	Nested?	Activity	Description	View ID	Action
15	false	12:26:58 PM - 12:27:01 PM	Book hotel: Hilton Diagonal Mar	/book.xhtml	Select Destroy

- Component

Select a component from one of the contexts below

[- Component \(org.jboss.seam.caughtException\)](#)

cause	java.lang.NoClassDefFoundError: org/slf4j/LoggerFactory
class	class javax.servlet.ServletException
localizedMessage	Servlet execution threw an exception
message	Servlet execution threw an exception
rootCause	java.lang.NoClassDefFoundError: org/slf4j/LoggerFactory
stackTrace	[org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:346), org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:248), org.jboss.seam.servlet.SeamFilter\$FilterChainImpl.doFilter(SeamFilter.java:83), org.jboss.seam.web.IdentityFilter.doFilter(IdentityFilter.java:40), [... rest of stacktrace omitted for display purposes]
toString()	javax.servlet.ServletException: Servlet execution threw an exception

+ Conversation Context (None selected)

+ Business Process Context

+ Session Context

+ Application Context

Figura 4.2. Informações sobre o componente org.jboss.seam.caughtException

3. Use a técnica descrita em [Seção 4.2.2, "Depuração e Solução de ClassNotFoundExceptions e NoClassDefFoundErrors"](#) para resolver as dependências do módulo.

No exemplo acima, a solução mais simples é adicionar **org.slf4j** ao **MANIFEST.MF**

```
Manifest-Version: 1.0
Dependencies: org.slf4j
```

Outra opção é adicionar uma dependência ao módulo no arquivo **jboss-deployment-structure.xml**:

```
<jboss-deployment-structure>
  <deployment>
    <dependencies>
      <module name="org.slf4j" />
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

```

    </dependencies>
  </deployment>
</jboss-deployment-structure>

```

[Reportar um erro](#)

4.3. REVISÃO DA MIGRAÇÃO DOS APLICATIVOS DE EXEMPLO

4.3.1. Revisão da Migração dos Aplicativos de Exemplo

Visão Geral

Segue abaixo uma lista dos aplicativos de exemplo do JBoss EAP 5.x que foram migrados para o JBoss EAP 6. Para ver os detalhes sobre o que mudou em um aplicativo em particular, clique nos links abaixo.

- [Seção 4.3.2, “Migração do Exemplo Seam 2.2 JPA para o JBoss EAP 6”](#)
- [Seção 4.3.3, “Migração de Exemplo do Seam 2.2 Booking para o JBoss EAP 6”](#)
- [Seção 4.3.4, “Migração do Arquivo do Seam 2.2 Booking para JBoss EAP 6: Instruções Passo a Passo”](#)

[Reportar um erro](#)

4.3.2. Migração do Exemplo Seam 2.2 JPA para o JBoss EAP 6

Sumário

A lista de tarefas a seguir resume as alterações necessárias para migrar com êxito o aplicativo de exemplo Seam 2.2 JPA para o JBoss EAP 6. Este aplicativo de exemplo pode ser encontrado na distribuição mais recente do JBoss EAP 5 sob **EAP5.x_HOME/jboss-eap-5.x/seam/examples/jpa/**



IMPORTANTE

Os aplicativos que usam Hibernate diretamente com Seam 2.2 podem usar uma versão do Hibernate 3 empacotada dentro do aplicativo. Hibernate 4, que é fornecido através do módulo org.hibernate do JBoss EAP 6, não é suportado pelo Seam 2.2. Este exemplo tem como objetivo ajudá-lo a executar o seu aplicativo no JBoss EAP 6 como uma primeira etapa. Lembre-se de que o empacotamento do Hibernate 3 com um aplicativo Seam 2.2 não possui uma configuração suportada.

Procedimento 4.6. Migração do Exemplo Seam 2.2 JPA

1. Remova o arquivo jboss-web.xml

Remova o arquivo **jboss-web.xml** do diretório **jboss-seam-jpa.war/WEB-INF/**. O carregamento da classe definido no **jboss-web.xml** é agora o comportamento padrão.

2. Modifique o arquivo jboss-seam-jpa.jar/META-INF/persistence.xml como a seguir.

- Remova ou converta em comentário a propriedade **hibernate.cache.provider_class** no arquivo **jboss-seam-jpa.war/WEB-INF/classes/META-INF/persistence.xml**:

```

<!-- <property name="hibernate.cache.provider_class"

```

```
value="org.hibernate.cache.HashtableCacheProvider"/> -->
```

- b. Adicione a propriedade do módulo do provedor ao arquivo **jboss-seam-booking.jar/META-INF/persistence.xml**:

```
<property name="jboss.as.jpa.providerModule" value="hibernate3-bundled" />
```

- c. Altere a propriedade **jta-data-source** para utilizar o nome JNDI da fonte de dados JDBC padrão:

```
<jta-data-source>java:jboss/datasources/ExampleDS</jta-data-source>
```

3. Adicione as dependências Seam 2.2

Copie os JARs a seguir da biblioteca de distribuição Seam 2.2, **SEAM_HOME/lib/**, no diretório **jboss-seam-jpa.war/WEB-INF/lib/**:

- o antlr.jar
- o slf4j-api.jar
- o slf4j-log4j12.jar
- o hibernate-entitymanager.jar
- o hibernate-core.jar
- o hibernate-annotations.jar
- o hibernate-commons-annotations.jar
- o hibernate-validator.jar

4. Crie um arquivo jboss-deployment-structure para adicionar as dependências restantes

Crie um arquivo **jboss-deployment-structure.xml** na pasta **jboss-seam-jpa.war/WEB-INF/** contendo os seguintes dados:

```
<jboss-deployment-structure>
  <deployment>
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
      <module name="com.sun.jsf-impl" slot="main"/>
      <module name="org.hibernate" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="org.apache.log4j" />
      <module name="org.dom4j" />
      <module name="org.apache.commons.logging" />
      <module name="org.apache.commons.collections" />
      <module name="javax.faces.api" slot="1.2"/>
      <module name="com.sun.jsf-impl" slot="1.2"/>
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```


Resultado:

O aplicativo de exemplo Seam 2.2 JPA implanta e executa com êxito no JBoss EAP 6.

[Reportar um erro](#)

4.3.3. Migração de Exemplo do Seam 2.2 Booking para o JBoss EAP 6

Sumário

A migração do Seam 2.2 Booking EAR é mais complicada do que o exemplo do Seam 2.2 JPA WAR. A documentação para a migração de exemplo do Seam 2.2 JPA WAR pode ser encontrada aqui [Seção 4.3.2, “Migração do Exemplo Seam 2.2 JPA para o JBoss EAP 6”](#). Para migrar o aplicativo, por favor realize o seguinte:

1. Inicialize JSF 1.2 ao invés do JSF 2 padrão.
2. Empacote as versões antigas dos JARs Hibernate ao invés de usar aquelas que foram enviadas junto com o JBoss EAP 6.
3. Altere as vinculações JNDI para uso da nova sintaxe portátil do Java EE 6 JNDI.

