# PLATAFORMA DE DESENVOLVIMENTO PINHÃO PARANÁ

## PROJETO SCRIBAPINHÃO – CELEPAR

## DEFINIÇÃO DE MOTOR DE REPOSITÓRIO DE CONTEÚDO PARA SISTEMAS CORPORATIVOS

**Abril/2006**

## Sumário de Informações do Documento

**Tipo do Documento:** Definição

**Título do Documento:** Definição de Motor de Repositório de Conteúdo para Sistemas corporativos

**Estado do Documento:** EB (Elaboração)

**Responsáveis**: Helio Silvio Piccinatto – Revisado Por: Cíntia Evangelista

**Palavras-Chaves: repositório; conteúdo; motor**

**Resumo:** Este documento descreve a prospecção praticada para levantamento das opções de motor para repositório de conteúdo a usar no ScribaPinhão, e as definições necessárias para entendimento e uso do mesmo.

**Número de páginas**: 61

**Software utilizados:** BrOffice 2.0

| Versão | Data | Mudanças |
|---|---|---|
| 1.0 | 10/04/06 | Versão inicial por Helio Silvio Piccinatto |
| 1.1 | 27/07/06 | Revisão Geral, inclusão de definições para os valores possíveis para o atributo "onParentVersion", utilizado na definição de itens (tipos de nó e propriedade).<br><br>Revisado por Elisabeth Hoffmann<br><br>Revisado e formatado por Cintia Evangelista |
| 1.2 | 13/03/2009 | Incluido opções de pesquisa ofertados pelo motor de buscas Lucene utilizado pelo Jackrabbit. |

# SUMÁRIO

**1.INTRODUÇÃO....4**

**2 SISTEMAS DE REPOSITÓRIO DE CONTEÚDO DISPONÍVEIS....4**

2.1 Apache Jackrabbit – O Repositório de Conteúdo Open Source para Java....4
2.2 Iniciando com Apache Jackrabbit....5
2.3 Visão Geral da Arquitetura Jackrabbit....5
2.4 Níveis da JSR170....6
*2.4.1 Nível 1 – Facilidade para adoção, cobertura de vários casos de uso....7*
*2.4.2 Nível 2 – Repositórios Manipuláveis....7*
*2.4.3 Opções Avançadas ....7*
2.5 Aplicações sobre Repositórios....7
2.6 Características....8
*2.6.1 Como Jackrabbit Atua....8*
*2.6.2 Persistência de Dados....10*
*2.6.3 Operações principais de Setup, Inicialização e Configuração....12*
*2.6.4 Gerenciador de Pesquisa (QUERY)....13*
*2.6.5 Exemplos de Montagem de Aplicações com Jackrabbit....14*

**3 CONFIGURANDO O JACKRABBIT....17**

3.1 Arquivo de Configuração do Repositório....18
*3.1.1 Definindo um local customizado no sistema de arquivos....22*
*3.1.2 Definindo Segurança de Acesso....23*
*3.1.3 Definindo Parâmetros para Armazenamento em Banco de Dados....24*
3.2 Tipos de Nó....25
*3.2.1 Definindo Tipos de Nó....25*
*3.2.1.1 Definição de Propriedades....26*
*3.2.1.2 Definição de Nó Filho....27*
*3.2.1.3 Características do atributo "onParentVersion"....28*
3.3 Notação XML para Tipos de Nó....31
3.4 Notação CND para Tipos de Nó....32
*3.4.1 Sintaxe Geral da Notação CND....33*
*3.4.2 Sintaxe da Notação CND em Detalhes....35*
*3.4.3 Comparação da Notação CND com a Notação Padrão XML....41*

**4 PRIMEIROS PASSOS – CODIFICANDO COM JACKRABBIT....42**

**5 PESQUISANDO CONTEÚDO NO REPOSITÓRIO....45**

5.1 Estrutura de uma Pesquisa....46
5.2 Facilidades de Pesquisa do Lucene....51
*Termos....51*
*Campos....51*
*Modificadores de Termos....52*
Buscas Curinga....52
Buscas Pertinentes (Fuzzy)....53
Pesquisa por Proximidade....53
Priorizando um Termo....54
Operadores Booleanos....54
Agrupamento....56
Caracteres Especiais....57

**6 REFERÊNCIAS....58**

**7 GLOSSÁRIO....58**

# 1.INTRODUÇÃO

Este documento visa a apresentar uma API para controle de repositório de conteúdo, escolhida como a mais promissora para uso no ScribaPinhão, após levantamento de uma série de sistemas de gerenciamento de repositório de conteúdo existentes, e verificar suas capacidades e funcionamento, para se avaliar a viabilidade de sua utilização no ScribaPinhão.

# 2 SISTEMAS DE REPOSITÓRIO DE CONTEÚDO DISPONÍVEIS

As seguintes soluções Open Source em Java foram encontradas:

- **Alfresco** – (http://www.alfresco.org).
- **Magnolia** – (http://www.magnolia.info).
- **jLibrary** – (http://jlibrary.sourceforge.net).
- **Liferay Portal** – (http://www.liferay.com/web/guest/home).
- **Apache Jackrabbit** – (http://jackrabbit.apache.org/).

## 2.1 Apache Jackrabbit – O Repositório de Conteúdo Open Source para Java

Apache Jackrabbit é um projeto da Apache Software Foundation.

O projeto Apache Jackrabbit é dedicado a construir e manter uma implementação de código aberto para Repositório de Conteúdo na API de tecnologia Java (JCR). A API JCR 1.0 está especificada na Java Specification Request 170 (JSR 170) e o trabalho na sua versão 2.0 já se iniciou na JSR 283.

A escolha do Jackrabbit sobre os outros sistemas gerenciadores de conteúdo disponíveis decorre de:

- Tratar-se de uma API pronta para o uso por outro sistema.
- É fortemente implementada sobre os conceitos definidos na especificação JSR170.
- Permitir liberdade para se construir a interface e a tecnologia da aplicação.

- Possuir opções para escolha do local e forma para armazenamento do conteúdo e persistência dos dados.
- Utilizar Lucene para indexação do conteúdo.
- Se encontrar num estágio maduro quanto à implementação.

## 2.2 Iniciando com Apache Jackrabbit

Para se familiarizar com o Jackrabbit é preciso conhecer melhor a API JCR. Baixar a especificação JSR 170 , explorar toda a documentação da API, verificar a introdução aos níveis de JCR, e também ler mais alguns artigos referentes a JCR para entendimento completo do modelo de repositório de conteúdo que Jackrabbit implementa. Também existem várias aplicações JCR e Jackrabbit onde se pode ter uma ideia melhor do que se pode fazer com um repositório de conteúdo.

## 2.3 Visão Geral da Arquitetura Jackrabbit

A arquitetura de uma aplicação com Jackrabbit pode ser dividida em três camadas:

•Camada de Aplicação de Conteúdo,

•Camada de API,

•Camada de implementação de Repositório de Conteúdo.

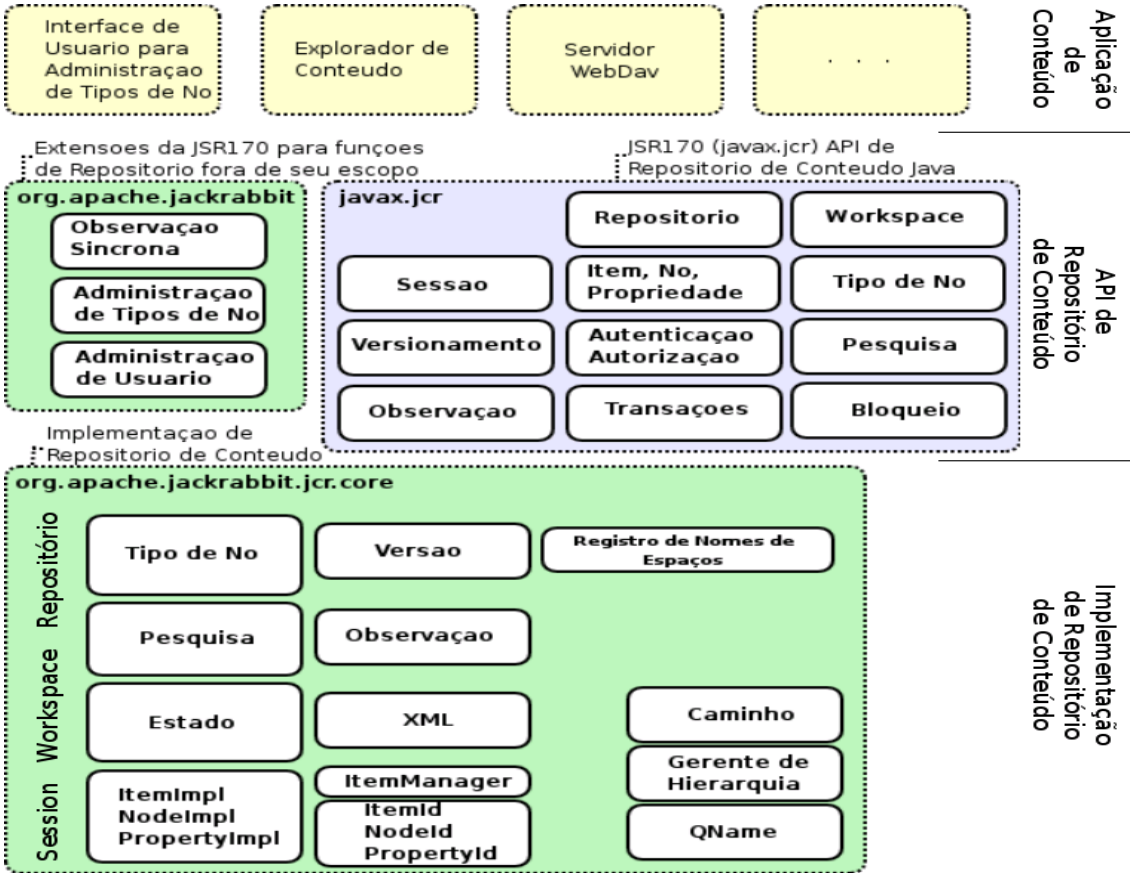


*Ilustração 2.3.1 Visão Geral da Arquitetura Jackrabbit*

A imagem reflete a maioria dos blocos construídos na implementação do jackrabbit, sendo os mais importantes componentes. O tamanho dos blocos simbolizam de uma forma bem grosseira o tamanho do código e sua complexidade de funcionalidade. Os nomes dados aos blocos não estão diretamente associados aos nomes das respectivas classes.

Os três escopos principais de um repositório de conteúdo são:

•Escopo do Repositório.

•Escopo de Workspace.

•Escopo de Sessão.

Cada funcionalidade operando no repositório pode ser atribuída a um destes escopos, sendo que algumas podem atuar em mais de um escopo.

## 2.4 Níveis da JSR170

A API de Repositório de Conteúdo para Tecnologia Java (JSR170) é dividida em diferentes níveis de concordância, para permitir que produtores de repositório adotem gradualmente JSR170 e evitar que se sobrecarreguem desnecessariamente, quando desejarem expor somente partes de seus repositórios através de uma interface compatível com a API.

A API especifica um Nível 1, um Nível 2 e um conjunto de blocos de recursos avançados. Jackrabbit está compatível com todos os níveis e com todos os blocos opcionais.

### 2.4.1 Nível 1 – Facilidade para adoção, cobertura de vários casos de uso

O escopo de Nível 1 da JSR170 cobre um grande número de aplicações simples, que precisam pesquisar e ler repositórios. O Nível 1 especifica uma API de somente leitura, que permite a introspecção aos tipos de Nós e Propriedades, e oferece acesso de leitura hierárquica aos conteúdos do repositório.

Permite que se gerem aplicações tais como portlets de pesquisa e leitura, Modelos de Conteúdo, Relatórios, Exportadores ou outras aplicações que tenham fortemente a necessidade de busca e mostra de informações em um ou mais repositórios.

### 2.4.2 Nível 2 – Repositórios Manipuláveis

O Nível 2 da JSR170 especifica todas as capacidades de escrita, necessárias para interagir bidirecionalmente com um repositório de conteúdo em um detalhado esquema. Aplicações desenvolvidas para o Nível 2 incluem sistemas de gerenciamento ou simplesmente armazenadores de dados, informações ou conteúdos para informações estruturadas ou não.

### 2.4.3 Opções Avançadas

Sobre os Níveis 1 e 2, um número de blocos funcionais serve para funcionalidades mais avançadas de um repositório. Nestas incluem: Versionamento, Transações, pesquisas com SQL, Bloqueio explícito e Observação do Conteúdo.

## 2.5 Aplicações sobre Repositórios

Aplicações de Conteúdo interagem através da API JSR170 com uma implementação de um Repositório de Conteúdo. Existem várias aplicações disponíveis para repositórios JSR170, algumas delas bem genéricas (como um servidor WebDAV) e outras bem específicas que fazem uso do repositório de conteúdo como um armazenador de informações úteis para as aplicações. Aplicações Java podem se beneficiar de um repositório de conteúdo JSR170 como um substituto genérico de arquivos de propriedades, configurações XML, porções de funcionalidades de bancos relacionais objetivando gerenciamento de arquivos. Utilizando um repositório de conteúdo, é possível para uma aplicação combinar um espaço hierarquicamente amplo em uma forma escalonável, automaticamente se beneficiando de serviços do repositório como: versionamento, pesquisa, transações e definições de nomes de espaços, o que torna um repositório de conteúdo um armazenador de dados ideal para várias aplicações.

Uma Aplicação Genérica de Conteúdo (aplicação que não foca nenhuma funcionalidade particular mas permite verificação e manipulação do repositório) utiliza as capacidades de tipos de nós, controle de acesso e outras facilidades para disponibilizar uma interface ao usuário ou um protocolo para usuário final, independentemente do que está sendo armazenado no repositório.

Uma Aplicação Especializada de Conteúdo parte do princípio que existem certos tipos de nó que são específicos, e que fazem parte de um modelo de dados explicados pela definição dos tipos de nó. A maioria destes tipos de nó são definidos pela própria aplicação e fazem parte dela. Estas aplicações usam um repositório de conteúdo como uma camada de persistência em uma evolução natural do uso de um SGBD ou um sistema de arquivos diretamente.

## 2.6 Características

### 2.6.1 Como Jackrabbit Atua

O diagrama abaixo demonstra quais componentes do Jackrabbit são usados quando um usuário da API JCR modifica o conteúdo do repositório. Esta é uma operação simples e muito comum, que abrange um grande número de componentes da implementação Jackrabbit. A forma de representação abaixo não foi definida pela JCR, mas foi feita com base na mesma.