As primeiras duas etapas acima foram feitas na migração de exemplo do Seam 2.2 JPA WAR. A terceira etapa é nova e necessária já que o EAR contém EJBs.



IMPORTANTE

Os aplicativos que usam Hibernate diretamente com Seam 2.2 podem usar uma versão do Hibernate 3 empacotada dentro do aplicativo. Hibernate 4, que é fornecido através do módulo org.hibernate do JBoss EAP 6, não é suportado pelo Seam 2.2. Este exemplo tem o propósito de ajudá-lo a executar o seu aplicativo no JBoss EAP 6 como uma primeira etapa. Você deve estar ciente de que o empacotamento do Hibernate 3 com um aplicativo Seam 2.2 não possui uma configuração suportada.

Procedimento 4.7. Migração de exemplo do Seam 2.2 Booking

1. Crie o arquivo `jboss-deployment-structure.xml`

Crie um novo arquivo nomeado `jboss-deployment-structure.xml` no `jboss-seam-booking.ear/META-INF/` e adicione o seguinte conteúdo:

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="com.sun.jsf-impl" slot="1.2"
export="true"/>
      <module name="org.apache.log4j" export="true"/>
      <module name="org.dom4j" export="true"/>
      <module name="org.apache.commons.logging" export="true"/>
      <module name="org.apache.commons.collections"
export="true"/>
    </dependencies>
    <exclusions>
      <module name="org.hibernate" slot="main"/>
    </exclusions>
  </deployment>
</jboss-deployment-structure>
```



```

</deployment>
<sub-deployment name="jboss-seam-booking.war">
  <exclusions>
    <module name="javax.faces.api" slot="main"/>
    <module name="com.sun.jsf-impl" slot="main"/>
  </exclusions>
  <dependencies>
    <module name="javax.faces.api" slot="1.2"/>
    <module name="com.sun.jsf-impl" slot="1.2"/>
  </dependencies>
</sub-deployment>
</jboss-deployment-structure>

```

2. Modifique o arquivo **jboss-seam-booking.jar/META-INF/persistence.xml** como a seguir.

- a. Remova ou converta em comentário a propriedade hibernate para a classe do provedor do cache:

```

<!-- <property name="hibernate.cache.provider_class"
value="org.hibernate.cache.HashtableCacheProvider"/> -->

```

- b. Adicione a propriedade de módulo do provedor ao arquivo **jboss-seam-booking.jar/META-INF/persistence.xml**:

```

<property name="jboss.as.jpa.providerModule" value="hibernate3-
bundled" />

```

- c. Altere a propriedade **jta-data-source** para usar o nome JNDI da fonte de dados JDBC padrão:

```

<jta-data-source>java:jboss/datasources/ExampleDS</jta-data-
source>

```

3. Copie os JARs da distribuição Seam 2.2

Copie os JARs a seguir da distribuição Seam 2.2 **EAP5.x_HOME/jboss-eap5.x/seam/lib/** no diretório **jboss-seam-booking.ear/lib**.

```

antlr.jar
slf4j-api.jar
slf4j-log4j12.jar
hibernate-core.jar
hibernate-entitymanager.jar
hibernate-validator.jar
hibernate-annotations.jar
hibernate-commons-annotations.jar

```

4. Altere os nomes de pesquisa JNDI

Altere as cadeias de pesquisa JNDI no arquivo **jboss-seam-booking.war/WEB-INF/components.xml**. Devido a novas regras portáteis JNDI, o JBoss EAP 6 agora vincula EJBs usando as regras de sintaxe portáteis JNDI e não é possível usar o `jndiPattern` que era usado no JBoss EAP 5. Segue abaixo as cadeias de pesquisa JNDI EJB do aplicativo que devem ser alteradas para o JBoss EAP 6:

```

java:global/jboss-seam-booking/jboss-seam-
booking/HotelSearchingAction!org.jboss.seam.example.booking.HotelSea
rching
java:app/jboss-seam-
booking/HotelSearchingAction!org.jboss.seam.example.booking.HotelSea
rching
java:module/HotelSearchingAction!org.jboss.seam.example.booking.Hote
lSearching
java:global/jboss-seam-booking/jboss-seam-
booking/HotelSearchingAction
java:app/jboss-seam-booking/HotelSearchingAction
java:module/HotelSearchingAction

```

As cadeias de pesquisa JNDI para os EJBs de estrutura Seam 2.2 devem ser alteradas conforme abaixo:

```

java:global/jboss-seam-booking/jboss-
seam/EjbSynchronizations!org.jboss.seam.transaction.LocalEjbSynchron
izations
java:app/jboss-
seam/EjbSynchronizations!org.jboss.seam.transaction.LocalEjbSynchron
izations
java:module/EjbSynchronizations!org.jboss.seam.transaction.LocalEjbS
ynchronizations
java:global/jboss-seam-booking/jboss-seam/EjbSynchronizations
java:app/jboss-seam/EjbSynchronizations
java:module/EjbSynchronizations

```

Você pode usar uma das seguintes abordagens:

a. **Adicionar os elementos do componente**

Você pode adicionar um **jndi-name** para cada EJB ao **WEB-INF/components.xml**:

```

<component
class="org.jboss.seam.transaction.EjbSynchronizations" jndi-
name="java:app/jboss-seam/EjbSynchronizations"/>
<component
class="org.jboss.seam.async.TimerServiceDispatcher" jndi-
name="java:app/jboss-seam/TimerServiceDispatcher"/>
<component
class="org.jboss.seam.example.booking.AuthenticatorAction" jndi-
name="java:app/jboss-seam-booking/AuthenticatorAction" />
<component
class="org.jboss.seam.example.booking.BookingListAction" jndi-
name="java:app/jboss-seam-booking/BookingListAction" />
<component
class="org.jboss.seam.example.booking.RegisterAction" jndi-
name="java:app/jboss-seam-booking/RegisterAction" />
<component
class="org.jboss.seam.example.booking.HotelSearchingAction" jndi-
name="java:app/jboss-seam-booking/HotelSearchingAction" />
<component
class="org.jboss.seam.example.booking.HotelBookingAction" jndi-
name="java:app/jboss-seam-booking/HotelBookingAction" />

```

```
<component
class="org.jboss.seam.example.booking.ChangePasswordAction" jndi-
name="java:app/jboss-seam-booking/ChangePasswordAction" />
```

- b. Você pode modificar o código adicionando a anotação `@JNDIName(value="")` e especificando o caminho JNDI. Segue abaixo um exemplo do código de bean de sessão sem monitorização de estado alterado. Uma descrição detalhada deste processo está disponível na documentação de referência do Seam 2.2.