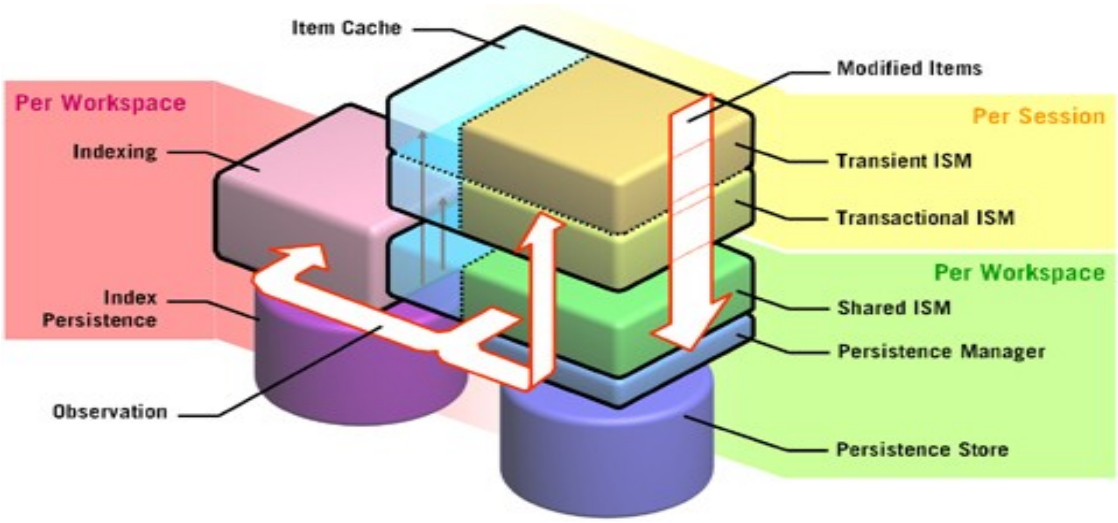


*Ilustração 2.6.1.1 Modelo de Atuação Jackrabbit*

Os componentes usados e suas respectivas funcionalidades aparecem por ordem do caso de uso de escrita ou modificação de conteúdo no repositório:

- **Transient Item State Manager** Uma vez que itens de conteúdo são lidos por uma sessão, eles são colocados em cache no Gerenciador de Estado de Item Transitório. Quando esses itens são modificados, a modificação é somente visível para a mesma sessão, no então chamado espaço "transitório".
- **Transactional Item State Manager** Quando a Aplicação persiste os itens modificados usando o JCR `Item.save()` ou `Session.save()`, os Itens transitórios são promovidos para o Transactional ISM. As modificações são mantidas visíveis somente no escopo desta transação, significando que outras sessões não verão as modificações até que elas sejam comitadas. O commit (fechamento) pode ter sido implícito no caso do Repositório de Conteúdo não estar sendo executado em um ambiente XA.

- **Shared Item State Manager** Uma vez que a transação é comitada, o Gerenciador de Estado de Item Compartilhado, recebe o log de mudança e publica as mudanças para todas as sessões logadas no mesmo workspace. Isto significa que todos os Estados de Itens que estão em cache e referenciados por outras sessões, são notificados e possivelmente atualizados ou invalidados. O Gerenciador de Estado de Item Compartilhado também dispara a observação e manipula o log de mudança sobre o gerenciador de persistência que está configurado para o workspace.
- **Persistence Manager** O Gerenciador de Persistência persiste todos os Estados dos Itens no log de mudanças fornecido pelo Shared ISM. Ele é uma interface simples, rápida e transacional que está em baixo nível e não requer entendimento da complexidade das operações do repositório, mas basicamente necessita ser capaz de persistir e recuperar um dado Item baseado em seu Id.
- **Observation** Quando uma transação é comitada o Gerenciador de Estado de Item Compartilhado dispara o mecanismo de Observação. Isto permite às aplicações enviar mudanças assincronamente no workspace. Jackrabbit também oferece observação sincronizada.
- **Query Manager / Index** Através do evento de observação sincronizada o Gerenciador de Pesquisa é instruído a indexar os itens novos ou modificados. Um índice de repositório de conteúdo é muito mais complexo que um índice em um SGBD clássico, uma vez que deve atender recursos específicos do repositório de conteúdo como hierarquia, herança de tipos de nó ou busca interna de textos (fulltext searches).

### 2.6.2 Persistência de Dados

Jackrabbit utiliza de um componente gerenciador de persistência de dados internos que manipula a armazenagem persistente de conteúdo para nós e propriedades. Essa persistência pode ser feita separadamente para cada workspace existente no repositório. Também é feita a persistência de versões em um gerenciador de persistência em separado.

O gerenciador de persistência se encontra na camada mais baixa da arquitetura Jackrabbit. Recuperação, integridade e performance de um gerenciador de persistência são cruciais para estabilidade e desempenho geral de um repositório. Por exemplo, se os dados em que o gerenciador controla possuem a permissão para serem alterados por outro agente externo, significa que a integridade de todo o repositório está em risco (pensando em toda a

integridade referencial entre nós, propriedades e conteúdos).

Na prática nada mais é do que alguma classe Java que implementa a interface PersistenceManager. Jackrabbit contêm um conjunto de classes de gerenciadores de persistência que cobrem a maioria das necessidades para deploy, via sistema de arquivos e/ou via alguns bancos de dados.

Jackrabbit possui um sistema de arquivamento interno, que trata-se de um componente que implementa operações padrões em sistemas de arquivo, sobre alguma forma de mecanismo de armazenagem (um sistema de arquivos normal, um banco de dados, um servidor webdav ou um formato customizado de arquivo). Esse componente nada mais é do que uma classe Java que implementa a interface FileSystem. Sistemas de arquivo são utilizados tanto como sub componentes de gerenciadores de persistência, como para necessidades gerais de armazenagem (por exemplo armazenar os índices de busca).

A tabela abaixo lista as formas de armazenamento de arquivos com suas vantagens e desvantagens:

| Sistema de Arquivo | Status | Vantagens | Desvantagens |
|---|---|---|---|
| LocalFileSystem | maduro | | Lento em ambiente Windows |
| DbFileSystem | maduro | Baseado em JDBC suportando grande variedade SGDB; automaticamente criado | Performance abaixo da de sistemas de arquivo nativos |
| | | | Configuração não documentada; Formato |

*Tabela 2.6.2.1 Formas de Armazenamento no Jackrabbit*

A tabela abaixo lista os gerenciadores de persistência disponíveis com suas vantagens e desvantagens.

| GP | Status | Vantagens | Desvantagens |
|---|---|---|---|
| SimpleDbPersistenceManager | maduro | GP padrão Jackrabbit | Utiliza esquema simples e não normalizado, com um formato de serialização binária, o que pode aparecer não agradável aos fãs da modelagem relacional. |
| | | Baseado em JDBC suportando variedade de SGBD | |
| | | Esquema automaticamente criado | |
| | | Transacional | |
| BerkeleyDBPersistence Manager | maduro | Persistência baseada btree (BerkeleyDB JE) | Utiliza serialização binária num formato com problemas de licenciamento |
| | | Transacional | |
| | | Automaticamente criado | |
| ObjectPersistenceManager | maduro | Baseado em sistema de arquivos | Utiliza formato de serialização binária |
| | | Configuração facilitada | Se o processo da JVM é abortado o repositório pode se tornar inconsistente |
| | | | Não transacional |
| XMLPersistenceManager | maduro | Baseado em sistema de arquivos | Se o processo da JVM é abortado o repositório pode se tornar inconsistente |
| | | Utiliza formato de serialização XML | Baixa performance |
| | | Configuração facilitada | Não transacional |
| ORM persistence manager | experimental & não finalizado | Baseado em ORM | Complexo para configurar |
| | | Transacional | Dúvidas sobre sua continuidade |

*Tabela 2.6.2.2 Gerenciadores de Persistência do Jackrabbit*

### 2.6.3 Operações principais de Setup, Inicialização e Configuração

O ciclo de vida de um Repositório de Conteúdo Jackrabbit inicia com uma chamada a um dos métodos RepositoryFactory.create(), passando opcionalmente o caminho da fonte do arquivo de configuração (o qual por convenção é chamado de \u201cconfig.xml\u201d) e o RepositoryFactoryHome, o qual aponta para um diretório do qual o Repositório continuará lendo as demais informações para inicialização e em vários casos armazenará os dados atuais que estão persistidos no repositório e em seus workspaces.

Não sendo fornecido o RepositoryFactoryHome, por padrão será utilizado o diretório home do usuário da propriedade de sistema \u201cuser.dir\u201d.

Não sendo fornecido o parâmetro do arquivo de configuração, por padrão será utilizado o valor da propriedade de sistema \u201cRepository.factory.config\u201d e se esta não estiver configurada, será obedecido por padrão o \u201cconfig.xml\u201d no

RepositoryFactoryHome.

A chamada do método create() instanciará o RepositoryFactory que através do método getRepository(String name), irá servir como o factory para as instâncias do Repositório.

Pelo config.xml um repositório é inicializado com o RepositoryStore respectivamente configurado. O RepositoryStore define onde a informação é armazenada no repositório as quais incluem: o UUID do nó root, propriedades do repositório, registro de nomes de espaços, definições de tipos de nós e o controle de versão, armazenados em uma estrutura de arquivos, como a seguir:

./meta:

rep.properties

rootUUID

./namespaces:

ns_reg.properties

./nodetypes:

custom_nodetypes.xml

./versions:

O RepositoryStore normalmente aponta para um arquivo regular local no sistema de arquivos. mas é abstraído através de um FileSystem abstrato que pode estar configurado para apontar

uma outra implementação de FileSystem, e no caso a informação acima poderia ser armazenada em um outro container de dados.

### 2.6.4 Gerenciador de Pesquisa (QUERY)

Jackrabbit implementa as sintaxes de pesquisa pela forma imposta Xpath e pela opcional SQL. A implementação atual do motor de pesquisa é independente da sintaxe usada, apesar que as pesquisas feitas internamente pelo Jackrabbit se utilizam da Xpath devido a estrutura hierarquizada de um JCR.

As implementações principais de pesquisa são:

- **XPath Parser (Verificador XPath)**: É baseado na definição de linguagem da W3C XQuery, a qual não se encontra totalmente finalizada mas pode ser baixada para verificação neste link. A razão do porquê Jackrabbit utiliza a linguagem XQuery, mais freqüentemente que a linguagem XPath, é que a JSR170 especifica uma cláusula "order by" para a sintaxe de pesquisa XPath. Esta cláusula "order by" é emprestada da sintaxe de expressão XQuery FLWOR. Antes de verificar a pesquisa via XPath, a expressão é contornada com um código auxiliar, para formar uma expressão XQuery FLWOR válida e então é passado ao verificador XQuery. O verificador atual é uma classe gerada por javacc, o qual usa a linguagem que pode ser encontrada em src/grammar/xpath. A expressão XPath verificada é então traduzida para uma Árvore de Pesquisa Abstrata, feita pela classe org.apache.jackrabbit.core.query.xpath.XPathQueryBuilder.
- **SQL Parser (Verificador SQL):** O verificador de pesquisa SQL é gerado de uma definição de linguagem localizada em src/grammar/sql. Após verificado, a Árvore de Sintaxe Abstrata é traduzida internamente para a Árvore de Pesquisa Abstrata, pela classe org.apache.jackrabbit.core.query.sql.JCRSQLQueryBulder.
- **Abstract Query Tree (Árvore de Pesquisa Abstrata)**: A Árvore de Pesquisa Abstrata é um formato comum de pesquisa que permite que Jackrabbit implemente um motor de pesquisa o qual é (para certas extensões) independente da sintaxe de pesquisa utilizada (XPath ou SQL). A Árvore de Pesquisa Abstrata consiste das classes derivadas de **org.apache.jackrabbit.core.query.QueryNode**.
- **Query engine (Motor de Pesquisa)**: A implementação atual do motor de pesquisa é configurável. A interface que necessita ser implementada é **org.apach.jackrabbit.core.query.QueryHandler**. Jackrabbit vem com uma implementação que usa um índice Lucene que é

**org.apache.jackrabbit.core.query.lucene.SearchIndex**. Este índice é independente do gerenciador de persistência utilizado. Entretanto é possível fazer uma implementação da **QueryHandler** de acordo com a forma de armazenamento praticado (por ex banco de dados) e executar a pesquisa no armazenamento "nativo". A classe **org.apache.core.query.lucene.LuceneQueryBuilder** traduz a Árvore de Pesquisa Abstrata em uma pesquisa que pode ser executada no índice Lucene. Jackrabbit implementa algumas extensões das classes primárias do Lucene, inicialmente para melhorar a performance em um ambiente com um indexação incremental como o Jackrabbit. Ao invés de índices simples, Jackrabbit utiliza geração de índices que otimiza a leitura e escrita de índices, verificar melhor pela classe **org.apache.jackrabbit.core.query.lucene.MultiIndex**. A geração de índice de busca mais recente é completamente mantida em memória, isso pode ser verificado na classe **org.apache.jackrabbit.core.query.lucene.VolatileIndex**. Ele pode ser comparado ao coletor de lixo do Java, onde gerações são utilizadas para mover objetos ativos entre as novas e as velhas gerações a todo instante. Pesquisas são então executadas em um MultiReader que verifica em todos os índices. Constantemente (dependendo dos parâmetros de configuração no arquivo workspace.xml) os índices são combinados e os nós marcados como excluídos, no índice são removidos. Isto se assemelha a como o Lucene combina seus segmentos internos.

- Utilitários: A classe org.apache.jackrabbit.core.query.QueryParser permite que seja traduzido uma expressão de pesquisa para uma Árvore de Pesquisa Abstrata e vice-versa. Isso é uma ferramenta útil para verificar como uma pesquisa em XPath se compara a uma em SQL ou em uma outra possível. A classe org.apache.jackrabbit.core.query.PropertyTypeRegistry fornece acesso rápido para informações com tipos baseados em nomes de propriedades. A implementação QueryHandler usa esta classe para transformar valores literais em outros tipos de valores.

### 2.6.5 Exemplos de Montagem de Aplicações com Jackrabbit

JSR170 explicita a liberdade para diferentes formas de montagem, o que significa que fica a cargo do implementador do repositório sugerir determinado modelo. Jackrabbit é construído de forma a suportar uma variedade de formas de montagem, podendo ser feita a migração entre elas, algumas das quais são descritas a seguir.

**1 – MODELO 1 – EMPACOTADO COM A APLICAÇÃO (WEB)**

Para muitas aplicações que seja em um contexto limitado, sem necessidade de

interação com outras aplicações ou fontes de dados, pode ser feita a escolha para que a aplicação porte o repositório. Isso é característico de aplicação comercial pronta para uso e que somente depende de suas próprias configurações.

A instância do Jackrabbit que está empacotada junto a aplicação está na mesma JVM (Java Virtual Machine) e não poderá ser acessada por outra aplicação. Desta forma não há necessidade de uma camada de comunicação remota.

As instâncias individuais do repositório são iniciadas e paradas pela própria aplicação, o que significa que a aplicação não somente se utiliza dos dados do repositório, mas também controla seu funcionamento.