```
@Stateless
@Name("authenticator")
@JndiName(value="java:app/jboss-seam-
booking/AuthenticatorAction")
public class AuthenticatorAction
    implements Authenticator
{
    ...
}
```

Resultado:

O aplicativo Seam 2.2 Booking é implantado e executado com êxito no JBoss EAP 6.

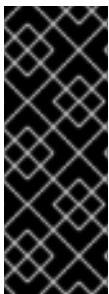
[Reportar um erro](#)

4.3.4. Migração do Arquivo do Seam 2.2 Booking para JBoss EAP 6: Instruções Passo a Passo

Este é um guia passo a passo sobre como portar o arquivo do aplicativo Seam 2.2 Booking do JBoss EAP 5.X para o JBoss EAP 6. Embora existam abordagens melhores para migrar aplicativos, muitos desenvolvedores podem ficar tentados a implantar o arquivo do aplicativo tal como está para o servidor do JBoss EAP 6 para ver o que acontece. A intenção desta documentação é apresentar os tipos de problema que você pode encontrar quando faz isto e como depurar e resolver esses problemas.

Para este exemplo, o aplicativo EAR é implantado no diretório

EAP6_HOME/standalone/deployments sem nenhuma alteração, além da extração dos arquivos. Isto permite que você modifique com facilidade os arquivos XML contidos nos arquivos à medida que você encontra e resolve os problemas.



IMPORTANTE

Os aplicativos que usam Hibernate diretamente com Seam 2.2 podem usar uma versão do Hibernate 3 empacotada dentro do aplicativo. Hibernate 4, que é fornecido através do módulo org.hibernate do JBoss EAP 6, não é suportado pelo Seam 2.2. Este exemplo tem o propósito de ajudá-lo executar o seu aplicativo no JBoss EAP 6 como um primeiro passo. Você deve estar ciente de que o empacotamento do Hibernate 3 com um aplicativo Seam 2.2 não possui uma configuração suportada.

Procedimento 4.8. Migração do aplicativo

1. [Seção 4.3.5, “Construção e Implantação da Versão JBoss EAP 5.X do Aplicativo Seam 2.2 Booking”](#)
2. [Seção 4.3.6, “Depuração e Resolução de Exceções e Erros de Implantação do Arquivo do Seam 2.2 Booking”](#)

3. [Seção 4.3.7, “Depuração e Resolução de Exceções e Erros de Tempo de Execução do Arquivo do Seam 2.2 Booking”](#)

A esta altura, é possível acessar o aplicativo com sucesso em um navegador usando o URL <http://localhost:8080/seam-booking/>. Faça o login como demo/demo e encontrará a página de boas vindas do Booking.

Revise o resumo das alterações

[Seção 4.3.8, “Revisão do Sumário das Alterações Feitas Quando Migrando o Aplicativo Seam 2.2 Booking”](#)

[Reportar um erro](#)

4.3.5. Construção e Implantação da Versão JBoss EAP 5.X do Aplicativo Seam 2.2 Booking

Antes de migrar este aplicativo, é necessário construir o aplicativo Seam 2.2 Booking do JBoss EAP 5.X, extrair o arquivo e copiá-lo em uma pasta de implantação do JBoss EAP 6.

Procedimento 4.9. Construção e implantação do EAR

1. Construção do EAR:

```
$ cd /EAP5_HOME/jboss-eap5.x/seam/examples/booking
$ ANT_HOME/ant explode
```

Substitua *jboss-eap5.x* pela versão do JBoss EAP a partir da qual você está migrando

2. Copie o EAR para o diretório de implantações *EAP6_HOME*:

```
$ cp -r EAP5_HOME/seam/examples/booking/exploded-archives/jboss-seam-booking.ear EAP6_HOME/standalone/deployments/
$ cp -r EAP5_HOME/seam/examples/booking/exploded-archives/jboss-seam-booking.war EAP6_HOME/standalone/deployments/jboss-seam.ear
$ cp -r EAP5_HOME/seam/examples/booking/exploded-archives/jboss-seam-booking.jar EAP6_HOME/standalone/deployments/jboss-seam.ear
```

3. Inicie o servidor do JBoss EAP 6 e verifique o log. Você encontrará o seguinte:

```
INFO [org.jboss.as.deployment] (DeploymentScanner-threads - 1) Found
jboss-seam-booking.ear in deployment directory.
    To trigger deployment create a file called jboss-seam-
booking.ear.dodeploy
```

4. Crie um arquivo vazio com o nome **jboss-seam-booking.ear.dodeploy** e copie-o no diretório **EAP6_HOME/standalone/deployments**. É preciso copiar este arquivo diversas vezes no diretório de implantações, enquanto estiver migrando este aplicativo. Portanto, mantenha-o em um local onde possa encontrá-lo com facilidade. No log, você deve encontrar as seguintes mensagens, indicando que a implantação está ocorrendo:

```
INFO [org.jboss.as.server.deployment] (MSC service thread 1-1)
Starting deployment of "jboss-seam-booking.ear"
INFO [org.jboss.as.server.deployment] (MSC service thread 1-3)
```

```
Starting deployment of "jboss-seam-booking.jar"
INFO [org.jboss.as.server.deployment] (MSC service thread 1-6)
Starting deployment of "jboss-seam.jar"
INFO [org.jboss.as.server.deployment] (MSC service thread 1-2)
Starting deployment of "jboss-seam-booking.war"
```

A esta altura, você encontrará seu primeiro erro de implantação. No próximo passo, será possível verificar cada problema e aprender como depurá-lo e resolvê-lo.

Consulte [Seção 4.3.6, “Depuração e Resolução de Exceções e Erros de Implantação do Arquivo do Seam 2.2 Booking”](#) para mais informações sobre como depurar e resolver problemas de implantação.

Clique aqui [Seção 4.3.4, “Migração do Arquivo do Seam 2.2 Booking para JBoss EAP 6: Instruções Passo a Passo”](#) para retornar ao tópico anterior.

[Reportar um erro](#)

4.3.6. Depuração e Resolução de Exceções e Erros de Implantação do Arquivo do Seam 2.2 Booking

Nos passos anteriores, [Seção 4.3.5, “Construção e Implantação da Versão JBoss EAP 5.X do Aplicativo Seam 2.2 Booking”](#), o aplicativo Seam 2.2 Booking do JBoss EAP 5.X foi construído e implantado na pasta de implantação do JBoss EAP 6. Neste passo, você depurará e resolverá cada erro de implantação que encontrar.



IMPORTANTE

Os aplicativos que usam Hibernate diretamente com Seam 2.2 podem usar uma versão do Hibernate 3 empacotada dentro do aplicativo. Hibernate 4, que é fornecido através do módulo org.hibernate do JBoss EAP 6, não é suportado pelo Seam 2.2. Este exemplo tem o propósito de ajudá-lo a executar o seu aplicativo no JBoss EAP 6 como um primeiro passo. Você deve estar ciente de que o empacotamento do Hibernate 3 com um aplicativo Seam 2.2 não possui uma configuração suportada.

Procedimento 4.10. Depuração e resolução de exceções e erros de implantação

1. Problema - java.lang.ClassNotFoundException: javax.faces.FacesException

Quando o aplicativo é implantado, o log contém o seguinte erro:

```
ERROR \[org.jboss.msc.service.fail\] (MSC service thread 1-1)
MSC00001: Failed to start service jboss.deployment.subunit."jboss-
seam-booking.ear"."jboss-seam-booking.war".POST_MODULE:
org.jboss.msc.service.StartException in service
jboss.deployment.subunit."jboss-seam-booking.ear"."jboss-seam-
booking.war".POST_MODULE:
Failed to process phase POST_MODULE of subdeployment "jboss-seam-
booking.war" of deployment "jboss-seam-booking.ear"
(.. additional logs removed ...)
Caused by: java.lang.ClassNotFoundException:
javax.faces.FacesException from \[Module "deployment.jboss-seam-
booking.ear:main" from Service Module Loader\]
```

```

        at
org.jboss.modules.ModuleClassLoader.findClass(ModuleClassLoader.java
:191)

```

O que significa:

`ClassNotFoundException` indica uma dependência ausente. Neste caso, a classe **`javax.faces.FacesException`** não pode ser encontrada e é necessário adicionar explicitamente a dependência.

Como resolver isto:

Localize o nome do módulo para aquela classe no diretório

`EAP6_HOME/modules/system/layers/base/` procurando por um caminho que coincida com a classe ausente. Neste caso, você encontra dois módulos que coincidem:

```

javax/faces/api/main
javax/faces/api/1.2

```

Ambos os módulos possuem o mesmo nome de módulo: **`javax.faces.api`**, porém aquele localizado no diretório principal é para JSF 2.0 e o outro, no diretório 1.2, é para JSF 1.2. Caso houvesse apenas um módulo disponível, seria possível simplesmente criar um arquivo **`MANIFEST.MF`** e adicionar a dependência do módulo. Mas, neste caso, você quer usar a versão JSF 1.2, e não a versão 2.0, no principal, então é necessário especificar uma e excluir a outra. Para isto, crie um arquivo **`jboss-deployment-structure.xml`** no diretório do **`META-INF/`** EAR que contenha os seguintes dados:

```

<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="javax.faces.api" slot="1.2"/>
    </dependencies>
  </sub-deployment>
</jboss-deployment-structure>

```

Na seção **`deployment`**, adicione a dependência para **`javax.faces.api`** para o módulo JSF 1.2. Adicione também a dependência para o módulo JSF 1.2 na seção de subimplantação para o WAR e exclua o módulo para JSF 2.0.

Reimplante o aplicativo deletando o arquivo

`EAP6_HOME/standalone/deployments/jboss-seam-booking.ear.failed` e criando um arquivo em branco **`jboss-seam-booking.ear.dodeploy`** no mesmo diretório.

2. Problema - java.lang.ClassNotFoundException: org.apache.commons.logging.Log

Quando o aplicativo é implantado, o log contém o seguinte erro:

```

ERROR [org.jboss.msc.service.fail] (MSC service thread 1-8)
MSC00001: Failed to start service jboss.deployment.unit."jboss-seam-
booking.ear".INSTALL:
org.jboss.msc.service.StartException in service
jboss.deployment.unit."jboss-seam-booking.ear".INSTALL:
Failed to process phase INSTALL of deployment "jboss-seam-
booking.ear"
    (... additional logs removed ...)
Caused by: java.lang.ClassNotFoundException:
org.apache.commons.logging.Log from [Module "deployment.jboss-seam-
booking.ear.jboss-seam-booking.war:main" from Service Module Loader]

```

O que significa:

ClassNotFoundException indica uma dependência ausente. Neste caso, a classe **org.apache.commons.logging.Log** não pode ser encontrada e é necessário adicionar explicitamente a dependência.

Como resolver isto:

Localize o nome do módulo para aquela classe no diretório

EAP6_HOME/modules/system/layers/base/ procurando por um caminho que coincida com a classe ausente. Neste caso, você encontra um módulo que coincide com o caminho **org/apache/commons/logging/**. O nome do módulo é "org.apache.commons.logging".

Modifique o arquivo **jboss-deployment-structure.xml** para adicionar a dependência do módulo à seção de implantação do arquivo.

```
<module name="org.apache.commons.logging" export="true"/>
```

jboss-deployment-structure.xml deve parecer-se com o seguinte:

```

<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="org.apache.commons.logging" export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="javax.faces.api" slot="1.2"/>
    </dependencies>
  </sub-deployment>
</jboss-deployment-structure>

```

Reimplante o aplicativo deletando o arquivo

EAP6_HOME/standalone/deployments/jboss-seam-booking.ear.failed e criando um arquivo em branco **jboss-seam-booking.ear.dodeploy** no mesmo diretório.

3. Problema - java.lang.ClassNotFoundException: org.dom4j.DocumentException

Quando o aplicativo é implantado, o log contém o seguinte erro:

```
ERROR [org.apache.catalina.core.ContainerBase.[jboss.web].[default-host].[/seam-booking]] (MSC service thread 1-3) Exception sending context initialized event to listener instance of class org.jboss.seam.servlet.SeamListener: java.lang.NoClassDefFoundError: org/dom4j/DocumentException
    (... additional logs removed ...)
Caused by: java.lang.ClassNotFoundException:
org.dom4j.DocumentException from [Module "deployment.jboss-seam-booking.ear.jboss-seam.jar:main" from Service Module Loader]
```

O que significa:

ClassNotFoundException indica uma dependência ausente. Neste caso, a classe **org.dom4j.DocumentException** não pode ser encontrada.

Como resolver isto:

Localize o nome do módulo no diretório **EAP6_HOME/modules/system/layers/base/** procurando pelo **org/dom4j/DocumentException**. O nome do módulo é "org.dom4j". Modifique o arquivo **jboss-deployment-structure.xml** para adicionar a dependência do módulo à seção de implantação do arquivo.