Como um exemplo deste modelo, vamos supor uma aplicação web já em um arquivo ".war" e que está em um container web, no qual também está o repositório de conteúdo. Segue figura ilustrativa:

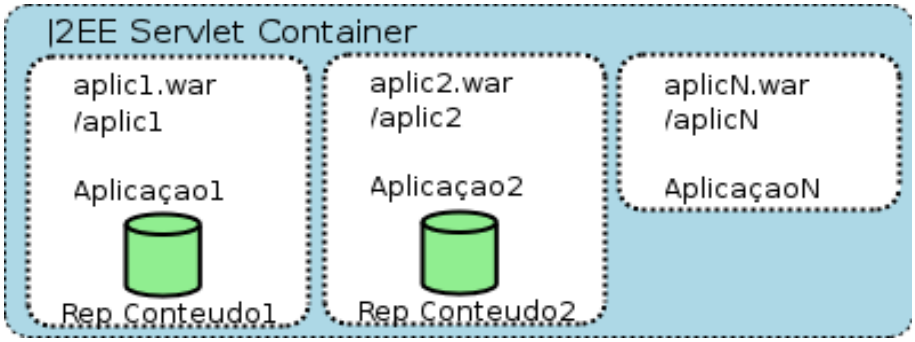


*Ilustração 2.6.5.1 Modelo 1 de Deploy do Jackrabbit*

As aplicações 1 e 2 possuem suas próprias instâncias de Repositório de Conteúdo distribuídos já como parte de seu ".war" e portanto carregado juntamente com a aplicação, tornando invisível para outras aplicações, ainda que elas estejam no mesmo web container.

**2 – MODELO 2 – RECURSO J2EE COMPARTILHADO**

Deste modo o repositório fica visível como um recurso para todas as aplicações web, sendo executadas no mesmo Container, pelo registro do Repositório como um Recurso

Adaptado para o Servidor de Aplicação.

Da mesma forma que no modelo anterior, este também não requer uma camada de comunicação remota se for considerado que os processos estejam no mesmo Servidor de Aplicação e portanto utilizando a mesma JVM.

O início e término do repositório é feito pelo Servidor de Aplicação, mas todas as aplicações podem se conectar ao repositório. A figura a seguir ilustra o modelo 2:

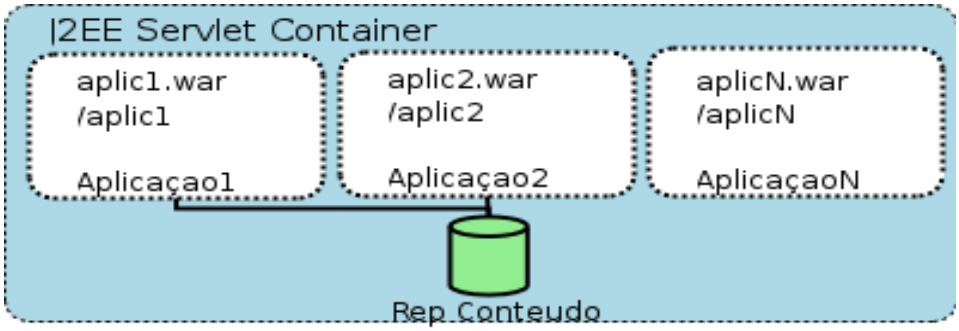


*Ilustração 2.6.5.2 Modelo 2 de Deploy do Jackrabbit*

Esta configuração também permite se beneficiar das facilidades do XA do Servidor de Aplicações e pode usar o mecanismo de login simples que é fornecido como parte do framework J2EE.

**3 – MODELO 3 – SERVIDOR DE REPOSITÓRIO**

Em ambientes empresariais o modelo client/server é amplamente usado por bancos de dados relacionais. Este modelo é praticado onde se deseja uma forte separação (física) entre a camada de conteúdo (dados) e a camada de apresentação, então o repositório de conteúdo pode ser acessado de diferentes aplicações e também individualmente.

A figura a seguir ilustra melhor o modelo 3:

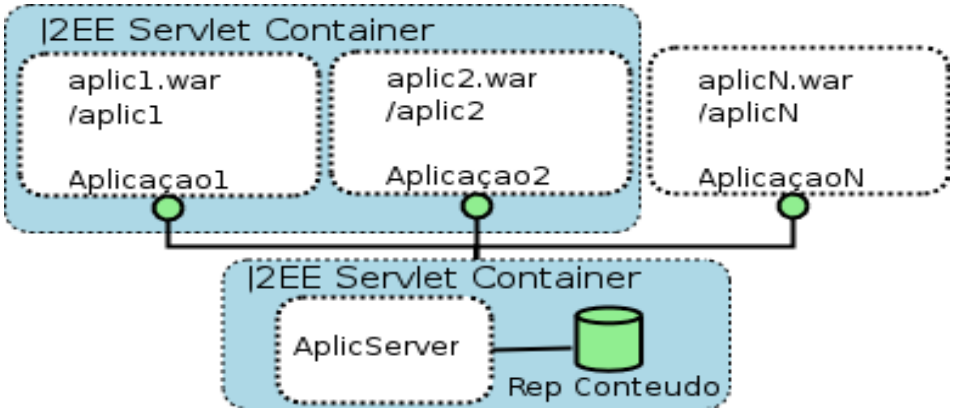


*Ilustração2.6.5.3 Modelo 3 de Deploy do Jackrabbit*

Este modelo assume que existe uma camada de comunicação entre o servidor de repositório de conteúdo e o seu respectivo cliente. Enquanto a comunicação da aplicação com o cliente do repositório é através da API JSR170, a comunicação entre o repositório cliente e o servidor é através de alguma forma de protocolo transparente residindo no servidor do repositório.

O protocolo de transporte entre o cliente e o servidor é alvo de implementação e não é imposto pela especificação JSR170. Alternativas comuns seriam RMI sobre JRMP ou IIOP, ou WebDAV e suas extensões.

# 3 CONFIGURANDO O JACKRABBIT

O Jackrabbit precisa de duas partes de informação para configurar uma instância de repositório de conteúdo em tempo de execução:

- **Diretório Home do Repositório:** O caminho no sistema de arquivos do diretório contento o repositório de conteúdo acessado pela instância de Jackrabbit. Este diretório normalmente contêm todos os repositórios de conteúdo, índices de busca, configurações internas, e outras informações de persistência gerenciadas pelo repositório de conteúdo. Observar que nem tudo isto é obrigatório e alguns gerenciadores de persistência e outros componentes do Jackrabbit podem ser configurados para acessar arquivos e até outros recursos (como base de dados remota) fora do diretório home do repositório. Um diretório home designado para o repositório entretanto, sempre é necessário mesmo se alguns

componentes escolham não fazerem uso dele. Jackrabbit incluirá no diretório home do repositório todos os arquivos e subdiretórios necessários quando o repositório for instanciado pela primeira vez.

- Arquivo de Configuração do Repositório: O caminho do sistema de arquivo para o arquivo XML de configuração do repositório. Este arquivo especifica os nomes de classes e propriedades de vários componentes do Jackrabbit utilizados para gerenciamento e acesso ao repositório de conteúdo. Jackrabbit valida este arquivo de configuração e instancia os componentes especificados quando a instância do repositório é criada em tempo de execução.

Estes parâmetros de configuração são passados para o Jackrabbit tanto diretamente na criação da instância, como indiretamente através de configurações de um objeto JNDI ou algum outro sistema gerenciador de componente.

## 3.1 Arquivo de Configuração do Repositório

O arquivo de configuração do repositório é um arquivo XML que especifica os componentes e suas configurações para o repositório de conteúdo Jackrabbit. Este arquivo se encontra normalmente em uma aplicação web em "*./WEB-INF/lib/repository/repository.xml*". A seguir um arquivo já configurado contendo uma Definição de Tipo de Documento (DTD) para o formato XML:

Obs: esta definição esta conforme versão 1.1 do jackrabbit sendo que atualmente a mesma já esta acima de 1.5 e outra forma de persistência e classes deve ser adotada, verificar melhores detalhes em http://jackrabbit.apache.org.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Repository [
<!-- o elemento Repositório configura uma instancia do repositório; workspaces individuais do
repositório são configurados em arquivos separados de configuração chamados workspace.xml os
quais estão localizados em um subdiretório do diretório root para workspaces (ver elemento
workspaces).
Ele consiste de
    –   Um elemento FileSystem (o sistema de arquivos virtual) utilizado pelo repositório para
        persistir estados globais como nomes de espaços (namespaces) registrados, tipos de nó
        customizados, etc...
    –   Um elemento de Segurança (Security) que especifica o nome da app-entry (entrada de
        aplicação) na configuração JAAS e no gerenciador de acesso.
    –   Um elemento Workspaces que especifica a localização de seu diretório root, o nome do
        workspace padrão (default) e opcionalmente a configuração do diretório root do
        workspace dentro do sistema de arquivos virtual do repositório
    –   Um elemento Workspace que é utilizado como um modelo para configuração do workspace; é
```

```
        usado para criação do workspace inicial se nenhum ainda existir e para criação de
        workspaces adicionais através da api.
    –   Um elemento SearchIndex que é utilizado para configuração de cada workspace.
    –   Configurações relacionadas a indexação.
    –   Um elemento Versioning que é utilizado para configuração relacionada a versionamento.
-->
<!ELEMENT Repository (FileSystem,Security,Workspaces,Workspace,Versioning,SearchIndex?)>
 <!--um sistema de arquivos virtual-->
 <!ELEMENT FileSystem (param*)>
 <!ATTLIST FileSystem class CDATA #REQUIRED>

 <!--o elemento Security especifica o nome (atributo appName) do app-entry da configuração
JAAS para este repositório. Também especifica o gerenciador de acesso a ser utilizado
(elemento AccessManager).-->
 <!ELEMENT Security (AccessManager, LoginModule?)>
 <!ATTLIST Security appName CDATA #REQUIRED>

 <!--o elemento AccessManager configura o gerenciador de acesso a ser usado pelo instância
deste repositório; o atributo da classe especifica o FQN da classe implementadora da interface
AccessManager.-->
 <!ELEMENT AccessManager (param*)>
 <!ATTLIST AccessManager class CDATA #REQUIRED>

 <!--parâmetros genéricos (name/value pair)-->
 <!ELEMENT param EMPTY>
 <!ATTLIST param name CDATA #REQUIRED value CDATA #REQUIRED>

 <!--o elemento LoginModule especificando opcionalmente um modulo de login JAAS para
autenticação de usuários Este recurso permite o uso do Jackrabbit em um ambiente não JAAS -->
 <!ELEMENT LoginModule (param*)>
 <!ATTLIST LoginModule class CDATA #REQUIRED>

 <!--o elemento Workspaces especifica o diretório root para workspaces (atributo rootPath), o
nome do workspace padrão (atributo defaultWorkspaces), e opcionalmente a configuração do
diretório root do workspace dentro do sistema de arquivos virtual do repositório (atributo
configRootPath).
   workspaces individuais são configurados individualmente através de arquivos workspace.xml
localizados nos subdiretórios de cada workspace
  a) o diretório root físico do workspaces
     ou, se configRootPath foi especificado,
  b) a configuração do diretório root dentro do sistema de arquivos virtual do repositório
 -->
 <!ELEMENT Workspaces EMPTY>
 <!ATTLIST Workspaces
        rootPath CDATA #REQUIRED
        defaultWorkspace CDATA #REQUIRED
        configRootPath CDATA #IMPLIED>

 <!--o elemento Workspaces serve como um modelo de configuração de workspace; ele é usado para
criar o workspace inicial se ainda não existir, e para criação de workspaces adicionais pela
api.
  -->
 <!ELEMENT Workspace (FileSystem,PersistenceManager,SearchIndex?)>
 <!ATTLIST Workspace      name CDATA #REQUIRED>

  <!--o elemento PersistenceManager configura o gerenciador de persistência a ser usado para o
workspace; o atributo de classe especifica o FQN da classe implementadora da interface
PersistenceManager    -->
  <!ELEMENT PersistenceManager (param*)>
  <!ATTLIST PersistenceManager      class CDATA #REQUIRED>

  <!--o elemento SearchIndex especifica a localização do índice de busca (usado pelo
QueryHandler); o atributo de classe especifica o FQN da classe implementadora da interface
QueryHandler.    -->
  <!ELEMENT SearchIndex (param*,FileSystem?)>
  <!ATTLIST SearchIndex      class CDATA #REQUIRED>

  <!--o elemento Versioning configura o gerenciador de persistência a ser usado para persistir
o estado de versão    -->
    <!ELEMENT Versioning (FileSystem, PersistenceManager)>
    <!ATTLIST Versioning      rootPath CDATA #REQUIRED>
```

```
]>
<!-- Exemplo Arquivo de Configuração do Repositório -->
<Repository>
 <!-- sistema de arquivo virtual onde o repositório armazena estados globais (namespaces registrados, tipos de nó customizados, etc) -->
 <FileSystem class="org.apache.jackrabbit.core.fs.db.DbFileSystem">
  <param name="driver" value="org.postgresql.Driver"/>
  <param name="url" value="jdbc:postgresql://localhost/NOME_BASE"/>
  <param name="schema" value="postgresql"/>
  <param name="user" value="postgres"/>
  <param name="password" value="postgres"/>
  <param name="schemaObjectPrefix" value="rep_"/>
 </FileSystem>
 <!-- outra forma de configuração, no caso informando outra classe de sistema de arquivo virtual do repositório
    <FileSystem class="org.apache.jackrabbit.core.fs.local.LocalFileSystem">
        <param name="path" value="${rep.home}/repository"/>
    </FileSystem>
 -->

 <!--configuração de segurança-->
 <Security appName="Jackrabbit">
  <!--gerenciador de acesso:
      class: FQN da classe que implementa a interface AccessManager -->
  <AccessManager class="org.apache.jackrabbit.core.security.SimpleAccessManager">
    <!-- <param name="config" value="${rep.home}/access.xml"/> -->
  </AccessManager>
  <LoginModule class="org.apache.jackrabbit.core.security.SimpleLoginModule">
   <!-- anonymous user name ('anonymous' é o valor padrão) -->
   <param name="anonymousId" value="anonymous"/>
  </LoginModule>
 </Security>

 <!--localização do diretório home para workspaces, configurações e nomes de workspaces padrão-->
 <Workspaces rootPath="${rep.home}/workspaces" defaultWorkspace="default" />
  <!-- configRootPath="/wspConfigs"/> -->
  <!-- modelo de configuração de workspace usado para criar o workspace inicial se não existir -->
  <Workspace name="${wsp.name}">
   <!--sistema de arquivo virtual do workspace:
       class: FQN da classe implementadora da interface FileSystem        -->
   <FileSystem class="org.apache.jackrabbit.core.fs.db.DbFileSystem">
    <param name="driver" value="org.postgresql.Driver"/>
    <param name="url" value="jdbc:postgresql://localhost/NOME_BASE"/>
    <param name="schema" value="postgresql"/>
    <param name="user" value="postgres"/>
    <param name="password" value="postgres"/>
    <param name="schemaObjectPrefix" value="${wsp.name}_"/>
   </FileSystem>
   <!-- outra forma de configuração, informando outra classe
    <FileSystem class="org.apache.jackrabbit.core.fs.local.LocalFileSystem">
     <param name="path" value="${wsp.home}"/>
    </FileSystem> -->
   <!--gerenciador de persistência do workspace:
       class: FQN da classe implementadora da interface PersistenceManager -->
   <PersistenceManager class="org.apache.jackrabbit.core.state.db.SimpleDbPersistenceManager">
    <param name="driver" value="org.postgresql.Driver"/>
    <param name="url" value="jdbc:postgresql://localhost/NOME_BASE"/>
    <param name="schema" value="postgresql"/>
    <param name="user" value="postgres"/>
    <param name="password" value="postgres"/>
    <param name="schemaObjectPrefix" value="${wsp.name}_"/>
    <param name="externalBLOBs" value="false"/>
   </PersistenceManager>
   <!--  outra forma de configuração, informando outra classe
    <PersistenceManager class="org.apache.jackrabbit.core.state.xml.XMLPersistenceManager" />
    -->
   <!--índice de busca e o sistema de arquivos que ele utiliza:
       class: FQN da classe implementadora da interface QueryHandler.
      Se requerido pela implementação QueryHandler, pode ser configurado o FileSystem que o
```

```
manipulador pode usar.
       Parâmetros suportados pelo índice de busca Lucene:
         –   path: localização do índice Este parâmetro é obrigatório!
         –   useCompoundFile: adverte o lucene para usar arquivos compostos para os arquivos de
             índice.
         –   minMergeDocs: número mínimo de nós em uma índice até que os segmentos sejam
             combinados.
         –   volatileIdleTime: tempo disponível em segundos para o índice volátil ser movido
             para um índice persistente mesmo se minMergeDocs não está disponível.
         –   maxMergeDocs: número máximo de nós em segmentos que serão combinados.
         –   mergeFactor: determina a freqüência de combinação dos segmentos de índice
         –   bufferSize: número máximo de documentos que são mantidos pendentes em fila ate
             serem adicionados ao índice
         –   cacheSize: tamanho do cache do número de documentos. Este cache mapeia uuids para
             os números dos documentos Lucene.
         –   forceConsistencyCheck:  executa  uma  validação  de  consistência  em  cada
             inicialização. Se false, uma verificação de consistência somente é feita quando o
             índice de busca detectar um principio de parada forçada (shutdown)
         –   autoRepair: erros detectados pela verificação de consistência são automaticamente
             reparados. Se false, os erros somente são registrados no log.
         –   analyzer: nome da classe do analizador Lucene usado para indexação de textos de
             arquivos binários (fulltext indexing of text).
         –   queryClass: nome da classe que implementa a interface javax.jcr.query.Query. Esta
             classe deve estender a classe: org.apache.jackrabbit.core.query.AbstractQueryImpl.
         –   idleTime: tempo disponível em segundos após o qual um manipulador de pesquisa não
             utilizado  é  liberado.  Se  o  manipulador  de  pesquisa  esta  atrasado  para  ser
             novamente  utilizado,  ele  é  inicializado  automaticamente.  Valor  padrão  (-1)
             desabilita este recurso.
         –   respectDocumentOrder: Se true e a pesquisa não conter a cláusula 'order by', os
             nós retornados ficarão ordenados por documento. Para melhorar a performance quando
             as pesquisas retornam muitos nós deixe configurado para 'false'.
             Observação: todos os parâmetros (exceto o path) nesta configuração de SearchIndex
estão com os valores padrão e portanto poderiam estar sendo omitidos-->
   <SearchIndex class="org.apache.jackrabbit.core.query.lucene.SearchIndex">
    <param name="path" value="${wsp.home}/index"/>
    <param name="useCompoundFile" value="true"/>
    <param name="minMergeDocs" value="100"/>
    <param name="volatileIdleTime" value="3"/>
    <param name="maxMergeDocs" value="100000"/>
    <param name="mergeFactor" value="10"/>
    <param name="bufferSize" value="10"/>
    <param name="cacheSize" value="1000"/>
    <param name="forceConsistencyCheck" value="false"/>
    <param name="autoRepair" value="true"/>
    <param name="analyzer" value="org.apache.lucene.analysis.standard.StandardAnalyzer"/>
    <param name="queryClass" value="org.apache.jackrabbit.core.query.QueryImpl"/>
    <param name="idleTime" value="-1"/>
    <param name="respectDocumentOrder" value="false"/>
    <param name="textFilterClasses"    value =
"org.apache.jackrabbit.core.query.MsExcelTextFilter,
org.apache.jackrabbit.core.query.MsPowerPointTextFilter,
org.apache.jackrabbit.core.query.MsWordTextFilter,
org.apache.jackrabbit.core.query.PdfTextFilter,
org.apache.jackrabbit.core.query.HTMLTextFilter,
org.apache.jackrabbit.core.query.XMLTextFilter,
org.apache.jackrabbit.core.query.RTFTextFilter,
org.apache.jackrabbit.core.query.OpenOfficeTextFilter" />
<!-- Para maiores informações veja nos JavaDocs em:
org.apache.jackrabbit.core.query.TextFilter -->
   </SearchIndex>
  </Workspace>
  <!--Configura o versionamento-->
  <Versioning rootPath="${rep.home}/version">
   <!--Configura o sistema de arquivos a ser usado para versionamento pelo respectivo
gerenciador de persistência-->
   <FileSystem class="org.apache.jackrabbit.core.fs.db.DbFileSystem">
    <param name="driver" value="org.postgresql.Driver"/>
    <param name="url" value="jdbc:postgresql://localhost/teste_ScribaPinhão"/>
    <param name="schema" value="postgresql"/>
    <param name="user" value="postgres"/>
    <param name="password" value="postgres"/>
```

```
    <param name="schemaObjectPrefix" value="version_"/>
   </FileSystem>
   <!-- outra forma de configuração informando outra classe
   <FileSystem class="org.apache.jackrabbit.core.fs.local.LocalFileSystem">
    <param name="path" value="${rep.home}/versions"/>
   </FileSystem>-->
   <!-- Configura o gerenciador de persistência a ser usado para persistir estado de versão
Note que a implementação corrente de versionamento esta baseado em um gerenciador 'normal' de
persistência, mas isto poderá ser alterado em uma implementação futura        -->
   <PersistenceManager class="org.apache.jackrabbit.core.state.db.SimpleDbPersistenceManager">
    <param name="driver" value="org.postgresql.Driver"/>
    <param name="url" value="jdbc:postgresql://localhost/teste_ScribaPinhão"/>
    <param name="schema" value="postgresql"/>
    <param name="user" value="postgres"/>
    <param name="password" value="postgres"/>
    <param name="schemaObjectPrefix" value="version_"/>
    <param name="externalBLOBs" value="false"/>
   </PersistenceManager>
   <!-- outra forma de configuração utilizando outra classe
   <FileSystem class="org.apache.jackrabbit.core.fs.local.LocalFileSystem">
    <param name="path" value="${rep.home}/versions"/>
   </FileSystem> -->
  </Versioning>

  <!--Índice de Busca para conteúdo que é compartilhado em todo o repositório (arvore
/jcr:system, contem as principais versões)
Os mesmos parâmetros utilizados na configuração de índice de busca dentro da definição do
elemento workspace, são suportados. Este elemento é opcional. Se omitido, a arvore /jcr:system
não será indexada e portanto não haverá retornos para pesquisas onde possa haver conteúdo
nesta arvore! -->
 <SearchIndex class="org.apache.jackrabbit.core.query.lucene.SearchIndex">
  <param name="path" value="${rep.home}/repository/index"/>
 </SearchIndex>
</Repository>
```

Os valores dos atributos XML no arquivo de configuração são interpretados como strings planas, exceto em algumas variáveis especiais. As seguintes variáveis no formato '${variável}' são substituídas antes do atributo string ser interpretado.

- **${rep.home}:** Diretório inicial do repositório. Esta variável é substituída pelo caminho do diretório inicial pego quando da instanciação do repositório. Esta variável torna possível o uso de um único arquivo de configuração para diferentes repositórios.
- **${wsp.name}:** Nome do workspace. Esta variável somente é disponibilizada dentro do elemento modelo **Workspace** e é substituída pelo nome do workspace sendo configurado.
- **${wsp.home}:** Diretório inicial do workspace. Esta variável é somente disponibilizada dentro do elemento modelo **Workspace** e é substituída pelo caminho do diretório atribuído pelo workspace sendo configurado.

No geral deve-se ter muito cuidado ao alterar o arquivo de configuração do repositório uma vez que ele já tenha sido inicializado. Normalmente, mas nem sempre, é seguro que sejam alteradas a forma de autenticação e controle de acesso no elemento Security e também alguns parâmetros de outros componentes (SearchIndex por ex). No mais é interessante

averiguar melhor os impactos de algumas alterações de parâmetros, pós inicialização do repositório na documentação das classes. No mínimo em se efetuando uma alteração dessas o arquivo workspace.xml que ficar abaixo da pasta destinada ao workspace devera ser atualizado manualmente.

### 3.1.1 Definindo um local customizado no sistema de arquivos

Existem alguns locais do arquivo de configuração para se definir o local desejado para armazenamento do conteúdo. Por exemplo, o dispositivo escolhido para conter o repositório pode ser o sistema de arquivos no seu caminho padrão. Veja abaixo:

```
<FileSystem class="org.apache.jackrabbit.core.fs.local.LocalFileSystem">
      <param name="path" value="${rep.home}/repository"/>
</FileSystem>
```

Como se percebe, a real localização do repositório no sistema de arquivos depende de onde a aplicação server se encontra, por exemplo, se for em Jboss, ela estará abaixo do diretório bin (${JBOSS_HOME}/bin). Caso exista um motivo para que esse não seja o local desejado, então a variável ${rep_home} deve ser substituída, em todas suas ocorrências, pelo caminho desejado para o armazenamento. Existe a possibilidade de usar diferentes localizações para diferentes propósitos. Verificar o exemplo de como se indicar um local diferente para armazenamento:

```
<FileSystem class="org.apache.jackrabbit.core.fs.local.LocalFileSystem">
      <param name="path" value="/home/ScribaPinhão/repositório"/>
</FileSystem>
```

### 3.1.2 Definindo Segurança de Acesso

O segurança de acesso ao repositório pode ser configurado via módulos de login JAAS. Por padrão a segurança de acesso aos conteúdos do repositório deve ser definido automaticamente pela aplicação. A forma de acesso inicial ao repositório é definida através da classe *SimpleLoginModule*:

```
<LoginModule class="org.apache.jackrabbit.core.security.SimpleLoginModule">
      <param name="anonymousId" value="anonymous" />
```

```
</LoginModule>
```

Essa forma é a que atende a necessidade básica de segurança. Existem outras formas para garantir essa autenticação inicial, várias podem ser as soluções, via diretório de usuários ou base de dados, enfim. Em todo o caso a alteração desta forma de autenticação é possível, e pode ser alterada da forma que convier. Segue um exemplo baseado em propriedades do módulo de login do Jboss:

```
<LoginModule class="org.jboss.security.auth.spi.UsersRolesLoginModule"
      <param name="usersProperties" value="users.properties" />
      <param name="unauthenticatedIdentity" value="nobody" />
</LoginModule>
```

Dependendo de seu Módulo de Login outros parâmetros podem ser necessários e o acima foi somente para ilustrar essa liberdade nesta escolha.

### 3.1.3 Definindo Parâmetros para Armazenamento em Banco de Dados

Várias podem ser as formas de configurar o armazenamento via banco de dados. Existem várias classes que promovem essa funcionalidade. A configuração pode ser para armazenamento de persistência e também de conteúdo do repositório. Segue configuração para definir armazenamento de conteúdo em banco de dados:

```
<FileSystem class="org.apache.jackrabbit.core.fs.db.DbFileSystem">
  <param name="driver" value="org.postgresql.Driver"/>
  <param name="url" value="jdbc:postgresql://localhost/NOME_BASE"/>
  <param name="schema" value="postgresql"/>
  <param name="user" value="postgres"/>
  <param name="password" value="postgres"/>
  <param name="schemaObjectPrefix" value="${wsp.name}_"/>
</FileSystem>
```

Segue configuração para definir armazenamento de persistência em banco de dados:

```
<PersistenceManager class="org.apache.jackrabbit.core.state.db.SimpleDbPersistenceManager">
  <param name="driver" value="org.postgresql.Driver"/>
  <param name="url" value="jdbc:postgresql://localhost/NOME_BASE"/>
  <param name="schema" value="postgresql"/>
  <param name="user" value="postgres"/>
  <param name="password" value="postgres"/>
  <param name="schemaObjectPrefix" value="version_"/>
  <param name="externalBLOBs" value="false"/>
</PersistenceManager>
```

Segue descrição dos parâmetros possíveis:

- **driver:** Nome da classe driver.
- **url:** O caminho da base.
- **schema:** O nome do esquema utilizado para criar a base de dados. Este nome depende da base de dados sendo utilizada.
- **schemaObjectPrefix:** O prefixo para as diferentes tabelas criadas pelo repositório.
- **user**: O usuário utilizado para acessar a base de dados.
- **password**: A senha do usuário.
- **externalBlobs**: Esta tag booleana define se os arquivos de conteúdo binário serão armazenados no banco de dados ou no sistema de arquivos.

## 3.2 Tipos de Nó

Cada nó na árvore de um workspace tem um tipo de nó que define os nós filhos e as propriedades que se pode ou deve ter. Desenvolvedores podem usar tipos de nó, para definir um modelo de conteúdo customizado, para o domínio de suas aplicações e ter o Jackrabbit aplicando as restrições do modelo no nível do repositório.

Existem duas categorias de tipos de nó, **primary** e **mixin**. Cada nó tem um tipo de nó associado na sua criação (maiores detalhes na API JCR em Node.addNode). Adicionalmente, um nó do tipo mixin pode ser adicionado a outro nó em qualquer momento dentro de seu ciclo de vida (maiores detalhes verificar no método Node.addMixin).

O tipo de nó **primary** usualmente define a estrutura do nó (nós filhos que são permitidos ou mandatórios e propriedades) relacionada ao domínio do problema sendo modelado. Por exemplo, um nó usado para armazenar conteúdo referente a contatos profissionais poderia ter o seu tipo primary como **myapp:Contato**, o qual define propriedades tais como **myapp:primeiroNome**, **myapp:ultimoNome** e assim por diante.

O tipo de nó **mixin** normalmente especifica propriedades adicionais ou nós filhos com capacidade de serem adicionados ao nó. Essas capacidades podem incluir funções genéricas a

nível de repositório como no caso de **mixins** embutidos: **mix:versionable** e **mix:lockable**; ou capacidades a nível de domínio com um (hipotético) tipo mixin **myapp:Comunicavel** que adiciona a propriedade **myapp:enderecoEmail** ao nó.

Tipos de nó **primary** são especificados com uma hierarquia de herança. Cada tipo de nó primário deve ser o sub tipo de nó de no mínimo um tipo de nó existente.

O tipo de nó embutido **nt:base** serve como o início (root) desta hierarquia. Jackrabbit suporta herança múltipla de tipos de nó, portanto tipos de nó podem ter mais que um super tipo.

Tipos de nó **mixin** normalmente não deveriam ter super tipos.

### 3.2.1 Definindo Tipos de Nó

Uma definição de tipo de nó tem os seguintes atributos:

- **Nome:** Cada tipo de nó registrado no repositório deve ter um nome único. A convenção de nomes para tipos de nó são as mesmas para itens (devem ter um prefixo delimitado por **':'**).
- **Supertipos:** Um tipo de nó primary (com a exceção de **nt:base**) deve estender outro tipo de nó.
- **Estado Mixin:** Um tipo de nó pode ser **primary** ou **mixin**.
- Estado Nó Child (Filho) Ordenável: Um tipo de nó pode especificar que os nós filho (child) são ordenáveis. Se este estado é configurado para true, então Node.orderBefore pode ser usado para setar a ordem dos nós filhos. Somente tipos de nó primary controlam um estado de nó nesta consideração. Esta configuração em um tipo de nó mixin não terá algum efeito sobre o nó.
- **Definições de Propriedade:** Um tipo de nó contém um conjunto de definições, especificando as propriedades dos nós deste tipo de nó, que são permitidas ou requeridas e as suas características.
- **Definições de Nó Filho:** Um tipo de nó contém definições especificando seus nós filhos e também as características que estes tipos de nó podem ter.
- **Nome de Item Primary:** Um tipo de nó pode especificar um item filho (nó ou propriedade) pelo nome como o item primário (primary). Esta indicação é utilizada pelo

método **Node.getPrimaryItem()**.

#### 3.2.1.1 Definição de Propriedades

Uma definição de propriedade (dentro de uma definição de tipo de nó) contém as seguintes informações:

- **Name (Nome):** O nome de uma propriedade na qual esta definição se aplica, ou **'*'** se esta definição é uma 'definição residual', significando que ela se aplica a qualquer propriedade adicional com qualquer nome, independente das demais já definidas de outra forma para este tipo de nó.
- **Required Type (Tipo requerido):** Um entre **String, Binary, Long, Double, Date, Path, Name, Reference ou Undefined**. Especificando um tipo **Undefined** significa que a propriedade pode ser de qualquer tipo.
- **Value Constraints (Restrições de Valores):** Define o intervalo de valores que podem ser atribuídos à propriedade.
- **Default Value (Valor Padrão):** O valor que a propriedade terá se ela for auto criada.
- **Auto-create Status (Estado de auto-criação)**: Define que a propriedade será criada automaticamente no momento da criação de seu nó pai. Somente propriedades com valor padrão podem ser auto-create (criadas automaticamente).
- **Mandatory Status (Estado Mandatório):** Define que a propriedade deve obrigatoriamente existir para o tipo de nó em que ela se encontra. Se o nó pai da propriedade contendo o estado mandatório for salvo sem que a propriedade com este estado tenha sido criada, será lançado um erro. Uma vez que o valor de uma propriedade ou existe ou não existe (não há o conceito de valor null para uma propriedade), isto implica que uma propriedade de um simples valor, deve ter um valor para existir. Uma propriedade mandatória de multi-valores (vetorizada), por outro lado pode ter zero ou mais valores.
- **On-Parent-Version Status (Estado Caso Versione o Pai):** Especifica o que acontece com a propriedade se uma nova versão de seu nó pai for criada.
- **Protected Status (Estado Protegido):** Uma propriedade protegida (protected) é uma que não pode ser modificada (ter nós filho ou propriedades adicionadas ou removidas) ou removida da estrutura de seu nó pai pela API JCR.
- Multiple Values Status (Estado de Multi Valor): Define se a propriedade pode ter múltiplos valores, significando que ela armazena um array de valores, não somente um.

Observe que este flag “multiple values” é especial em um dado tipo de nó em que podem haver duas definições idênticas de propriedade em vários aspectos exceto neste tipo de estado “multiple values”. Por exemplo, um tipo de nó pode especificar duas propriedades string chamadas X, uma delas é multi valor e a outra não é. Um exemplo de um tipo de nó com essa característica em termos de propriedade é o embutido nt:unstructured.

#### 3.2.1.2 Definição de Nó Filho

Uma definição de nó filho (incluído em uma definição de tipo de nó) contém as seguintes informações:

- **Name (Nome):** O nome de um nó filho no qual esta definição se aplica ou **'*'** se esta definição é uma 'definição residual', significando que se aplica a qualquer nó filho adicional com qualquer nome independente das demais definições neste tipo de nó.
- **Required Primary Types (Tipos Primários Requeridos):** Se é especificado somente um tipo simples de nó 'N' então o tipo primary do nó deste nó filho deve ser 'N' ou um sub tipo de 'N'. Se este atributo especifica tipos múltiplos de nó 'N1, N2,...Nm', então o tipo primary do nó deste nó filho deve ser um sub tipo de todos os tipos “N1, N2,...Nm'. Observe que isto é possível porque Jackrabbit suporta herança múltipla entre os tipos de nós e que cada nó permanece tendo somente um tipo primary de nó.
- **Default Primary Type (Tipo Primário Padrão):** Este é o tipo primário de nó automaticamente atribuído, se nenhuma informação de tipo de nó é especificada quando o nó é criado.
- **Auto-create Status (Estado de Auto Criação):** Controla quando este nó filho poderá ser auto criado juntamente com a criação de seu nó pai.
- **Mandatory Status (Estado Mandatório):** Controla quando o nó filho é mandatório. Um nó filho mandatório é um que deve existir. Se um nó filho mandatório não existe para um nó pai então o salvamento deste nó pai falhará.
- **On-Parent-Version Status (Estado no Caso de Versionamento Pai):** Especifica o que fazer com o nó filho se seu nó pai for versionado.
- **Protected Status (Estado Protegido):** Controla se o nó filho está protegido. Um nó protegido é aquele que não pode ser modificado (ter nós filhos ou propriedades adicionadas ou removidas dele) ou removido da estrutura de seu pai através da API JCR.
- **Same-Name_Siblings Status (Estado Irmãos com Mesmo Nome):** Controla se o nó filho pode ter o mesmo nome de um outro nó filho já existente no mesmo nível deste (filhos do mesmo nó pai).

#### 3.2.1.3 Características do atributo "onParentVersion"

Este atributo merece um capítulo para sua descrição, com relação a seus possíveis valores, pelo fato de que dependendo desses valores o comportamento de versionamento do nó/propriedade será afetado.

Todo item do repositório (nó ou propriedade) tem um indicador de situação que indica o que acontecerá ao mesmo, caso seu nó pai seja versionado. Este indicador é definido pelo atributo "onParentVersion" na definição de propriedade ou de nó em que se aplica o item em questão.

Por exemplo, deixemos um nó "**N**" como um nó versionável, significando que ele tem um tipo de nó "mix:versionable" atribuído a ele. Também deixemos "**N**" ter um tipo primário de nó que permite a ele ter uma propriedade chamada "**P**" e um nó filho chamado "**C**" (child).

O que acontecerá a "**P**" e "**C**" quando uma nova versão de "**N**" for persistida, dependerá de seus respectivos atributos "**onParentVersion**" definidos, na definição de propriedade para "**P**" e na definição de nó para "**C**".

Os possíveis valores para o atributo "onParentVersion" são: **COPY, VERSION, INITIALIZE, COMPUTE, IGNORE E ABORT**. Usando então o caso acima, segue a descrição para cada possível valor de "onParentVersion", o que ocorrerá em "**C**" e em "**P**" quando:

- "**N.checkin()**" é acionado, criando a nova versão "$\mathbf{V_N}$" e incluindo-a para o histórico de versão.
- "**N.restore($V_N$ , b)**" é acionado, restaurando a versão "$\mathbf{V_N}$" (o parâmetro booleano "**b**" indica o que ser feito no caso de colisão de UUID).

- COPY
    - NÓ FILHO: No checkin (entrega) de "N", "C" e todos seus itens descendentes,

descendo nas ramificações de sua sub-árvore, serão copiados para o armazenamento de versão como uma sub-árvore filha de "$V_N$". A cópia de "C" e sua sub-árvore não terá seu próprio histórico de versão, mas será parte do estado preservado em "$V_N$". "C" por si próprio não necessita ser versionável. Na restauração (restore) de "$V_N$", a cópia de "C" e sua sub-árvore armazenada serão restauradas também, substituindo o "C" atual e sua sub-árvore no workspace.

- **PROPRIEDADE**: No checkin de "**N**", "**P**" será copiado para a versão armazenada como um filho de "$\mathbf{V_N}$". Esta cópia de "**P**" é parte do estado preservado em "$\mathbf{V_N}$". Na restauração de "$\mathbf{V_N}$", a cópia de "**P**" armazenada como sua filha será restaurada também, substituindo o "**P**" atual do workspace.

- VERSION
  - **NÓ FILHO**: No checkin de "**N**", o nó "$\mathbf{V_N}$" pegará um sub-nó do tipo "**nt:versionedChild**" com o mesmo nome como "**C**". A propriedade simples deste nó, "**jcr:childVersionHistory**", é uma REFERENCE (referência) para o histórico de versão de "***C***" (não para "**C**" ou qualquer outra versão atual de "**C**"). Isto também requer que o próprio "**C**" seja versionável (senão ele não poderia ter um histórico de versão). Se "**C**" não é versionável então o comportamento de **COPY** se aplica no checkin, entretanto a cópia recursiva termina a cada nó versionável encontrado mais abaixo na sub-árvore, aos quais o padrão de comportamento **VERSION** é novamente aplicado.

```
NodeTypeName
nt:versionedChild
Supertypes
nt:base
IsMixin
false
HasOrderableChildNodes
false
PrimaryItemName
null
PropertyDefinition
Name jcr:childVersionHistory
RequiredTypeREFERENCE
ValueConstraint["nt:versionHistory"]
DefaultValuesnull
AutoCreatedtrue
Mandatory true
OnParentVersionABORT
Protectedtrue
Multiplefalse
```

Na restauração de "$\mathbf{V_N}$", se o workspace atualmente já possui um nó existente correspondente a uma versão histórica de "**C**" e o flag "**removeExisting**" do método "**restore**" é "**true**", então aquela instância de "**C**" torna o filho do "**N**" restaurado.

Se o workspace atualmente já possui um nó existente correspondente a uma versão histórica de "**C**" e o flag "**removeExisting**" do método "**restore**" é "**false**", então uma exceção "**ItemExistsException**" é lançada.

Se o workspace não tem uma instância de "**C**" então uma versão histórica de "**C**" é restaurada. O workspace no qual o método "**restore**" é acionado determinará qual versão em particular de "**C**" será restaurada. Este comportamento depende da configuração do workspace e da forma de escopo que está especificado neste documento (é específico da implementação do fornecedor da API).

- PROPRIEDADE: No caso de propriedades, um atributo "**onParentVersion**" como **VERSION** tem o mesmo efeito já citado do **COPY**.

- INITIALIZE
  - **NÓ FILHO**: No checkin de "**N**", um novo nó "**C**" será criado e colocado em versão armazenada como um filho de "$\mathbf{V_N}$". Este novo "**C**" será inicializado tal qual tivesse sido criado normalmente no workspace. Nenhuma informação de estado do "**C**" atual no workspace é preservada. Na restauração de "$\mathbf{V_N}$", o "**C**" armazenado como seu filho será ignorado, e o "**C**" atual no workspace ficará inalterado.
  - **PROPRIEDADE**: No checkin de "**N**", um novo "**P**" será criado e colocado em versão armazenada como um filho de "$\mathbf{V_N}$". O novo "**P**" será inicializado tal qual tivesse sido criado normalmente no workspace. Na restauração de "$\mathbf{V_N}$", o "**P**" armazenado como seu filho será ignorado, e o "**P**" atual no workspace ficará inalterado.

- COMPUTE
  - **NÓ FILHO**: No checkin de "**N**", um novo nó "**C**" será criado e colocado em versão armazenada como um filho de "$\mathbf{V_N}$". Este novo "**C**" será inicializado por algum procedimento definido para aquele tipo de nó filho. Na restauração de "$\mathbf{V_N}$", o "**C**" armazenado como seu filho será ignorado, e o "**C**" atual no workspace ficará inalterado.
  - **PROPRIEDADE**: No checkin de "**N**", um novo "**P**" será criado e colocado em versão armazenada como um filho de "$\mathbf{V_N}$". O novo "**P**" será inicializado por algum procedimento definido para aquele tipo de propriedade. Na restauração de "$\mathbf{V_N}$", o "**P**" armazenado como seu filho será ignorado, e o "**P**" atual no workspace

ficará inalterado.

- IGNORE
  - **NÓ FILHO:** No checkin de "N", nenhuma informação de estado sobre "C" será armazenada em "$V_N$". Na restauração de "$V_N$", o nó filho "C" do "N" atual permanecerá e não será removido.
  - **PROPRIEDADE**: No checkin de "**N**", nenhuma informação de estado sobre "**P**" será armazenada em "$\mathbf{V_N}$". Na restauração de "$\mathbf{V_N}$", a propriedade "**P**" do "**N**" atual permanecerá e não será removido.

- ABORT
  - **NÓ FILHO ou PROPRIEDADE:** No checkin de "N" uma "VersionException" será lançada. Tendo um nó filho ou uma propriedade com um atributo "onParentVersion" como ABORT previne/impede que o nó pai seja entregado (checked-in).

### 3.3 Notação XML para Tipos de Nó

A notação XML para definição dos tipos de nó é a disponibilizada pela especificação JSR170 e compatível com a API de repositório JCR, portanto é a mais recomendada por ter todos os parâmetros especificados com seus valores. Segue um exemplo de corpo de um arquivo contendo todas as opções existentes numa declaração de tipos de nó:

```
<?xml version="1.0" encoding="UTF-8"?>
<nodeTypes
 xmlns:rep="internal"
 xmlns:nt="http://www.jcp.org/jcr/nt/1.0"
 xmlns:mix="http://www.jcp.org/jcr/mix/1.0"
 xmlns:jcr="http://www.jcp.org/jcr/1.0"
 xmlns:escriba="http://www.ScribaPinhao.pr.gov.br/1.0"
 xmlns:prop="http://www.ScribaPinhao.pr.gov.br/prop/1.0">
<!-- Especifica -->
 <nodeType name="escriba:base" isMixin="false" hasOrderableChildNodes="false"
primaryItemName="">
  <supertypes>
  <!-- Especifica que o nó terá as definições já dadas em nt:hierarchyNode -->
   <supertype>nt:hierarchyNode</supertype>
  </supertypes>
  <propertyDefinition name="prop:criador" requiredType="Reference" autoCreated="false"
mandatory="false" onParentVersion="VERSION" protected="false" multiple="false" >
   <valueConstraints>
   <!-- Especifica que a propriedade somente poderá ter valores que referenciem nós que sejam
do tipo escriba:usuario -->
    <valueConstraint>escriba:usuario</valueConstraint>
```