```
<module name="org.dom4j" export="true"/>
```

O arquivo **jboss-deployment-structure.xml** deve parecer-se com o seguinte agora:

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="org.apache.commons.logging" export="true"/>
      <module name="org.dom4j" export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="javax.faces.api" slot="1.2"/>
    </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

Reimplante o aplicativo deletando o arquivo

EAP6_HOME/standalone/deployments/jboss-seam-booking.ear.failed e criando um arquivo em branco **jboss-seam-booking.ear.dodeploy** no mesmo diretório.

4. Problema - java.lang.ClassNotFoundException: org.hibernate.validator.InvalidValue

Quando o aplicativo é implantado, o log contém o seguinte erro:

```
ERROR [org.apache.catalina.core.ContainerBase.[jboss.web].[default-
```



```
host].[/seam-booking]] (MSC service thread 1-6) Exception sending
context initialized event to listener instance of class
org.jboss.seam.servlet.SeamListener: java.lang.RuntimeException:
Could not create Component:
org.jboss.seam.international.statusMessages
(... additional logs removed ...)
Caused by: java.lang.ClassNotFoundException:
org.hibernate.validator.InvalidValue from [Module "deployment.jboss-
seam-booking.ear.jboss-seam.jar:main" from Service Module Loader]
```

O que significa:

ClassNotFoundException indica uma dependência ausente. Neste caso, a classe **org.hibernate.validator.InvalidValue** não pode ser encontrada.

Como resolver isto:

Existe um módulo “org.hibernate.validator”, mas o JAR não possui a classe **org.hibernate.validator.InvalidValue**, portanto a adição da dependência do módulo não resolve este problema. Neste caso, o JAR contendo a classe fazia parte da implantação do JBoss EAP 5.X. Procure pelo JAR que contém a classe ausente no diretório **EAP5_HOME/seam/lib/**. Para fazer isto, abra um console e digite o seguinte:

```
$ cd EAP5_HOME/seam/lib
$ grep 'org.hibernate.validator.InvalidValue' `find . -name '*.jar'`
```

O resultado apresenta o seguinte:

```
$ Binary file ./hibernate-validator.jar matches
$ Binary file ./test/hibernate-all.jar matches
```

Neste caso, copie **hibernate-validator.jar** para o diretório **jboss-seam-booking.ear/lib/**

```
$ cp EAP5_HOME/seam/lib/hibernate-validator.jar jboss-seam-
booking.ear/lib
```

Reimplante o aplicativo deletando o arquivo

EAP6_HOME/standalone/deployments/jboss-seam-booking.ear.failed e criando um arquivo em branco **jboss-seam-booking.ear.dodeploy** no mesmo diretório.

5. Problema - java.lang.InstantiationException: org.jboss.seam.jsf.SeamApplicationFactory

Quando o aplicativo é implantado, o log contém o seguinte erro:

```
INFO [javax.enterprise.resource.webcontainer.jsf.config] (MSC
service thread 1-7) Unsanitized stacktrace from failed start...:
com.sun.faces.config.ConfigurationException: Factory
'javax.faces.application.ApplicationFactory' was not configured
properly.
    at
com.sun.faces.config.processor.FactoryConfigProcessor.verifyFactorie
sExist(FactoryConfigProcessor.java:296) [jsf-impl-2.0.4-b09-
jbossorg-4.jar:2.0.4-b09-jbossorg-4]
(... additional logs removed ...)
```

```

Caused by: javax.faces.FacesException:
org.jboss.seam.jsf.SeamApplicationFactory
    at
    javax.faces.FactoryFinder.getImplGivenPreviousImpl(FactoryFinder.java:606) [jsf-api-1.2_13.jar:1.2_13-b01-FCS]
    (... additional logs removed ...)
    at
    com.sun.faces.config.processor.FactoryConfigProcessor.verifyFactoryExist(FactoryConfigProcessor.java:294) [jsf-impl-2.0.4-b09-jbossorg-4.jar:2.0.4-b09-jbossorg-4]
    ... 11 more
Caused by: java.lang.InstantiationException:
org.jboss.seam.jsf.SeamApplicationFactory
    at java.lang.Class.newInstance0(Class.java:340) [:1.6.0_25]
    at java.lang.Class.newInstance(Class.java:308) [:1.6.0_25]
    at
    javax.faces.FactoryFinder.getImplGivenPreviousImpl(FactoryFinder.java:604) [jsf-api-1.2_13.jar:1.2_13-b01-FCS]
    ... 16 more

```

O que significa:

com.sun.faces.config.ConfigurationException e **java.lang.InstantiationException** indicam um problema de dependência. Neste caso, a causa não é óbvia.

Como resolver isto:

Procure pelo módulo que contém as classes **com.sun.faces**. Ainda que não exista um módulo **com.sun.faces**, existem dois módulos **com.sun.jsf-impl**. Uma verificação rápida do **jsf-impl-1.2_13.jar** no diretório 1.2 mostra que ele contém as classes **com.sun.faces**. Assim como fez com **javax.faces.FacesException** **ClassNotFoundException**, você deve usar a versão 1.2 JSF e não, a versão 2.0 JSF no principal, portanto você deve especificar uma e excluir a outra. A modificação do **jboss-deployment-structure.xml** para a adição da dependência do módulo é necessária na seção de implantação do arquivo. Você também precisa adicioná-la à subimplantação WAR e excluir o módulo 2.0 JSF. O arquivo deve parecer-se com o seguinte agora:

```

<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="com.sun.jsf-impl" slot="1.2" export="true"/>
      <module name="org.apache.commons.logging" export="true"/>
      <module name="org.dom4j" export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
      <module name="com.sun.jsf-impl" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="javax.faces.api" slot="1.2"/>

```

```

        <module name="com.sun.jsf-impl" slot="1.2"/>
    </dependencies>
</sub-deployment>
</jboss-deployment-structure>

```

Reimplante o aplicativo deletando o arquivo

EAP6_HOME/standalone/deployments/jboss-seam-booking.ear.failed e criando um arquivo em branco **jboss-seam-booking.ear.dodeploy** no mesmo diretório.

6. Problema - java.lang.ClassNotFoundException: org.apache.commons.collections.ArrayStack

Quando o aplicativo é implantado, o log contém o seguinte erro:

```

ERROR [org.apache.catalina.core.ContainerBase.[jboss.web].[default-
host].[/seam-booking]] (MSC service thread 1-1) Exception sending
context initialized event to listener instance of class
com.sun.faces.config.ConfigureListener: java.lang.RuntimeException:
com.sun.faces.config.ConfigurationException: CONFIGURATION FAILED!
org.apache.commons.collections.ArrayStack from [Module
"deployment.jboss-seam-booking.ear:main" from Service Module Loader]
(... additional logs removed ...)
Caused by: java.lang.ClassNotFoundException:
org.apache.commons.collections.ArrayStack from [Module
"deployment.jboss-seam-booking.ear:main" from Service Module Loader]

```

O que significa:

ClassNotFoundException indica que uma dependência ausente. Neste caso, a classe **org.apache.commons.collections.ArrayStack** não pode ser encontrada.

Como resolver isto:

Localize o nome do módulo no diretório **EAP6_HOME/modules/system/layers/base/** procurando pelo caminho **org/apache/commons/collections**. O nome do módulo é "org.apache.commons.collections". Modifique **jboss-deployment-structure.xml** para adicionar a dependência do módulo à seção de implantação do arquivo.