```
   </valueConstraints>
  </propertyDefinition>
  <propertyDefinition name="prop:descricao" requiredType="String" autoCreated="false"
mandatory="false" onParentVersion="VERSION" protected="false" multiple="false"/>
  <propertyDefinition name="prop:ativo" requiredType="Boolean" autoCreated="true"
mandatory="true" onParentVersion="VERSION" protected="false" multiple="false">
   <defaultValues>
    <!-- Especifica que o valor padrao para a propriedade, ao ser criada será true -->
    <defaultValue>true</defaultValue>
   </defaultValues>
  </propertyDefinition>
  <childNodeDefinition name="_Permissao" defaultPrimaryType="escriba:seguranca"
autoCreated="true" mandatory="false" onParentVersion="VERSION" protected="false"
sameNameSiblings="false">
   <requiredPrimaryTypes>
   <!-- Especifica que o nó filho deve ser do tipo escriba:seguranca, ou derivado do mesmo -->
    <requiredPrimaryType>escriba:seguranca</requiredPrimaryType>
   </requiredPrimaryTypes>
  </childNodeDefinition>
 </nodeType>
</nodeTypes>
```

## 3.4 Notação CND para Tipos de Nó

A notação CND (Compact Namespace and Node Type Definition) fornece um padrão compactado de sintaxe para definição de tipos de nó e declarações de namespaces. O Jackrabbit tem suporte a esta notação, sendo que nativamente a API JCR suporta a notação padrão em linguagem XML.

Segue um exemplo usando todas configurações possíveis na notação CND:

```
// A declaração de namespace
<ns = 'http://namespace.com/ns'>

// Nome do tipo de Nó
[ns:TipodeNoX]

// Supertipos
> ns:TipoPai1, ns:TipoPai2

// Este tipo de nó suporta nós filhos ordenáveis
orderable

// Este é um tipo de nó mixin
mixin

// Nós deste tipo tem uma propriedade chamada 'ex:propriedade' do tipo String
- ex:propriedade (string)

// Os valores padrão para esta propriedade multi-valor são...
= 'padrão1', 'padrão2'

// Esta propriedade é definida como o item primário
primary
```

```
// e também tem as demais características...
mandatory autocreated protected

// e a característica de poder ser multi valorizada
multiple

// Ela tem configuração para versionamento do pai...
version

// Suas restrições são...
< 'restricao1', 'restricao2'

// Nós deste tipo tem nó filho chamado 'ns:node o que deve ser ao menos de um
dos tipos de nó: ns:tipoRequerido1 e ns:tipoRequerido2
+ ns:node (ns:tipoRequerido1, ns:TipoRequerido2)

// e o tipo de nó primário padrão do nó filho é...
= ns:tipoPrimarioPadraoFilho

// Este nó filho é, por sua vez...
mandatory autocreated protected

// e suporta irmãos com mesmo nome
multiple

// e tem um versionamento de acordo com o do seu pai ...
version
```

A definição acima pode ser escrita de uma maneira compacta, com identação e sem comentários:

```
<ns = 'http://namespace.com/ns'>
[ns:TipodeNoX] > ns:TipoPai1, ns:TipoPai2
  orderable mixin
  - ex:propriedade (string)
  = 'padrão1', 'padrão2'
    primary mandatory autocreated protected multiple
    version
    < 'restricao1', 'restricao2'
  + ns:node (ns:tipoRequerido1, ns:tipoRequerido2)
    = ns:tipoPrimarioPadraoFilho
    mandatory autocreated protected multiple version
```

Ou ainda mais compactado, usando a forma simplificada de definição de atributos:

```
<ns='http://namespace.com/ns'>
[ns:TipodeNoX]>ns:TipoPai1, ns:TipoPai2 o m
-ex:propriedade='padrão1','padrão2' ! m a p * version
 <'restricao1', 'restricao2'
+ns:node(ns:tipoRequerido1,ns:tipoRequerido2)=ns:tipoPrimarioPadraoFilho
 m a p *  version
```

### 3.4.1 Sintaxe Geral da Notação CND

Os símbolos de terminadores estão com aspas duplas:

```
cnd ::= {ns_mapping | node_type_def}

ns_mapping ::= "<" prefix "=" uri ">"

prefix ::= string

uri ::= string

node_type_def ::= node_type_name [supertypes] [options]
                  {property_def | child_node_def}

node_type_name ::= "[" string "]"

supertypes ::= ">" string_list

options ::= orderable_opt | mixin_opt | orderable_opt
            mixin_opt | mixin_opt orderable_opt

orderable_opt ::= "orderable" | "ord" | "o"

mixin_opt ::= "mixin" | "mix" | "m"

property_def ::= "-" property_name [property_type_decl]
                 [default_values] [attributes]
                 [value_constraints]

property_name ::= string

property_type_decl ::= "(" property_type ")"

property_type ::= "STRING" | "String" |"string" |
                  "BINARY" | "Binary" | "binary" |
                  "LONG" | "Long" | "long" |
                  "DOUBLE" | "Double" | "double" |
                  "BOOLEAN" | "Boolean" | "boolean" |
                  "DATE" | "Date" | "date" |
                  "NAME | "Name | "name" |
                  "PATH" | "Path" | "path" |
                  "REFERENCE" | "Reference" |
                     "reference" |
                  "UNDEFINED" | "Undefined" |
                     "undefined" | "*"

default_values ::= "=" string_list

value_constraints ::= "<" string_list

node_def ::= "+" node_name [required_types]
             [default_type] [attributes]

node_name ::= string

required_types ::= "(" string_list ")"

default_type ::= "=" string

attributes ::= "primary" | "pri" | "!" |
               "autocreated" | "aut" | "a" |
               "mandatory" | "man" | "m" |
               "protected" | "pro" | "p" |
               "multiple" | "mul" | "*" |
               "COPY" | "Copy" | "copy" |
               "VERSION" | "Version" | "version" |
               "INITIALIZE" | "Initialize" |
                  "initialize" |
```

```
                "COMPUTE" | "Compute" | "compute" |
                "IGNORE" | "Ignore" | "ignore" |
                "ABORT" | "Abort" | "abort"

string_list ::= string {"," string}

string ::= quoted_string | unquoted_string

quoted_string :: = "'" unquoted_string "'"

unquoted_string ::= /* uma string */
```

### 3.4.2 Sintaxe da Notação CND em Detalhes

```
Constituição de um cnd

cnd ::= {ns_mapping | node_type_def}
```

Um CND consiste de zero ou mais blocos, cada um pode ser: declaração de namespaces ou definição de tipo de nó. Os prefixos de namespace referenciados no bloco de definição de tipo de nó, devem ser previamente declarados em um bloco de declaração de namespace.

- Declaração de NameSpace

```
ns_mapping ::= "<" prefix "=" uri ">"

prefix ::= string

uri ::= string
```

Uma declaração de namespace consiste de um par prefixo/uri. O prefixo (prefix) deve ser um prefixo de namespace JCR válido, o qual é o mesmo para um prefixo de namespace válido para arquivo XML. A uri pode na verdade ser qualquer string. Tal qual para XML, não é necessário que seja uma uri real, mas aconselha-se que o seja.

- Definição de Tipo de Nó

```
node_type_def ::= node_type_name [super_types] [options]

                  {property_def | child_node_def}
```

Uma definição de tipo de nó(node_type_def) consiste de um nome para o tipo de nó

(node_type_name) seguido por um bloco opcional de supertipos (super_types), um bloco opcional de opções (options) e zero ou mais blocos de definição de propriedade ou nó.

- Nome de Tipo de Nó

```
node_type_name ::= "[" string "]"
```

O nome do tipo de nó é delimitado por colchetes e deve ser um nome JCR válido. Pode estar entre aspas simples. Este elemento é o único estritamente requerido dentro de uma definição de tipo de nó, então uma definição consistindo somente de um nome de tipo de nó poderia simplesmente definir um novo tipo de nó idêntico a **nt:base.**

- SuperTipos

```
supertypes ::= ">" string_list
```

Após o nome do tipo de nó vem a lista opcional de supertipos. Se este elemento não está presente e o nó não é do tipo mixin, então um super tipo nt:base é assumido. Se presente, o elemento consiste de um sinal de "maior que" seguido de uma lista de nomes de tipos de nó delimitados por vírgula, cada um podendo estar entre aspas. Em Jackrabbit, a herança múltipla de tipos de nó é suportada, então esta lista pode ter mais que um item em sua extensão.

- Opções

```
options ::= orderable_opt | mixin_opt | orderable_opt
            mixin_opt | mixin_opt orderable_opt

orderable_opt ::= "orderable" | "ord" | "o"

mixin_opt ::= "mixin" | "mix" | "m"
```

A indicação de **options** segue o nome do tipo de nó e a lista opcional de **supertypes**.

Se a palavra **orderable** (ou em seu formato abreviado) está presente, então a definição de ordenação do nó filho do tipo de nó é true (verdadeiro). Se esta palavra não está informada, então a definição fica como false.

Se a palavra **mixin** (ou sua abreviação) está presente, então este é um nó do tipo mixin. Se não está presente, então este é um tipo de nó primary (primário).

- Definição de Propriedade

```
property_def ::= "-" property_name [property_type_decl]
                 [default_values] [attributes]
                 [value_constraints]
```

Uma definição de propriedade consiste de um sinal de menos "-" seguido de um nome de propriedade, por sua vez seguido de elementos opcionais que definem seu tipo de propriedade, seus valores padrão, seus atributos e restrições.

- Nome de Propriedade

```
property_name ::= string
```

O nome da propriedade deve ser um nome JCR válido ou '*', para indicar uma definição de propriedade residual. Deve estar entre aspas.

- Tipo de Propriedade

```
property_type_decl ::= "(" property_type ")"

property_type ::= "STRING" | "String |"string" |
                  "BINARY" | "Binary" | "binary" |
                  "LONG" | "Long" | "long" |
                  "DOUBLE" | "Double" | "double" |
                  "BOOLEAN" | "Boolean" | "boolean" |
                  "DATE" | "Date" | "date" |
                  "NAME | "Name | "name" |
                  "PATH" | "Path" | "path" |
                  "REFERENCE" | "Reference" | "reference" |
                  "UNDEFINED" | "Undefined" | "undefined" | "*"
```

O tipo da propriedade é indicada por uma palavra delimitada por parênteses. Se a declaração do tipo da propriedade é omitida, o tipo STRING é assumido.

- Valores Padrão

```
default_values ::= "=" string_list
```

O valor padrão, ou os valores padrão no caso de propriedade multi-valor, são indicadas por um sinal de igual “=” seguido tanto por um simples valor num formato string ou uma lista de valores delimitada por vírgulas. Os valores podem estar entre aspas. Se a definição do valor padrão é omitida então nenhum valor padrão é gravado.

- Atributos

```
attributes ::= "primary" | "pri" | "!" |
               "autocreated" | "aut" | "a" |
               "mandatory" | "man" | "m" |
               "protected" | "pro" | "p" |
               "multiple" | "mul" | "*" |
               "COPY" | "Copy" | "copy" |
               "VERSION" | "Version" | "version" |
               "INITIALIZE" | "Initialize" | "initialize" |
               "COMPUTE" | "Compute" | "compute" |
               "IGNORE" | "Ignore" | "ignore" |
               "ABORT" | "Abort" | "abort"
```

A indicação attribute descreve as características da propriedade. A presença da palavra attribute indica que as características correspondentes se aplicam para a propriedade. Sua omissão indica nenhuma característica correspondente.

A palavra primary indica que esta propriedade é o item primário. Ele pode aparecer no máximo, em uma propriedade ou em uma definição de nó filho, dentro de uma definição de tipo de nó.

A palavra multiple indica que esta propriedade é multi-valor.

No máximo uma indicação de version pode estar presente. Se for omitida então a definição de copy é assumida.

- Restrições de Valores

```
value_constraints ::= "<" string_list
```

Restrições de valores são especificados por um sinal de menor “<” seguido de uma

lista de restrições separada por vírgula, cada uma opcionalmente entre aspas.

- Definição de Nó Filho

```
child_node_def ::= "+" node_name [required_types]
             [default_type] [attributes]
```

Uma definição de nó filho consiste de um sinal de mais “+” seguido de um nome de propriedade, seguido por sua vez por elementos opcionais definindo os tipos de nó primário, o tipo padrão, e os atributos do nó.

- Nome do Nó

```
node_name ::= string
```

O nome do nó deve ser um nome JCR válido ou “*”, para indicar uma definição de nó filho residual. Pode estar entre aspas.

- Tipos de Nó Primários Requeridos

```
required_types ::= "(" string_list ")"
```

Os tipos de nó requeridos de um nó filho são indicados por uma lista de tipos de nó delimitados por vírgula, entre parênteses. Se este elemento é omitido então um tipo de nó primário como nt:base é assumido. Esta é a mínima definição restritiva possível.

- Tipo de Nó Primário Padrão

```
default_type ::= "=" string
```

O tipo de nó primário padrão é indicado por um sinal de igual “=” seguido por um nome de tipo de nó, o qual pode estar entre aspas. Se este elemento é omitido então não há tipo de nó primário padrão.

- Atributos

```
attributes ::= "primary" | "pri" | "!" |
               "autocreated" | "aut" | "a" |
               "mandatory" | "man" | "m" |
               "protected" | "pro" | "p" |
               "multiple" | "mul" | "*" |
               "COPY" | "Copy" | "copy" |
               "VERSION" | "Version" | "version" |
               "INITIALIZE" | "Initialize" | "initialize" |
               "COMPUTE" | "Compute" | "compute" |
               "IGNORE" | "Ignore" | "ignore" |
               "ABORT" | "Abort" | "abort"
```

A indicação attribute descreve as características do nó filho. A presença de uma palavra attribute indica que as características correspondentes se aplicam ao nó filho. Sua omissão indica que não existem características correspondentes a serem aplicadas.

A palavra primary indica que este nó filho é o item primário. Ele pode aparecer no máximo em uma propriedade ou na definição de nó filho dentro da definição do tipo de nó.

A palavra multiple indica que este nó filho pode ter irmãos de mesmo nome.

O máximo de uma indicação on-version pode estar presente. Se omitida então uma definição de copy é assumida.

- Aspas

```
string_list ::= string {"," string}

string ::= quoted_string | unquoted_string

quoted_string :: = "'" unquoted_string "'"

unquoted_string ::= /* a string */
```

Aspas simples (') são usadas para permitir strings (nomes, prefixos, uri's, valores ou restrições) com caracteres que poderiam ser interpretados como delimitadores.

- Escapes

A seqüência de escapes padrão Java também é suportada.

`\n` (nova linha)

`\t` (tabulação)

`\b` (retorno de espaço)

`\f` (passo de formulário)

`\r` (retorno, return)

`\"` (aspas duplas)

`\'` (aspa simples)

`\\` (back slash)

`\uHHHH` (caracter Unicode em hexadecimal)

- Comentários

Comentários também estão incluídos na notação podendo ser igual ao padrão Java:

```
// Comentário 1

/* Outra forma de comentário */
```

- Espaços em Branco e Termos Abreviados

A notação pode ser compactada omitindo os espaços em branco após os caracteres chave (`[ ] > , - ( ) = <`), novas linhas e identações que não são requeridas. Então a seguinte expressão também é válida:

```
[x]>y,z orderable mixin -p(date)=a,b primary mandatory autocreated protected
multiple version <c,d
```

Adicionalmente, podem se abreviar as palavras chaves (orderable, mixin, date, mandatory, etc.), separando-as com espaços em branco. então a seguinte expressão é válida:

```
[x]>y,z o m-p(date)=a,b ! m a p * version <c,d
```

Mas não é recomendada...

### 3.4.3 Comparação da Notação CND com a Notação Padrão XML

Segue definição do tipo de nó interno nt:resource utilizando a notação padrão XML:

```
<nodeType name="nt:resource"
  isMixin="false"
  hasOrderableChildNodes="false"
  primaryItemName="jcr:data">
```

```
 <supertypes>
  <supertype>nt:base</supertype>
  <supertype>mix:referenceable</supertype>
 </supertypes>
 <propertyDefinition name="jcr:encoding"
                     requiredType="String"
                     autoCreated="false"
                     mandatory="false"
                     onParentVersion="COPY"
                     protected="false"
                     multiple="false"/>
 <propertyDefinition name="jcr:mimeType"
                     requiredType="String"
                     autoCreated="false"
                     mandatory="true"
                     onParentVersion="COPY"
                     protected="false"
                     multiple="false"/>
 <propertyDefinition name="jcr:data"
                     requiredType="Binary"
                     autoCreated="false"
                     mandatory="true"
                     onParentVersion="COPY"
                     protected="false"
                     multiple="false"/>
 <propertyDefinition name="jcr:lastModified"
                     requiredType="Date"
                     autoCreated="false"
                     mandatory="true"
                     onParentVersion="IGNORE"
                     protected="false"
                     multiple="false"/>
</nodeType>
```

E o mesmo tipo de nó sendo expressado na notação CND:

```
[nt:resource] > mix:referencable
- jcr:encoding
- jcr:mimeType mandatory
- jcr:data (binary) mandatory
- jcr:lastModified (date) mandatory ignore
```

# 4 PRIMEIROS PASSOS – CODIFICANDO COM JACKRABBIT

De posse da seguinte lista de dependências podemos construir uma simples aplicação com Jackrabbit:

**jackrabbit-core-1.0-SNAPSHOT.jar,**
**jcr-1.0.jar (JCR 1.0 API),**
**log4j-1.2.8.jar,**
**commons-collections-3.1.jar,**
**xercesImpl-2.6.2.jar, xmlParserApis-2.0.2.jar,**
**derby-10.1.1.0.jar,**
**concurrent-1.3.4.jar,**

**lucene-1.4.3.jar**
Observação: Atualmente verificar as dependências do jackrabbit em sua versao 1.4 – 1.5.

Esta parte é para demonstrar a criação de um Repositório e iniciar uma sessão para acessá-lo da maneira mais simples, seguido de uma breve explicação de alguns comandos:

```
import javax.jcr.Repository;
import javax.jcr.Session;
import org.apache.jackrabbit.core.TransientRepository;
/**
 * Primeiro exemplo, logando em um repositório de conteúdo e imprimindo uma
 * mensagem de status
 */
public class PrimeirosPassos {
 /**
 * Ponto principal da aplicação exemplo.
 *
 * @param args argumento de linha de comando (ignorado)
 * @throws Exception se um erro ocorrer
 */
 public static void main(String[] args) throws Exception {
  Repository repository = new TransientRepository();
  Session session = repository.login();
  try {
   String user = session.getUserID();
   String name = repository.getDescriptor(Repository.REP_NAME_DESC);
   System.out.println("Logado como " + user + " em um repositório " + name);
  } finally {
    session.logout();
  }
 }
}
```

Executando o código acima tem-se a seguinte saída:

"Logado como anonymous em um repositório Jackrabbit"

Adicionalmente à produção da linha de status acima, a aplicação copia o arquivo de configuração padrão do repositório para repository.xml e cria um repositório inicial de conteúdo Jackrabbit no subdiretório de repository. Podem ser especificados arquivos alternativos de configuração e locais para o diretório do repositório diferentes nos arquivos de propriedades em **org.apache.jackrabbit.repository.conf** e **org.apache.jackrabbit.repository.home .**

Segue maior detalhamento do código:

**import javax.jcr.Repository;**
**import javax.jcr.Session;**

As interfaces da API JCR estão no pacote javax.jcr entrados no arquivo jcr-1.0.jar. A promessa desta API é que ao utilizar somente estas interfaces em sua aplicação de conteúdo, isso a manterá independente sobre a implementação da camada do repositório de conteúdo.

A interface Repository representa uma dada instância de um repositório de conteúdo e a interface Session representa uma única sessão de login para acessar o repositório. Uma sessão é necessária para acessar qualquer conteúdo dentro de um repositório.

Observe que uma instância Session não é multi thread, então devem ser iniciadas múltiplas sessões se necessitar acessar o conteúdo do repositório de diferentes threads. Este detalhe é especialmente importante no caso de aplicações web.

**import org.apache.jackrabbit.core.TransientRepository;**

A melhor prática para deploy do Jackrabbit é utilizar JNDI ou algum outro mecanismo de configuração, em um ambiente de aplicação para que a aplicação não seja dependente direta do Jackrabbit, mas neste caso de estarmos criando uma simples aplicação de exemplo, podemos aproveitar a facilidade utilizando a classe TransientRepository do núcleo Jackrabbit.

**public class FirstHop**
**public static void main(String[] args) throws Exception**

Este exemplo é para uma aplicação independente que se enquadra somente no método main() e permite que a JVM cuide das possíveis exceções. Outras aplicações reais devem ser escritas com diferentes setups e padrões de manipuladores de erros.

**Repository repository = new TransientRepository();**

A classe TransientRepository implementa a interface JCR Repository, então pode ser simplesmente atribuído uma instância TransientRepository para uma variável Repository. Seu default constructor (construtor padrão) contém um recurso que cuidará da configuração inicial e construção do repositório quando a primeira sessão for iniciada. Assim não se necessita de configuração manual por enquanto, a menos que se queira controle direto sobre o setup do

repositório.

A implementação de TransientRepository também cuidará do fechamento (shutdown) do repositório quando a última sessão for fechada. Observe que no diretório do repositório Jackrabbit existe um arquivo de lock (bloqueio) que evita que ele seja acessado simultaneamente por múltiplos processos. Se não forem fechadas todas as sessões ou baixado o repositório apropriadamente antes de finalizar o processo que o acessa, na próxima inicialização ocorrerá uma exceção causada devido a esse arquivo de lock. Normalmente neste caso pode ser manualmente removido o arquivo ".lock", mas isso em freqüência pode ocasionar uma chance de corrupção nos dados do repositório, principalmente se é utilizada uma persistência não transacional.

**Session session = repository.login();**

O método padrão Repository.login() inicia a sessão no repositório usando o workspace padrão e sem credenciais. Jackrabbit tenta usar autenticação JAAS nesses casos, mas utiliza usuário padrão se um objeto JAAS não é encontrado.

Estar usando a classe TransientRepository como implementação do Repository, este passo provoca também a inicialização do repositório.

**try { ... } finally { session.logout(); }**

É uma boa prática liberar todos os recursos adquiridos, e as sessões JCR para evitar exceções. O trecho try-finally é uma forma de garantir que os recursos foram realmente liberados, e se não pode capturar a exceção podendo fazer seu tratamento. O método Session.logout() no trecho finally fecha a sessão, uma vez que esta é a única sessão que temos, o repositório automaticamente será finalizado.

**String user = session.getUserID();**

O username ou identificador do usuário associado a sessão é disponibilizado com o método Session.getUserID(). Jackrabbit retorna "anonymous" como padrão.

**String name = repository.getDescriptor(Repository.REP_NAME_DESC);**

Cada implementação de repositório de conteúdo pode publicar um número de strings descritoras, que contém várias propriedades da implementação, como o nível de implementação e os recursos adicionais de JCR. Verificar a interface Repository para uma lista dos descritores padrões do repositório. O descritor REP_NAME_DESC contém o nome da implementação do repositório, neste caso "Jackrabbit".

# 5 PESQUISANDO CONTEÚDO NO REPOSITÓRIO

Como já dito, o repositório suporta linguagens de pesquisa no padrão SQL e Xpath.

A linguagem de pesquisa XPath foi originalmente definida para busca de elementos a partir de um documento XML. Como um workspace, tal qual um documento XML, pode ser visualizado como uma estrutura em árvore, a linguagem XPath fornece uma sintaxe conveniente para buscas de conteúdo neste workspace.

A execução de uma pesquisa XPath é sobre uma visão de um documento no workspace. Considerando um worspace com a seguinte estrutura:

```
<?xml version="1.0" encoding="UTF-8"?>
<jcr:root xmlns:jcr="http://www.jcp.org/jcr/1.0"
 xmlns:nt="http://www.jcp.org/jcr/nt/1.0"
 xmlns:mix="http://www.jcp.org/jcr/mix/1.0"
 xmlns:myapp="http://mycorp.com/myapp"
 xmlns:mynt="http://mycorp.com/mynt"
 jcr:primaryType="nt:unstructured">
 <myapp:documento jcr:primaryType="mynt:documento">
  myapp:titulo="JSR 170"
  myapp:objetivo="Repositório de Conteúdo">
  <myapp:corpo jcr:primaryType="mynt:corpo">
   <myapp:paragrafo jcr:primaryType="mynt:paragrafo"
    myapp:titulo="Tipos de Nó"
    myapp:texto="Um importante recurso..."/>
  </myapp:corpo>
 </myapp:documento>
</jcr:root>
```

(Observando que o XML foi formatado para melhor legibilidade. Um stream XML não pode possuir espaços em branco entre os elementos ou atributos).

Para localizar o nó de nome myapp:paragrafo, a seguinte expressão XPath pode ser

utilizada:

```
//element(*, mynt:paragrafo)[@myapp:titulo="Tipos de Nó"]
```

Esta pesquisa retornará o nó com a sua localização no workspace:

```
/myapp:documento/myapp:corpo/myapp:paragrafo
```

## 5.1 Estrutura de uma Pesquisa

A estrutura de uma pesquisa, independente se XPath ou SQL, especifica um subconjunto de nós dentro de um workspace, chamado de conjunto de resultados. O conjunto de resultados é constituído por todos os nós do workspace que batem com as restrições da pesquisa. Essas restrições podem estar em três categorias:

- Restrição de Tipo: Limita os nós retornados a serem de um tipo particular de nó primário (e possivelmente, limita os nós com um tipo particular de nó “mixin”).
- Restrição de Propriedade: Limita os nós retornados a serem aqueles com uma propriedade particular tendo um valor particular.
- Restrição de Localização: Limita os nós retornados a serem aqueles dentro de certas subarvores definidas no workspace.

Um resultado de pesquisa é retornado de duas formas paralelas: um iterador sobre o conjunto de nós resultantes e uma tabela onde cada linha corresponde a um nó no conjunto de resultados (cada coluna na tabela corresponde a uma propriedade definida ou herdada do seu tipo de nó primary (primário) e ou mixto (mixin)). A expressão de pesquisa também define aspectos de como estes dois objetos retornados são estruturados através de duas formas de especificadores de apresentação:

- Especificador de Coluna: Especifica o conjunto de propriedades que formarão as colunas da tabela retornada.
- Especificador de Ordenação: Define a ordem dos nós no iterador e linhas na tabela.

**1 – Especificador de Coluna**

É a parte da expressão de pesquisa que define quais propriedades serão retornadas como colunas na tabela de resultado. Se nenhuma especificação de coluna é especificada

então ao menos o conjunto padrão de colunas será retornado. O conjunto padrão é definido como todos valores simples, propriedades não residuais declaradas no tipo de nó ou herdadas especificadas na restrição de tipo. Em ambos os casos (com a especificação ou com o conjunto padrão) uma pseudo propriedade "jcr:path" sempre será retornada como uma coluna. "jcr:path" é uma coluna especial que não corresponde a nenhuma propriedade do nó, ela possui o caminho absoluto ao nó representado por cada linha. Da mesma forma, uma coluna de pontuação (jcr:score) será sempre incluída. O especificador de coluna não tem efeito no conteúdo ou forma do objeto NodeIterator resultante da pesquisa.

No XPath o mecanismo especificador de coluna não é definido, no SQL o especificador de coluna é o termo SELECT. Para selecionar as colunas padrão o "*" é utilizado.

Exemplos:

| ***SQL*** | ***XPath*** |
|---|---|
| SELECT * FROM nt:base | //* |
| SELECT * FROM meu:tipo | //element(*, meu:tipo) |
| SELECT meu:titulo FROM meu:tipo | //element(*, meu:tipo)/@meu:titulo |
| SELECT meu:titulo, meu:texto FROM meu:tipo | //element(*, meu:tipo)/(@meu:titulo \| @meu:texto) |

**2 – Restrição de Tipo**

Especifica o tipo primário de nó comum dos nós retornados, mais, possivelmente, tipos mixin adicionais que podem ter. Restrições de tipo são herdáveis, onde uma restrição especificada em um tipo de nó X incluirá todos os nós explicitamente declarados para serem do tipo X, bem como todos os nós de subtipos de X.

Em XPath o termo "element" é utilizado para restringir o tipo de nó. Em SQL ele é definido no termo "FROM".

Exemplos:

| ***SQL*** | ***XPath*** |
|---|---|
| SELECT * FROM meu:tipo | //element(*, meu:tipo) |

| *SQL* | *XPath* |
|---|---|
| SELECT * FROM meu:tipo WHERE jcr:path LIKE '/nos[%]/%' | /jcr:root/nos//element(*, meu:tipo) |

**3 – Restrição de Propriedade**

Uma pesquisa pode especificar várias restrições sobre os nós resultantes através da restrição de propriedades.

Em XPath um predicado (@) que testa os atributos no fim da declaração expressão de restrição de propriedade.

Em SQL a cláusula WHERE forma a expressão de restrição.

Exemplos:

| *SQL* | *XPath* |
|---|---|
| SELECT * FROM meu:tipo WHERE meu:titulo='JSR 170' | //element(*,meu:tipo) [@meu:titulo = 'JSR 170'] |

Em ordem, para garantir tradução mútua entre XPath e SQL, os elementos de comparação são padronizados (=, !=, <, <=, >, >=). A única diferença entre os operadores é no de “não-igual”, enquanto que no XPath ele é “!=”, no SQL ele é “<>”.

Seguem os tipos e as comparações possíveis para os mesmos:

```
STRING: =, != (<>), <, <=, >, >=, jcr:like()(LIKE)
LONG: =, != (<>), <, <=, >, >=
DOUBLE: =, != (<>), <, <=, >, >=
DATE: =, != (<>), <, <=, >, >=
NAME: =, != (<>)
PATH: =, != (<>), (adicionalmente, em SQL, LIKE é utilizado ao invés de jcr:path)
REFERENCE: =, != (<>)
BOOLEAN: =, != (<>)
BINARY: (none)
```

A função “jcr:like” do XPath corresponde ao operador “LIKE” do SQL.

A função “jcr:contains()” corresponde a função “CONTAIS()” do SQL, e é necessária para propriedades do nó, nas quais existe um índice implementado.

Exemplos:

| *SQL* | *XPath* |
|---|---|
| meu:titulo = 'JSR 170' | @meu:titulo = 'JSR 170' |
| meu:titulo <> 'JSR 170' | @meu:titulo != 'JSR 170' |
| meu:titulo < 'JSR 170' | @meu:titulo < 'JSR 170' |
| meu:titulo <= 'JSR 170' | @meu:titulo <= 'JSR 170' |
| meu:titulo > 'JSR 170' | @meu:titulo > 'JSR 170' |
| meu:titulo >= 'JSR 170' | @meu:titulo >= 'JSR 170' |
| meu:titulo = 'JSR 170' AND meu:autor = 'JCP' | @meu:titulo = 'JSR 170' and @meu:autor = 'JCP' |
| meu:titulo = 'JSR 170' OR meu:autor = 'JCP' | @meu:titulo = 'JSR 170' or @meu:autor = 'JCP' |
| NOT (meu:titulo = 'JSR 170') | not(@meu:titulo >= 'JSR 170') |
| meu:titulo IS NOT NULL | @meu:titulo |
| meu:titulo IS NULL | not(@meu:titulo) |
| meu:titulo LIKE 'JSR 170' | jcr:like (@meu:titulo, 'JSR 170') |
| CONTAINS(*, 'JSR 170') | jcr:contains ( . , 'JSR 170') |

**4 – Restrição de Caminho**

Restringe o resultado para um escopo especificado pela expressão de caminho (path). Os seguintes termos são suportados:

- Exato
- Nós Filhos
- Descendentes
- Descendentes ou Recursivo

Em XPath os passos de localização especificam a restrição do caminho.

Em SQL a restrição ocorre como um termo de teste “AND” dentro de uma clausula WHERE da pseudo propriedade “jcr:path” usando tanto o operador “=” ou o operador “LIKE”.

Exemplos:

| *SQL* | *XPath* |
|---|---|
| SELECT * FROM meu:tipo WHERE jcr:path LIKE '/alguns[%]/nós[%]' | /jcr:root/algunsnós/element(nós, meu:tipo) |
| SELECT * FROM meu:tipo WHERE jcr:path = '/alguns/nós' | /jcr:root/algunsnós[1]/element(nós, meu:tipo) [1] |

Exemplos para restrições de caminho para nós filhos:

| ***SQL*** | ***XPath*** |
|---|---|
| SELECT * FROM meu:tipo WHERE jcr:path LIKE '/alguns[%]/nós[%]/%' AND NOT jcr:path LIKE '/alguns[%]/nós[%]/%/%' | /jcr:root/alguns/nós/element(*, meu:tipo) |
| SELECT * FROM meu:tipo WHERE jcr:path LIKE '/alguns/nós/%' AND NOT jcr:path LIKE '/alguns/nós/%/%' | /jcr:root/alguns[1]/nós[1]/element(*, meu:tipo) |

Exemplos para restrições de caminhos descendentes:

| ***SQL*** | ***XPath*** |
|---|---|
| SELECT * FROM meu:tipo WHERE jcr:path LIKE '/alguns[%]/nós[%]/%/%' | /jcr:root/alguns/nós//element(*, meu:tipo) |
| SELECT * FROM meu:tipo WHERE jcr:path LIKE '/alguns/nós/%' | /jcr:root/some[1]/nós[1]//element(*, meu:tipo) |

Exemplos de restrições para caminhos descendentes ou recursivos:

| ***SQL*** | ***XPath*** |
|---|---|
| SELECT * FROM meu:tipo WHERE jcr:path LIKE '/alguns[%]/nós[%]' OR jcr:path LIKE '/alguns[%]/nós[%]/%' | /jcr:root/alguns/nós//element(*, meu:tipo) |
| SELECT * FROM meu:tipo WHERE jcr:path LIKE '/alguns/nós' OR jcr:path LIKE '/alguns/nós/%' | /jcr:root/alguns[1]/nós[1]//element(*, meu:tipo) |

**5 – Especificador de Ordenação**

Especifica a ordenação dos nós resultantes de acordo com a ordenação natural dos valores de uma ou mais propriedades dos nós resultantes. Se não informado em uma expressão de pesquisa, a ordenação será conforme o padrão definido pelo repositório.

Em ambos, XPath e SQL, a ordenação é especificada por uma cláusula especial que lista um ou mais nomes de propriedade e para cada uma define se é ascendente ou descendente. Se não informado o tipo da ordenação, o padrão é ascendente.

Exemplos:

| *SQL* | *XPath* |
|---|---|
| SELECT * FROM meu:tipo ORDER BY meu:titulo | //element(*, meu:tipo) order by @meu:titulo |
| SELECT * FROM meu:tipo ORDER BY meu:data DESC, meu:titulo ASC | //element(*, meu:tipo) order by @meu:data descending, @meu:titulo ascending |
| SELECT * FROM meu:tipo WHERE CONTAINS(., 'jcr') ORDER BY jcr:score DESC | //element(*, meu:tipo) [jcr:contais(., 'jcr')] order by jcr:score() descending |

## 5.2 Facilidades de Pesquisa do Lucene

O motor de busca utilizado pelo Jackrabbit é o Lucene (http://lucene.apache.org) atualmente na versao 2.2.1.

As facilidades para retorno de um resultado oferecidas pelo Lucene podem ser transferidas para a aplicação feita com o motor Jackrabbit (por exemplo Scriba e aplicações que se utilizam do Scriba), descritas com suas definições a seguir:

(traduzido da fonte original: http://lucene.apache.org/java/2_3_2/queryparsersyntax.html)

**Termos**

Uma pesquisa é dividida entre termos e operadores.

Existem dois tipos de termos: Simples e Frases

Um termo SIMPLES é uma única palavra tal qual “teste” ou “primeiro”.

Uma FRASE é um grupo de palavras cercadas por aspas duplas tal qual “teste primeiro”.

Múltiplos termos podem ser combinados juntamente com operadores booleanos para formarem uma pesquisa complexa que será visto mais adiante.

Observação: O analisador que cria os índices é baseado em termos e frases. Portanto dependendo da necessidade pode ser feita a implementação de um analisador específico para as necessidades desde que este não interfira nestes termos a serem utilizados em uma expressão de busca.

**Campos**

O Lucene suporta dados para determinados campos. Quando se efetua uma pesquisa pode ser especificado um campo ou definido que deve ser um campo padrão. Os nomes de campos para isso são dependentes da implementação utilizada.

Pode ser pesquisado qualquer campo informando o seu nome seguido por “:” e entao o termo que esta sendo procurado.

Como exemplo, suponha um índice do Lucene contendo dois campos, 'titulo' e 'descricao' e o campo 'descricao' é o definido como padrão. Se você pretende encontrar um documento como título de “No Caminho Certo” e que contem “não siga por aqui”, pode ser feita as expressões:

titulo:“No Caminho Certo” AND descricao:siga

ou

titulo:“Certo no Caminho” AND Certo

Pelo fato do campo 'descricao' ser considerado o padrão então o mesmo não precisa ser informado.

Observação: O campo somente considera o termo que diretamente o sucede, portanto a pesquisa:

titulo:No Caminho Certo

Somente irá considerar “No” para o campo 'titulo'. E irá buscar por “caminho” e “certo” no campo definido como padrão (neste caso o campo 'descricao').

**Modificadores de Termos**

Lucene suporta a modificação de termos na pesquisa fornecendo uma gama de opções de busca, descritas na sequencia.

## Buscas Curinga

Lucene suporta um ou mais caracteres curinga para buscas entre os termos simples (não dentro de frases).

Para uma busca em que um caractere possa ser substituido por qualquer outro deve ser utilizado o símbolo "?".

Para uma busca em que vários caracteres podem assumir o restante do termo deve ser utilizado o símbolo "*".

Para pesquisar por termos onde um único caractere pode ter o seu valor desconhecido, por exemplo o termo "cazuza" ou "casuza" pode ser utilizado: ca?uza .

Para pesquisar por termos onde diversos caracteres (a partir de 0 caracteres) possam estar contidos na continuação do termo, por exemplo o termo "casa" ou "casarão", pode ser utilizado: casa* .

O curinga para múltiplos caracteres pode estar no meio de um termo: cas*o .

Observação: Não é possível utilizar um símbolo "*" ou "?" como sendo o primeiro caractere de uma pesquisa.

### Buscas Pertinentes (Fuzzy)

Lucene suporta pesquisas pertinentes com base no algorítmo "Levenshtein Distance", ou "Edit Distance algorithm". Para uma pesquisa pertinente deve ser utilizado o símbolo "~" no final de um termo SIMPLES. Por exemplo para pesquisar por termos similares na pronúncia por "versão" utilize:

versão~

Isto trará, entre outros, o termo "Cleverson" além de "versão".

Um parâmetro adicional pode ser informado para especificar o nível de similaridade desejada. O valor aceito é entre 0 e 1, sendo que quanto maior esse valor, mais será a filtragem

de similaridade dos termos. Por padrão esse valor é 0.5. Para efetuar uma busca com esse critério utilize no padrão, por exemplo:

versão~0.8

### Pesquisa por Proximidade

Lucene suporta buscas de palavras dentro de uma distância específica. Para efetuar uma busca por proximidade deve ser utilizado o símbolo "~" no ginal de uma FRASE. Por exemplo, para buscar por "empresa" e "finalidade" contendo 10 palavras entre os termos dentro de algum documento, deve ser utilizado:

"empresa finalidade"~10

### Priorizando um Termo

Lucene fornece um nível de relevância para escolher documentos baseados nos termos encontrados. Para que determinado termo tenha uma maior relevância dentro da busca, utilize o símbolo "^" juntamente com o fator de relevância no final. Quanto maior o fator, mais relevante será o termo dentro daquela busca.

Por exemplo, se estiver buscando por

empresa finalidade

e quiser que o termo "empresa" tenha mais relevância, utilize o símbolo "^" com o fator desejado:

empresa^4 finalidade

Isto fará que os documentos encontrados com o termo empresa sejam mais relevantes.

Pode ser dada a uma FRASE também, como:

"empresa publica"^4 "finalidade lucrativa"

Por padrão, o fator é 1. Embora deva ser positivo, pode ser informado como sendo menor que 1, por exemplo 0.2 .

## Operadores Booleanos

Lucene suporta a lógica booleana em seus operadores, tais como:

AND, “&&”

“+”

“-”

OR, “||”

NOT, “!”

Observação: Os operadores booleanos devem estar em CAIXA ALTA.

O operador OR é definido como padrão. Significando que se não houver operadores booleanos entre dois termos, eles estão sendo combinados segundo a lógica do operador OR.

Esse operador junta os termos e encontra documentos onde um dos termos seja encontrado. Isso equivale a uma união de conjuntos.

Para pesquisar por documentos contendo ambos “empresa publica” ou somente “empresa” use:

“empresa publica” empresa

ou

“empresa publica” OR empresa

ou ainda

“empresa publica” || empresa

O operador AND encontra documentos onde ambos termos existam em algum ponto do texto do documento. Isto equivale a uma interseção de conjuntos. Para pesquisar por documentos que contenham “empresa publica” e “COPEL energia” :

“empresa publica” AND “COPEL energia”

ou

“empresa publica” && “COPEL energia”

O operador “+” indica a obrigatoriedade do termo que o seguir existir em algum lugar no documento. Para pesquisar documentos que devem conter “empresa” e podem conter “publica”, utilize:

+empresa publica

O operador NOT filtra os documentos que contiverem o termo que o suceder. Isto equivale a uma diferença entre conjuntos. Para pesquisar por documentos que contenham “empresa publica” mas não documentos que contenham “COPEL” utilize:

“empresa publica” NOT “COPEL”

ou

“empresa publica” ! “COPEL”

Observação: O operador NOT não deve ser utilizado para somente um termo, por exemplo, a busca a seguir não obterá efeito:

NOT “empresa publica”

O operador “-” indica a proibição da existência do termo que o seguir, nos documentos

pesquisados. Para pesquisar documentos que contenham “empresa publica” mas não contenham “COPEL” utilize:

“empresa publica” -“COPEL”

## Agrupamento

Lucene suporta a utilização de parênteses para agrupar cláusulas que formem subpesquisas. Isso pode ser útil se desejar controlar logicamente uma pesquisa. Para pesquisar por qualquer um dos termos “empresa” ou “publica” e “COPEL” utilize:

(empresa OR publica) AND COPEL

Com isso se consegue evitar alguma confusão tornnando mais claro que COPEL deve existir e algum dos termos empresa ou publica podem existir.

## Caracteres Especiais

Lucene suporta forçar caracteres especiais como parte de um termo da busca. Os caracteres especiais são:

+ - && || ! ( ) { } [ ] ^ ” ~ * ? : \

Para forçar o uso desses caracteres como parte do termo, utilize o símbolo “\” antes do caractere a ser forçado. Por exemplo, para buscar por (1+1):2 utilize:

\(1\+1\)\:2

# 6 REFERÊNCIAS

http://jackrabbit.apache.org/
http://wiki.apache.org/jackrabbit/
http://www.jcp.org/en/jsr/detail?id=170

# 7 GLOSSÁRIO

- **JCR** – JCR é uma abreviação para JSR 170: Content Repository for Java technology API, (API de Repositório de Conteúdo para tecnologia Java), uma interface padrão para acesso a repositórios de conteúdo.
- **Repositório de Conteúdo** – É um sistema gerenciador de informação que fornece vários serviços para armazenar, acessar e gerenciar conteúdo. Em adição ao sistema de armazenagem hierarquicamente estruturada, outros serviços comuns de um repositório de conteúdo são: versionamento, controle de acesso, pesquisa de texto em arquivos binários e monitoração de eventos. Um repositório de conteúdo não é um Sistema Gerenciador de Conteúdo, entretanto a maior parte de SGC existentes contêm uma implementação de recursos de repositório de conteúdo. Um SGC usa um repositório de conteúdo como um componente interno para suas necessidades de apresentação, regras e outras.
- **Jackrabbit** – O projeto Jackrabbit iniciou quando a empresa Day Software, um dos líderes da especificação JSR-170, licenciou sua implementação inicial de referência a JCR. Desde então o código de Jackrabbit tem sido utilizado como referência oficial de implementação (RI) e kit de compatibilidade de tecnologia (TCK - technology compatibility kit) liberado no decorrer da finalização da API JCR.
- **Portlet** – São componentes web, como servlets, especificamente desenhados para serem agregados no contexto de uma página web, que processam requisições e geram conteúdo dinâmico. O usuário final basicamente visualiza um portlet como sendo uma área de conteúdo especializado, dentro de uma página web, que ocupa uma pequena porção da tela na página de um portal.
- **UUID** – Identificador Universalmente Único (Universally Unique Identifier). Um UUID é essencialmente um número de 16 bytes e seu formato é parecido com

  550E8400-E29B-11D4-A716-446655440000.