```

<module name="org.apache.commons.collections" export="true"/>

```

O arquivo **jboss-deployment-structure.xml** deve parecer-se com o seguinte agora:

```

<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="com.sun.jsf-impl" slot="1.2"
export="true"/>
      <module name="org.apache.commons.logging" export="true"/>
      <module name="org.dom4j" export="true"/>
      <module name="org.apache.commons.collections"
export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>

```

```

        <module name="javax.faces.api" slot="main"/>
        <module name="com.sun.jsf-impl" slot="main"/>
    </exclusions>
    <dependencies>
        <module name="javax.faces.api" slot="1.2"/>
        <module name="com.sun.jsf-impl" slot="1.2"/>
    </dependencies>
</sub-deployment>
</jboss-deployment-structure>

```

Reimplante o aplicativo deletando o arquivo

EAP6_HOME/standalone/deployments/jboss-seam-booking.ear.failed e criando um arquivo em branco **jboss-seam-booking.ear.dodeploy** no mesmo diretório.

7. Problema - Serviços com dependências indisponíveis/faltantes

Quando o aplicativo é implantado, o log contém o seguinte erro:

```

ERROR [org.jboss.as.deployment] (DeploymentScanner-threads - 2)
{"Composite operation failed and was rolled back. Steps that
failed:" => {"Operation step-2" => {"Services with
missing/unavailable dependencies" =>
["jboss.deployment.subunit.\"jboss-seam-booking.ear\".\"jboss-seam-
booking.jar\".component.AuthenticatorAction.START missing [
jboss.naming.context.java.comp.jboss-seam-booking.\"jboss-seam-
booking.jar\".AuthenticatorAction.\"env/org.jboss.seam.example.booki
ng.AuthenticatorAction/em\" ]\", \"jboss.deployment.subunit.\"jboss-
seam-booking.ear\".\"jboss-seam-
booking.jar\".component.HotelSearchingAction.START missing [
jboss.naming.context.java.comp.jboss-seam-booking.\"jboss-seam-
booking.jar\".HotelSearchingAction.\"env/org.jboss.seam.example.book
ing.HotelSearchingAction/em\" ]\", \"
(... additional logs removed ...)
\"jboss.deployment.subunit.\"jboss-seam-booking.ear\".\"jboss-seam-
booking.jar\".component.BookingListAction.START missing [
jboss.naming.context.java.comp.jboss-seam-booking.\"jboss-seam-
booking.jar\".BookingListAction.\"env/org.jboss.seam.example.booking
.BookingListAction/em\" ]\", \"jboss.persistenceunit.\"jboss-seam-
booking.ear/jboss-seam-booking.jar#bookingDatabase\" missing [
jboss.naming.context.java.bookingDatasource ]"]}}}

```

O que significa:

Quando aparecer um erro “Serviços com dependências ausentes/indisponíveis”, observe o texto entre parênteses após "missing" (ausente). Segue abaixo um exemplo:

```

missing [ jboss.naming.context.java.comp.jboss-seam-booking.\"jboss-
seam-
booking.jar\".AuthenticatorAction.\"env/org.jboss.seam.example.booki
ng.AuthenticatorAction/em\" ]

```

O “/em” indica um problema de fonte de dados e Gerenciador de Entidade.

Como resolver isto:

No JBoss EAP 6, a configuração da fonte de dados foi alterada e precisa ser definida no arquivo

EAP6_HOME/standalone/configuration/standalone.xml. Já que o JBoss EAP 6 é enviado junto com um exemplo de fonte de dados que já está definido no arquivo **standalone.xml**, modifique o arquivo **persistence.xml** para usar o exemplo da fonte de dados neste aplicativo. Quando pesquisando o arquivo **standalone.xml**, você verá que **jndi-name** para o exemplo da fonte de dados é **java:jboss/datasources/ExampleDS**. Modifique o arquivo **jboss-seam-booking.jar/META-INF/persistence.xml** para comentar o elemento **jta-data-source** existente e substituí-lo pelo o seguinte:

```
<!-- <jta-data-source>java:/bookingDataSource</jta-data-source> -->
<jta-data-source>java:jboss/datasources/ExampleDS</jta-data-source>
```

Reimplante o aplicativo deletando o arquivo

EAP6_HOME/standalone/deployments/jboss-seam-booking.ear.failed e criando um arquivo em branco **jboss-seam-booking.ear.dodeploy** no mesmo diretório.

8. A esta altura, o aplicativo é implantado sem erros, porém quando acessar o URL <http://localhost:8080/seam-booking/> em um navegador e tentar "Entrar na Conta", um erro de tempo de execução "A página não está sendo redirecionada de forma apropriada" será exibido. No próximo passo, você aprenderá como depurar e resolver os erros de tempo de execução.

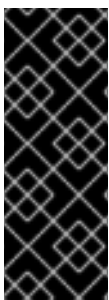
Consulte [Seção 4.3.7, "Depuração e Resolução de Exceções e Erros de Tempo de Execução do Arquivo do Seam 2.2 Booking"](#) para mais informações sobre como depurar e resolver problemas de tempo de execução.

Clique aqui [Seção 4.3.4, "Migração do Arquivo do Seam 2.2 Booking para JBoss EAP 6: Instruções Passo a Passo"](#) para retornar ao tópico anterior.

[Reportar um erro](#)

4.3.7. Depuração e Resolução de Exceções e Erros de Tempo de Execução do Arquivo do Seam 2.2 Booking

Na etapa anterior, [Seção 4.3.6, "Depuração e Resolução de Exceções e Erros de Implantação do Arquivo do Seam 2.2 Booking"](#), foi demonstrado como depurar os erros de implantação. Nessa etapa, será demonstrado como depurar e resolver os erros de tempo de execução.



IMPORTANTE

Os aplicativos que usam Hibernate diretamente com o Seam 2.2 podem usar uma versão do Hibernate 3 empacotada dentro do aplicativo. O Hibernate 4, que é fornecido através do módulo org.hibernate do JBoss EAP 6, não é suportado pelo Seam 2.2. Este exemplo tem a intenção de ajudá-lo a executar o seu aplicativo no JBoss EAP 6 como um primeiro passo. Observe que o empacotamento do Hibernate 3 com um aplicativo Seam 2.2 não possui uma configuração suportada.

Procedimento 4.11. Depuração e resolução de exceções e erros de tempo de execução

A esta altura, quando o aplicativo é implantado, você não vê erro algum no log. No entanto, quando o URL do aplicativo é acessado, os erros aparecem no log.

1. Problema - javax.naming.NameNotFoundException: Nome 'jboss-seam-booking' não encontrado em contexto"

Quando um URL <http://localhost:8080/seam-booking/> é acessado em um navegador, a mensagem "A página não está sendo redirecionada de forma apropriada" aparece e o log contém o seguinte erro:

```
SEVERE [org.jboss.seam.jsf.SeamPhaseListener] (http--127.0.0.1-8080-1) swallowing exception: java.lang.IllegalStateException: Could not start transaction
    at
    org.jboss.seam.jsf.SeamPhaseListener.begin(SeamPhaseListener.java:598) [jboss-seam.jar:]
    (... log messages removed ...)
Caused by: org.jboss.seam.InstantiationException: Could not instantiate Seam component:
    org.jboss.seam.transaction.synchronizations
    at org.jboss.seam.Component.newInstance(Component.java:2170) [jboss-seam.jar:]
    (... log messages removed ...)
Caused by: javax.naming.NameNotFoundException: Name 'jboss-seam-booking' not found in context ''
    at
    org.jboss.as.naming.util.NamingUtils.nameNotFoundException(NamingUtils.java:109)
    (... log messages removed ...)
```

O que significa:

Um **NameNotFoundException** indica um problema de nomeação JNDI. As regras de nomeação JNDI foram alteradas no JBoss EAP 6, portanto é necessário modificar os nomes de busca para que as novas regras sejam cumpridas.

Como resolver isto:

Para depurar isto, veja qual vinculação JNDI foi usada no rastreamento de log do servidor. Ao observar o log do servidor, você encontra o seguinte:

```
15:01:16,138 INFO
[org.jboss.as.ejb3.deployment.processors.EjbJndiBindingsDeploymentUnitProcessor] (MSC service thread 1-1) JNDI bindings for session bean named RegisterAction in deployment unit subdeployment "jboss-seam-booking.jar" of deployment "jboss-seam-booking.ear" are as follows:
    java:global/jboss-seam-booking/jboss-seam-booking.jar/RegisterAction!org.jboss.seam.example.booking.Register
    java:app/jboss-seam-booking.jar/RegisterAction!org.jboss.seam.example.booking.Register
    java:module/RegisterAction!org.jboss.seam.example.booking.Register
    java:global/jboss-seam-booking/jboss-seam-booking.jar/RegisterAction
    java:app/jboss-seam-booking.jar/RegisterAction
    java:module/RegisterAction
    [JNDI bindings continue ...]
```

Existe um total de oito vinculações INFO JNDI listadas no log, uma para cada bean de sessão: RegisterAction, BookingListAction, HotelBookingAction, AuthenticatorAction, ChangePasswordAction, HotelSearchingAction, EjbSynchronizations e TimerServiceDispatcher.

Será necessário modificar o arquivo **lib/components.xml** WAR para usar as novas vinculações JNDI. No log, observe que todas as vinculações EJB JNDI começam com "java:app/jboss-seam-booking.jar". Substitua o elemento **core:init** como a seguir:

```
<!--      <core:init jndi-pattern="jboss-seam-booking/#{
ejbName}/local" debug="true" distributable="false"/> -->
<core:init jndi-pattern="java:app/jboss-seam-booking.jar/#{ejbName}"
debug="true" distributable="false"/>
```

Em seguida, é preciso adicionar as vinculações JNDI EjbSynchronizations e TimerServiceDispatcher. Adicione os seguintes elementos do componente ao arquivo:

```
<component class="org.jboss.seam.transaction.EjbSynchronizations"
jndi-name="java:app/jboss-seam/EjbSynchronizations"/>
<component class="org.jboss.seam.async.TimerServiceDispatcher" jndi-
name="java:app/jboss-seam/TimerServiceDispatcher"/>
```

O arquivo components.xml deve parecer-se com o seguinte:

```
<?xml version="1.0" encoding="UTF-8"?>
<components xmlns="http://jboss.com/products/seam/components"
xmlns:core="http://jboss.com/products/seam/core"
xmlns:security="http://jboss.com/products/seam/security"
xmlns:transaction="http://jboss.com/products/seam/transaction"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation=
"http://jboss.com/products/seam/core
http://jboss.com/products/seam/core-2.2.xsd
http://jboss.com/products/seam/transaction
http://jboss.com/products/seam/transaction-2.2.xsd
http://jboss.com/products/seam/security
http://jboss.com/products/seam/security-2.2.xsd
http://jboss.com/products/seam/components
http://jboss.com/products/seam/components-2.2.xsd">

  <!-- <core:init jndi-pattern="jboss-seam-booking/#{
ejbName}/local" debug="true" distributable="false"/> -->
  <core:init jndi-pattern="java:app/jboss-seam-booking.jar/#{
ejbName}" debug="true" distributable="false"/>
  <core:manager conversation-timeout="120000"
concurrent-request-timeout="500"
conversation-id-parameter="cid"/>
  <transaction:ejb-transaction/>
  <security:identity authenticate-method="#
{authenticator.authenticate}"/>
  <component
class="org.jboss.seam.transaction.EjbSynchronizations"
jndi-name="java:app/jboss-seam/EjbSynchronizations"/>
  <component class="org.jboss.seam.async.TimerServiceDispatcher"
jndi-name="java:app/jboss-
seam/TimerServiceDispatcher"/>
</components>
```


Reimplante o aplicativo excluindo o arquivo **standalone/deployments/jboss-seam-booking.ear.failed** e criando um arquivo em branco **jboss-seam-booking.ear.dodeploy** no mesmo diretório.

2. Problema - O aplicativo é implantado e executado sem erros. Quando o URL <http://localhost:8080/seam-booking/> é acessado em um navegador e você tenta fazer o login, ocorre uma falha e a seguinte mensagem aparece "Falha ao fazer login. Falha na transação". Você deve encontrar um rastreo de exceções no log do servidor:

```
13:36:04,631 WARN [org.jboss.modules] (http-/127.0.0.1:8080-1)
Failed to define class
org.jboss.seam.persistence.HibernateSessionProxy in Module
"deployment.jboss-seam-booking.ear.jboss-seam.jar:main" from Service
Module Loader: java.lang.LinkageError: Failed to link
org/jboss/seam/persistence/HibernateSessionProxy (Module
"deployment.jboss-seam-booking.ear.jboss-seam.jar:main" from Service
Module Loader)
....
Caused by: java.lang.LinkageError: Failed to link
org/jboss/seam/persistence/HibernateSessionProxy (Module
"deployment.jboss-seam-booking.ear.jboss-seam.jar:main" from Service
Module Loader)
...
Caused by: java.lang.NoClassDefFoundError:
org/hibernate/engine/SessionImplementor
  at java.lang.ClassLoader.defineClass1(Native Method)
[rt.jar:1.7.0_45]
...
Caused by: java.lang.ClassNotFoundException:
org.hibernate.engine.SessionImplementor from [Module
"deployment.jboss-seam-booking.ear.jboss-seam.jar:main" from Service
Module Loader]
...
```

O que significa:

ClassNotFoundException indica a falta de uma biblioteca Hibernate. Neste caso, trata-se de **hibernate-core.jar**.

Como resolver isto:

Copie o **hibernate-core.jar** JAR do diretório **EAP5_HOME/seam/lib/** para o diretório **jboss-seam-booking.ear/lib**.

Reimplante o aplicativo excluindo o arquivo **standalone/deployments/jboss-seam-booking.ear.failed** e criando um arquivo em branco **jboss-seam-booking.ear.dodeploy** no mesmo diretório.

3. Problema - O aplicativo é implantado e executado sem erros. Quando o URL <http://localhost:8080/seam-booking/> é acessado em um navegador, você consegue efetuar o login com sucesso. No entanto, ao tentar reservar um hotel, você encontra um rastreo de exceções.

Para depurar isto, você deve remover primeiro **jboss-seam-booking.ear/jboss-seam-booking.war/WEB-INF/lib/jboss-seam-debug.jar**, já que mascara o erro verdadeiro. A esta altura, o seguinte erro pode ser observado:

-


```
java.lang.NoClassDefFoundError:
org/hibernate/annotations/common/reflection/ReflectionManager
```

O que significa:

NoClassDefFoundError indica a falta de uma biblioteca Hibernate.

Como resolver isto:

Copie os JARs **hibernate-annotations.jar** e **hibernate-commons-annotations.jar** do diretório **EAP5_HOME/seam/lib/** para o diretório **jboss-seam-booking.ear/lib**.

Reimplante o aplicativo excluindo o arquivo **standalone/deployments/jboss-seam-booking.ear.failed** e criando um arquivo em branco **jboss-seam-booking.ear.dodeploy** no mesmo diretório.

- Os erros do aplicativo e de tempo de execução devem ser resolvidos.

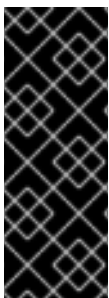
A esta altura, o aplicativo é implantado e executado sem erros.

Clique aqui [Seção 4.3.4, “Migração do Arquivo do Seam 2.2 Booking para JBoss EAP 6: Instruções Passo a Passo”](#) para retornar ao tópico anterior.

[Reportar um erro](#)

4.3.8. Revisão do Sumário das Alterações Feitas Quando Migrando o Aplicativo Seam 2.2 Booking

Embora seja mais eficiente determinar previamente as dependências e adicionar as dependências implícitas em uma única etapa, essa prática mostra como os problemas aparecem no log e fornece informações sobre como depurá-los e resolvê-los. Segue abaixo um sumário das alterações feitas ao aplicativo quando migrando-o para o JBoss EAP 6.



IMPORTANTE

Os aplicativos que usam Hibernate diretamente com Seam 2.2 podem usar uma versão do Hibernate 3 empacotada dentro do aplicativo. O Hibernate 4, que é fornecido através do módulo org.hibernate do JBoss EAP 6, não é suportado pelo Seam 2.2. Este exemplo tem a intenção de ajudá-lo a executar o seu aplicativo no JBoss EAP 6 como um primeiro passo. Observe que o empacotamento do Hibernate 3 com um aplicativo Seam 2.2 não possui uma configuração suportada.

- Você criou um arquivo **jboss-deployment-structure.xml** no diretório **META-INF/** EAR. **<dependencies>** e **<exclusions>** foram adicionadas para resolver **ClassNotFoundExceptions**. Este arquivo contém os seguintes dados:

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="com.sun.jsf-impl" slot="1.2" export="true"/>
      <module name="org.apache.commons.logging" export="true"/>
      <module name="org.dom4j" export="true"/>
```

```

        <module name="org.apache.commons.collections"
export="true"/>
    </dependencies>
</deployment>
<sub-deployment name="jboss-seam-booking.war">
    <exclusions>
        <module name="javax.faces.api" slot="main"/>
        <module name="com.sun.jsf-impl" slot="main"/>
    </exclusions>
    <dependencies>
        <module name="javax.faces.api" slot="1.2"/>
        <module name="com.sun.jsf-impl" slot="1.2"/>
    </dependencies>
</sub-deployment>
</jboss-deployment-structure>

```

- Os seguintes JARs foram copiados do diretório **EAP5_HOME/jboss-eap-5.X/seam/lib/** (substitua 5.X pela versão do EAP 5 da qual você está migrando) para o diretório **jboss-seam-booking.ear/lib/** para resolver **ClassNotFoundExceptions**:

- o hibernate-core.jar
- o hibernate-validator.jar

- O arquivo **jboss-seam-booking.jar/META-INF/persistence.xml** foi modificado como a seguir.

- O elemento **jta-data-source** foi alterado para usar o banco de dados do exemplo que é enviado junto com o JBoss EAP 6:

```

<!-- <jta-data-source>java:/bookingDatasource</jta-data-source> -
-->
<jta-data-source>java:jboss/datasources/ExampleDS</jta-data-
source>

```

- A propriedade **hibernate.cache.provider_class** foi convertida em comentário:

```

<!-- <property name="hibernate.cache.provider_class"
value="org.hibernate.cache.HashtableCacheProvider"/> -->

```

- O arquivo **lib/components.xml** WAR foi modificado para usar as novas vinculações JNDI

- O elemento **core:init** foi substituído como a seguir:

```

<!-- <core:init jndi-pattern="jboss-seam-booking/#
{ejbName}/local" debug="true" distributable="false"/> -->
<core:init jndi-pattern="java:app/jboss-seam-booking.jar/#
{ejbName}" debug="true" distributable="false"/>

```

- Os elementos do componente para as vinculações JNDI "EjbSynchronizations" e "TimerServiceDispatcher" foram adicionados:

```

<component class="org.jboss.seam.transaction.EjbSynchronizations"
jndi-name="java:app/jboss-seam/EjbSynchronizations"/>
<component class="org.jboss.seam.async.TimerServiceDispatcher"

```

```
jndi-name="java:app/jboss-seam/TimerServiceDispatcher"/>
```

[Reportar um erro](#)

APÊNDICE A. HISTÓRICO DE REVISÃO

Revisão 6.4.0-11.1 Translated EAP 6.4.0 Migration Guide into pt-BR	Thu Nov 05 2015	maria andrada
Revisão 6.4.0-11 Red Hat JBoss Enterprise Application Platform 6.4.0.GA	Tuesday April 14 2015	Lucas Costi